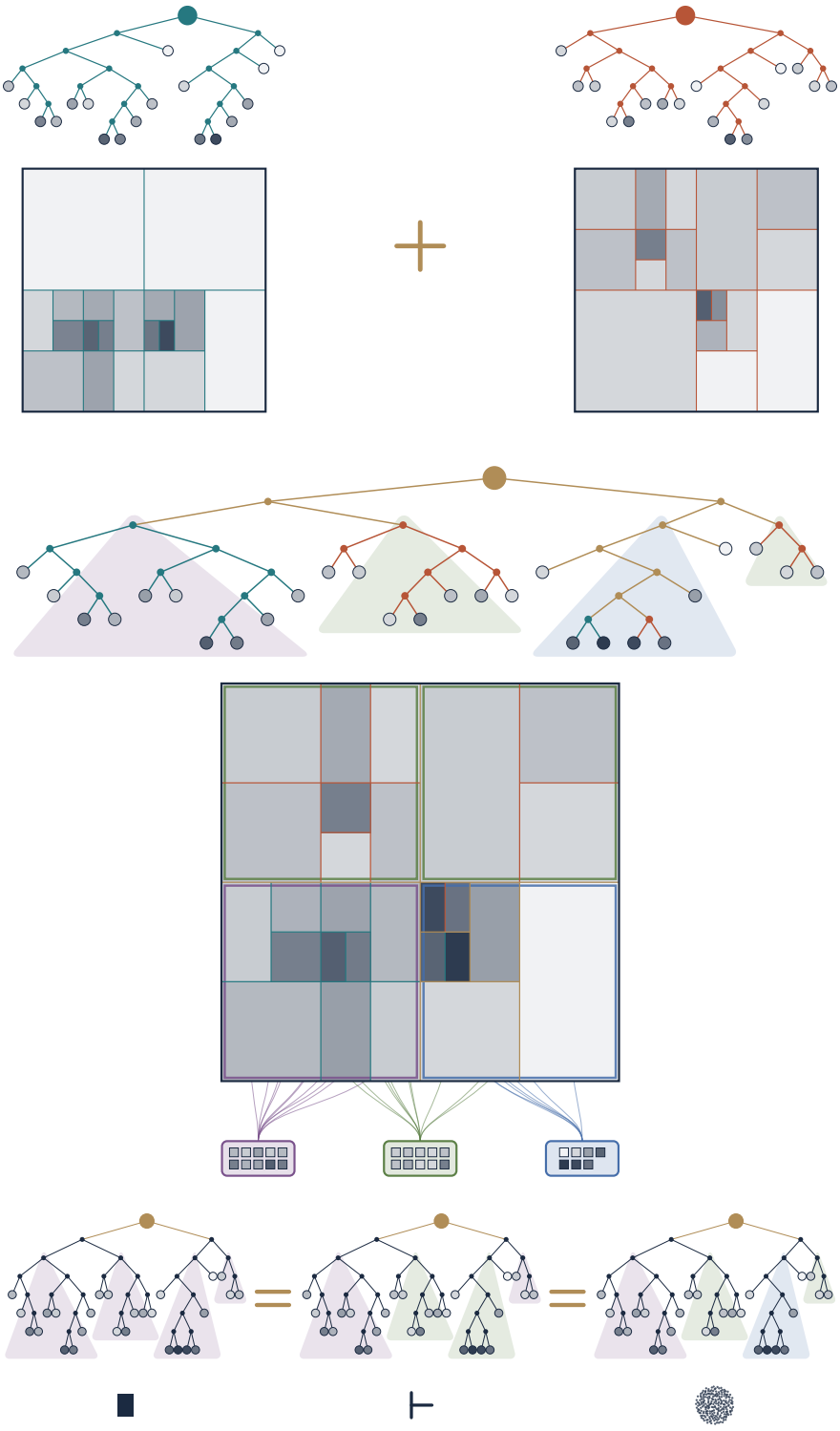
A 1ArchCodeMathsProveDog Book

Arithmetic, Algebra and Analysis with Scalable Sparse Catalan Binary Trees for Reliable, Statistical and Distributed Computing

Proved, Machine-Checked and Simulated



Raazesh Sainudiin — An \|/*ias*/|\ Commons Continuum
Laboratory for Mathematical Statistical Experiments, Research Division, VakeWorks

Arithmetic, Algebra and Analysis with Scalable Sparse Catalan Binary Trees for Reliable, Statistical and Distributed Computing

Proved, Machine-Checked and Simulated

Raazesh Sainudiin — An \|/*ias*/|\ Commons Continuum
Laboratory for Mathematical Statistical Experiments, Research Division, VakeWorks

November 17, 2026

source date 2026-11-17T16:35:57Z · source commit 96a8411c7178 · sources sha256:d4231aed248ae27f · this PDF's checksum: see the VERIFY file published beside it

Abstract

This book is about one data structure and what it lets a computer do with data and with probability. The structure is a finite binary tree, in which every branching point splits into exactly two. Such a tree records how a box (an interval, a rectangle, or its analogue in more dimensions) is cut in half at the midpoint of a widest side, and how the halves are cut again: finely where something happens and coarsely elsewhere. Each cell at a tip of the tree, a leaf, carries a value, such as a count of data points or a probability. Every cell is named by a short string of bits, 0 for the left half and 1 for the right. So the tree can be stored sparsely, by its labelled leaves alone, and split into pieces that live on different machines.

The same trees serve the three kinds of computing that the title names. *Reliable computing:* a sampler draws random samples exactly from a probability density given by a formula, enclosing the density with the rigorous bounds of interval arithmetic. When its bounds are too loose to succeed within its budget, it returns no sample rather than a wrong one. *Statistical computing:* histograms of massive data, such as the trajectories of aircraft; posterior averages of histograms over the space of such trees; the co-trajectories of moving objects, whose parts can be swapped to hinder re-identification while every decision based on a chosen statistical model of the data stays the same; and an estimator of an unknown density from data. Among the histograms it grows, the estimator picks one whose error, in the L_1 distance, is, with high probability, within a constant factor of the best of them plus an estimation term, without assuming the shape of the unknown density. *Distributed computing:* the estimator, in its final form, runs on many machines at once and computes exactly what one machine would. Its answer holds at the resolution that the data and the machines allow, and it says which resolution it reached.

Arithmetic, algebra and analysis are the mathematics underneath. The arithmetic adds, subtracts and collates these tree-valued functions exactly from the trees and their labels, and on the regular paving marginalises and slices them too. The algebra is that of Catalan binary trees: they are counted by the Catalan numbers, their pairs represent the elements of Thompson's group F, and cutting boxes in half is one realisation of them among many. The analysis gives the L_1 guarantees and the algorithm's cost on real machines.

The work began in 2002 and is told in the order in which it happened, which is also the clearest order in which to learn it. Every theorem about the final distributed algorithm is checked three ways, each recorded by a mark: proved on paper, machine-checked in the Lean proof assistant, and checked by simulation. Where a mark is missing, the theorem says which and why.

Copyright

© 2026 VakeWorks AB. The text, mathematics, figures and student exercises of this book are licensed under the Creative Commons Attribution–ShareAlike 4.0 International License (CC BY–SA 4.0), https://creativecommons.org/licenses/by-sa/4.0/. Runnable code is licensed separately (Apache-2.0, AGPL-3.0-or-later unless noted); product and company names are trademarks and are not covered by the content licence. See the LICENSE file accompanying the book source.

Contents

1 How to read this monograph **7**

2 Experiments on machines **9**
2.1 Two limits . . . 9
2.2 Enclosures . . . 9
2.3 The machine interval experiment . . . 10

3 Simulation: the Moore rejection sampler **11**
3.1 Rejection . . . 11
3.2 The Moore rejection sampler . . . 11
3.3 Trans-dimensional targets . . . 12
3.4 No sample rather than a wrong one . . . 12
3.5 Markov chains at a resolution . . . 12
3.6 Distributed exact sampling . . . 13

4 Catalan binary trees, regular pavings and labels **14**
4.1 Catalan binary trees . . . 14
4.2 Split paths and the Catalan coefficient . . . 17
4.3 Distributions on Catalan binary trees . . . 19
4.4 Pairs of trees and Thompson's group F . . . 23
4.5 Regular pavings . . . 24
4.6 Labels and orders . . . 27
4.7 Other cut rules and other domains . . . 30
4.8 Counting and random regular pavings . . . 31

5 Mapped trees **33**
5.1 Cells, values and the function a mapped tree represents . . . 33
5.2 The algebra . . . 34
5.3 The canonical form . . . 36
5.4 Approximation . . . 37
5.5 Interval enclosures . . . 42
5.6 Kinds of values . . . 44
5.7 Sparse and labelled mapped trees . . . 44

6 The arithmetic, algorithm by algorithm **46**
6.1 Spark operations and partition invariance . . . 47
6.2 The union and the common refinement . . . 49
6.3 Point location and the pointwise image . . . 53
6.4 MinimiseLeaves . . . 54
6.5 The pointwise operation . . . 56
6.6 Unary maps: MRPTransform . . . 58

6.7 Pushforward: relabelling, coarsening, counting and folds . . . 59
6.8 Marginals . . . 62
6.9 Slices and conditional densities . . . 64
6.10 Collation and averages . . . 66
6.11 Labelled arithmetic over a forest of root boxes . . . 67
6.12 Level sets, highest density regions and a centre-outward median . . . 80
6.13 UniformApprox . . . 82
6.14 Split, propagate hulls, prune . . . 84
6.15 What runs where . . . 88

7 Statistical regular pavings in use **91**
7.1 The statistical regular paving . . . 91
7.2 Trajectories as pavings . . . 92
7.3 Arithmetic on trajectories . . . 92
7.4 A dynamic airspace . . . 94
7.5 Posterior-mean histograms . . . 94
7.6 What sparse labels change . . . 97
7.7 Other uses of the arithmetic . . . 98
7.8 Co-trajectories, privacy and decision problems . . . 98

8 Estimation I: consistent histograms **102**
8.1 The priority queue SEBTreeMC . . . 102
8.2 Threshold pavings, from above and from below . . . 103
8.3 The theorem of Lugosi and Nobel . . . 107
8.4 SEBTreeMC is consistent . . . 108
8.5 Consistency by coarsening . . . 110
8.6 Counts already on a grid . . . 115
8.7 The braiding: the queue on the uniform law . . . 115

9 Estimation II: minimum distance **123**
9.1 The distance . . . 123
9.2 Where the method comes from . . . 123
9.3 The minimum distance estimate and its guarantees . . . 124
9.4 Computing it on pavings . . . 126
9.5 The zoom-in search . . . 126
9.6 Simulation and the theorems: `CheckBounds` . . . 127
9.7 What the estimator does . . . 128

10 The final distributed algorithm **129**
10.1 Where it comes from . . . 129
10.2 The pipeline . . . 129
10.3 Subtree partitioning . . . 132
10.4 The search and its guarantees . . . 133
10.5 The code at `c0a1837`, against the final algorithm . . . 137

11 Exactness of the distributed computation **139**
11.1 The merge walk . . . 139
11.2 Safe depth limits . . . 144
11.3 Without condition (G) . . . 145
11.4 The backtrack . . . 148
11.5 The Yatracos counts . . . 148
11.6 The rounds of the search . . . 153

11.7 The conditions on the execution, and the theorem . . . 155
11.8 The other refinements . . . 158

12 Resolution and resources **161**
12.1 Condition (G) and the (G)-depth . . . 161
12.2 Resources . . . 163
12.3 When (G) fails at every representable depth . . . 163
12.4 What a run estimates at resolution D . . . 165
12.5 The selection checks its own resolution . . . 166
12.6 The duality with the auto-validating sampler . . . 167

13 Cost on real machines **169**
13.1 Layer 1: the record . . . 170
13.2 Layer 2: the machine . . . 171
13.3 Layer 3: the operators . . . 174
13.4 Layer 4: the pipeline . . . 176
13.5 Layer 5: calibration . . . 179

14 Scale: our own runs **182**
14.1 The runs of the final algorithm . . . 182
14.2 Earlier runs . . . 183
14.3 What the earlier runs do not establish . . . 184

15 Machine-checked certificates **185**
15.1 The decision-theory chain . . . 185
15.2 The specification of the pipeline . . . 187
15.3 The cost model . . . 199
15.4 What is not checked . . . 200

16 Directions **202**
16.1 Trans-dimensional estimation . . . 202
16.2 Many samples, many treatments . . . 202
16.3 A revived C++ library . . . 203
16.4 Moving objects: air traffic and images . . . 203
16.5 The surface of the sphere . . . 203
16.6 The same trees, other cuts . . . 205
16.7 Data near lower-dimensional sets . . . 205
16.8 When the coarsening conditions are necessary . . . 205
16.9 Tighter bounds for the search . . . 206
16.10 Conditional densities for sequence models . . . 206
16.11 Contacts and transmission trees . . . 206
16.12 Halo-aware subtree sampling . . . 207
16.13 Adaptive approximation, and values beyond constants . . . 207
16.14 The selected leaf count as a complexity measure . . . 208

A Algorithms: index and pseudocode **217**
A.1 Conventions . . . 218
A.2 The Moore rejection sampler . . . 218
A.3 Statistical regular pavings, sparse form . . . 218
A.4 The stay-split-merge chain on pavings . . . 219
A.5 The Yatracos matrix along one path, and the objective . . . 219
A.6 The zoom-in search . . . 220

A.7 `CheckBounds`, as the repair runs it . . . 220
A.8 The final distributed algorithm . . . 220
A.9 Subtree partitioning . . . 221
A.10 The gap search . . . 222
A.11 The rows stated in the body . . . 222
A.12 SEBTreeMC . . . 223
A.13 The merge walk $W(\delta)$ . . . 223
A.14 The (G)-depth, in two passes . . . 224

B Proofs **225**
B.1 The weighted walk . . . 225
B.2 Longest-processing-time assignment . . . 228
B.3 The lookup . . . 230
B.4 From sampled loads to received cells . . . 231

Chapter 1

How to read this monograph

This monograph is about one data structure and what it lets a machine do with statistical decision problems: estimation and testing, in the decision-theoretic sense of [Sai05, §1.1.1]. The data structure is a finite rooted binary tree whose leaves carry values: a *mapped* binary tree. Its nodes are named by short binary labels, and it is stored sparsely, by its labelled leaves alone. The problems it serves are: - drawing samples that are exact in distribution; - estimating an unknown density from data, with guarantees in L_1; - comparing, combining and conditioning such densities; - doing all of this on one machine or on many.

The work began in 2002 and has run for more than twenty years. It is told here in the order in which it happened, because that order is also the clearest order in which to learn it.

Simulation came first. A computer works on a finite screen of floating-point numbers. A statistical experiment run on a computer must respect two limits: what can be observed and what can be computed (Chapter 2). Interval arithmetic makes those limits rigorous [Moo66, JKDW01]. For a function given by an arithmetical expression whose interval extension is well defined on a box, it computes an interval guaranteed to contain the function's range over the box.

The Moore rejection sampler is built on such enclosures [Sai05, SY13]; "Moore" honours Ramon E. Moore, a founder of interval analysis. The sampler: - partitions its domain into boxes by bisecting a side of maximal width; - puts a constant ceiling over each box; - proposes from the resulting histogram.

Its samples are exact, its domain may have several labelled parts of different dimensions, and when its enclosures are too loose to accept within its budget of trials, it returns no sample rather than a wrong one (Chapter 3).

The partitions became the object. Partitions of a box by repeated bisection form a *regular sub-paving* (Jaulin, Kieffer, Didrit and Walter [JKDW01]). With values attached to the leaves, the tree becomes a mapped regular paving, and an exact arithmetic on it needs only the tree [HST12]: adding, subtracting, collating, marginalising, slicing (Chapters 4 to 6). It was implemented in the C++ library MRS [SYH+18], whose name first stood for the Moore rejection sampler. On it were built: - histograms of massive data, such as the trajectories of aircraft [TKS12] (Chapter 7); - posterior means of random histograms over the space of trees [STHL13] (Chapter 7); - a randomised priority queue that grows a histogram consistently in L_1 [THS14] (Chapter 8); - a selection rule among the histograms on its path that satisfies an oracle inequality: within a constant factor of the best histogram on the path, plus an estimation term [ST19] (Chapter 9).

Estimation is the inverse problem of simulation. The sampler builds a histogram over a binary tree *from a density*, from above. The estimator builds one *from data*. The tree, the labels and the arithmetic are the same. So is the honesty about resolution: - a sampler whose

enclosures are too loose returns no sample within its budget; - an estimator that cannot resolve its data finely enough says at which resolution its answer holds (Chapter 12).

Distribution came last. - Stored as a sorted list of labelled leaves, the tree is sparse and can be cut into contiguous partitions, several per machine. - The Spark line of the work began with integer node labels [STW20] and grew in two theses [Gra22, San23] (Chapter 10). - Chapter 10 gives the distributed algorithm in its final form, with the proved refinements built in. - Chapter 11 proves that it computes exactly what one machine would. - Chapter 13 costs it on real machines, layer by layer. - Chapter 14 will report it run at terabyte scale.

Three marks. Every theorem about the final algorithm is checked three ways, each recorded by a mark: - **proved**, in the text or in Appendix B; - **certified**, by a machine-checked proof in Lean (Chapter 15); - **simulated**, checked by simulation.

Where a mark is missing, the theorem says which and why. Checking the mathematics by simulation is a habit of this line of work: MRS checked the bounds of its minimum-distance estimates by simulation (§9.6). The loop is iterative: prove, simulate, learn from the simulation, and prove again.

The binary trees are the deeper structure. The trees here are Catalan binary trees: finite rooted binary trees in which every internal node has two children. The regular paving is one geometric realisation of such a tree, by cutting a box at the midpoint of a side of maximal width. Other cut rules realise the same trees (Chapter 4). The counting of these trees, their distributions [CFGS15, SV15], and their pairs, which represent the elements of Thompson's group F [CFGS15], tie the work to combinatorics and algebra (Chapter 4, Chapter 16).

Decision problems. The methods of this monograph are made to be used in decisions. - One is the minimum distance selection of Chapter 9, whose guarantee holds for every density under the hypotheses of §9.3. - Another is the sufficient sanitizer of §7.8. It swaps parts of trajectories to hinder re-identification, while keeping the sufficient statistic of a chosen statistical experiment. Then "any decision based on the estimated probability model of the data will be the same as without sanitization" [SMT+20, §1].

What this monograph is not. It is not a survey of tree-based density estimation. Each idea is credited at the paper where it first appeared; the co-authors and students of those papers are named there.

How to read. - A reader who wants the data structure and its algorithms reads Chapters 4 to 7. - A reader who wants the estimator and its guarantees reads Chapter 8, Chapter 9, Chapter 10, Chapter 11 and Chapter 12. - A reader who wants to run it at scale reads Chapter 10, Chapter 13 and Chapter 14. - Appendix A collects every algorithm in pseudocode, in three forms: the MRS2 form, the sparse form and the Spark form. - The sections build on one another in this order: Chapter 2 → Chapter 3 → Chapter 4 → Chapter 5 → Chapter 6 → {Chapter 7, Chapter 8} → Chapter 9 → Chapter 10 → Chapter 11 → {Chapter 12, Chapter 13} → Chapter 14. Chapter 15 and Chapter 16 look back over all of them. - The distributed story in two pages is the two tables of §10.5: what the code at `c0a1837` does, step by step, against the final algorithm of Chapter 10.

Chapter 2

Experiments on machines

2.1 Two limits

The experiment. A statistical experiment is a triple $(\mathbb{X}, \mathcal{F}_{\mathbb{X}}, \mathcal{P})$: a sample space of the possible observations, a σ-algebra on it, and a family $\mathcal{P} = \{P_\theta : \theta \in \Theta\}$ of probability measures indexed by Θ [Sai05, Definition 2]. Given an observation x, the likelihood $\ell(\theta, x)$ is proportional to the density of x under P_θ [Sai05, Definition 3]. Run on a computer, the experiment meets two limits.

Limits on empirical resolution. An observation in the continuum is never known to infinite precision. "The empirically indiscernible sets, rather than the infinitely precise points, must be allowed to enter the statistical experiment as data" [Sai05, §1.1.2]. The dissertation's example is a light bulb's burn-out time recorded by a clock that only sounds on the hour, so that each datum is an interval between two hours [Sai05, §1.3.1].

Limits on numerical resolution. A computer represents real numbers on a finite screen $\mathcal{R}(2, p, \underline{e}, \overline{e})$ of floating-point numbers, with a mantissa of p bits and exponents between $\underline{e}$ and $\overline{e}$ [Sai05, §1.1.3]. A numerical inference procedure faces "at least five major sources of errors": - roundoff; - truncation of limits; - conversion, since 0.1 has no finite binary expansion; - an ill-posed, for instance non-identifiable, model; - the limits on empirical resolution.

A local search that evaluates a function at finitely many points cannot rule out that the function plunges between them [Sai05, §1.1.3].

2.2 Enclosures

Intervals and boxes. An interval $X = [\underline{x}, \overline{x}]$ stands for every real number between its bounds. Its width is $\mathrm{wid}(X) = \overline{x} - \underline{x}$. A box is a vector of intervals, and its volume is the product of its widths. The arithmetic operations and the elementary functions are extended to intervals, and on a machine they are computed with outward rounding, so that the computed interval always contains the exact result [Sai05, §1.2.4], [SY13, §3].

The natural interval extension of an arithmetical expression f replaces each variable by a box and each operation by its interval version [Sai05, Definition 9], [SY13, §3]. The fundamental theorem of interval analysis, due to Moore [Moo66], states: if the natural interval extension F of f is well defined on a box Y, then for boxes $X \subseteq X' \subseteq Y$, - **inclusion isotony:** $F(X) \subseteq F(X')$; - **range enclosure:** $\{f(x) : x \in X\} \subseteq F(X)$.

[SY13, Theorem 2], [Sai05, Theorem 4]

The second property is the one on which everything later rests. Over any box on which the

extension is defined, a machine can compute an interval *guaranteed* to contain the range of f, however f behaves between the points it could sample. A function f is *Lipschitz* (in the class $\mathcal{E}_L$ of the dissertation, Definition 12) when every sub-expression is Lipschitz. For such f, the width of the enclosure shrinks at least linearly with the width of the box [Sai05, Theorem 5], [SY13, Theorem 3].

2.3 The machine interval experiment

The dissertation's claim is that interval methods answer *both* limits at once [Sai05, §1.3]. Extending the likelihood $\ell(\theta, x)$ to its interval extension $L(\Theta, X)$, which is inclusion-isotone and encloses the range [Sai05, Theorem 6], turns an experiment on points into an experiment on boxes. Boxes of parameters enter, and so do boxes of data that carry the empirical resolution. The *machine interval experiment* is this extended experiment computed on the floating-point screen [Sai05, Definition 13]. With rigorous global optimisation it "allows a statistical decision made with the aid of a computer to be equivalent to a numerical proof of its global optimality" [Sai05, abstract].

Three open problems in computational statistics were solved with it [Sai05, abstract]: - parametric bootstraps of likelihood-ratio test statistics for finite mixtures; - rigorous maximum-likelihood estimates of the branch lengths of a phylogenetic tree with a fixed topology (which, the dissertation says, partly addresses an open problem in phylogenetics, §3.2.4); - Monte Carlo sampling from multi-modal targets with sharp peaks.

One remark of that work recurs in this monograph. In a machine interval experiment, "the estimator $\widehat{\Theta}_n$ may not converge in probability to a thin box even if the true measure θ^* is a singleton and all observations are measurable exactly, since the machine precision $p < \infty$. The diameter $d(\widehat{\Theta}_n)$ reflects the limit of numerical resolution" [Sai05, §1.3.3]. An honest estimator carries its resolution with it. Chapter 12 makes the same point for the L_1 estimator: when the data cannot be resolved at the depth they need, the estimate says at which depth it holds.

Chapter 3

Simulation: the Moore rejection sampler

3.1 Rejection

To draw from a density $p = p^*/N_p$ known only up to its normalising constant N_p, von Neumann's rejection method [vN63] needs: - a proposal density q from which samples can be drawn; - an envelope $\hat{g} \geq p^*$ proportional to q.

Draw $T \sim q$ and $H \sim \text{Uniform}[0, \hat{g}(T)]$; accept T if $H \leq p^*(T)$, else repeat. The accepted T has density p. The probability of acceptance is $A = N_p/N_{\hat{g}}$, the ratio of the areas under the two curves. The number of proposals per sample is geometric with success probability A, so its mean is $1/A = N_{\hat{g}}/N_p$ [Sai05, §2.3.1], [SY13, §2]. The exactness rests on one observation, made rigorous in the proofs cited below: a pair (T, H) drawn this way is uniform in the region under $\hat{g}$, so an accepted pair is uniform in the region under p^*, and its first coordinate has density p [Sai05, proof of Theorem 7], [SY13, proof of Theorem 1].

Everything depends on the envelope. When one substitutes floating-point for real arithmetic and builds the envelope from finitely many evaluations, "it is generally not possible to guarantee the envelope property and thereby ensure that samples are drawn from the desired target density" [SY13, §6]; a violated envelope gives samples from the wrong density, silently.

3.2 The Moore rejection sampler

Interval enclosures repair this [Sai05, §2.3.2], [SY13, §4].

Construction. 1. Partition the domain into boxes $\Theta^{(1)}, \ldots, \Theta^{(W)}$. 2. Enclose the range of p^* over each box by its interval extension, $p^*(\Theta^{(i)}) \subseteq [\underline{P}^*(\Theta^{(i)}), \overline{P}^*(\Theta^{(i)})]$. 3. Take as envelope the simple function $\hat{g} = \sum_i \overline{P}^*(\Theta^{(i)})\, \mathbf{1}_{\Theta^{(i)}}$, an upper histogram over the partition [Sai05, Eq. 2.1–2.3]. 4. Take as proposal its normalisation: choose a box with probability proportional to $\text{vol}(\Theta^{(i)})\, \overline{P}^*(\Theta^{(i)})$, then a uniform point in it. Walker's alias method draws the box in constant time per draw, after $O(W)$ preprocessing [SY13, §4.2.3 and §6].

Properties. - The envelope holds by construction, and the samples are exact [Sai05, Theorem 7], [SY13, Theorem 4]. - The acceptance probability is at least $\sum_i \text{vol}(\Theta^{(i)})\underline{P}^*(\Theta^{(i)}) / \sum_i \text{vol}(\Theta^{(i)})\overline{P}^*(\Theta^{(i)})$. - It tends to 1 at least linearly with the mesh of the partition, for a Lipschitz target [SY13, Theorem 4]. On an interval, for $p^* \in \mathcal{E}_L$ and a uniform partition into W pieces, it is $1 - O(1/W)$ [Sai05, Theorem 8]. - In d dimensions a uniform

partition into W boxes has mesh of order $W^{-1/d}$, so the rate is $1 - O(W^{-1/d})$. For example, $p^*(x, y) = x$ on $[0, 1]^2$ with W equal squares and exact enclosures gives $A = 1 - \Theta(W^{-1/2})$.

Adaptive refinement. The partition is refined adaptively, and the refinement is the seed of what follows. - **The 2005 rule:** bisect a box "along the side with the maximal diameter, only if" the width of its range enclosure exceeds a tolerance [Sai05, p.72]. An adaptive partition of $[-100, 100]^2$ into 150 rectangles for a two-component bivariate Gaussian mixture is made this way; by inspection of the figure it consists of midpoint bisections [Sai05, Fig. 2.8]. - **The 2013 rule:** a priority queue holds the boxes. The box with the largest uncertainty in the integral over it, $\mathrm{vol}(\Theta^{(i)}) \cdot \mathrm{wid}\, P^*(\Theta^{(i)})$, is bisected first, at the midpoint of its *first* side of maximal width. Refinement stops when the boxes reach a number within the machine's memory, or when the acceptance bound passes a threshold [SY13, §4.2.1–4.2.2].

With the "first side of maximal width" convention, the partitions are regular pavings in the sense of Chapter 4.

3.3 Trans-dimensional targets

Many targets of interest live on several domains of different dimensions: - a model in which two success probabilities are equal (one dimension) against one in which they differ (two); - a star tree on three leaves against a resolved one.

A *labelled* box carries the label k of its domain ${}^k\mathbb{T} \subseteq \mathbb{R}^{d_k}$. The trans-dimensional rejection sampler (Algorithm 1) proposes a labelled point and accepts it as before. For any envelope satisfying the envelope condition over the labelled domains, its samples are exact for the trans-dimensional density [SY13, Theorem 1]. With an envelope built from interval enclosures over labelled boxes, it is the Moore rejection sampler for any bounded density whose shape is a locally Lipschitz arithmetical expression [SY13, Theorem 4]. A *locally Lipschitz* expression is one whose interval extension is Lipschitz on every box of the domain [SY13, §3].

The sampler drew exact samples from densities previously considered unsamplable, among them the multivariate witch's hat, and from small trans-dimensional posteriors over binomial partition models and over phylogenetic tree spaces [SY13, §5]. The phylogenetic example used five tree spaces on three leaves: the star tree, three rooted clock trees, and the unrooted triplet [SY13, §5.8]. The labels return in Chapter 6 and Chapter 7: a label on a root box lets one paving arithmetic carry many objects at once.

3.4 No sample rather than a wrong one

The sampler's guarantee has a price, and the price is the lesson of this section. The sampler returns "possibly one sample" within a budget of trials [SY13, Algorithm 1]. When the target's expression is large, its interval extension can over-enclose the range badly, and the partition must become very fine before acceptance is efficient. In the paper's words: "one will not obtain samples in a reasonable time rather than produce samples from some unknown and undesired target" [SY13, §6]. The estimator of this monograph obeys the same rule (Chapter 12).

3.5 Markov chains at a resolution

The dissertation also built Metropolis–Hastings chains from the same enclosures [Sai05, §2.3.3–2.3.4].

The chain. A Metropolis–Hastings chain proposes a move from a base chain and accepts it with a probability that makes the target its stationary distribution [Sai05, §2.3.3]. Its distance from

stationarity in total variation is bounded, for an independent base chain, by Liu's inequality in terms of the constant $W_q = \max_i \pi_r(i)/q(i)$ [Sai05, Theorem 9].

The setting. On $\Theta = [0,1]$, choosing dyadic partitions "to simplify the notation and the analysis" [Sai05, pp.81–82]: - **target resolution:** the target is discretised as π_r on the partition of mesh 2^{-R}; - **probing resolution:** a coarser partition of mesh 2^{-S}, $S \leq R$, supplies the proposals through the enclosures of the target over its cells; - **the map between them:** a target cell i lies in the probing cell $K(i) = \lceil i/2^{R-S} \rceil$.

The general setting allows any partitions [Sai05, p.79].

The efficiency bound. For the independent Moore–Metropolis–Hastings *max* chain, with $p \in \mathcal{E}_L$ on $[0,1]$ and as $R \to \infty$, $W_q \leq 1 + O(2^{-S})$ [Sai05, Theorem 10].

An observation of this monograph, not of the source. After the shift $K(i) - 1 = \lfloor (i-1)/2^{R-S} \rfloor$, the map drops the last $R - S$ bits of the R-bit expansion of $i - 1$. It is the truncation of a depth-R dyadic label to depth S, the ancestor map that Chapter 4 introduces for the labels of a binary tree.

In practice, the dissertation notes, one keeps a chain's samples only "to count the number of visits to each bin in some appropriate binning" of the parameter space [Sai05, p.80]. So a finest resolution, coarsening by truncation, and counts on a partition (dyadic in the setting above), the working parts of the estimator of Chapters 8 to 14, are present on the sampling side in 2005.

3.6 Distributed exact sampling

A/B testing compares a model in which two treatments share a success probability, $\theta_A = \theta_B$ (one dimension), with a model in which they differ, $(\theta_A, \theta_B) \in [0,1]^2$ (two dimensions), so the posterior over the two models is trans-dimensional [AS18]. The Moore rejection sampler samples it exactly, "even when the posterior does not have analytic solution" [AS18], and without the posterior's normalising constant, which rejection never needs [SY13, §1.1].

Run on Apache Spark, the sampler is fault-tolerant and parallel [AS18]. It handles: - more than two treatments; - any trans-dimensional posterior given by a locally Lipschitz expression; - within the memory and processors of each node, data streaming in bursts through time.

That note states no theorem and reports no experiment. It marks the point at which the simulation side reached distributed computing, which the estimation side reaches in Chapter 10.

Chapter 4

Catalan binary trees, regular pavings and labels

The structure at the core of this monograph is a finite rooted binary tree in which every internal node has exactly two children. Such a tree is regular in the Catalan sense: the trees with k internal nodes are counted by the Catalan number C_k, they grow by splitting a leaf, and they shrink by merging a cherry. Cutting boxes at their midpoints is one way to turn such a tree into a partition of a box, the regular paving. It is not the only one, but its cuts are fixed in advance, and so the union of two trees is the overlay of their partitions (Remark 4.30). This section studies the trees first: their moves and their number (§4.1), the number of ways to grow each one (§4.2), the distributions that random growth induces (§4.3), and the pairs of trees that describe Thompson's group F (§4.4). It then realises the trees as regular pavings (§4.5), names and orders their nodes (§4.6), states what other cut rules keep (§4.7), and counts and randomises regular pavings (§4.8).

4.1 Catalan binary trees

Write the nodes of the infinite binary tree as ρ followed by a finite word over $\{\mathsf{L}, \mathsf{R}\}$. The children of u are $u\mathsf{L}$ and $u\mathsf{R}$, and $u\mathsf{L}$ is drawn to the left of $u\mathsf{R}$. The parent of $u \neq \rho$ is u with its last letter removed. The depth $|u|$ is the length of the word, so $|\rho| = 0$. For a node u and a word w, uw is the node whose word is that of u followed by w. Write $v \sqsubseteq u$ when $v = uw$ for some word w; then u is an ancestor of v, a strict one when $v \neq u$ (written $v \sqsubset u$). Two nodes are *nested* when one is an ancestor of the other, and *disjoint* otherwise, that is, when neither word is a prefix of the other.

Definition 4.1 (Catalan binary tree)**.** *A* Catalan binary tree s *is a finite set $V(s)$ of nodes that contains ρ, contains the parent of each of its nodes other than ρ, and contains either both children of a node or neither. Its* leaves $\mathcal{L}(s)$ *are the nodes of $V(s)$ with no child in $V(s)$, and its* internal nodes *are the others, $I(s) = V(s) \setminus \mathcal{L}(s)$. A* cherry *is an internal node whose two children are leaves. The* number of splits *of s is $|I(s)|$. Write $\mathbb{S}_k$ for the set of Catalan binary trees with k splits, and $\mathbb{S}_{0:\infty} = \bigcup_{k\geq 0} \mathbb{S}_k$. The* root tree*, with $V = \{\rho\}$, is the only element of $\mathbb{S}_0$.*

These trees go by several names. They are the plane binary trees of enumerative combinatorics [Sta99, Ex. 6.19(d), p. 220] and the finite rooted binary trees (frb-trees) of geometric group theory [Mei08, Ch. 10], as [HST12] note, and also the binary search trees of computer science [Mah92], as [CFGS15] add. The words are the addresses of [CFGS15]: they fix the planar embedding. The trees are called Catalan binary trees here, after the numbers that count them

(Proposition 4.5).

The *subtree* of s at $u \in V(s)$ is the Catalan binary tree $s[u]$ with nodes $\{\rho w : uw \in V(s)\}$: the nodes of s at or below u, renamed so that u becomes the root. If s has a split, its leaves are those of $s[\rho\mathsf{L}]$ followed by those of $s[\rho\mathsf{R}]$, each one level deeper. Listing them in this order, recursively, gives the *left-to-right order* of the leaves; §4.6 shows that it is the left-right order $\prec$ restricted to the leaves. The *leaf-depth string* of s lists the depths of its leaves in this order. For example, the tree with internal nodes $\rho, \rho\mathsf{L}, \rho\mathsf{LR}, \rho\mathsf{R}$ has the leaves $\rho\mathsf{LL}, \rho\mathsf{LRL}, \rho\mathsf{LRR}, \rho\mathsf{RL}, \rho\mathsf{RR}$, the leaf-depth string 23322, and the cherries $\rho\mathsf{LR}$ and $\rho\mathsf{R}$ [HST12].

Lemma 4.2 (Internal nodes, leaves and depths)**.** *Let s be a Catalan binary tree.*

- *(i) $V(s) = \{\rho\} \cup I(s) \cup \{u\mathsf{L}, u\mathsf{R} : u \in I(s)\}$. A finite set of nodes is $I(s)$ for some Catalan binary tree s exactly when it contains the parent of each of its elements other than ρ, and then s is unique. The empty set is I of the root tree.*
- *(ii) $|\mathcal{L}(s)| = |I(s)| + 1$ and $|V(s)| = 2|I(s)| + 1$.*
- *(iii) $\sum_{\ell \in \mathcal{L}(s)} 2^{-|\ell|} = 1$.*
- *(iv) The leaf-depth string of s determines s.*
- *(v) The leaf set determines s: $I(s)$ is the set of strict ancestors of the leaves of s.*
- *(vi) Every node of the infinite binary tree either lies in $I(s)$ or lies at or below exactly one leaf of s, and not both. In particular, if $I(s) \subseteq I(s')$, every leaf of s' lies at or below a leaf of s.*

Proof.

- *(i).* A node of $V(s)$ other than ρ is a child of its parent, which lies in $V(s)$ and has a child, so is internal. Both children of an internal node lie in $V(s)$. This gives the formula, and with it the uniqueness. The parent of an internal node $u \neq \rho$ lies in $V(s)$ and has the child u, so it is internal. Conversely, let I be a finite set that contains the parent of each of its elements other than ρ, and put $V = \{\rho\} \cup I \cup \{u\mathsf{L}, u\mathsf{R} : u \in I\}$. V contains ρ and the parent of each of its other nodes. Every element of I has both children in V. A node $w \in V \setminus I$ has no child in V: such a child would lie in I, and then its parent w would lie in I, or it would be a child of an element of I, and that element would be w. A node outside V has no child in V either, since V contains the parent of each of its nodes other than ρ. So every node has both children in V or neither, and V is a Catalan binary tree with internal nodes I. A finite set contains the parent of each of its elements other than ρ exactly when it is closed under taking ancestors.
- *(ii).* Each node other than ρ has exactly one parent, which is internal, and each internal node has exactly two children. So $|V(s)| - 1 = 2|I(s)|$, and $|\mathcal{L}(s)| = |V(s)| - |I(s)| = |I(s)| + 1$.
- *(iii).* By induction on the number of splits. The root tree has the one leaf ρ. Otherwise the leaves of s are those of $s[\rho\mathsf{L}]$ and of $s[\rho\mathsf{R}]$, each one level deeper, and these subtrees have fewer splits. The sum is $\frac{1}{2} + \frac{1}{2}$.
- *(iv).* By induction on the number of splits. The root tree has the string (0), and every other string has no entry 0. Otherwise the string of s is that of $s[\rho\mathsf{L}]$ followed by that of $s[\rho\mathsf{R}]$, each entry increased by one. By (iii) for $s[\rho\mathsf{L}]$, the first part is the prefix $(y_1, \ldots, y_j)$ with $\sum_{i \leq j} 2^{-y_i} = \frac{1}{2}$. The partial sums increase strictly, so this prefix is unique. The string thus determines the strings of the two subtrees, by induction the two subtrees, and so s.
- *(v).* By (i), every strict ancestor of a node of $V(s)$ lies in $V(s)$ and has a child there, so it is internal. Conversely, an internal node u is a strict ancestor of a leaf, reached from u by following left children, since $V(s)$ is finite. For the root tree both sets are empty.
- *(vi).* Let $u \notin I(s)$, and let v be the shallowest node with $u \sqsubseteq v$ and $v \notin I(s)$. Either $v = \rho$, and then $I(s) = \emptyset$ and v is the only leaf, or the parent of v lies in $I(s)$, and then $v \in V(s) \setminus I(s)$ is a leaf. Two distinct leaves are disjoint, since a strict ancestor of a node

of $V(s)$ is internal; so u lies below at most one leaf. A node $u \in I(s)$ lies below no leaf ℓ, since ℓ would then be an ancestor of an internal node, hence internal. For the last claim, a leaf of s' is not in $I(s') \supseteq I(s)$. □

Definition 4.3 (Split and merge)**.** *For a leaf ℓ of s, the* split *of s at ℓ is the tree $\bigtriangledown_\ell s$ with $V(\bigtriangledown_\ell s) = V(s) \cup \{\ell\mathsf{L}, \ell\mathsf{R}\}$. For a cherry c of s, the* merge*, or* reunion*, of s at c is the tree $\bigtriangleup_c s$ with $V(\bigtriangleup_c s) = V(s) \setminus \{c\mathsf{L}, c\mathsf{R}\}$.*

In terms of internal nodes, $I(\bigtriangledown_\ell s) = I(s) \cup \{\ell\}$ and $I(\bigtriangleup_c s) = I(s) \setminus \{c\}$. Both sets contain the parent of each of their elements other than ρ: the parent of ℓ is internal, and no internal node has the parent c, since the children of c are leaves. So both are Catalan binary trees (Lemma 4.2(i)). A split adds one internal node and one leaf, and a merge removes one of each. The two moves are inverse: ℓ is a cherry of $\bigtriangledown_\ell s$ and $\bigtriangleup_\ell \bigtriangledown_\ell s = s$, and c is a leaf of $\bigtriangleup_c s$ and $\bigtriangledown_c \bigtriangleup_c s = s$. In the notation of [HST12], splitting the root gives $\bigtriangledown(\rho) = \{\rho\mathsf{L}, \rho\mathsf{R}\}$, and merging its children gives back $\bigtriangleup(\rho\mathsf{L}, \rho\mathsf{R}) = \rho$.

Proposition 4.4 (The state space of trees)**.** *Make $\mathbb{S}_{0:\infty}$ a directed graph with an arrow $s \to \bigtriangledown_\ell s$ for each tree s and each leaf ℓ of s.*

- *(i) A tree in $\mathbb{S}_k$ has $k+1$ outgoing arrows, to $k+1$ distinct trees of $\mathbb{S}_{k+1}$.*
- *(ii) A tree in $\mathbb{S}_{k+1}$ has one incoming arrow for each of its cherries, from distinct trees of $\mathbb{S}_k$. Every tree with a split has a cherry.*
- *(iii) Every $s \in \mathbb{S}_k$ is reached from the root tree by a path of arrows, and every such path has k arrows.*

Proof. (i) The tree has $k+1$ leaves (Lemma 4.2(ii)), and distinct leaves ℓ give distinct sets $I(s) \cup \{\ell\}$. (ii) An arrow $s \to t$ splits a leaf ℓ of s; then ℓ is a cherry of t and $s = \bigtriangleup_\ell t$. Conversely, each cherry c of t gives the arrow $\bigtriangleup_c t \to t$. Distinct cherries c give distinct sets $I(t) \setminus \{c\}$. A deepest internal node is a cherry, since its children are deeper and so are not internal. (iii) By (ii), merging cherries one at a time takes s to the root tree in k steps; read backwards, these are k arrows. Each arrow adds one split. □

Table 4.1 shows the graph up to three splits. There may be more than one path to a tree: there are two paths to the tree 2222, splitting $\rho\mathsf{L}$ before $\rho\mathsf{R}$ or after [HST12]. The randomised algorithms of [HST12], and the priority queue SEBTreeMC of Definition 8.1, are Markov chains on this graph: they refine a partition by moving along the arrows. Markov chain Monte Carlo over pavings moves along them and against them [SYH+18]. The threshold and distributed forms of later sections reach the same trees, but not one split at a time.

Table 4.1: The Catalan binary trees with at most three splits, named by their leaf-depth strings, the trees that their splits reach, and their Catalan coefficients B (§4.2). The two paths to 2222 give $B(2222) = 2$.

k	tree in $\mathbb{S}_k$	its splits reach	B
0	0	11	1
1	11	221, 122	1
2	221	3321, 2331, 2222	1
2	122	2222, 1332, 1233	1
3	3321, 2331, 1332, 1233		1
3	2222		2

Proposition 4.5 (Catalan count)**.** *For every $k \geq 0$, $|\mathbb{S}_k| = C_k = \frac{1}{k+1}\binom{2k}{k}$, and $C_k \leq 4^k$.*

The count is classical [Sta99, Ex. 6.19(d)], and [HST12] states it for regular pavings. The short proof below is Rémy's bijection by grafting [R85].

Proof. For $s \in \mathbb{S}_k$, a node $v \in V(s)$ and a side $\sigma \in \{\mathsf{L}, \mathsf{R}\}$, with $\bar{\sigma}$ the other side, let $g(s, v, \sigma)$ be the tree obtained by inserting a new internal node at v. Each node vw of s becomes $v\sigma w$, the node v and the leaf $v\bar{\sigma}$ are added, and the nodes not below v are unchanged. So the old subtree at v hangs on side σ of the new node, whose other child $v\bar{\sigma}$ is a new leaf. The new node set contains ρ and the parent of each of its other nodes (the parent of $v\sigma w$ is $v\sigma w'$ when vw' is the parent of vw, and the parents of $v\sigma$ and $v\bar{\sigma}$ are v), and every node has both children in it or neither (those of $v\sigma w$ correspond to those of vw, v has both, and $v\bar{\sigma}$ has none). So the result is a Catalan binary tree. Its internal nodes are v and the moved internal nodes of s, so it lies in $\mathbb{S}_{k+1}$, and $v\bar{\sigma}$ is one of its leaves. Conversely, let $t \in \mathbb{S}_{k+1}$ and let ℓ be a leaf of t; then $\ell \neq \rho$, so $\ell = v\bar{\sigma}$ for a node v and a side $\bar{\sigma}$, and v is internal in t. Removing ℓ, and moving the subtree at $v\sigma$ up to v (each $v\sigma w$ becomes vw), keeps the same two properties by the same correspondence and leaves one internal node fewer, so it gives the one $s \in \mathbb{S}_k$ with $g(s, v, \sigma) = t$. So g is a bijection from the triples (s, v, σ), of which there are $2(2k+1)|\mathbb{S}_k|$ (Lemma 4.2(ii)), onto the pairs (t, ℓ), of which there are $(k+2)|\mathbb{S}_{k+1}|$. Hence $|\mathbb{S}_{k+1}|/|\mathbb{S}_k| = 2(2k+1)/(k+2)$ and $|\mathbb{S}_0| = 1$. The numbers C_k satisfy the same: $C_0 = 1$ and

$$\frac{C_{k+1}}{C_k} = \frac{(2k+2)(2k+1)}{(k+2)(k+1)} = \frac{2(2k+1)}{k+2}.$$

Finally $C_k \leq \binom{2k}{k} \leq \sum_j \binom{2k}{j} = 4^k$. □

4.2 Split paths and the Catalan coefficient

A tree can be grown in many orders. Counting the orders gives the coefficient that turns probabilities of growth histories into probabilities of trees, as binomial coefficients turn probabilities of sequences of coin tosses into probabilities of their sums [CFGS15].

Definition 4.6 (Split path, ranking and Catalan coefficient)**.** *A* split path *to $s \in \mathbb{S}_k$ is a path $s^{(0)} \to s^{(1)} \to \cdots \to s^{(k)} = s$ of arrows from the root tree $s^{(0)}$. A* ranking *of s is a bijection $r : I(s) \to \{1, \ldots, k\}$ with $r(u) < r(w)$ whenever u is a strict ancestor of w; a tree with a ranking is a* ranked tree*. The* Catalan coefficient *$B(s)$ is the number of split paths to s.*

Lemma 4.7 (Split paths are rankings)**.** *(i) Let $s \in \mathbb{S}_k$. Sending a split path to s to the ranking $r(u) =$ (the step at which u is split) is a bijection from the split paths to s onto the rankings of s. (ii) There are exactly $k!$ split paths with k arrows, so $\sum_{s \in \mathbb{S}_k} B(s) = k!$.*

Proof. (i) Splits only add nodes, so a node once internal stays internal. Along a split path to s, each step splits a leaf and makes it internal, so the k steps split the k internal nodes of s, each exactly once. A node other than ρ appears only when its parent is split, so an ancestor is split before its descendants, and r is a ranking. Conversely, given a ranking r, split $r^{-1}(1), \ldots, r^{-1}(k)$ in turn. At step i the node $u = r^{-1}(i)$ is present, since $u = \rho$ or its parent has a smaller rank and has been split, and it is a leaf, since its children appear only when it is split. The two constructions are inverse. (ii) A tree with i splits has $i+1$ leaves, so a split path with k arrows is a sequence of choices among $1, 2, \ldots, k$ leaves, and distinct sequences give distinct paths. Each path ends at one tree. □

Ranked trees are the finest of four resolutions of the trees [CFGS15]. Forgetting the ranks gives the Catalan binary trees, and forgetting in addition which subtree is on the left gives the shapes of §4.3. With k splits there are $k!$ ranked trees and C_k Catalan binary trees. The ranked

trees without left and right are counted by the Euler zigzag numbers, and the shapes by the Wedderburn–Etherington numbers [CFGS15].

Remark 4.8 (Ranked trees and permutations)**.** Reading the ranks of a ranked tree in symmetric (in-fix) order, the left subtree, then the node, then the right subtree, gives a permutation of $\{1,\dots,k\}$. This is a bijection between the ranked trees with k splits and the permutations of $\{1,\dots,k\}$, the increasing binary tree lifting [FS09, Ex. 17, p. 132]. Its inverse splits a word at its smallest letter, which ranks the root, and recurses on the words to its left and to its right. Indeed, the root has the smallest rank, and the in-fix reading lists the ranks of the left subtree before it and those of the right subtree after it. So any distribution on ranked trees is also a distribution on permutations. The Yule model of Proposition 4.16 gives the uniform one. The Beta-splitting model of Proposition 4.17 gives a family of distributions on permutations with two parameters, which contains the uniform one at $\alpha = \beta = 0$. The report proposes it for studying randomised algorithms, sorting algorithms among them, that usually assume uniformly random inputs [CFGS15].

Lemma 4.9 (Catalan coefficient)**.** *Let $s \in \mathbb{S}_k$. For $u \in I(s)$ let $k_s(u)$ be the number of internal nodes of s at or below u, and let n_u^{L} and n_u^{R} be the numbers of internal nodes of s at or below $u\mathsf{L}$ and $u\mathsf{R}$, so that $k_s(u) = 1 + n_u^{\mathsf{L}} + n_u^{\mathsf{R}}$.*

- *(i)* $B(s) = \dfrac{k!}{\prod_{u\in I(s)} k_s(u)} = \displaystyle\prod_{u\in I(s)} \binom{n_u^{\mathsf{L}} + n_u^{\mathsf{R}}}{n_u^{\mathsf{L}}}$.
- *(ii) B of the root tree is 1, and for $k \geq 1$, $B(s) = \sum_c B(\triangle_c s)$, the sum over the cherries c of s.*
- *(iii)* $\sum_{s\in\mathbb{S}_k} B(s) = k!$.
- *(iv) $B(s)$ does not change when the two subtrees at an internal node are exchanged.*

Proof.

- *(i).* By Lemma 4.7(i), $B(s)$ is the number of rankings of s. Let $k \geq 1$, $k_{\mathsf{L}} = n_\rho^{\mathsf{L}}$ and $k_{\mathsf{R}} = n_\rho^{\mathsf{R}}$, so $k_{\mathsf{L}} + k_{\mathsf{R}} = k - 1$. In a ranking the root has rank 1, since it is an ancestor of every other internal node. The ancestor conditions relate the root to every node, and otherwise only nodes of the same subtree. So a ranking of s is the choice of the k_{L} ranks among $2,\dots,k$ that go to the left subtree, together with a ranking of each of $s[\rho\mathsf{L}]$ and $s[\rho\mathsf{R}]$ in the relative order of its ranks. Hence

  $$B(s) = \binom{k-1}{k_{\mathsf{L}}} B(s[\rho\mathsf{L}])\, B(s[\rho\mathsf{R}]).$$

  Applying this at every internal node gives the product of binomial coefficients. For the first formula, use induction on k: an internal node of $s[\rho\mathsf{L}]$ has the same count k_s as the node of s it comes from, so $B(s[\rho\mathsf{L}]) = k_{\mathsf{L}}!/\prod k_s(u)$ over the internal nodes u of s below $\rho\mathsf{L}$, an empty product when $k_{\mathsf{L}} = 0$; likewise on the right. Then $B(s) = (k-1)!/\prod_{u\in I(s)\setminus\{\rho\}} k_s(u) = k!/\prod_{u\in I(s)} k_s(u)$, as $k_s(\rho) = k$.
- *(ii).* A split path to s ends with an arrow $\triangle_c s \to s$ for exactly one cherry c (Proposition 4.4(ii)). The split paths through $\triangle_c s$ are the split paths to $\triangle_c s$ followed by that arrow.
- *(iii).* Lemma 4.7(ii).
- *(iv).* The exchange maps internal nodes to internal nodes and keeps every count k_s; apply (i). □

The first formula of (i) is [CFGS15, Lemma 2.3], the product of binomial coefficients appears in [SV15], and (iv) is observed in [CFGS15]. The coefficient has several earlier names, which the

report [CFGS15] consolidates. It is the shape functional of a binary search tree [DF95, Cor. 4.1], whose proof the report's proof follows. For a rooted binary phylogenetic tree with $k+1$ labelled leaves, drawn as a plane tree s, it is the number of rankings of that phylogenetic tree [SS03, Prop. 2.3.2]. It can also be proved by counting linear extensions of a partial order [Sta97, Ch. 3, Ex. 1.b, p. 312]. The Catalan coefficients are sequence A185155 of the On-Line Encyclopedia of Integer Sequences, whose entry describes them as the numbers of distinct paths from the root tree to each of the C_k trees with k splits [Sai12]. Recursion (ii) is also how the coefficient is computed: the MRS2 example `CatalanCoeff` [SYH+18] stores each tree as its leaf-depth string, runs (ii) forward over the arrows of Proposition 4.4, adding the count of each tree into the counts of the trees its splits reach, and tabulates how many trees share each value. The MRS2 class `MultiTreeManager` finds the same values by brute force, growing one tree for every split path.

Example 4.10 (Catalan coefficients)**.** For the complete tree with 7 splits, whose 8 leaves all have depth 3, the counts k_s are $7, 3, 3, 1, 1, 1, 1$, and $B = 7!/(7 \cdot 3 \cdot 3) = 80$. Among the trees with 7 splits the value 80 occurs once. The tree with internal nodes $\rho, \rho\mathsf{L}, \rho\mathsf{R}, \rho\mathsf{LL}, \rho\mathsf{RR}$, with leaf-depth string 332233, has counts $5, 2, 2, 1, 1$ and $B = 5!/(5 \cdot 2 \cdot 2) = 6$; four of the 42 trees with 5 splits have $B = 6$ [CFGS15]. In Table 4.1, $B(2222) = 3!/(3 \cdot 1 \cdot 1) = 2$.

Proposition 4.11 (Trees with one split path)**.** *Let $k \geq 1$. Then $B(s) = 1$ exactly when the internal nodes of s form a chain $u_1 = \rho, u_2, \ldots, u_k$, each a child of the one before. There are 2^{k-1} such trees in $\mathbb{S}_k$.*

Proof. By the recursion in the proof of Lemma 4.9(i), $B(s) = 1$ exactly when $\binom{k-1}{k_\mathsf{L}} = 1$ and both subtrees have coefficient 1. The binomial coefficient is 1 exactly when one of the two subtrees has no internal node. By induction, $B(s) = 1$ exactly when every internal node has at most one internal child, that is, when $I(s)$ is such a chain. Each of $u_2, \ldots, u_k$ is the left or the right child of the node before it. □

The count 2^{k-1}, with its reason, is in [CFGS15].

At the other extreme, let $M_k = \max_{s \in \mathbb{S}_k} B(s)$. By the recursion in the proof of Lemma 4.9(i), the two subtrees of the root can be chosen independently, so $M_0 = 1$ and

$$M_k = \max_{0 \leq j \leq k-1} \binom{k-1}{j} M_j M_{k-1-j}.$$

The tree whose internal nodes are the first k nodes in heap order (ρ, then the nodes of depth one, and so on, each depth from left to right; the nodes with integer labels $1, \ldots, k$ of §4.6) attains the number of heaps on k elements, since a ranking of it is exactly a heap. So M_k is at least that number, the sequence A056971, $1, 1, 1, 2, 3, 8, 20, 80, 210, 896, 3360, 19200, 79200$ for $k = 0, \ldots, 12$ [OEI]. That M_k equals it was observed, with trees indexed by their leaves, in [CFGS15]. The recursion above, computed exactly, confirms it for every $k \leq 400$; no proof for all k is given here.

Under the Yule model of Proposition 4.16, a tree $s \in \mathbb{S}_k$ has probability $Q(s) = B(s)/k!$, a measure of its balance. Fill [Fil96] gives the limit in probability of $-\ln Q(S)/k$, both when S is uniform on $\mathbb{S}_k$ and when S follows the Yule model, and, under the Yule model, a limiting normal law for $-\ln Q(S)$, centred and scaled [Fil96, Thm. 4.1], as the report records [CFGS15].

4.3 Distributions on Catalan binary trees

Definition 4.12 (Splitting model)**.** *A* Markov splitting model *assigns to each tree $s \in \mathbb{S}_{0:\infty}$ a probability distribution $P(s, \cdot)$ on its leaves. It defines a Markov chain $S^{(0)}, S^{(1)}, \ldots$ on*

$\mathbb{S}_{0:\infty}$: *$S^{(0)}$ is the root tree, and $S^{(i)} = \bigtriangledown_\ell S^{(i-1)}$ with ℓ drawn from $P(S^{(i-1)}, \cdot)$. A split path $\pi = (s^{(0)}, \dots, s^{(k)})$ that splits $\ell_1, \dots, \ell_k$ in turn has probability $p(\pi) = \prod_{i=1}^{k} P(s^{(i-1)}, \ell_i)$. The model is* split-exchangeable *when $p(\pi)$ depends on π only through its last tree $s^{(k)}$.*

In a general splitting model the distribution of the next leaf may depend on the whole split path so far. A Markov splitting model is a choice, for each k and each of the C_k trees of $\mathbb{S}_k$, of a point in the simplex of distributions on $k+1$ leaves [CFGS15].

To forget left and right, let the *swap* at an internal node u exchange the subtrees at $u\mathsf{L}$ and $u\mathsf{R}$: it maps each node $u\mathsf{L}w$ to $u\mathsf{R}w$ and back, and fixes the other nodes. Two trees have the same *shape* when one is obtained from the other by finitely many swaps. Shapes are the rooted binary trees without left and right, the trees of [CFGS15] and the tree shapes of phylogenetics [SS03, §2.4]. An internal node u of s is a *symmetry node* when $s[u\mathsf{L}]$ and $s[u\mathsf{R}]$ have the same shape, and $\mathrm{sym}(s)$ is the number of symmetry nodes of s. A splitting model is *plane-invariant* when $\Pr\{S^{(k)} = s\} = \Pr\{S^{(k)} = s'\}$ whenever s and s' have the same shape.

Lemma 4.13 (Trees of one shape)**.** *Let $s \in \mathbb{S}_k$.*

- *(i) For $k \geq 1$, a tree $s' \in \mathbb{S}_k$ has the shape of s exactly when the shapes of $s'[\rho\mathsf{L}]$ and $s'[\rho\mathsf{R}]$ are those of $s[\rho\mathsf{L}]$ and $s[\rho\mathsf{R}]$, in some order.*
- *(ii)* sym *and B take the same value on all trees of one shape.*
- *(iii) The shape of s is the shape of exactly $2^{k-\mathrm{sym}(s)}$ trees in $\mathbb{S}_k$.*
- *(iv) There are exactly $B(s)\,2^{k-\mathrm{sym}(s)}$ split paths to trees of the shape of s.*

Proof.

- *(i).* A swap at ρ exchanges the two root subtrees. A swap at a node at or below $\rho\mathsf{L}$ is a swap of $s[\rho\mathsf{L}]$, which keeps its shape, and leaves $s[\rho\mathsf{R}]$ unchanged; likewise at or below $\rho\mathsf{R}$. So swaps keep the unordered pair of root-subtree shapes. Conversely, if the pairs agree, swaps at or below $\rho\mathsf{L}$ and $\rho\mathsf{R}$, followed if needed by a swap at ρ, turn s into s'.
- *(ii).* For sym, by induction on k: $\mathrm{sym}(s) = \mathrm{sym}(s[\rho\mathsf{L}]) + \mathrm{sym}(s[\rho\mathsf{R}]) + [\rho \text{ is a symmetry node}]$, and by (i) and the induction hypothesis the right side depends only on the unordered pair of root-subtree shapes. For B, Lemma 4.9(iv).
- *(iii).* By induction on k; the root tree is the only tree of its shape, and $\mathrm{sym} = 0$. For $k \geq 1$ let the root subtrees of s have the shapes A and A', with k_A and $k_{A'}$ splits, $k_A + k_{A'} = k-1$, and let $N(A)$ be the number of trees of shape A. By (i), the trees of the shape of s are the trees whose root subtrees have the shapes A and A' in some order. If $A \neq A'$ there are $2N(A)N(A')$ of them, and $\mathrm{sym}(s) = \mathrm{sym}(A) + \mathrm{sym}(A')$. If $A = A'$ there are $N(A)^2$, and $\mathrm{sym}(s) = 2\,\mathrm{sym}(A) + 1$. In both cases the induction hypothesis $N(A) = 2^{k_A - \mathrm{sym}(A)}$ gives $2^{k-\mathrm{sym}(s)}$.
- *(iv).* By (ii) and (iii). □

Theorem 4.14 (Split-exchangeable trees)**.** *Under a split-exchangeable Markov splitting model, $\Pr\{S^{(k)} = s\} = B(s)\,p(\pi)$ for every $s \in \mathbb{S}_k$ and every split path π to s.*

Proof. The event $\{S^{(k)} = s\}$ is the disjoint union, over the $B(s)$ split paths to s, of the events that the chain follows them, and each has the probability $p(\pi)$. □

Theorem 4.15 (Split-exchangeable, plane-invariant shapes)**.** *Under a split-exchangeable and plane-invariant Markov splitting model, the probability that $S^{(k)}$ has the shape of $s \in \mathbb{S}_k$ is $B(s)\,2^{k-\mathrm{sym}(s)}\,p(\pi)$, for every split path π to s.*

Proof. By plane-invariance, the $2^{k-\mathrm{sym}(s)}$ trees of this shape (Lemma 4.13(iii)) all have the

probability of s, which is $B(s)\,p(\pi)$ by Theorem 4.14. □

The report's point is that these theorems give the probabilities of trees, and of shapes, at the coarser resolutions from those of split paths [CFGS15]. Three models show the range. One of them needs the cells of a tree on $[0,1]$. The *dyadic interval* of a node u of depth h, whose word has the letters $b_1 \cdots b_h$ (reading L as 0 and R as 1), is $\mathbf{x}_u = [m2^{-h}, (m+1)2^{-h})$ with $m = \sum_j b_j 2^{h-j}$, closed on the right when $m = 2^h - 1$. A split replaces the interval of a leaf by its left and right halves, in this order. So, by Proposition 4.4(iii) and induction on the number of splits, the leaf intervals of any tree partition $[0,1]$, in the left-to-right order of the leaves. They are the cells of the regular paving of $[0,1]$ (§4.5).

Proposition 4.16 (Three splitting models)**.** *Let $s \in \mathbb{S}_k$.*

- *(i)* Yule. *$P(t,\ell) = 1/(j+1)$ for every leaf ℓ of $t \in \mathbb{S}_j$. Every split path with k arrows has probability $1/k!$, and the model is split-exchangeable and plane-invariant.* $\Pr\{S^{(k)} = s\} = B(s)/k!$, *and the shape of s has probability $2^{k-\mathrm{sym}(s)} / \prod_{u \in I(s)} k_s(u)$.*
- *(ii)* Statistically equivalent blocks (SEB). *Let f be a probability density on $[0,1]$ and $P(t,\ell) = \int_{\mathbf{x}_\ell} f$. A split path to s has probability $\prod_{u \in I(s)} \int_{\mathbf{x}_u} f$, so the model is split-exchangeable and* $\Pr\{S^{(k)} = s\} = B(s) \prod_{u \in I(s)} \int_{\mathbf{x}_u} f$. *For the uniform density these are $\prod_{u \in I(s)} 2^{-|u|}$ and $B(s)\, 2^{-\sum_{u \in I(s)} |u|}$, and the model is plane-invariant. For $f = 2 \cdot \mathbf{1}_{[0,1/2)}$ it is not.*
- *(iii)* Depth-proportional. *Split the root tree at ρ, and otherwise let $P(t,\ell) = |\ell| / \sum_{\ell' \in \mathcal{L}(t)} |\ell'|$. The model is not split-exchangeable.*

Proof.

- *(i).* At step i there are i leaves, so a split path with k arrows has probability $\prod_{i=1}^{k} 1/i = 1/k!$, which does not depend on the path. Theorem 4.14 gives $B(s)/k!$, which takes one value on each shape by Lemma 4.13(ii). Theorem 4.15 and Lemma 4.9(i) give the probability of the shape.
- *(ii).* The leaf intervals partition $[0,1]$, so $P(t,\cdot)$ is a probability. Along a split path to s, each internal node u of s is split exactly once, as a leaf of probability $\int_{\mathbf{x}_u} f$; the product does not depend on the order. For the uniform density, $\int_{\mathbf{x}_u} 1 = 2^{-|u|}$. A swap maps internal nodes to internal nodes of the same depth and keeps B, so the trees of one shape have one probability. For $f = 2 \cdot \mathbf{1}_{[0,1/2)}$ and $k = 2$, the tree with internal nodes $\rho, \rho\mathsf{L}$ has probability $1 \cdot 1 \cdot 1 = 1$, and the tree with internal nodes $\rho, \rho\mathsf{R}$, of the same shape, has probability 0.
- *(iii).* Take the tree with internal nodes $\rho, \rho\mathsf{L}, \rho\mathsf{R}, \rho\mathsf{RR}$ and leaf-depth string 22233; it has three split paths. Splitting $\rho, \rho\mathsf{L}, \rho\mathsf{R}, \rho\mathsf{RR}$ in turn has probability $1 \cdot \frac{1}{2} \cdot \frac{1}{5} \cdot \frac{2}{8} = \frac{1}{40}$, and splitting $\rho, \rho\mathsf{R}, \rho\mathsf{L}, \rho\mathsf{RR}$ has the same. Splitting $\rho, \rho\mathsf{R}, \rho\mathsf{RR}, \rho\mathsf{L}$ has probability $1 \cdot \frac{1}{2} \cdot \frac{2}{5} \cdot \frac{1}{9} = \frac{1}{45}$. □

Model (i) is Yule's model of speciation [Yul25] and the random permutation model of binary search trees [DF95], as the report notes. For $k = 3$, the shape with two cherries has probability $1/3$ under it, and the other shape $2/3$. Model (ii) is related to the statistically equivalent blocks of Gessaman [Ges70], a consistent partitioning rule in density estimation [CFGS15]. It draws the leaf to split at random, with probability its mass under f. The priority queue SEBTreeMC of Definition 8.1 is a different chain on the same trees: it splits a leaf of largest count, choosing uniformly among ties. In model (iii), deeper leaves split more often.

The fourth model cuts at random points instead of midpoints. Blum and François introduced an evolutionary Beta-splitting model [BF06], based on ideas of Kirkpatrick and Slatkin [KS93] and of Aldous [Ald01], and Sainudiin and Véber extended it to two parameters [SV15]. Write

$\mathcal{B}(a,b) = \int_0^1 x^{a-1}(1-x)^{b-1}dx$ for the Beta function, and let the Beta$(\alpha+1,\beta+1)$ law have the density $x^\alpha(1-x)^\beta/\mathcal{B}(\alpha+1,\beta+1)$ on $[0,1]$, for $\alpha,\beta > -1$.

Proposition 4.17 (Beta-splitting)**.** *Let $V_1, V_2, \dots$ be i.i.d. with the Beta$(\alpha+1,\beta+1)$ law, and $U_1, U_2, \dots$ i.i.d. uniform on $[0,1]$ and independent of them. Start from the root tree, with the root carrying the interval $[0,1]$. At step i, split the leaf whose interval contains U_i; if that interval has the ends $y_0 < y_1$ and $c = y_0 + (y_1 - y_0)V_i$, give its left child the interval $[y_0, c)$ and its right child the rest of it, $[c, y_1)$ or $[c, y_1]$. With $n_u^{\mathsf{L}}, n_u^{\mathsf{R}}$ as in Lemma 4.9:*

- *(i) a split path π to $s \in \mathbb{S}_k$ has probability*

$$p_{\alpha,\beta}(\pi) = \prod_{u \in I(s)} \frac{\mathcal{B}(n_u^{\mathsf{L}} + \alpha + 1,\, n_u^{\mathsf{R}} + \beta + 1)}{\mathcal{B}(\alpha+1,\beta+1)},$$

  which depends on π only through s; write it $p_{\alpha,\beta}(s)$;
- *(ii) the trees $S^{(0)}, S^{(1)}, \dots$ form the Markov splitting model with $P(t,\ell) = p_{\alpha,\beta}(\bigtriangledown_\ell t)/p_{\alpha,\beta}(t)$; it is split-exchangeable for all α,β, and plane-invariant exactly when $\alpha = \beta$;*
- *(iii) for $\alpha = \beta = 0$, every split path with k arrows has probability $1/k!$, as under the Yule model;*
- *(iv) as $\alpha = \beta \to \infty$, $p_{\beta,\beta}(\pi) \to \prod_{u\in I(s)} 2^{-(n_u^{\mathsf{L}} + n_u^{\mathsf{R}})} = \prod_{u \in I(s)} 2^{-|u|}$, the probability of π under the uniform SEB model of Proposition 4.16(ii).*

Proof.

- *(i).* The V_i lie in $(0,1)$ almost surely, so every interval of the construction is nonempty and the leaf intervals partition $[0,1]$, as in the dyadic case; the leaf containing U_i is unique. Write $r(w)$ for the step at which the internal node w is split. Given $V = (V_1, V_2, \dots)$, the interval of a node v has the length $\prod_w \lambda_w(v)$ over its strict ancestors w, where $\lambda_w(v) = V_{r(w)}$ if $v \sqsubseteq w\mathsf{L}$ and $1 - V_{r(w)}$ otherwise. U_i is uniform and independent of V and of $U_1, \dots, U_{i-1}$. So, given V and the first $i-1$ steps, step i splits a given leaf with probability its length. Multiplying over the k steps, the conditional probability of π is the product of the lengths of the intervals of the internal nodes of s. Collecting the factors by ancestor, it is $\prod_{w\in I(s)} V_{r(w)}^{n_w^{\mathsf{L}}}(1 - V_{r(w)})^{n_w^{\mathsf{R}}}$. The ranks $r(w)$ are distinct and the V_i are i.i.d., so the expectation is the product of $\mathbb{E}[V_1^{n_w^{\mathsf{L}}}(1-V_1)^{n_w^{\mathsf{R}}}] = \mathcal{B}(n_w^{\mathsf{L}} + \alpha + 1, n_w^{\mathsf{R}} + \beta + 1)/\mathcal{B}(\alpha+1,\beta+1)$, a quantity that depends on π only through s.
- *(ii).* Let π be a split path to t. By (i), the probability that the process follows π is $p_{\alpha,\beta}(t) > 0$, and the probability that it follows π and then splits the leaf ℓ is $p_{\alpha,\beta}(\bigtriangledown_\ell t)$. So, given that it has followed π, it splits ℓ with probability $p_{\alpha,\beta}(\bigtriangledown_\ell t)/p_{\alpha,\beta}(t)$, which depends on the past only through t; and these numbers sum to 1 over the leaves of t, since the events that the next step splits ℓ partition the event that the process follows π. So the trees form the Markov splitting model with this kernel, and by (i) it is split-exchangeable. For $\alpha = \beta$, each factor $\mathcal{B}(n_u^{\mathsf{L}} + \beta + 1, n_u^{\mathsf{R}} + \beta + 1)$ is symmetric in n_u^{L} and n_u^{R}, since $\mathcal{B}(a,b) = \mathcal{B}(b,a)$. A swap at u exchanges n_u^{L} and n_u^{R} and leaves the pair $(n^{\mathsf{L}}, n^{\mathsf{R}})$ of every other internal node, at its new position, unchanged; so $p_{\beta,\beta}$, and with Theorem 4.14 and Lemma 4.13(ii) the probability of a tree, take one value on each shape. For $\alpha \neq \beta$ and $k = 2$, using $\mathcal{B}(a+1,b)/\mathcal{B}(a,b) = a/(a+b)$, the tree with internal nodes $\rho, \rho\mathsf{L}$ has probability $(\alpha+1)/(\alpha+\beta+2)$ and the tree with internal nodes $\rho, \rho\mathsf{R}$, of the same shape, has $(\beta+1)/(\alpha+\beta+2)$.
- *(iii).* For integers $n, m \geq 0$, $\mathcal{B}(n+1, m+1) = n!\,m!/(n+m+1)!$ and $\mathcal{B}(1,1) = 1$. So $p_{0,0}(\pi) = \prod_{u\in I(s)} n_u^{\mathsf{L}}!\, n_u^{\mathsf{R}}!/k_s(u)!$. Each internal node $u \neq \rho$ is a child of exactly one internal node, and its count $k_s(u)$ appears there as n^{L} or n^{R}; a leaf child contributes $0! = 1$. All factorials cancel except $k_s(\rho)! = k!$ in the denominator.

- *(iv).* Let X have the Beta$(\beta+1,\beta+1)$ law. Then $\mathcal{B}(a+\beta+1,b+\beta+1)/\mathcal{B}(\beta+1,\beta+1) = \mathbb{E}[X^a(1-X)^b]$. X has mean $\frac{1}{2}$ and variance $1/(4(2\beta+3)) \to 0$, so $X \to \frac{1}{2}$ in probability, and since $x^a(1-x)^b$ is bounded and continuous on $[0,1]$, the expectation tends to 2^{-a-b}. Both $\sum_{u\in I(s)}(n_u^{\mathsf{L}}+n_u^{\mathsf{R}})$ and $\sum_{w\in I(s)}|w|$ count the pairs (u,w) of internal nodes with u a strict ancestor of w: every strict ancestor of an internal node is internal, and w has $|w|$ of them. Proposition 4.16(ii) gives the last equality. □

In the limit $\beta \to \infty$ balanced trees are favoured [SV15]. A split path to a comb, a tree whose internal nodes form a chain (Proposition 4.11), has probability $2^{-k(k-1)/2}$ in the limit. A split path to the complete tree with $k = 2^N - 1$ splits has probability $2^{-(N-2)2^N-2}$. In the words of [SV15], "for large β's the tree distribution is concentrated on balanced trees". For $k = 3$, for instance, the complete tree 2222 has probability $B \cdot 2^{-2} = \frac{1}{2}$ in the limit, against $\frac{1}{3}$ under the Yule model and $\frac{1}{5}$ under the uniform law on $\mathbb{S}_3$, while each of the four combs has probability $\frac{1}{8}$.

At the other end, as $\alpha = \beta \to -1$ the trees become combs. With X as in the proof of (iv), $\mathbb{E}[X(1-X)] = \frac{1}{4} - 1/(4(2\beta+3)) \to 0$. Since $x^a(1-x)^b \le x(1-x)$ for $a,b \ge 1$ and $0 \le x - x^a \le (a-1)\,x(1-x)$ on $[0,1]$, the factor of an internal node u in (i) tends to 0 when $n_u^{\mathsf{L}}, n_u^{\mathsf{R}} \ge 1$, to $\frac{1}{2}$ when exactly one of them is 0, and to 1 when both are. So a split path to a comb has limit probability $2^{-(k-1)}$, a split path to any other tree has limit 0, and each of the 2^{k-1} combs, with $B = 1$, has limit probability $2^{-(k-1)}$. [SV15] describe this limit through cuts that fall at 0 or 1.

4.4 Pairs of trees and Thompson's group F

Thompson's group F is the group, under composition, of the piecewise-linear orientation-preserving homeomorphisms of $[0,1]$ whose slopes are powers of two and whose breakpoints are dyadic rationals [CFP96, CFGS15]. Its elements are described by pairs of Catalan binary trees with the same number of leaves, through the dyadic intervals of §4.3.

Proposition 4.18 (Tree pairs)**.** *Let $s,t \in \mathbb{S}_k$, with leaves $\ell_1,\dots,\ell_{k+1}$ and $m_1,\dots,m_{k+1}$ in left-to-right order. Let $\varphi_{s,t}$ map each dyadic interval $\mathbf{x}_{\ell_i}$ increasingly and affinely onto $\mathbf{x}_{m_i}$.*

- *(i) $\varphi_{s,t} \in F$, with slope $2^{|\ell_i|-|m_i|}$ on $\mathbf{x}_{\ell_i}$.*
- *(ii) If s' and t' split the i-th leaf of s and of t, then $\varphi_{s',t'} = \varphi_{s,t}$.*
- *(iii) Every element of F is $\varphi_{s,t}$ for some pair [CFP96].*

Proof. (i) The leaf intervals of each tree partition $[0,1]$ in left-to-right order (§4.3), so the right end of $\mathbf{x}_{\ell_i}$ is the left end of $\mathbf{x}_{\ell_{i+1}}$, and likewise for t. Hence $\varphi_{s,t}$ is continuous and increasing, with $\varphi_{s,t}(0) = 0$ and $\varphi_{s,t}(1) = 1$. Its slope on $\mathbf{x}_{\ell_i}$ is the ratio of the widths, $2^{-|m_i|}/2^{-|\ell_i|}$, and its breakpoints are ends of dyadic intervals, which are dyadic rationals. (ii) An increasing affine map of an interval onto another maps the midpoint to the midpoint, so it maps the two halves of $\mathbf{x}_{\ell_i}$ affinely onto the two halves of $\mathbf{x}_{m_i}$. (iii) is cited. □

Example 4.19 (A tree pair)**.** The trees 221 and 122 of Table 4.1 give the partitions $(0, \frac{1}{4}, \frac{1}{2}, 1)$ and $(0, \frac{1}{2}, \frac{3}{4}, 1)$ of $[0,1]$ [CFGS15]. Their element of F maps $[0, \frac{1}{4}]$ onto $[0, \frac{1}{2}]$ with slope 2, $[\frac{1}{4}, \frac{1}{2}]$ onto $[\frac{1}{2}, \frac{3}{4}]$ with slope 1, and $[\frac{1}{2}, 1]$ onto $[\frac{3}{4}, 1]$ with slope $\frac{1}{2}$.

By Proposition 4.18(ii), many pairs give one element. The report lists three ways in which elements of F have been sampled at random [CFGS15]. The first samples words at random from the balls of the word metric for the generators $\{x_0, x_1\}$. The other two draw pairs of trees and reduce them, undoing splits made at the same position in both trees, before reading off the element: pairs of size k have been drawn uniformly [CERT10, CC13, CRW13], and by a

bifurcation process modelled on the Yule distribution [CPT15].

Remark 4.20 (The bias between two samplers of tree pairs)**.** Draw the two trees of a pair independently from $\mathbb{S}_k$. Under the uniform law on $\mathbb{S}_k$, the pair (s, t) has probability $1/C_k^2$. Under the Yule model run for k splits, it has probability $B(s)B(t)/(k!)^2$ (Proposition 4.16(i)). So, where uniform sampling gives every pair the weight 1, Yule sampling gives the pair (s, t) the weight $B(s)B(t)$, the number of ways in which the bifurcation process can produce it. In the words of the report, the Catalan coefficients "exactly describe the sampling bias between the two methods of tree pair generation for unreduced tree pair diagrams". The remark concerns the pairs as drawn, before reduction.

[HST12] point to the proof of [Mei08, Prop. 10.3] for the closure of these trees under union (Proposition 6.5) and its connection with Thompson's group. Further connections of the trees with algebra are left to Chapter 16.

4.5 Regular pavings

A regular paving realises a Catalan binary tree as a partition of a box by cutting each box at the midpoint of its first widest side [HST12, JKDW01]. An interval $\mathbf{x}_j$ with ends $\underline{x}_j < \overline{x}_j$ is here $[\underline{x}_j, \overline{x}_j)$ or $[\underline{x}_j, \overline{x}_j]$, and its width is $\mathrm{wid}(\mathbf{x}_j) = \overline{x}_j - \underline{x}_j$. A box is a product $\mathbf{x} = \mathbf{x}_1 \times \cdots \times \mathbf{x}_d$ of such intervals. Its *first widest coordinate* is $\iota = \min \operatorname{argmax}_j \mathrm{wid}(\mathbf{x}_j)$. Its *regular bisection* replaces $\mathbf{x}_\iota$, with midpoint $m = (\underline{x}_\iota + \overline{x}_\iota)/2$, by $[\underline{x}_\iota, m)$ in the *left child* box and by $[m, \overline{x}_\iota)$ or $[m, \overline{x}_\iota]$, as $\mathbf{x}_\iota$ is open or closed on the right, in the *right child* box, and keeps the other coordinates. The left child gets the half-open interval at the cut. The two children are disjoint, their union is the parent, and they have the same widths, those of the parent with the ι-th halved.

The root box $\mathbf{x}_\rho = [\underline{x}_1, \overline{x}_1] \times \cdots \times [\underline{x}_d, \overline{x}_d]$ is closed, with positive widths. Every box below it is then a product of half-open intervals, except that an interval reaching the upper end of the root box on its coordinate stays closed. MRS2 and the Spark implementation locate points by this convention: a point goes to the right child when its coordinate on the split axis is at least the midpoint. In MRS2 [SYH+18], `SPnode::nodeContains` (`spnode.cpp`) puts a point in the left child when it is strictly below the split value, and in the right child otherwise. In SparkDensityTree [SGWS23a], `quickDescendBox` (`SpatialTree.scala` lines 149–179 at commit `c0a1837`) takes the right child when `point(axis) >= mid` (line 170). In the example notebooks the root box is the box hull of the training and validation data. A half-open root box $[\underline{x}_1, \overline{x}_1) \times \cdots \times [\underline{x}_d, \overline{x}_d)$ makes every box a product of half-open intervals. The two conventions differ on the upper faces of the root box. That set has volume zero, so they give the same integrals and the same laws for densities. They do not give the same counts: when the root box is the box hull of the data, each coordinate attains its maximum at a sample point, which lies on an upper face and which the half-open box does not contain. The closed convention is the one that counts every point. The Lean module for Corollary 8.11 uses the half-open cube $[-1, 1)^d$, which suits a law with a density (Chapter 15).

Definition 4.21 (Splitting scheme)**.** *Write the nodes of the infinite binary tree as ρ followed by a word over $\{\mathsf{L}, \mathsf{R}\}$; $u\mathsf{L}$ and $u\mathsf{R}$ are the children of u, and the depth $|u|$ is the length of the word (§4.1). A* (binary) splitting scheme *on a set $\mathbf{x}_\rho$ assigns to every node u a cell $\mathbf{x}_u$, with $\mathbf{x}_u = \mathbf{x}_{u\mathsf{L}} \sqcup \mathbf{x}_{u\mathsf{R}}$ for every u. The scheme is* fixed *when it is chosen before the data are seen. A* paving *is a finite set of nodes that is the leaf set of a finite full binary tree rooted at ρ, that is, of a Catalan binary tree (Definition 4.1); its leaves partition $\mathbf{x}_\rho$. The scheme has* shrinking cells *when $\max_{|u|=k} \mathrm{diam}(\mathbf{x}_u) \to 0$ as $k \to \infty$.*

A paving determines its Catalan binary tree, whose internal nodes are the strict ancestors of the leaves (Lemma 4.2(v)), and the tree determines the paving. So a paving s has internal nodes

$I(s)$, leaves $\mathcal{L}(s)$ and nodes $V(s)$, those of its tree, and a paving with k splits is identified with a tree of $\mathbb{S}_k$; a paving is also determined by $I(s)$ (Lemma 4.2(i)), and, for a fixed scheme with nonempty cells, by its leaf cells (Proposition 4.33). The leaf cells partition $\mathbf{x}_\rho$ because a split replaces the cell of a leaf by the disjoint union of the cells of its children (Proposition 4.4(iii)). Cells along a path are nested, so the largest diameter at depth k does not increase with k. Cells shrink as soon as they are small at the depths of some subsequence, for example when dyadic equipartitions recur along a subsequence of depths.

Definition 4.22 (Regular paving)**.** *Bisecting a box at the midpoint of its first widest side gives a left and a right child box; the* regular scheme *on the root box $\mathbf{x}_\rho$ gives ρ the cell $\mathbf{x}_\rho$, and gives $u\mathsf{L}$ and $u\mathsf{R}$ the left and right child boxes of $\mathbf{x}_u$. A* regular paving *(RP) of $\mathbf{x}_\rho$ is a finite binary tree whose root is $\mathbf{x}_\rho$ and in which every internal node has exactly its two children: a Catalan binary tree s whose node u carries the box $\mathbf{x}_u$ of the regular scheme. Nodes are labelled by binary strings: the root is ρ, and the children of u are $u\mathsf{L}$ and $u\mathsf{R}$. The depth $|u|$ is the length of the label after ρ. The leaves $\mathcal{L}(s)$ of an RP s partition $\mathbf{x}_\rho$.*

Regular bisection is a fixed splitting scheme, and its cells shrink (Lemma 4.23). Figure 4.1 shows the construction on a square; its last paving is the tree with leaf-depth string 23322 of §4.1. The box of a node depends only on the node, not on the tree that contains it, and it has the volume $\mathrm{vol}(\mathbf{x}_u) = 2^{-|u|}\mathrm{vol}(\mathbf{x}_\rho)$, since each bisection halves the volume [HST12]. On $[0, 1]$ the boxes are the dyadic intervals of §4.3.

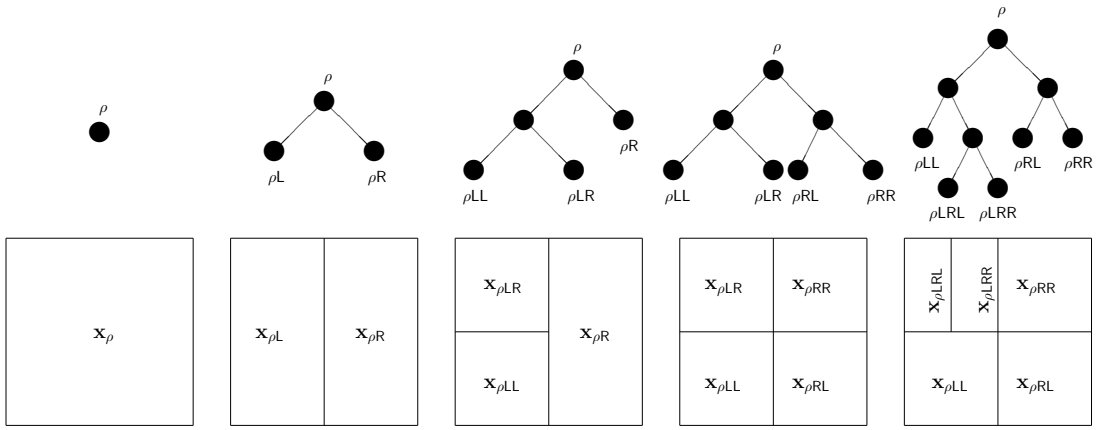


Figure 4.1: Successive bisections of boxes (nodes) along the first widest coordinate, starting from the root box (root node), produce a regular paving. From the joint manuscript [STW+26].

Lemma 4.23 (Aspect ratio and diameter)**.** *The aspect ratio of a box is its widest width over its narrowest. Let a_ρ be that of the root box, $a = \max(2, a_\rho)$, and $W_\rho = \max_j(\overline{x}_j - \underline{x}_j)$ the widest width of the root box.*

- *(i) Every box of the regular scheme has aspect ratio at most a.*
- *(ii) All boxes of one depth k have the same widths. So they are all split along one coordinate ι_k, the* split axis of depth k*, and the split axes to depth D are computed once, in $O(dD)$ operations.*
- *(iii) A box of depth k has volume $2^{-k}\mathrm{vol}(\mathbf{x}_\rho)$ and diameter at most $\mathrm{diam}_k = \sqrt{d}\, a\, (2^{-k}\mathrm{vol}(\mathbf{x}_\rho))^{1/d}$, which tends to 0. So the regular scheme has shrinking cells.*
- *(iv) Every width of a box of depth k is at most $W_\rho\, 2^{-\lfloor k/d \rfloor}$, so its diameter is at most $\sqrt{d}\, W_\rho\, 2^{-\lfloor k/d \rfloor}$.*

Proof. (i) By induction on depth. Bisection halves a widest width w. If $w/2$ is at least the

narrowest width, the narrowest width is unchanged and the widest does not increase, so the aspect ratio does not increase. Otherwise $w/2$ becomes the narrowest width while every width is at most w, so the aspect ratio is at most 2. (ii) The bisection of a box depends only on its widths, and the two children have the same widths; by induction on depth all boxes of one depth have the widths of the depth-k box on the root path of left turns, and the split axis is computed from those widths by halving one of them per depth. (iii) Each bisection halves the volume. Let w be the widest width of the box. Every width is at least w/a by (i), so $(w/a)^d \le 2^{-k}\mathrm{vol}(\mathbf{x}_\rho)$, and the diameter is at most $\sqrt{d}\,w \le \mathrm{diam}_k$. (iv) Let a box have widest width W, and follow any d further bisections below it. A width never grows, and a width of at most $W/2$ stays at most $W/2$. While some width exceeds $W/2$, the widest width exceeds $W/2$ and is at most W, and halving it makes it at most $W/2$; so each bisection lowers by one the number of widths above $W/2$, until there are none. There are at most d of them at the start, so after d bisections every width is at most $W/2$. By induction, after k bisections from the root every width is at most $W_\rho 2^{-\lfloor k/d \rfloor}$. □

The bound (iv) is the better one for a cube, where (iii) is $2\sqrt{d}\,W_\rho 2^{-k/d} \ge \sqrt{d}\,W_\rho 2^{-\lfloor k/d \rfloor}$, and for every root box with $a_\rho \ge 2^{d-1}$, since $\mathrm{vol}(\mathbf{x}_\rho) \ge W_\rho^d / a_\rho^{d-1}$. The bound (iii) is the one that Lemma 4.25 states; Theorems 5.7 and 5.17 use (iv).

Marks. Proved above. [LEAN: (i)–(iv) certified for a general root box, `ProveDog.ProveDog.RegularPaving.Aspect`, built at `bd5b09442` (Chapter 15): (i) `lem_aspect_i`; (ii) `lem_aspect_ii`, `lem_aspect_ii_same`, `widths_succ`, `firstWidest_le`, `box_cons` (all boxes of one depth have the width vector w_k and are split along one axis; the $O(dD)$ count is not modelled); (iii) the closed box of depth k has volume $2^{-k}\mathrm{vol}(\mathbf{x}_\rho)$, $\mathbf{x}_\rho$ closed, `lem_aspect_iii_volume`, and so has the half-open cell, `lem_aspect_iii_volume_cell`; the diameter as a bound on pairs of points of the closure, in the Euclidean distance, `lem_aspect_iii_diam`, `lem_aspect_iii_diam_closure`, `pair_le`; $\mathrm{diam}_k \ge 0$ and $\mathrm{diam}_k \to 0$, `diamK_nonneg`, `diamK_small`, `lem_aspect_iii_tendsto`; shrinking cells, `lem_aspect_iii_shrink`, `lem_aspect_iii_shrink_cell`; (iv) `lem_aspect_iv_width`, `lem_aspect_iv_diam`, `lem_aspect_iv_diam_closure`; with `widths_pos` and `narrowest_widths_pos`. The cells of the closed convention, which lie between the half-open cell and the closed box, are not sets of the module; nor are the comparison after the proof and the bridge to the cube of `VC.RegularPaving` (`small_of_deep`, the cube case of (iv)).] [SIM: the widths of the depth-k boxes, computed from the split axes, against (i), (iii) and (iv), for random root boxes with $d \le 10$ and $k \le 60$; to come.]

Projections of the regular scheme onto some of the coordinates are regular schemes again, and marginals and slices (§§ 6.8 and 6.9) rest on this.

Lemma 4.24 (Kept axes)**.** *Let $K \subseteq \{1, \dots, d\}$ be nonempty, renumbered in increasing order, let π_K be the projection onto the coordinates in K, and give $\mathbf{x}'_\rho = \pi_K \mathbf{x}_\rho$ its regular scheme, with cells $\mathbf{x}'_v$. Let $t_0 < t_1 < \cdots$ be the depths t with $\iota_t \in K$. Then $\iota_{t_0}, \iota_{t_1}, \dots$ are the split axes of the regular scheme of $\mathbf{x}'_\rho$, and for every node u, $\pi_K \mathbf{x}_u = \mathbf{x}'_{\kappa_K(u)}$, where $\kappa_K(u)$ is the node whose turns are the turns of u at the depths $t_j < |u|$, in order.*

Proof. By Lemma 4.23(ii), the widths of the boxes of depth t form one vector $w_t \in \mathbb{R}^d$, and ι_t is its first widest coordinate. The widths on K change only at the depths t_j. If $\iota_t \in K$, then ι_t attains the largest width over all coordinates, hence over K, and every coordinate of K before it has a smaller width, since ι_t is the first maximiser overall; so ι_t is the first widest coordinate of $\pi_K w_t$. Each coordinate is split infinitely often (Lemma 4.23(iv)), so the depths t_j form an infinite sequence. By induction on j, $\pi_K w_{t_j}$ is the width vector of depth j of the scheme of $\mathbf{x}'_\rho$, and ι_{t_j} its split axis. Along a node u, a turn at a depth t_j selects the lower or the upper half of the interval on coordinate ι_{t_j}, with the same convention at the cut on both sides, and a turn at any other depth leaves $\pi_K \mathbf{x}_u$ unchanged; by induction on $|u|$, $\pi_K \mathbf{x}_u = \mathbf{x}'_{\kappa_K(u)}$. □

Marks. Proved above. [LEAN: not yet authored (proposed with `Marginal`).] [SIM: $\pi_K \mathbf{x}_u$ against $\mathbf{x}'_{\kappa_K(u)}$ for random root boxes, random K and random nodes to depth 40; to come.]

For a scheme of boxes whose cuts are at midpoints along a split axis that depends on the depth alone, the last two sentences of the proof still give $\pi_K \mathbf{x}_u = \mathbf{x}'_{\kappa_K(u)}$, for the scheme of $\mathbf{x}'_\rho$ that cuts at depth j at the midpoint along ι_{t_j}: a fixed scheme, though not in general the regular one. `UniformSplitTree` is such a scheme (Remark 4.30).

Lemma 4.25 (Regular pavings approximate every density)**.** *Let s_k be the complete RP of depth k, whose leaves are the 2^k nodes of depth k, and let $\mathrm{Av}_k g$ be the function equal on each leaf of s_k to the average of g over that leaf. (i) The closures of the leaves of s_k are translates of one closed box, of diameter at most $\mathrm{diam}_k = \sqrt{d}\, a\, (2^{-k}\mathrm{vol}(\mathbf{x}_\rho))^{1/d}$, where $a = \max(2, a_\rho)$ and a_ρ is the aspect ratio of the root box. (ii) If f is continuous on $\mathbf{x}_\rho$, then $\sup |f - \mathrm{Av}_k f| \le \omega_f(\mathrm{diam}_k)$, with ω_f the modulus of continuity of f. (iii) For every density f on $\mathbf{x}_\rho$, $\int |f - \mathrm{Av}_k f| \to 0$ as $k \to \infty$. (iv) If f is constant on the leaves of s_{k_0}, then $\mathrm{Av}_k f = f$ for every $k \ge k_0$.*

Proof. (i) By Lemma 4.23(ii) the leaves of s_k have the same widths, so their closures are translates of one closed box, and by Lemma 4.23(iii) its diameter is at most diam_k. (ii) For x in a leaf C, $|f(x) - \mathrm{Av}_k f(x)| \le \sup_{y \in C} |f(x) - f(y)| \le \omega_f(\mathrm{diam}_k)$. (iii) Av_k is linear and positive and preserves the integral over each leaf, so $\int |\mathrm{Av}_k g| \le \int |g|$. Given $\epsilon > 0$, choose g continuous on $\mathbf{x}_\rho$ with $\int |f - g| < \epsilon$, as continuous functions are dense in L_1 of a box. As $\mathbf{x}_\rho$ is compact, g is uniformly continuous (Heine–Cantor), so $\omega_g(\delta) \to 0$ as $\delta \to 0$. Then $\int |f - \mathrm{Av}_k f| \le \int |f - g| + \int |g - \mathrm{Av}_k g| + \int |\mathrm{Av}_k(g - f)| \le 2\epsilon + \mathrm{vol}(\mathbf{x}_\rho)\, \omega_g(\mathrm{diam}_k)$, which is below 3ϵ for large k. (iv) The average of a constant is that constant. □

Marks. Proved above. [LEAN: (iv), and (iii) on the finest cells, certified (`IasCert.L1MDE.Approx`, Chapter 15); (i), (ii) and the limit in (iii) for a general density not yet authored.] [SIM: not needed.]

The lemma holds for every density, not only for continuous ones. Alternatively, $\mathrm{Av}_k f$ is the conditional expectation of f given the depth-k cells, a martingale in k. The depth-k cells generate the Borel sets of $\mathbf{x}_\rho$ as $k \to \infty$, since their diameters tend to 0 (Lemma 4.23(iii)), so the martingale converges to f in L_1 and almost everywhere. With a fixed strictly increasing bijection of each axis onto a bounded open interval, continuously differentiable with a positive derivative, such as $t(x) = x/(1 + |x|)$ (Chapter 1), the transformed law has a density and the L_1 error is unchanged [DL01, §5.3]; so the lemma covers every density on $\mathbb{R}^d$. It is the approximation half of the guarantees: the first term of the oracle inequality Theorem 9.1, $3 \min_\theta \int |f_\theta - f|$, compares the estimate with the best candidate, and the lemma says that regular pavings contain candidates as close to f as wanted. Whether the data find them is the estimation half (Theorem 8.9 and Corollary 8.11).

4.6 Labels and orders

Each node of an RP is named by the sequence of choices of child from the root: a string over $\{\mathsf{L}, \mathsf{R}\}$ after ρ [HST12], the address of [CFGS15]. The distributed histograms of [STW20] name a node by a positive integer instead: the root is 1, and the children of n are $2n$ and $2n + 1$.

Labels as the code encodes them. The implementation stores a node as `NodeLabel(lab: BigInt)` (`NodeLabel.scala` line 34 at commit `c0a1837`), the heap encoding: the root is 1, and the children of n are $2n$ (left) and $2n + 1$ (right) (lines 37–38). Below its leading one, the bits of `lab` are the turns from the root, the least significant bit being the deepest turn, so the depth is the bit length minus one (line 55), the sibling flips the last bit (lines 45–46), and the ancestor at depth t is a right shift (`truncate`, lines 60–63). The encoding is a bijection from the nodes onto

the positive integers. Write $u|_t$ for the ancestor of u at depth $t \le |u|$ (and $u|_t = u$ for $t \ge |u|$, as the code does), and $u \wedge v$ for the deepest common ancestor (`join`, lines 119–124). The relation $v \sqsubseteq u$ of §4.1 gives $\mathbf{x}_v \subseteq \mathbf{x}_u$. Two nodes are either nested ($u \sqsubseteq v$ or $v \sqsubseteq u$) or disjoint, a property of their words (§4.1); disjoint nodes lie below different children of $u \wedge v$, so their cells are disjoint. An *antichain* is a set of pairwise disjoint nodes; the leaves of a paving form one.

For a node u and $D \ge |u|$, the depth-D descendants of u have the consecutive heap codes

$$\mathrm{blk}_D(u) = \big[\,\mathrm{lab}(u)\,2^{D-|u|},\ (\mathrm{lab}(u)+1)\,2^{D-|u|}\big),$$

the *dyadic block* of u at depth D. Blocks of two nodes at a common depth D are nested exactly when the nodes are, and disjoint otherwise. On $[0,1]$ the heap code gives the dyadic interval directly: with $h = |u|$, $\mathbf{x}_u = [(\mathrm{lab}(u) - 2^h)2^{-h}, (\mathrm{lab}(u) - 2^h + 1)2^{-h})$, closed at 1 for the last node of each depth, and $\mathrm{blk}_D(u)$ lists the heap codes of the depth-D dyadic intervals inside $\mathbf{x}_u$.

Definition 4.26 (Left-right order)**.** *For nodes u, v and any $D \ge \max(|u|, |v|)$, write $u \prec v$ when every element of $\mathrm{blk}_D(u)$ is smaller than every element of $\mathrm{blk}_D(v)$.*

The relation does not depend on D, since passing from D to $D+1$ doubles every block endpoint.

Proposition 4.27 (The left-right order is an interval order)**.** *(i) $\prec$ is a strict partial order on all nodes. (ii) Two distinct nodes are $\prec$-incomparable exactly when they are nested. (iii) The restriction of $\prec$ to a set of nodes is total exactly when the set is an antichain. (iv) For disjoint u, v, $u \prec v$ exactly when u lies in the left subtree of $u \wedge v$. (v) On nodes of one depth, $\prec$ is the integer order of the heap codes. (vi) If $a_1 \prec \cdots \prec a_N$ is an antichain and u is a node, the indices i with $a_i \sqsubseteq u$ form an interval.*

Proof.

- *(i)–(iii).* $\prec$ is the interval order of the family of blocks $\mathrm{blk}_D(u)$, which are nonempty integer intervals. So it is irreflexive and transitive: if $\max \mathrm{blk}_D(u) < \min \mathrm{blk}_D(v)$ and $\max \mathrm{blk}_D(v) < \min \mathrm{blk}_D(w)$ then $\max \mathrm{blk}_D(u) < \min \mathrm{blk}_D(w)$. Two blocks are incomparable exactly when they intersect, and dyadic blocks intersect exactly when they are nested, that is, when the nodes are. A set is totally ordered exactly when no two of its blocks intersect.
- *(iv).* With $w = u \wedge v$, one of u, v lies below $w\mathsf{L}$ and the other below $w\mathsf{R}$, and $\mathrm{blk}_D(w\mathsf{L})$ precedes $\mathrm{blk}_D(w\mathsf{R})$.
- *(v).* At depth D the block of u is $\{\mathrm{lab}(u)\}$.
- *(vi).* Let $i < k < j$ with $a_i, a_j \sqsubseteq u$. Then $\mathrm{blk}_D(a_k)$ lies between $\mathrm{blk}_D(a_i)$ and $\mathrm{blk}_D(a_j)$, inside the interval $\mathrm{blk}_D(u)$, so a_k and u are nested. If $a_k \sqsupset u$ then $a_k \sqsupset a_i$, against the antichain property. So $a_k \sqsubseteq u$. □

By (iv), the left-to-right order of the leaves of §4.1, the leaves of the left subtree before those of the right, recursively, is $\prec$ on the leaves. By (vi), the leaves of a paving below any node form a contiguous run of its sorted leaf list: a subtree is a block of consecutive leaves. On $[0,1]$, $u \prec v$ for disjoint u, v means that $\mathbf{x}_u$ lies to the left of $\mathbf{x}_v$.

The code compares labels with `isStrictLeftOf(a, b)`, which compares the labels of `a.truncate(|b|)` and `b.truncate(|a|)` (lines 133–134), and sorts with `leftRightOrd` (lines 207–210), whose `compare` returns 0 on equal labels, -1 when `isLeftOf(a, b)` holds, and $+1$ otherwise (line 209). Since `isLeftOf(a, b)` is `a == b || isStrictLeftOf(a, b)` (line 137), on distinct labels it is `isStrictLeftOf(a, b)`. In Lemma 4.28(c), as in Appendix B, $I(u)$ is a set of list indices, not a set of internal nodes.

Lemma 4.28 (Left-right order in the code)**.** *(a) For all nodes a, b, `isStrictLeftOf(a, b)` holds exactly when $a \prec b$. (b) If a is a strict ancestor of b, then `compare(a, b)` and `compare(b, a)` both return $+1$, so `leftRightOrd` is not a lawful ordering on arbitrary labels; on every antichain, repetitions allowed, it is a lawful total order. (c) Let $\ell_0, \ldots, \ell_{N-1}$ be a list whose distinct entries form an antichain, listed in left-right order with equal entries adjacent, and let u be a node of which no ℓ_i is a strict ancestor. Then $I(u) = \{i : \ell_i \sqsubseteq u\}$ is an interval $[\mathrm{lo}(u), \mathrm{up}(u))$, and `subtreeAt` (`SubtreePartitioner.scala` lines 289–294) returns it after at most $2(\lceil \log_2 N \rceil + 1)$ evaluations of label predicates.*

Proof.

- **(a)** Let $t = \min(|a|, |b|)$. The calls truncate the shallower node to its own depth, which leaves it unchanged, and the deeper one to depth t; so the code compares the heap codes of $a|_t$ and $b|_t$, two nodes of depth t. If a, b are nested, both truncations are the shallower node, the codes are equal, and the test fails; by Proposition 4.27(ii) neither $a \prec b$ nor $b \prec a$. If a, b are disjoint, $a|_t$ and $b|_t$ are disjoint nodes of one depth containing a and b, so by Proposition 4.27(v) the test holds exactly when $\mathrm{blk}_D(a|_t)$ precedes $\mathrm{blk}_D(b|_t)$, which happens exactly when $a \prec b$.
- **(b)** By (a), on a nested pair of distinct labels `isStrictLeftOf` fails both ways, and so does `isLeftOf`, which equals it on distinct labels; so `compare` returns $+1$ both ways. On an antichain with repetitions, `compare` is 0 on equal entries and otherwise follows the total order $\prec$ of Proposition 4.27(iii).
- **(c), interval.** As in Proposition 4.27(vi), using that copies of one label are adjacent and that no ℓ_i strictly contains u.
- **(c), searches.** The first search of `subtreeAt` tests "not $\ell_i \prec u$". By (a) and the hypothesis on u this is false exactly before $I(u)$. The second search starts at $\mathrm{lo}(u)$ and tests $\ell_i \not\sqsubseteq u$, which is false exactly on $I(u)$. Both predicates are false and then true, as `binarySearchWithin` (`Truncation.scala` lines 46–60) requires. It halves the index range at each evaluation and returns the first true index after at most $\lceil \log_2 N \rceil + 1$ evaluations. □

Marks. Proposition 4.27 and (a), (b) proved above and certified (`IasCert.L1MDE.Labels`, `IasCert.L1MDE.PropBlock`, Chapter 15). [LEAN: (c), `subtreeAt`, not yet authored.] [SIM: a randomised test of `isStrictLeftOf`, `leftRightOrd` and `subtreeAt` against Definition 4.26 and Proposition 4.27(vi), on random labels and random antichains, to come.]

So every sort by `leftRightOrd` must be a sort of an antichain, possibly with repetitions. The audit of the code at commit `c0a1837` found that every such sort is of an antichain under one invariant.

Definition 4.29 (Inv-Depth)**.** *All data labels lie at one common finest depth D, and every leaf of a stored paving and every subtree root has depth at most D.*

The sorts, and why each is of an antichain under Inv-Depth:

- `Truncation.fromLeafSet` (`Truncation.scala` line 266; its sort, `leaves.toVector.sorted(leftRightOrd)`, is line 267): the caller's obligation; the code warns that it is not checked, and the antichain and order invariants of `Truncation` are never checked (lines 264–265).
- `rpUnion` (`Truncation.scala` lines 437 and 439): a node, its sibling and the siblings of its ancestors up to a given depth, a family of pairwise disjoint nodes.
- the backtracks of `Histogram` (`Histogram.scala` lines 191, 195, 267, 296, 300, 374) and `LeafMap.fromNodeLabelMap` (`LeafMap.scala` line 157, with `sortWith(is-`

`StrictLeftOf`): the leaves of one paving.

- the sampling of `SubtreePartitioner` (`SubtreePartitioner.scala` line 119, `candidates.sorted`, which sorts (label, weight) pairs by label under the ordering `leftRightOrd` declared implicitly at line 76, then by weight) and `findSubtree` (lines 410 and 458): depth-D keys, or a truncated key compared with a subtree root of the same depth.
- `MinimumDistanceEstimation` (`MinimumDistanceEstimation.scala` lines 482 and 590): the validation records merged to the refinement leaves (an antichain by Theorem 11.12(iii)), and the depth-D validation labels, with repetitions.
- `mergeLeavesRDD` (`MergeEstimator.scala` line 314), the only site that is not local: it sorts the subtree roots collected from all partitions. They form an antichain only if the local outputs of different partitions are disjoint. Under a safe depth limit they are (Lemma 11.4, distributed execution), provided no partition relabels its single subtree as the root (Chapter 11, the driver-pass condition).

Union. Two pavings of one scheme have a coarsest common refinement, the paving whose internal nodes are the union of theirs; by Lemma 4.2(i) this union is again the internal-node set of a paving. Its statement and proof, and the sparse form for supports that need not cover the root box, are in §6.2 (Propositions 6.5 and 6.7). [HST12] compute it by descending both trees at once, matching left children with left children and right with right.

4.7 Other cut rules and other domains

Remark 4.30 (Other cut rules realise the same trees)**.** The trees, their counts and their Catalan coefficients (§§ 4.1 and 4.2), and the labels and orders of §4.6, do not use the midpoint rule. Of the distributions of §4.3, only the SEB model reads the cells of a fixed scheme, and it extends to any scheme with measurable cells by $P(s, \ell) = \int_{\mathbf{x}_\ell} f$, with the same proof of split-exchangeability. Any splitting scheme (Definition 4.21) realises every Catalan binary tree as a partition, with the same labels: a label names a node, and the scheme gives the node its cell. The union, the pointwise operation, the pushforward and the collation of Chapter 6 are computed from the tree and the labels alone, so they hold for any fixed scheme, provided both operands use the same one. The proviso is essential. Two pavings with one tree but different cuts have equal labels on different cells, and the union of their trees is not the overlay of their partitions. With the cut coordinate and the cut point free at each node, the union of two pavings could be guaranteed to be such a paving only when both had the same cuts at every pair of nodes with equal labels [HST12]. A scheme fixed in advance removes the proviso, since each tree then represents one partition. Marginals and slices need more (Remark 4.31): for a scheme of boxes, the splits of the kept axes, read in order of depth, must also be the splits of a fixed scheme of the projected box, as Lemma 4.24 proves for regular bisection and for every scheme whose split axis depends on the depth alone. The implementation has two schemes, the regular paving `WidestSplitTree` and `UniformSplitTree` (`SpatialTree.scala` lines 74 and 210 at commit `c0a1837`), each a midpoint split along an axis that depends on the depth alone: the first widest coordinate of the boxes of that depth, and the depth modulo d. Its labels, merging and counting use only the labels, so they hold for both. The Beta-splitting trees of Proposition 4.17 cut $[0, 1]$ at random points. They are Catalan binary trees like the others, but their cells depend on the draws, so two of them share an arithmetic only when they share their cut points.

Remark 4.31 (What the general setting needs)**.** The results of §6.1 and Chapter 11, and the union, the pointwise operation and the pushforward of Chapter 6, use only that the scheme is fixed and binary: labels, the left-right order, the nesting of cells, and the monotonicity of counts along ancestry. Marginals and slices use in addition that cells are boxes split along coordinate axes. Closure under overlay is then automatic, since the union of two internal-node sets is closed

under taking ancestors (Proposition 6.5). Theorem 8.10 uses in addition only that the cells shrink: its part (b) counts Catalan binary trees (Proposition 4.33), which holds for any fixed binary scheme, and its part (c) uses the diameter bound of regular bisection (Lemma 4.23), which shrinking cells replace. For a scheme with more than two children per cell the count would change; no such count is found in the sources, and none is stated here. Histograms also need the volume of each cell, which a scheme supplies through `volumeAt` in the implementation.

Remark 4.32 (Partitions of the sphere)**.** Planetary computation needs partitions of the surface of a sphere. Three ways to bring the trees there are proposed here. They are options, not results of this monograph:

1. a radar range, mapped to a rectangle that is then paved;
2. a polar projection onto a box, excluding the polar caps;
3. trees that partition the compact surface of the sphere itself.

The first two pave a box and reach the sphere through a fixed map. The third needs a splitting scheme (Definition 4.21) whose cells lie on the sphere. Chapter 16 returns to them.

4.8 Counting and random regular pavings

Proposition 4.33 (Counting regular pavings)**.** *Fix a splitting scheme whose cells are nonempty, for example the regular scheme on a root box.*

- *(i) Distinct nodes have distinct cells, and a paving is determined by its partition:* $s \mapsto \{\mathbf{x}_\ell : \ell \in \mathcal{L}(s)\}$ *is injective on* $\mathbb{S}_{0:\infty}$.
- *(ii) The pavings with* k *splits give exactly* $C_k \le 4^k$ *distinct partitions of* $\mathbf{x}_\rho$*. In particular a root box has exactly* C_k *regular pavings with* k *splits [HST12].*

Proof. (i) Both children of a cell are nonempty and they partition it, so each is a proper subset of it, and a strict descendant has a strictly smaller cell. Disjoint nodes have disjoint nonempty cells. So $u \mapsto \mathbf{x}_u$ is injective, and the partition determines $\mathcal{L}(s)$, which determines the tree (Lemma 4.2(v)). (ii) By (i) and Proposition 4.5, since the leaf set of every Catalan binary tree is a paving of the scheme. □

Marks. Proved above. [LEAN: the bound, that a family of distinct pavings with k internal nodes has at most $C_k \le 4^k$ members, is certified (`IasCert.L1MDE.PavingCount`, Chapter 15); the equality with C_k and (i) not yet authored.] [SIM: not needed.]

So the state space of regular pavings with a given root box is $\mathbb{S}_{0:\infty}$ itself, with the arrows of Proposition 4.4, and every Markov splitting model of §4.3 gives a random regular paving: run the model on the trees and read each tree through the regular scheme. The Yule model splits a uniformly chosen leaf. The SEB model extends to a probability density f on a box $\mathbf{x}_\rho$, with $P(s,\ell) = \int_{\mathbf{x}_\ell} f$. The proof of Proposition 4.16(ii) applies unchanged: a split path to s has probability $\prod_{u\in I(s)} \int_{\mathbf{x}_u} f$, so the model is split-exchangeable, and Theorem 4.14 gives the probability of the paving s as $B(s)\prod_{u\in I(s)} \int_{\mathbf{x}_u} f$. For the uniform density on $\mathbf{x}_\rho$, $\int_{\mathbf{x}_u} f = \mathrm{vol}(\mathbf{x}_u)/\mathrm{vol}(\mathbf{x}_\rho) = 2^{-|u|}$, so the law of the tree is that of the uniform SEB model on $[0,1]$, whatever the dimension and the root box. Beta-splitting cuts at random points, so its own partitions are not regular pavings. Its trees are Catalan binary trees all the same, read through the regular scheme like any others. As $\alpha = \beta \to \infty$, the law of its trees tends to that of the uniform SEB model (Proposition 4.17(iv), with Theorem 4.14), and its cut points tend in probability to the midpoints, since the Beta$(\beta+1, \beta+1)$ law concentrates at $\frac{1}{2}$.

Priors on pavings use the same counts. MRS2 gives each paving with k splits the prior probability proportional to C_k^{-2}, so that the pavings with k splits together carry mass proportional to $1/C_k$,

for its Markov chains over pavings [SYH+18]. A split from k to $k+1$ splits multiplies the prior by $(C_k/C_{k+1})^2 = \big((k+2)/(2(2k+1))\big)^2$, the ratio in the proof of Proposition 4.5.

Chapter 5

Mapped trees

A *mapped tree* is a Catalan binary tree whose nodes carry values. Realised as a regular paving of a root box (Chapter 4), it represents a function on the box that is constant on each leaf cell, and an operation on such functions becomes an operation on trees. This section fixes the objects: the two forms in which a mapped tree is stored (§5.1), the algebra of the functions they represent (§5.2), the canonical form (§5.3), what they approximate (§5.4), interval enclosures (§5.5), the kinds of values (§5.6), and the sparse and labelled forms that scale, with the unary maps that act on them (§5.7). The algorithms are in Chapter 6, which also treats labelled arithmetic over a forest of root boxes (§6.11).

Mapped regular pavings and their arithmetic are [HST12], implemented in the C++ library MRS [SYH+18]. The sparse form on integer labels is [STW20], and its leaf-list arithmetic is [Gra22].

5.1 Cells, values and the function a mapped tree represents

Throughout, $\mathbf{x}_\rho = [\underline{x}_1, \overline{x}_1] \times \cdots \times [\underline{x}_d, \overline{x}_d]$ is a closed root box with $\underline{x}_j < \overline{x}_j$ for every j, and the cells $\mathbf{x}_u$ are those of its regular scheme (§4.5).

- **Splitting.** A cell $\mathbf{x}_u$ is split along its first widest coordinate ι at the midpoint c of that side. The left child takes the interval $[\underline{x}_{u,\iota}, c)$ on coordinate ι. The right child takes $[c, \overline{x}_{u,\iota}]$ or $[c, \overline{x}_{u,\iota})$: it keeps the upper end of the parent, closed or open.
- **Cells.** So every cell is a product of intervals, each of the form $[a, b)$ or $[a, b]$, the closed ones reaching the upper faces of the root box. The children of a cell partition it, and the leaves of every regular paving partition $\mathbf{x}_\rho$.
- **Point location.** A point x of $\mathbf{x}_u$ lies in the right child exactly when $x_\iota \geq c$. This is the rule by which MRS [SYH+18] and SparkDensityTree [SGWS23a] locate points.
- **Notation.** $\overline{\mathbf{x}}_u$ is the closure of a cell, the compact box with the same ends; $\operatorname{wid} \mathbf{x}_u$ is its largest side; $\operatorname{mid} \mathbf{x}_u$ is the centre of $\overline{\mathbf{x}}_u$, which lies in $\mathbf{x}_u$.
- **Split axes and cell size.** All cells of depth k have the same side lengths and are split along one axis ι_k, the split axis of depth k; the split axes to depth D are computed once, in $O(dD)$ operations, and the cell of a label follows from its turns (Lemma 4.23(ii)). A cell of depth k has all its sides at most $W_\rho\, 2^{-\lfloor k/d \rfloor}$, where $W_\rho = \max_j(\overline{x}_j - \underline{x}_j)$, and so diameter at most $\sqrt{d}\, W_\rho\, 2^{-\lfloor k/d \rfloor}$ (Lemma 4.23(iv)).

The root box is closed, so it is compact and its upper faces belong to the rightmost cells, as in the implementations. Compactness is used in Theorem 5.7 and in Lemma 4.25(iii). With a half-open root box, Theorems 5.7, 5.15 and 5.17 would need g continuous on, and G defined on boxes of, the closure of the root box.

Below a paving. Let s be a regular paving. Every node u of the infinite binary tree either lies in $I(s)$ or lies at or below exactly one leaf of s, and if $I(s) \subseteq I(s')$, every leaf of s' lies in a leaf of s (Lemma 4.2(vi)).

A mapped tree is stored in one of two forms.

Definition 5.1 (Mapped regular paving in multi-resolution form)**.** *For a nonempty set $\mathbb{A}$, an* $\mathbb{A}$-mapped regular paving in multi-resolution form *is a pair (s, f) of a regular paving s of $\mathbf{x}_\rho$ and a map $f : V(s) \to \mathbb{A}$ on all its nodes, $V(s) = I(s) \cup \mathcal{L}(s)$. Forgetting the values at the internal nodes gives its* leaf-valued form *$(s, f|_{\mathcal{L}(s)})$.*

The multi-resolution form is that of [HST12] and of MRS, where every node of a pointer tree carries a value (the class template `MappedSPnode<T>`, `mappedspnode.hpp`). The leaf-valued form is that of the sparse arithmetic and of SparkDensityTree.

Definition 5.2 (Mapped and statistical regular pavings)**.** *For a set $\mathbb{A}$, an* $\mathbb{A}$-mapped regular paving *($\mathbb{A}$-MRP) is an RP s together with a map $\mathcal{L}(s) \to \mathbb{A}$. A* statistical regular paving *(SRP) of a sample $x_1, \ldots, x_n \in \mathbf{x}_\rho$ is the $\mathbb{Z}_{\geq 0}$-MRP with leaf values $\#(u) = |\{i : x_i \in \mathbf{x}_u\}|$. Its* histogram *is $f_{n,s}(x) = \#(u)/(n \operatorname{vol}(\mathbf{x}_u))$ for $x \in \mathbf{x}_u$, $u \in \mathcal{L}(s)$. The* sparse *representation stores only the leaves with $\#(u) > 0$, keyed by label.*

For any node u (leaf or not), $\#(u)$ is the number of sample points in $\mathbf{x}_u$, so counts are monotone along ancestry: if u is an ancestor of w then $\#(u) \geq \#(w)$. Every argument in Chapter 11 rests on this monotonicity. In the multi-resolution form of an SRP every node carries its count, as in MRS, where each node of the class `SPSnode` (`spsnode.hpp`) holds the count of its cell.

The function a mapped tree represents. For $x \in \mathbf{x}_\rho$ let $\eta_s(x)$ be the leaf of s whose cell contains x. It is unique because the leaves partition the root box. An MRP in either form represents the function

$$\bar{f} : \mathbf{x}_\rho \to \mathbb{A}, \qquad \bar{f}(x) = f(\eta_s(x)).$$

- Only the leaf values enter $\bar{f}$. - The internal values of the multi-resolution form are views of f at coarser resolutions. What they mean depends on the kind of value (§5.6): for counts, the count of the cell; for an interval enclosure, an enclosure over the cell; for a height, whatever an algorithm last wrote there. - Two MRPs are *equivalent* when they represent the same function. - $\mathcal{F}_\mathbb{A}$ denotes the set of functions represented by $\mathbb{A}$-MRPs of $\mathbf{x}_\rho$: the functions $\mathbf{x}_\rho \to \mathbb{A}$ that are constant on each leaf of some regular paving.

If $\bar{f}$ is constant on the leaves of s and $I(s) \subseteq I(s')$, then $\bar{f}$ is constant on the leaves of s', since each of them lies in a leaf of s. The union of finitely many internal-node sets is again the internal-node set of a paving (Lemma 4.2(i)), so finitely many elements of $\mathcal{F}_\mathbb{A}$ are represented on one common paving, the union of theirs (Proposition 6.5 names it). This is all that the algebra needs.

5.2 The algebra

Proposition 5.3 (The algebra of mapped regular pavings)**.** *Let $\mathcal{F}_\mathbb{A}$ be as above.*

- *(i) $\mathcal{F}_\mathbb{A}$ contains the constant functions, and for every $n \geq 1$ and every map $\star : \mathbb{A}^n \to \mathbb{A}$ it is closed under the pointwise operation $\star(\phi_1, \ldots, \phi_n)(x) = \star(\phi_1(x), \ldots, \phi_n(x))$. Read every relation on $\mathbb{A}$ pointwise on functions. Then every law that holds in $\mathbb{A}$ and has the form "$t\ R\ t'$" or "if $t_1\ R_1\ t_1', \ldots, t_m\ R_m\ t_m'$, then $t\ R\ t'$", for terms in the operations and the constants and relations R, R_i such as $=$ or $\subseteq$, holds in $\mathcal{F}_\mathbb{A}$.*

- *(ii) If* $(\mathbb{A}, +, \cdot, 0, 1)$ *is a commutative ring, so is* $\mathcal{F}_{\mathbb{A}}$ *under pointwise operations. Its zero and one are the MRPs on the root alone with values* 0 *and* 1*. If* $\mathbb{A}$ *is a field* $\mathbb{K}$*,* $\mathcal{F}_{\mathbb{K}}$ *is a commutative* $\mathbb{K}$*-algebra, with scalar multiplication by the constant functions.*
- *(iii)* $\mathcal{F}_{\mathbb{K}}$ *is not a field and not an integral domain:* $\mathbf{1}_{\mathbf{x}_{\rho\mathsf{L}}} \cdot \mathbf{1}_{\mathbf{x}_{\rho\mathsf{R}}} = 0$ *with both factors nonzero. Its units are the functions with no zero value, and the inverse of a unit is represented on the same paving.*
- *(iv) With* $\vee$*,* $\wedge$ *and* $\neg$*, the set* $\mathcal{F}_{\mathbb{B}}$ *of Boolean MRPs,* $\mathbb{B} = \{\text{True}, \text{False}\}$*, is a Boolean algebra. The map* $\phi \mapsto \{x : \phi(x) = \text{True}\}$ *is an isomorphism onto the algebra of finite unions of cells.*

Proof.

- *(i).* A constant is represented on the root alone. Let $\phi_1, \ldots, \phi_n$ be represented on pavings $s_1, \ldots, s_n$, and let s be the paving with $I(s) = I(s_1) \cup \cdots \cup I(s_n)$ (Lemma 4.2(i)). By Lemma 4.2(vi), every leaf of s lies in a leaf of each s_i. So every ϕ_i is constant on each leaf of s, and so is $\star(\phi_1, \ldots, \phi_n)$. A law of either form holds at every point x, because its hypotheses, read pointwise, hold at x and it holds in $\mathbb{A}$; so it holds for the functions.
- *(ii).* The functions $\mathbf{x}_\rho \to \mathbb{A}$ form a commutative ring under pointwise operations. By (i), $\mathcal{F}_{\mathbb{A}}$ contains 0 and 1 and is closed under $+$, $-$ and $\cdot$, so it is a subring. The scalar $c \in \mathbb{K}$ acts as multiplication by the constant function c.
- *(iii).* The cells $\mathbf{x}_{\rho\mathsf{L}}$ and $\mathbf{x}_{\rho\mathsf{R}}$ are nonempty and disjoint. Their indicators are represented on the paving with leaves $\rho\mathsf{L}, \rho\mathsf{R}$, and their product is 0. If $\phi\chi = 1$, then $\phi(x) \neq 0$ for every x. Conversely, if ϕ has no zero, $1/\phi$ is constant on the leaves on which ϕ is constant.
- *(iv).* The map is injective, and it sends $\vee, \wedge, \neg$ to union, intersection and complement. The set where ϕ is true is a union of leaves of a paving, so a finite union of cells. Conversely, let $\mathbf{x}_{u_1} \cup \cdots \cup \mathbf{x}_{u_m}$ be a finite union of cells, and let s be the paving whose internal nodes are the strict ancestors of $u_1, \ldots, u_m$ (Lemma 4.2(i)). Each u_j is a node of s, and $\mathbf{x}_{u_j}$ is the union of the cells of the leaves of s at or below u_j. So the union is a union of leaves of s, the set where a Boolean MRP on s is true. □

Marks. Proved here. [LEAN: Operate algebra_closure, algebra_law, inF_const, inF_comp, inF_map$_2$ ((i), on a splitting scheme with nonempty cells), algebraSubring, algebraSubalgebra, algebraMap_const, algebra_smul ((ii)), algebra_iii, algebra_units, inF_indicator_child ((iii)) — built at `b6ea25813`; to write — (iv), the Boolean algebra] [SIM: random triples of MRPs, the ring laws checked at random points and on the common refinement]

Intervals. Compact intervals $\mathbb{IR}$ do not form a ring: $X - X \neq [0, 0]$ unless X is a single point, and multiplication is only subdistributive, $X(Y + Z) \subseteq XY + XZ$ [Moo66]. By (i), $\mathcal{F}_{\mathbb{IR}}$ has the same laws as $\mathbb{IR}$: - commutative monoids under $+$ and under $\cdot$; - subdistributivity, a law $t \subseteq t'$; - inclusion isotony, pointwise, a law with hypotheses: if $F \subseteq F'$ and $H \subseteq H'$ at every point, then $F \star H \subseteq F' \star H'$.

[HST12] call this the inclusion algebra of $\mathbb{IR}$-MRPs, a specialisation, to pieces formed by regular pavings, of the algebra PC_k of [Neu90, 2.2.2, p. 43].

Booleans. A Boolean MRP is a set built from cells. The regular subpavings of set inversion [JKDW01] are Boolean MRPs whose true leaves are the kept boxes. [HST12] write $+$, $-$, $\times$ and $\div$ for $\vee$, exclusive or, $\wedge$ and $a \wedge \neg b$, that is, for union, symmetric difference, intersection and set difference.

Colours and vectors. Colours in $[0, 1]^3$ and vectors in $\mathbb{R}^3$ [HST12] carry whatever operation is chosen on the value set, such as a componentwise sum of colours clipped to $[0, 1]$, or the cross product of vectors. By (i), the MRPs inherit exactly its laws, and no more. The cross product is neither associative nor commutative, and neither is its pointwise extension.

Unary maps act value by value; they are treated with the sparse forms (Definition 5.21 and Proposition 5.22).

5.3 The canonical form

A function in $\mathcal{F}_{\mathbb{A}}$ is represented on many pavings, since any refinement of a representing paving also represents it. One of them is smallest, and MinimiseLeaves finds it.

MinimiseLeaves [HST12] (Algorithm 5) visits the nodes in post-order. At a *cherry*, an internal node whose two children are leaves, it acts only if the two leaves carry equal values: it gives the cherry that value and prunes the two leaves. Each such *merge* keeps the paving a regular paving and keeps the represented function.

Lemma 5.4 (A paving with no mergeable cherry is canonical)**.** *Let $\phi \in \mathcal{F}_{\mathbb{A}}$, and let $\mathcal{N}(\phi)$ be the set of nodes u of the infinite binary tree such that ϕ is not constant on $\mathbf{x}_u$. Let s be a paving on whose leaves ϕ is constant. Then $\mathcal{N}(\phi) \subseteq I(s)$. If, in addition, no cherry of s has two leaves on which ϕ takes the same value, then $I(s) = \mathcal{N}(\phi)$.*

Proof. A node $u \notin I(s)$ lies at or below a leaf of s (Lemma 4.2(vi)), on whose cell ϕ is constant, so $u \notin \mathcal{N}(\phi)$. Hence $\mathcal{N}(\phi) \subseteq I(s)$. Suppose now that $I(s) \neq \mathcal{N}(\phi)$, and take $u \in I(s) \setminus \mathcal{N}(\phi)$ of greatest depth. Then ϕ is constant on $\mathbf{x}_u$, hence on $\mathbf{x}_{u\mathsf{L}}$ and $\mathbf{x}_{u\mathsf{R}}$, so neither child lies in $\mathcal{N}(\phi)$. Neither child lies in $I(s) \setminus \mathcal{N}(\phi)$ either, by the choice of u. So both children are leaves of s, u is a cherry, and ϕ takes one value on both of its leaves. □

Theorem 5.5 (Canonical form)**.** *Let $\phi \in \mathcal{F}_{\mathbb{A}}$ and let $\mathcal{N}(\phi)$ be as in Lemma 5.4.*

- *(i) $\mathcal{N}(\phi)$ is finite and closed under taking ancestors. So it is the internal-node set of exactly one regular paving $s^*(\phi)$, and ϕ is constant on each leaf of $s^*(\phi)$.*
- *(ii) A paving s represents ϕ exactly when $I(s) \supseteq \mathcal{N}(\phi)$. Hence $s^*(\phi)$ is the unique paving with fewest leaves that represents ϕ, and every paving that represents ϕ refines it.*
- *(iii) For every MRP (s, f) representing ϕ, in either form, MinimiseLeaves returns the paving $s^*(\phi)$ with the values of ϕ on its leaves. The tree shape and the leaf values are therefore unique. The values at internal nodes are not determined by ϕ, and MinimiseLeaves leaves those of the surviving internal nodes as they were.*

Proof.

- *(i).* ϕ is represented on some paving s, and $\mathcal{N}(\phi) \subseteq I(s)$ by Lemma 5.4, so $\mathcal{N}(\phi)$ is finite. If ϕ is not constant on $\mathbf{x}_u$, it is not constant on any larger cell, so $\mathcal{N}(\phi)$ is closed under ancestors. A finite set of nodes closed under ancestors is the internal-node set of exactly one paving (Lemma 4.2(i)). A leaf of that paving is not in $\mathcal{N}(\phi)$, so ϕ is constant on it.
- *(ii).* If s represents ϕ, then $I(s) \supseteq \mathcal{N}(\phi)$ by Lemma 5.4. Conversely, if $I(s) \supseteq \mathcal{N}(\phi)$, a leaf of s is not in $I(s)$, hence not in $\mathcal{N}(\phi)$, and ϕ is constant on it. A paving has one more leaf than internal nodes, so the fewest leaves go with the smallest internal-node set, which is $\mathcal{N}(\phi)$.
- *(iii).* Every merge keeps the represented function, so the output represents ϕ. Let u be a cherry of the output. The call at u tested its children after both recursive calls had returned. After those calls, the subtrees at $u\mathsf{L}$ and $u\mathsf{R}$ change only if a call at u or above prunes them. So at the test both children were leaves with their final values, and the test failed: their values differ. By Lemma 5.4 the output paving is $s^*(\phi)$. Only merged cherries receive new values, so the other internal nodes keep theirs. □

Marks. Proved here. [LEAN: Canonical nset_ancClosed, thm_canonical_i ((i)); nset_subset,

terminal_eq_canonical (Lemma 5.4); thm_canonical_ii, thm_canonical_ii_min ((ii)); thm_canonical_iii, canonical_unique, with minimiseAt_noCherry, the post-order pass ((iii), pointer form; the leaf-list form is Proposition 6.11) — built at `dbf40e07f`. The root box is a splitting scheme with nonempty cells (`Scheme.cell_nonempty`), which a box of positive widths has; Lem-Terminal's equality, (iii), canonical_unique and Proposition 6.11 use it (with an empty cell the MRP $\rho\mathsf{L} \mapsto 1$, $\rho\mathsf{R} \mapsto 7$ represents $\phi \equiv 1$ and MinimiseLeaves keeps it; a root box that is the hull of coinciding points is one point, so the box must have positive widths). The dyadic scheme on the half-open $[0,1)$ is an instance: CanonicalScheme dyadic (the cells $[k/2^h, (k+1)/2^h)$, dyadic_cell, with root cell $[0,1)$, dyadic_root), dyCell_split, dyCell_disj, dyCell_nonempty, and thm_canonical_iii_dyadic ((iii) on it, so the hypothesis of nonempty cells is met) — built at `15a5577ef`.] [SIM: random MRPs built by refining canonical ones and relabelling values at random; MinimiseLeaves against $\mathcal{N}(\phi)$ computed by brute force to depth 12]

The theorem is what makes $\mathcal{F}_{\mathbb{A}}$ computable: two MRPs represent the same function exactly when their canonical forms have the same leaves and leaf values, which is a comparison of two sorted leaf lists. The canonical form need not be the most useful representation. A histogram is usually kept on the paving that the data chose, and MinimiseLeaves is applied when size matters, for instance before a broadcast.

Remark 5.6 (Reuniting empty nodes)**.** For aircraft trajectories, [TKS12] reunite every pair of sibling leaves whose counts are both zero (their ReuniteEmptyNodes). This is MinimiseLeaves restricted to merges of value 0, and it cascades upwards, since it visits the nodes in post-order. On a sparse MRP with background 0 (Definition 5.20) it changes nothing: the zero cells are not stored, before or after.

5.4 Approximation

Theorem 5.7 (Uniform approximation of continuous functions)**.** *Let $g : \mathbf{x}_\rho \to \mathbb{R}$ be continuous, with modulus of continuity $\omega_g(\delta) = \sup\{|g(x) - g(y)| : x, y \in \mathbf{x}_\rho,\ |x - y| \le \delta\}$. Let s_k be the complete regular paving of depth k, and let f_k be the $\mathbb{R}$-MRP on s_k with $f_k(u) = g(\operatorname{mid}\mathbf{x}_u)$. Then*

$$\sup_{x \in \mathbf{x}_\rho} |\bar{f}_k(x) - g(x)| \le \omega_g\big(\sqrt{d}\, W_\rho\, 2^{-\lfloor k/d \rfloor}\big) \longrightarrow 0 \qquad (k \to \infty).$$

So every continuous function on the root box is a uniform limit of $\mathbb{R}$-MRPs: $C(\mathbf{x}_\rho, \mathbb{R})$ lies in the closure of $\mathcal{F}_{\mathbb{R}}$ in the supremum norm on bounded functions. The class $\mathcal{F}_{\mathbb{R}}$ is not a subset of $C(\mathbf{x}_\rho, \mathbb{R})$: $\mathcal{F}_{\mathbb{R}} \cap C(\mathbf{x}_\rho, \mathbb{R})$ consists of the constant functions.

Proof. The root box is compact, so g is uniformly continuous (Heine–Cantor), and $\omega_g(\delta) \to 0$ as $\delta \to 0$. For x in a leaf u of s_k, both x and $\operatorname{mid}\mathbf{x}_u$ lie in the closure of $\mathbf{x}_u$, so their distance is at most the diameter of the cell, which is at most $\sqrt{d}\, W_\rho\, 2^{-\lfloor k/d \rfloor}$ by Lemma 4.23(iv). Hence $|\bar{f}_k(x) - g(x)| = |g(\operatorname{mid}\mathbf{x}_u) - g(x)|$ is at most ω_g of that bound. For the last claim, a continuous function with finitely many values on the connected set $\mathbf{x}_\rho$ is constant, since the preimage of each value is open and closed. □

Marks. Proved here. [LEAN: to write — `UniformLimit`, from `Aspect` and Mathlib's `CompactSpace.uniformContinuous_of_continuous`] [SIM: $\sup|f_k - g|$ on a fine grid against the bound, for smooth and for Lipschitz g in $d \le 3$]

[HST12] obtain the approximation of continuous functions by these step functions through the Stone–Weierstrass theorem; the statement here adds the rate. The cell averages of Lemma 4.25(ii) obey the same bound with the diameter of Lemma 4.23(iii). Theorem 5.17(iii) bounds a midpoint MRP too, but with a tolerance fixed in advance and certified by the enclosure and the test, not

by the modulus of continuity. Lemma 4.25(iii) is the L_1 statement for every density, continuous or not, which is the one an estimator needs.

Three topologies. What the $\mathbb{R}$-MRPs on the regular pavings can approximate depends on the norm, and the three norms give three different answers, in order of what matters to this monograph. - *In L_1: every density, with no assumption.* The cell averages of any integrable f converge to f in L_1 (Proposition 5.8(i)). This is the topology of Chapters 8 to 10, whose guarantees hold for every density; it is the topology in which "all densities" is attainable, and no estimator is consistent for all densities in a stronger one. - *Almost everywhere: every density* (Proposition 5.8(ii)), by the same martingale. - *Uniformly: exactly the functions continuous on the doubled box.* Theorem 5.7 is one half of this: the uniform closure is larger than $C(\mathbf{x}_\rho, \mathbb{R})$ and has an exact description, the functions whose oscillation over the cells of depth k tends to zero with k (Theorem 5.10); an unbounded density, or one with a jump off the cut hyperplanes, is never a uniform limit (Remark 5.13). This is a fact about the supremum norm, not about the trees: no method with bounded pieces does better. The supremum norm is the norm of the enclosures of §5.5, where the target is a continuous expression; the estimator never uses it.

Rates are a fourth matter: in every topology a rate needs a smoothness class. In L_1 no rate holds for all densities, since for every prescribed sequence some density misses it (the no-rate theorem of [DG85]; its number is to be verified before the release candidate); Hölder classes give $2^{-\alpha k/d}$ with saturation at order one (Corollary 5.11), bounded variation gives 2^{-k} in one dimension (Lemma 5.12), and the adaptive pavings reach the Besov classes (§16.13). The estimator pays the rate through the approximation term of Theorem 9.1, which it minimises on its path without knowing the class.

Proposition 5.8 (Every density is approximated in L_1 and almost everywhere)**.** *Let $f \in L_1(\mathbf{x}_\rho)$, and let f_k be its cell averages on the complete paving of depth k: $f_k = \int_{\mathbf{x}_v} f/\mathrm{vol}(\mathbf{x}_v)$ on each depth-k cell $\mathbf{x}_v$, an $\mathbb{R}$-MRP as in Lemma 4.25 (the density at resolution k of Definition 12.3). - (i) $\int_{\mathbf{x}_\rho} |f_k - f| \to 0$ as $k \to \infty$. - (ii) $f_k \to f$ almost everywhere on $\mathbf{x}_\rho$. - (iii) $\sup |f_k - f| \to 0$ exactly when $\omega_k^{\square}(f) \to 0$ (Theorem 5.10(ii)); so not for an unbounded f, nor for one with a jump off the cut hyperplanes (Remark 5.13).*

So every law on a box dominated by Lebesgue measure, that is every density, is the L_1 limit and the almost-everywhere limit of $\mathbb{R}$-MRPs on the regular pavings, whatever its smoothness; it is their uniform limit exactly when it is $G \circ \iota$ for a continuous G on the end space K of Remark 5.13, the doubled box.

Proof. Let λ be Lebesgue measure normalised on $\mathbf{x}_\rho$ and $\mathcal{F}_k$ the finite σ-field generated by the depth-k cells. Then $f_k = \mathbb{E}_\lambda[f \mid \mathcal{F}_k]$, the $\mathcal{F}_k$ increase with k, and $(f_k)_k$ is a martingale under λ with $\int |f_k| \le \int |f| < \infty$ for every k. By Lévy's upward theorem [Wil91, Theorem 14.2], $f_k \to \mathbb{E}_\lambda[f \mid \mathcal{F}_\infty]$ almost everywhere and in L_1, where $\mathcal{F}_\infty = \sigma(\bigcup_k \mathcal{F}_k)$. The cells shrink (Lemma 4.23(iii)), so every open subset of $\mathbf{x}_\rho$ is a countable union of cells, $\mathcal{F}_\infty$ is the Borel σ-field of the box, and $\mathbb{E}_\lambda[f \mid \mathcal{F}_\infty] = f$. This is (i) and (ii). (iii) is Theorem 5.10(ii) with the examples of Remark 5.13. □

Marks. Proved here. (i) is also Lemma 12.4(iii), proved there by continuous approximation; the martingale proof gives (ii) with it. No rate holds for all densities: Corollary 5.11 and Lemma 5.12 give rates under Hölder and bounded-variation assumptions, and the saturation of Corollary 5.11 bounds what constant values can do. [LEAN: to write, on Mathlib's `MeasureTheory.Integrable.tendsto_ae_condExp` and `Integrable.tendsto_eLpNorm_condExp` (`Mathlib/Probability/Martingale/Convergence.lean`; the names to verify at the pin), with `Lem-Aspect`(iii) for the identification of $\mathcal{F}_\infty$.] [SIM: $f(x) = 1/(2\sqrt{x})$ on $(0, 1]$, unbounded and in L_1: $\int |f_k - f|$ against k (exact, by the cell integrals $\sqrt{b} - \sqrt{a}$), and $\sup |f_k - f| = \infty$ at

every k; $f = \mathbf{1}_{[0,1/3)} \cdot 3$: $\int |f_k - f| = O(2^{-k})$ against the uniform error $\frac{3}{2}$ at every k; to run, L1MDE-PROOF-SIMULATION.]

Definition 5.9 (Dyadic oscillation modulus)**.** *For a bounded $g : \mathbf{x}_\rho \to \mathbb{R}$ and a cell C, let* $\operatorname{osc}(g, C) = \sup_{x,y \in C} |g(x) - g(y)|$. *The* dyadic oscillation modulus *of g at depth k is $\omega_k^{\square}(g) = \max_{|u|=k} \operatorname{osc}(g, \mathbf{x}_u)$, the largest oscillation of g over a cell of depth k of the regular scheme. The* best uniform error at depth k *is $E_k(g) = \inf\{\sup_{x \in \mathbf{x}_\rho} |\bar{f}(x) - g(x)| : f$ an $\mathbb{R}$-MRP on $s_k\}$.*

Cells nest, so $\omega_k^{\square}(g)$ does not increase with k; and an $\mathbb{R}$-MRP on any paving is one on s_k for k its depth, so $E_k(g)$ is also the best error over all pavings of depth at most k. For continuous g, $\omega_k^{\square}(g) \le \omega_g(\operatorname{diam}_k)$, which is the bound Theorem 5.7 uses.

Theorem 5.10 (The uniform closure of the mapped pavings)**.** *Let $g : \mathbf{x}_\rho \to \mathbb{R}$ be bounded.*

- *(i) For every k, $E_k(g) = \frac{1}{2}\omega_k^{\square}(g)$. The infimum is attained by the midrange MRP, $f(u) = \frac{1}{2}(\sup_{\mathbf{x}_u} g + \inf_{\mathbf{x}_u} g)$ for $|u| = k$; the midpoint MRP of Theorem 5.7, $f(u) = g(\operatorname{mid} \mathbf{x}_u)$, has error at most $\omega_k^{\square}(g)$.*
- *(ii) g is a uniform limit of $\mathbb{R}$-MRPs exactly when $\omega_k^{\square}(g) \to 0$ as $k \to \infty$.*
- *(iii) If $\omega_k^{\square}(g) \to 0$, then g is continuous at every point of $\mathbf{x}_\rho$ that lies on no cut hyperplane of the scheme, a set whose complement has volume zero, and for every x and every $\epsilon > 0$ some cell containing x has $\operatorname{osc}(g, \cdot) < \epsilon$.*
- *(iv) For $d = 1$ and $\mathbf{x}_\rho = [a, b]$: g is a uniform limit of $\mathbb{R}$-MRPs exactly when g is regulated, that is, has a left limit $g(t-)$ at every $t > a$ and a right limit $g(t+)$ at every $t < b$, is continuous at every t that is not a cut point $a + (b-a)m2^{-j}$ (a and b among them), and is right-continuous at every cut point $t < b$ (with $g(b) = g(b-)$).*

Proof.

- *(i).* On a cell C with $M = \sup_C g$ and $m = \inf_C g$, a constant c has $\sup_C |g - c| \ge \max(M - c, c - m) \ge \frac{1}{2}(M - m)$, with equality at $c = \frac{1}{2}(M + m)$. Taking the maximum over the cells of s_k gives $E_k(g) = \frac{1}{2}\omega_k^{\square}(g)$, attained by the midrange values. For the midpoint value, $|g(x) - g(\operatorname{mid} \mathbf{x}_u)| \le \operatorname{osc}(g, \mathbf{x}_u)$ for $x \in \mathbf{x}_u$.
- *(ii).* If $\omega_k^{\square}(g) \to 0$, the midrange MRPs on s_k converge uniformly to g by (i). Conversely, let f be an MRP with $\sup |\bar{f} - g| \le \epsilon$, of depth k_0. On a cell C of depth $k \ge k_0$, $\bar{f}$ is constant, so $|g(x) - g(y)| \le |g(x) - \bar{f}(x)| + |\bar{f}(y) - g(y)| \le 2\epsilon$ for $x, y \in C$; hence $\omega_k^{\square}(g) \le 2\epsilon$ for all $k \ge k_0$.
- *(iii).* The cut hyperplanes of the scheme are $\{x : x_i = c\}$ for the finitely many cut values c of each depth, a countable family of sets of volume zero. A point x on none of them lies in the interior of its cell at every depth; those cells shrink to x (Lemma 4.23(iii)), so every neighbourhood of x contains one of them, and $|g(y) - g(x)| \le \omega_k^{\square}(g)$ on it. For any x, its cell of depth k has oscillation at most $\omega_k^{\square}(g)$.
- *(iv), necessity.* Let $\omega_k^{\square}(g) \to 0$. The cells of depth k to the right of $t < b$ are intervals $[t', t' + w_k)$ with $t' \le t < t' + w_k$ when t is not a cut point of depth k, and $[t, t + w_k)$ when it is; in both cases g varies by at most $\omega_k^{\square}(g)$ on $(t, t + \delta)$ for δ small, so $g(t+)$ exists (the Cauchy criterion), and likewise $g(t-)$ on $(t - \delta, t)$. At a non-cut point t the cells containing t shrink to t with t in their interior, so g is continuous at t (this is (iii)). A cut point $t < b$ is the left end of its cell $[t, t + w_k)$ at every depth from its own on, so $|g(t) - g(s)| \le \omega_k^{\square}(g)$ for $s \in [t, t + w_k)$: $g(t) = g(t+)$. The last cell is $[b - w_k, b]$ (Definition 4.22), which gives $g(b) = g(b-)$.
- *(iv), sufficiency.* Suppose $\omega_k^{\square}(g) \not\to 0$: there are $\epsilon > 0$, depths $k \to \infty$ and points x_k, y_k in one cell C_k of depth k with $|g(x_k) - g(y_k)| > \epsilon$. By compactness, along a subsequence $x_k \to t$, and $y_k \to t$ too since the widths $w_k \to 0$. If t is not a cut point, g is continuous

at t and $|g(x_k) - g(y_k)| \to 0$, a contradiction. If t is a cut point, the one-sided limits give $\eta > 0$ with $|g(s) - g(t+)| < \epsilon/3$ on $(t, t+\eta)$ and $|g(s) - g(t-)| < \epsilon/3$ on $(t-\eta, t)$, and right-continuity gives $g(t) = g(t+)$ (at $t = b$, $g(b) = g(b-)$). For large k, $C_k \subseteq (t-\eta, t+\eta)$, since C_k contains $x_k \to t$ and has width $w_k \to 0$; and since t is an end of the cells of every depth from its own on, C_k lies in $[t, t+\eta)$ or in $(t-\eta, t)$, or in $(b-\eta, b]$ at $t = b$. On each, $\operatorname{osc}(g, C_k) \le 2\epsilon/3 < \epsilon$, against $|g(x_k) - g(y_k)| > \epsilon$. □

Marks. Proved here; new as stated (the one-dimensional case is the classical fact that the regulated functions are the uniform limits of step functions, with the breakpoints now fixed at the cut points, which is the whole price). [LEAN: to write — `DyadicClosure`: (i) on finite cell sets; (ii) from (i); (iv) with the one-dimensional cells of `VC.RegularPaving`.] [SIM: E_k and $\frac{1}{2}\omega_k^{\square}$ computed on a fine grid for Lipschitz, Hölder and dyadic-step test functions in $d \le 3$, and for a step function with one non-dyadic jump, whose E_k must not decrease below half the jump; to run, L1MDE-PROOF-SIMULATION.]

Two consequences of (iv) fix the picture. Every MRP is in the closure, as it must be, and so is every function whose restriction to each half-open cell of some paving extends continuously to the closed cell: a function may jump wherever the scheme cuts. It may not jump anywhere else: one jump at a point that is no cut point, however small, keeps $\omega_k^{\square}(g)$ above the size of the jump at every depth, since the cell containing the point does. The regular scheme fixes where the approximation is allowed to be discontinuous, and that is all it fixes.

Corollary 5.11 (Rates, and the limit of constant values)**.** *Let $g : \mathbf{x}_\rho \to \mathbb{R}$.*

- *(i) If $|g(x) - g(y)| \le L\|x - y\|_2^\alpha$ for some $0 < \alpha \le 1$ (Hölder; Lipschitz at $\alpha = 1$), then $E_k(g) \le \frac{1}{2} L \operatorname{diam}_k^\alpha \le \frac{1}{2} L \big(\sqrt{d}\, W_\rho 2^{-\lfloor k/d \rfloor}\big)^\alpha$. With $m = 2^k$ leaves this is of order $m^{-\alpha/d}$.*
- *(ii) By Theorem 5.10(i), $E_k(g)$ and $\omega_k^{\square}(g)$ have the same order. The class with $E_k(g) = O(2^{-\alpha k/d})$ contains the Hölder-α functions and, by Theorem 5.10(iv), functions that jump on the cut hyperplanes.*
- *(iii) Saturation. If g is continuously differentiable on $\mathbf{x}_\rho$ and $\nabla g(x_0) \ne 0$ at some interior point x_0, then $E_k(g) \ge c\, 2^{-k/d}$ for all large k, with $c > 0$ depending on g and the root box. So no $\mathbb{R}$-MRP on s_k approximates such a g faster than $2^{-k/d}$, whatever its values.*

Proof. (i) $\operatorname{osc}(g, C) \le L \operatorname{diam}(C)^\alpha \le L \operatorname{diam}_k^\alpha$ (Lemma 4.23(iii), (iv)); apply Theorem 5.10(i). (ii) is Theorem 5.10(i) read both ways. (iii) Let $e = \nabla g(x_0)/|\nabla g(x_0)|$ and $\gamma = |\nabla g(x_0)|$. By continuity of ∇g there is a ball B around x_0 on which $\nabla g \cdot e \ge \gamma/2$. For large k the cell C of depth k containing x_0 lies in B (Lemma 4.23(iii)). The cell C has aspect ratio at most a and volume $2^{-k}\operatorname{vol}(\mathbf{x}_\rho)$ (Lemma 4.23(i), (iii)); its widest side $s_{\max}$ satisfies $s_{\max}^d \ge \operatorname{vol}(C)$, so its narrowest side is $s_{\min} \ge s_{\max}/a \ge a^{-1}(2^{-k}\operatorname{vol}(\mathbf{x}_\rho))^{1/d}$. The segment through the centre of C in direction e has length at least $s_{\min}$, and along it g increases at rate at least $\gamma/2$; so $\operatorname{osc}(g, C) \ge \frac{1}{2}\gamma s_{\min}$, and Theorem 5.10(i) gives $E_k(g) \ge c\, 2^{-k/d}$ with $c = \gamma \operatorname{vol}(\mathbf{x}_\rho)^{1/d}/(4a)$. □

Marks. Proved here. [LEAN: to write, after `DyadicClosure`.] [SIM: E_k along k for $g(x) = \|x\|_2$ and $g(x) = x_1$ in $d \le 3$, against both bounds; to run, L1MDE-PROOF-SIMULATION.]

The saturation is the limit of constant values, not of the trees: an MRP whose values are polynomials of degree r on each cell keeps the arithmetic of Chapter 6 (the pointwise operations act on a common refinement, cell by cell) and approximates C^{r+1} functions at the rate $2^{-(r+1)k/d}$. That extension, and the adaptive pavings that are the real gain, are §16.13.

The L_1 rate for bounded variation. Proposition 5.8(i) (and Lemma 4.25(iii)) says every density is an L_1 limit of the cell averages, with no rate. For functions of bounded variation on an interval the rate is the width of a cell.

Lemma 5.12 (Cell averages of a function of bounded variation)**.** *Let $d = 1$, $\mathbf{x}_\rho = [a, b]$, and let $f : [a, b] \to \mathbb{R}$ have total variation $\mathrm{TV}(f) < \infty$. Then for every k, and for every $\mathbb{R}$-MRP h on s_k whose value on each cell lies between the infimum and the supremum of f there (the cell average $\mathrm{Av}_k f$ among them),*

$$\int_a^b |f - \bar{h}| \le \sum_{|u|=k} \mathrm{osc}(f, \mathbf{x}_u)\,\mathrm{vol}(\mathbf{x}_u) \le \mathrm{TV}(f)\,(b-a)\,2^{-k}.$$

Proof. On a cell C, $|f - \bar{h}| \le \mathrm{osc}(f, C)$, which gives the first inequality. Every cell of depth k has width $(b-a)2^{-k}$. The oscillation of f over a cell is at most the variation of f over its closure, and the closures of the cells of s_k are adjacent intervals whose variations add to the variation over $[a, b]$, since the variation is additive at a common endpoint; so $\sum_{|u|=k} \mathrm{osc}(f, \mathbf{x}_u) \le \mathrm{TV}(f)$. □

Marks. Proved here. [LEAN: to write.] [SIM: $\int |f - \mathrm{Av}_k f|$ for a density with finitely many jumps and a monotone density, against $\mathrm{TV}(f)(b-a)2^{-k}$; to run, L1MDE-PROOF-SIMULATION.]

With $m = 2^k$ leaves the rate is $1/m$, which is of the best order for a piecewise constant on m cells of equal width and a function with a jump. In d dimensions the same first inequality holds, and the sum of the oscillations over the depth-k grid is controlled by the variation of f along each axis; that statement is not made here. For the densities of Chapter 9, the lemma bounds the resolution bias $\int |f_k - f|$ of Lemma 12.4(ii) by $\mathrm{TV}(f)(b-a)2^{-k}$; the distance of the empirical candidate on s_k to f adds the estimation term $\int |f_{n',s_k} - f_k|$, which Lemma 12.4(iv) sends to 0 as $n' \to \infty$.

Remark 5.13 (What cannot be approximated uniformly)**.** Each of the following has $\omega_k^{\square}(g)$ bounded away from zero, so by Theorem 5.10(ii) no sequence of $\mathbb{R}$-MRPs converges to it uniformly, whatever the values and however deep the pavings.

- *A jump off the cut hyperplanes.* $g = \mathbf{1}_{[0,1/3)}$ on $[0, 1]$: the cell containing $1/3$ has oscillation 1 at every depth, so $E_k(g) = \frac{1}{2}$ for all k. Likewise $\mathbf{1}_{\{x_1+x_2<1\}}$ on $[0, 1]^2$, the indicator of a ball, or of a half-space whose normal is not a coordinate axis: any set whose boundary is not contained in the cut hyperplanes. Bounded variation does not help; compare Lemma 5.12, where the same functions have L_1 error $O(2^{-k})$.
- *A jump on a cut hyperplane, with the wrong side.* $g = \mathbf{1}_{[0,1/2]}$ on $[0, 1]$, closed at $1/2$: the cell $[1/2, 1/2 + w_k)$ contains the point $1/2$, where $g = 1$, and points where $g = 0$, so its oscillation is 1 at every depth. The jump must be right-continuous, as Theorem 5.10(iv) says; the half-open convention of Definition 4.22 is in the closure.
- *Bounded oscillation at a point.* $g(x) = \sin(1/x)$ on $(0, 1]$ with any value at 0: continuous except at one point, with oscillation 2 on the cell containing 0 at every depth.
- *Dense discontinuities with jumps that do not shrink.* The Dirichlet function $\mathbf{1}_{\mathbb{Q}}$ has oscillation 1 on every cell. Thomae's function, $1/q$ at p/q in lowest terms and 0 elsewhere, is excluded twice over: it is discontinuous at $1/3$, which is no cut point, and at a cut point t it has $T(t+) = 0 \neq T(t)$. Dense discontinuities alone do not exclude. Enumerate the dyadic rationals of (a, b) by depth, $t_1, t_2, \ldots$ (depth j before depth $j + 1$), and let $g = \sum_j 2^{-j} \mathbf{1}_{[t_j,b]}$: monotone, right-continuous, and discontinuous at every dyadic rational, a dense set; but a cell of depth k contains, apart from its left end, only dyadics of depth greater than k, whose indices are at least 2^k (there are $2^k - 1$ of depth at most k), so $\omega_k^{\square}(g) \le \sum_{j \ge 2^k} 2^{-j} = 2^{1-2^k}$ and g is approximable. What decides is where the jumps are and how large they are on each cell, not how many there are.
- *Unbounded functions*, trivially: $1/x$ on $(0, 1]$.

Behind these is one statement. Let K be the set of ends of the infinite paving tree, the nested sequences of cells, one per depth: $\{\mathsf{L},\mathsf{R}\}^{\mathbb{N}}$ with the product topology, a Cantor space. Every $x \in \mathbf{x}_\rho$ has a canonical end $\iota(x)$, the half-open cells that contain it (Definition 4.22); the other ends over x are those whose closed cells all contain x, 2^j of them in all for a point on j cut hyperplanes and one off them (Lemma 4.23(iii)). For $d = 1$, K is the interval with each interior cut point doubled into the end of the cell on its left and the start of the cell on its right, the* doubled box*, and ι picks the right copy. An $\mathbb{R}$-MRP of depth k is $G \circ \iota$ for the function G on K constant on the cylinder of each depth-k cell, and the locally constant functions are dense in $C(K)$ for a totally disconnected compact space; so the uniform closure of the $\mathbb{R}$-MRPs is exactly $\{G \circ \iota : G \in C(K)\}$, and g is of this form exactly when $\omega_k^{\square}(g) \to 0$: G is then the limit of g along each end, and continuity of G at an end is the oscillation of g along its half-open cells tending to 0, uniform in the end by compactness. (The composite the other way, with the projection of an end to its point, would force continuity at every cut point; the half-open convention enters through ι.) The examples above are the functions that are not of this form, and Proposition 5.8's last sentence says the same for densities.

5.5 Interval enclosures

Interval arithmetic (§2.2) gives, over a box, an interval that contains the range of a function. Mapped trees with interval values carry such enclosures cell by cell. Write $\mathbb{IR}$ for the compact intervals $X = [\underline{X}, \overline{X}]$, with $\operatorname{wid} X = \overline{X} - \underline{X}$ and $\operatorname{mid} X = (\underline{X} + \overline{X})/2$, and $X \sqcup Y = [\min(\underline{X}, \underline{Y}), \max(\overline{X}, \overline{Y})]$ for the interval hull. For a set S write $g(S) = \{g(x) : x \in S\}$.

Definition 5.14 (Inclusion function)**.** *Let $g : \mathbf{x}_\rho \to \mathbb{R}$, and let G assign to every compact box $\mathbf{y} \subseteq \mathbf{x}_\rho$ an interval $G(\mathbf{y}) \in \mathbb{IR}$.*

- *G is an* inclusion function *of g when it encloses the range: $g(\mathbf{y}) \subseteq G(\mathbf{y})$ for every such box.*
- *G is* inclusion isotone *when $\mathbf{y} \subseteq \mathbf{y}'$ implies $G(\mathbf{y}) \subseteq G(\mathbf{y}')$.*
- *G is* convergent *when for every $\epsilon > 0$ there is a width $w_\epsilon > 0$ with $\operatorname{wid} G(\mathbf{y}) \le \epsilon$ whenever $\operatorname{wid} \mathbf{y} \le w_\epsilon$.*
- *G is* Lipschitz *with constant $\operatorname{Lip}(G)$ when $\operatorname{wid} G(\mathbf{y}) \le \operatorname{Lip}(G) \operatorname{wid} \mathbf{y}$ for every such box.*

A Lipschitz inclusion function is convergent, with $w_\epsilon = \epsilon/\operatorname{Lip}(G)$. Isotony compares an extension with itself on two boxes; it is a property of G, not of g, whose ranges are always isotone.

Theorem 5.15 (Natural interval extensions [Moo66])**.** *Let g be given by an arithmetical expression in $x_1, \dots, x_d$, built from constants and the variables by $+$, $-$, $\cdot$, $/$ and elementary functions. Let G be its* natural interval extension*, obtained by replacing every operation by its interval counterpart, evaluated with outward rounding. If $G(\mathbf{x}_\rho)$ is defined, that is, if no division by an interval containing 0 occurs and no elementary function is applied outside its domain, then $G(\mathbf{y})$ is defined for every box $\mathbf{y} \subseteq \mathbf{x}_\rho$, and G is an inclusion isotone inclusion function of g.*

This is the fundamental theorem of interval analysis, cited and not proved here (see also [JKDW01]). When the elementary functions of the expression are Lipschitz on the intervals over which they are evaluated, the natural extension is Lipschitz [Neu90]. These are the locally Lipschitz arithmetical expressions of the Moore rejection sampler (§3.3, [SY13]).

Definition 5.16 (Enclosure by an interval MRP)**.** *An $\mathbb{IR}$-MRP (s, F)* encloses *g when $g(x) \in \bar{F}(x)$ for every $x \in \mathbf{x}_\rho$. Its* volume of enclosure *is $\Xi(s, F) = \sum_{l \in \mathcal{L}(s)} \operatorname{wid} F(l) \operatorname{vol}(\mathbf{x}_l) = \int \operatorname{wid} \bar{F}$.*

UniformApprox [HST12] (Algorithm 17) refines an $\mathbb{IR}$-MRP until every leaf passes a test. It

sets $F(u) \leftarrow G(\overline{\mathbf{x}}_u)$ at each node it visits. It splits a leaf u, and then visits its children, while

$$\max\{\overline{F(u)} - g(\xi_u),\ g(\xi_u) - \underline{F(u)}\} > \epsilon, \qquad \xi_u = \operatorname{mid} \mathbf{x}_u.$$

Theorem 5.17 (Enclosure by UniformApprox)**.** *Let G be a convergent inclusion function of $g : \mathbf{x}_\rho \to \mathbb{R}$, and let $\epsilon > 0$. Take $w_\epsilon > 0$ with $\operatorname{wid} G(\mathbf{y}) \le \epsilon$ whenever $\operatorname{wid} \mathbf{y} \le w_\epsilon$, and let $k_\epsilon = d\lceil \log_2^+(W_\rho/w_\epsilon)\rceil$, where $\log_2^+ t = \max(0, \log_2 t)$. Started from any $\mathbb{IR}$-MRP of $\mathbf{x}_\rho$ whose paving has depth D_0, UniformApprox terminates, and its output (s, F) has depth at most $\max(k_\epsilon, D_0)$. For every leaf u of s and every $x \in \mathbf{x}_u$:*

- *(i) $g(x) \in F(u) = G(\overline{\mathbf{x}}_u)$, so (s, F) encloses g;*
- *(ii) $F(u) \subseteq [g(\xi_u) - \epsilon,\ g(\xi_u) + \epsilon]$, so $\operatorname{wid} \bar{F}(x) \le 2\epsilon$;*
- *(iii) the $\mathbb{R}$-MRP f on s with $f(u) = g(\xi_u)$ satisfies $\sup_{x\in\mathbf{x}_\rho} |\bar{f}(x) - g(x)| \le \epsilon$.*

Started from the root alone, the output has at most 2^{k_ϵ} leaves.

Proof.

- *Termination and depth.* If $\operatorname{wid} G(\overline{\mathbf{x}}_u) \le \epsilon$, the test passes. Indeed $\xi_u \in \mathbf{x}_u$, so $g(\xi_u) \in G(\overline{\mathbf{x}}_u)$ by range enclosure, and both deviations are at most the width. A cell of depth at least k_ϵ has all sides at most $W_\rho 2^{-\lfloor k_\epsilon/d \rfloor} \le w_\epsilon$ by Lemma 4.23(iv), so the test passes there. Hence no leaf of depth k_ϵ or more is split, and every new leaf has depth at most k_ϵ. The algorithm visits each node of the finite output once, so it terminates.
- *(i).* This is range enclosure on $\overline{\mathbf{x}}_u \supseteq \mathbf{x}_u$.
- *(ii).* Every leaf of the output passed the test, which says exactly $F(u) \subseteq [g(\xi_u) - \epsilon, g(\xi_u) + \epsilon]$.
- *(iii).* By (i) and (ii), $g(x) \in [g(\xi_u) - \epsilon, g(\xi_u) + \epsilon]$ for $x \in \mathbf{x}_u$. □

Marks. Proved here, in exact arithmetic. [LEAN: to write — `UniformApprox`: termination from `Aspect` and convergence, (i)–(iii)] [SIM: UniformApprox in MRS for polynomial and trigonometric expressions in $d \le 3$; the enclosure checked at random points with C-XSC, the depth against k_ϵ]

The enclosure by UniformApprox, with its widths at most 2ϵ and the midpoint MRP within ϵ of g, is that of [HST12]. Without a convergence hypothesis the theorem fails. The constant inclusion function $G(\mathbf{y}) = [-1, 2]$ of $g(x) = x$ on $[0, 1]$ is inclusion isotone and encloses every range, but it never passes the test with $\epsilon < 1$, since $2 - g(\xi) \ge 1$.

Remark 5.18 (On a machine)**.** Three things change in floating-point arithmetic.

- *Unsplittable cells.* MRS splits a cell only while its widest side contains a floating-point number strictly between its ends and its volume is at least twice the smallest positive normalised number (`SPnode::isSplittableNode`, `spnode.cpp`, with `cxsc::MinReal`). It stops at cells that cannot be split. UniformApprox then terminates in any case, but (ii) and (iii) can fail on such cells. A run reports them, in the spirit of §3.4 and Chapter 12: what cannot be resolved is said, not hidden.
- *Enclosure.* Part (i) holds for every tree whatever the test decided, provided G is evaluated with outward rounding. Only the widths in (ii) depend on the test.
- *The midpoint value.* A machine evaluates $g(\xi_u)$ with rounding error. A rigorous test encloses it too: with $[a, b] = G(\{\xi_u\})$, which contains $g(\xi_u)$, testing $\overline{F(u)} - a \le \epsilon$ and $b - \underline{F(u)} \le \epsilon$ guarantees (ii). Part (iii) then holds with the same ϵ for any value $f(u)$ in $[a, b]$: for $x \in \mathbf{x}_u$, $g(x) - f(u) \le \overline{F(u)} - a \le \epsilon$ and $f(u) - g(x) \le b - \underline{F(u)} \le \epsilon$.

5.6 Kinds of values

Definition 5.19 (Intensive and extensive values)**.** *A value is* intensive *when refining a cell copies it unchanged to both children: heights of a function or a density, Booleans, colours, intervals copied as enclosures. A value is* extensive *when coarsening sums it: its value set carries a commutative monoid* $(\mathbb{M}, +, 0)$*, and the value of a cell is the sum of the values of its children, as for counts and masses. An extensive value has no refinement rule of its own: a count must be re-read from the data, and a mass can be split in proportion to volume only by treating it as a density.*

A density is an extensive value divided by volume: the histogram $\#(u)/(n \operatorname{vol}(\mathbf{x}_u))$ is intensive while the count $\#(u)$ is extensive. The distinction matters for the arithmetic. Copying a coarse count into each finer cell, as a pointwise operation with copy-down does, multiplies its mass. Adding count tables label by label is a histogram operation only when they are at one common depth, or after both are coarsened to a common paving.

In the multi-resolution form the kind of value says what an internal value should be.

- *Extensive values.* An internal value is *consistent* when it is the sum of its children's values; then coarsening is pruning, since the parent already holds the sum. This is how MRS keeps counts.
- *Interval enclosures.* An internal value is consistent when it contains the hull of its children's values. HullPropagate makes it equal to that hull (§6.14).
- *Intensive heights.* There is no rule. An internal height is a free annotation.

The fold of §6.7 (Algorithm 10) makes internal values consistent for any associative operation with an identity.

5.7 Sparse and labelled mapped trees

Definition 5.20 (Sparse and labelled MRPs)**.** *Let* $e \in \mathbb{A}$ *be a* background *value. A* sparse $\mathbb{A}$-MRP *is a finitely supported map* f *from nodes to* $\mathbb{A} \setminus \{e\}$ *whose support* $\operatorname{supp} f$ *is an antichain; it represents the function equal to* $f(a)$ *on* $\mathbf{x}_a$ *for* $a \in \operatorname{supp} f$ *and to* e *elsewhere. Its* sorted leaf list *is the list of pairs* $(a, f(a))$ *in left-right order. For a set* $\mathbb{I}$ *of ids and a commutative monoid* $\mathbb{M}$*, a* labelled sparse MRP *is a sparse MRP whose values are finitely supported maps* $\mathbb{I} \to \mathbb{M}$*, the background being the zero map; equivalently, a finitely supported map on* $\mathbb{I} \times$ (*nodes*) *with values in* $\mathbb{M}$*.*

The "equivalently" of the definition is a correspondence of represented families, not of data. A labelled sparse MRP f gives the map $(i, a) \mapsto f(a)(i)$, whose support meets each id in an antichain. Conversely, a finitely supported map c on $\mathbb{I} \times$ (nodes) whose support meets each id in an antichain represents, id by id, the same family of functions as a labelled sparse MRP: its collation (Definition 6.29), on the sparse closure of the union of the supports. Its node support over all ids need not be an antichain.

The background is not stored, so every operation must fix it.

- A binary operation $\star$ keeps the sparse form exactly when $e \star e = e$, so that regions stored in neither operand need not be visited.
- A unary map τ keeps it exactly when $\tau(e) = e$ (Proposition 5.22(iv) below).
- A coarsening monoid keeps it when e is its identity.

Sums with $e = 0$, $\vee$ and $\wedge$ with $e =$ False, and products with $e = 0$ (sparse on the intersection of the supports) all qualify. The SRP of a sample is the sparse $\mathbb{Z}_{\geq 0}$-MRP with $e = 0$; its sparse

representation is that of Definition 5.2. A *count map* is a finitely supported map c from labels to $\mathbb{Z}_{\geq 0}$; at one depth D its support is automatically an antichain.

A dense MRP is the case in which the support is the leaf set of a paving; then no background is needed. Every sparse MRP is a dense one in disguise. The paving whose internal nodes are the strict ancestors of the support has as leaves the support and the gaps between its elements, and giving the gaps the value e represents the same function.

Labelled MRPs are the right object for families of maps on one root box. The collated regular paving of [ST19], which carries at each leaf the densities of several candidate histograms, is a labelled MRP whose ids are the candidates (§6.10). A set of aircraft trajectories is one whose ids are flights (§6.11).

Unary maps. A unary map acts value by value.

Definition 5.21 (MRPTransform)**.** *For a map $\tau : \mathbb{A} \to \mathbb{B}$,* MRPTransform$(f, \tau)$ *is the MRP on the same paving with value $\tau(f(u))$ at every node u (multi-resolution form) or at every leaf (leaf-valued form).*

Proposition 5.22 (Laws of MRPTransform)**.** *(i)* MRPTransform(f, τ) *represents $\tau \circ \bar{f}$. (ii)* MRPTransform(MRPTransform$(f, \tau), \tau')$ $=$ MRPTransform$(f, \tau' \circ \tau)$*, and* MRPTransform$(f, \mathrm{id}) = f$*. (iii) If $\tau(a \star b) = \tau(a) \star' \tau(b)$ for all a, b, then $\tau \circ (\phi \star \chi) = (\tau \circ \phi) \star' (\tau \circ \chi)$ for all $\phi, \chi \in \mathcal{F}_{\mathbb{A}}$. (iv) Applied to the stored values of a sparse MRP with background e (Definition 5.20), it gives the sparse MRP with background $\tau(e)$ once the entries with $\tau(f(a)) = \tau(e)$ are dropped; it keeps the background e exactly when $\tau(e) = e$. (v) Up to the values at internal nodes,* MinimiseLeaves(MRPTransform(f, τ)) $=$ MinimiseLeaves(MRPTransform(MinimiseLeaves$(f), \tau)$)*; if τ is injective,* MRPTransform *maps canonical MRPs to canonical MRPs.*

Proof. (i)–(iii) hold pointwise. (iv) Outside the support the represented function is $\tau(e)$. (v) Both sides represent $\tau \circ \bar{f}$, and the canonical form of a function is unique (Theorem 5.5). An injective τ keeps the values of two sibling leaves unequal, so by Lemma 5.4 a canonical MRP stays canonical. □

Marks. Proved here. [LEAN: (v) Canonical prop_transform_v, prop_transform_v_inj, with transform_represents ((i) as (v) uses it), for an MRP that represents some ϕ — built at `dbf40e07f`; (v) for every MRP, CanonicalScheme represents_leafFn (every MRP whose set of internal nodes is ancestor-closed represents its own leaf function, on every scheme), exists_represents, leafFn_comp, prop_transform_v_all, prop_transform_v_inj_all and prop_transform_v_mrp (the first law with no scheme) — built at `15a5577ef`; to write — (i)–(iv) and `Transform`.] [SIM: MRPTransform against pointwise evaluation for $\tau \in \{\exp,\ y \mapsto cy,\ y \mapsto \mathbf{1}[y \geq c]\}$]

With $\tau(y) = cy$ MRPTransform scales; with $\tau(y) = y/\int \bar{f}$ it normalises a nonnegative MRP to a density; and with $\tau(y) = \mathbf{1}[y \geq c]$ it turns a real MRP into the Boolean MRP of a superlevel set (§6.12). The exponential of a sparse real MRP with background 0 is, by (iv), the sparse MRP with background $\exp 0 = 1$ on the same stored labels. It is dense, with every cell stored, only when background 0 is required, for instance to combine it with other operands of background 0.

Chapter 6

The arithmetic, algorithm by algorithm

This section gives the operations on mapped trees one at a time. Each is presented in the same order: - its definition; - the recursive algorithm on a pointer tree, the form of MRS [SYH+18]; - the algorithm on a sorted leaf list with integer labels, the sparse form of SparkDensityTree [SGWS23a]; - its law, with a proof or a precise citation; - its cost; - its Spark form, where one exists, with the partition-invariance law it owes; - a small worked example; - the implementations.

The root box is fixed before the data are seen. Each output is again a mapped tree, computed exactly from the labels, so a joint estimate yields its marginal and conditional densities in closed form, not by numerical integration. Each algorithm is stated once; Appendix A collects them.

What uses only the labels holds for any fixed binary splitting scheme: the union, the pointwise operation, the pushforward, the folds, collation, the forest arithmetic and the Spark laws (Remark 4.31). Marginals and slices use in addition that cells are boxes split along coordinate axes (Remark 4.31), with the splits of the kept axes forming a fixed scheme of the projected box (Lemma 4.24 and Remark 4.30). The following parts also use the regular scheme itself, or at least its boxes and volumes: - the point-location rule $x_\iota \geq c$ at a midpoint; - UniformApprox and the enclosures, which evaluate interval extensions on boxes, and their termination, which uses Lemma 4.23(iv); - the volume-of-enclosure priority, whose strictness in Proposition 6.51(v) uses that a child has half its parent's volume; - the level sets and coverage values, which use the volumes of cells; - every worked example.

The implementation pointers name MRS 2.0 [SYH+18], with paths relative to `mrs-2.0/src/`, and SparkDensityTree [SGWS23a] at commit `c0a1837`, with line numbers at that commit.

Definition 6.1 (Cost model)**.** *Costs are counted in operations on words of ω bits in a random-access machine. A label of depth at most D occupies $\lambda = \lceil (D+2)/\omega \rceil$ words, and each label primitive (depth, truncate, join, equality, comparison, ancestor test, hashing) costs $O(\lambda)$ word operations. An arithmetic operation or a comparison on a count, a weight or a density costs $O(1)$. A hash-table operation on a label key costs $O(\lambda)$ in expectation (ideal hashing).*

For the pointer form the cost counts node visits, value operations, and evaluations of the functions supplied. A child's cell is computed from its parent's in $O(1)$ by halving the side along the split axis of its depth (Lemma 4.23(ii)), or in $O(d)$ when the box is copied.

Labels. The sparse algorithms use the heap codes of Chapter 4: the root is 1, and the children of n are $2n$ (left) and $2n+1$ (right). In these codes: - the depth is the bit length minus one; -

$\mathrm{sib}(u)$ flips the last bit; - the parent is a right shift; - $u|_t$ is the ancestor at depth t; - $u \wedge v$ is the deepest common ancestor; - $\prec$ is the left-right order, decided by Lemma 4.28.

A *sorted leaf list* is a list of pairs $(a, f(a))$ whose labels form an antichain in left-right order. It is *dense* when its labels are the leaves of a paving.

Fills. For nodes $a \sqsupseteq b$, let: - $\mathrm{FillLeft}(a, b)$ be the list, top-down, of the left siblings of the right children on the path from a (exclusive) down to b (inclusive); - $\mathrm{FillRight}(a, b)$ be the list, bottom-up, of the right siblings of the left children on the same path.

Both are empty when $a = b$.

Lemma 6.2 (Fills)**.** *For $a \sqsupseteq b$, the list $\mathrm{FillLeft}(a, b) + [b] + \mathrm{FillRight}(a, b)$ is the leaf list, in left-right order, of the paving of $\mathbf{x}_a$ whose internal nodes are the strict ancestors of b at or below a. It is computed with $O(\lambda(|b| - |a| + 1))$ word operations.*

Proof. The leaves of that paving are b and the siblings of the nodes on the path from a (exclusive) to b (inclusive). The sibling of a path node v precedes b when v is a right child, and follows it when v is a left child (Proposition 4.27(iv)). Consider two left siblings, of path nodes $v_1 \sqsupset v_2$. The shallower one precedes v_1, which contains the deeper one; so top-down is left to right. Symmetrically, for right siblings bottom-up is left to right. In heap codes a path node is a right child exactly when its last bit is 1. The path is read from the bits of b below the prefix a, and each sibling costs one bit flip. □

6.1 Spark operations and partition invariance

Spark holds data as a resilient distributed dataset (RDD) split into partitions, one task per partition, and recomputes a failed task from its lineage. It is modelled here as relational algebra over multisets [GCI+17].

Definition 6.3 (RDD and partition invariance)**.** *An RDD of elements of T with q partitions is a list $r = (r_1, \ldots, r_q)$ of finite lists of elements of T. Its* content *is the multiset of the elements of the concatenation $r_1 r_2 \cdots r_q$. A computation on RDDs is* partition-invariant *when its result depends on its input only through the content.*

A Spark program is partition-invariant when each of its steps is. Each kind of step owes one law, which is the proof obligation for that step (Table 6.1).

Two steps used below are not rows of the table. `zipPartitions`(g) applies g to the pairs of corresponding partitions of two RDDs and owes what `mapPartitions` owes; for the operations on a common refinement, the exactness theorem is the locality lemma of the partitioner (Lemma 6.4). A floating-point sum computed off the driver is made partition-invariant by an exact accumulator, which makes the sum a commutative monoid rounded once [HHKR95].

Which operations distribute. The operations of this section fall into five classes by the communication they need.

- *(a) Pointwise maps on records* need none: labelling a point, a unary map τ with $\tau(e) = e$, a slice, point location against a broadcast list.
- *(b) Reductions per id* of a labelled MRP need a shuffle by id.
- *(c) Pushforwards* need one shuffle by the image label, with a commutative monoid on the values: coarsening, summing out ids, counting, and the hulls of interval values (the compact intervals with the empty interval adjoined as the identity of $\sqcup$, §6.7).

Table 6.1: Spark operations, their mathematical meaning, and the law each owes for partition invariance.

Scala/Spark	Mathematics	Law owed
`map`, `filter`, `flatMap`	the same map on every partition	none: commutes with concatenation
`reduceByKey`, `aggregate`, `fold`, `reduce`	key-wise sum of finitely supported maps	the combiner is a commutative monoid [Lin13, CSW16]
accumulator `add`	sum in a commutative monoid	a commutative monoid, and every update applied once; Spark guarantees this only for updates made inside actions [The]
`mapPartitions`(g)	g on each partition	g a homomorphism for concatenation; otherwise an exactness theorem
`collect`	the content as a list, up to permutation	every consumer permutation-invariant, or a sort first
`sortBy`, `repartitionAndSortWithinPartitions`	sorting	a lawful total order on the values present: an antichain of labels, or labels of one depth (Lemma 4.28)
`broadcast`	a value read by every partition	none
floating-point sums	sums in $\mathbb{R}$	exact only in $\mathbb{R}$; every floating-point sum of the pipeline is sequential on the driver, hence deterministic

- *(d) Operations on the common refinement* are local to the partitions when both operands are partitioned by the same subtree partitioner and every stored leaf has depth at least its depth limit δ (Lemma 6.4): union, MRPOperate, collation, and the local passes of MinimiseLeaves and of the folds. Leaves shallower than δ must first be split to depth δ (intensive values) or broadcast.
- *(e) Global operations* need a sort, a selection or a pass on the driver: the thresholds of level sets, and the priority queues.

Proposition 6.51 moves the priority queues with strict priorities, such as the volume of enclosure, from a sequence of global steps to a threshold search: rounds of class (a), counts of class (c), and one selection of class (e), under the hypotheses stated with their Spark forms in §6.14 (a positive threshold, finite search sets). Subtree partitioning itself is Chapter 10 (§10.3).

Lemma 6.4 (Common refinements are local to subtrees)**.** *Let N be a finite set of nodes, all of depth at least δ. For a node w of depth δ let $N_w = \{a \in N : a \sqsubseteq w\}$. Then every cell of* $\mathrm{cl}(N)$ *lies below exactly one node of depth δ, and the cells of* $\mathrm{cl}(N)$ *below w are exactly those of* $\mathrm{cl}(N_w)$*:*

$$\mathrm{cl}(N) = \bigsqcup_{|w|=\delta} \mathrm{cl}(N_w).$$

For sparse MRPs f_1, f_2 with stored leaves of depth at least δ, the value of $\mathrm{MRPOperate}(f_1, f_2, \star, e)$ *at a cell below w depends only on the entries of f_1 and f_2 below w.*

Proof. Every cell of $\mathrm{cl}(N)$ lies in an element of N (Proposition 6.7(i)). That element has depth at least δ, so the cell lies below a unique node of depth δ. Let $a \in N_w$. Its strict descendants in N lie below w, so they are in N_w. Hence $a \in \max N$ exactly when $a \in \max N_w$, and $\mathrm{gaps}_N(a) = \mathrm{gaps}_{N_w}(a)$, since the gaps of a depend only on a and on the elements of N strictly below it. Conversely, a gap c of some $a \in N$ with $c \sqsubseteq w$ comes from an a nested with w and of depth at least δ, so $a \sqsubseteq w$. For MRPOperate, the element of $\mathrm{supp}\, f_k$ that contains a cell

below w has depth at least δ, so it lies below w. □

Marks. Proved here. [LEAN: Union lem_localRefine, cl_below_unique, cl_local (every cell of cl(N) lies below exactly one depth-δ node w, and the cells below w are those of cl(N_w); N need not be an antichain), union_local, operate_local (two pairs of sparse MRPs with the same entries below w have the same MRPOperate cells and values below w) — built at `050df0219`] [SIM: random pairs of sparse MRPs split at a random depth δ; the union per depth-δ block against the global union]

So let two RDDs be partitioned by one subtree partitioner with depth limit δ, every stored label having depth at least δ. Its lookup depends on such a label only through the label's ancestor at depth δ (Theorem B.8(iv)), so all the labels below one node of depth δ go to one partition. Then `zipPartitions` with the local merge computes the common refinement, and the result is partition-invariant: it does not depend on how the depth-δ blocks are grouped into partitions.

6.2 The union and the common refinement

Definition. The union $s^{(1)} \cup s^{(2)}$ of two pavings of one root box is the paving with $I(s^{(1)} \cup s^{(2)}) = I(s^{(1)}) \cup I(s^{(2)})$. For sparse supports the common refinement is the sparse closure (Definition 6.6).

Proposition 6.5 (Union of pavings)**.** *Let $s^{(1)}, s^{(2)}$ be pavings of one fixed splitting scheme, and let s be the paving with $I(s) = I(s^{(1)}) \cup I(s^{(2)})$. Then s is the coarsest common refinement of $s^{(1)}$ and $s^{(2)}$, and its leaves are the nonempty intersections of a leaf of $s^{(1)}$ with a leaf of $s^{(2)}$. Union is associative, commutative and idempotent, with the root paving s_0 as identity; so the pavings of a fixed root form a commutative monoid, indeed a join-semilattice with least element.*

Proof. $I(s)$ is finite and closed under taking ancestors, so s is a paving. A paving refines $s^{(i)}$ exactly when its internal-node set contains $I(s^{(i)})$, so s is the coarsest common refinement. Leaves of $s^{(1)}$ and $s^{(2)}$ are nested or disjoint, so a nonempty intersection of $l_1 \in \mathcal{L}(s^{(1)})$ and $l_2 \in \mathcal{L}(s^{(2)})$ is the smaller of the two, say $l_1 \sqsubseteq l_2$. No node at or below l_2 lies in $I(s^{(2)})$, since that set is closed under ancestors and does not contain the leaf l_2; so $l_1 \notin I(s^{(2)})$. Also $l_1 \notin I(s^{(1)})$, and its parent, if it has one, lies in $I(s^{(1)}) \subseteq I(s)$. Hence l_1 is a leaf of s. The nonempty intersections partition $\mathbf{x}_\rho$ and each is a leaf of s; a partition whose cells all belong to another partition of the same set equals it. Union of pavings is union of internal-node sets, which is associative, commutative and idempotent, with identity $\emptyset = I(s_0)$. □

Marks. Proved here. [LEAN: Union prop_union, refines_iff, union_monoid, refines_order, card_leaves_union_le (refinement stated on labels: every leaf of the finer paving lies at or below a leaf of the coarser; read on cells, the leaves-as-intersections clause needs the scheme's cells nonempty, as §5.1 sets them) — built at `050df0219`] [SIM: Algorithm 1 against the union of internal-node sets on random pairs of pavings]

The closure of regular pavings under union is [HST12]. The statement that union makes the regular pavings of a root box a commutative monoid is [Gra22]. The number of leaves satisfies $|\mathcal{L}(s^{(1)} \cup s^{(2)})| \leq |\mathcal{L}(s^{(1)})| + |\mathcal{L}(s^{(2)})| - 1$, since a paving has one more leaf than internal nodes.

Pointer form. Algorithm 1 recurses on pairs of nodes with one cell. Its output's internal nodes are those internal in either input, by induction on depth.

Cost. One node visit per node of the output, $O(|V(s^{(1)} \cup s^{(2)})|)$, with cells computed from the split axes.

Sparse form. For sparse MRPs the operands are antichains that need not cover the root box. The sparse common refinement and its leaf-list arithmetic extend the union of [HST12] to sparse maps [Gra22].

Algorithm 1: RPUnion(u_1, u_2): the union of two pavings, pointer form [HST12]

Input: nodes u_1, u_2 of two regular pavings of one root box, with $\mathbf{x}_{u_1} = \mathbf{x}_{u_2}$ (initially the two roots).

Output: a new node u with $\mathbf{x}_u = \mathbf{x}_{u_1}$, the root of the union of the subtrees at u_1 and u_2.

```
if u1 and u2 are leaves then
    return a new leaf with cell x_{u1};
else if u2 is a leaf then
    return Copy(u1);
else if u1 is a leaf then
    return Copy(u2);
else
    u ← a new node with cell x_{u1};
    uL ← RPUnion(u1L, u2L); uR ← RPUnion(u1R, u2R);
    return u;
```

Definition 6.6 (Sparse closure and common refinement)**.** *For a finite set N of nodes, let* $\max N$ *be the elements of N with no strict descendant in N, and for $a \in N$ let* $\mathrm{gaps}_N(a)$ *be the maximal nodes strictly below a that are disjoint from every element of N strictly below a. The* sparse closure *of N is*

$$\mathrm{cl}(N) = \max N \ \cup \bigcup_{a \in N \setminus \max N} \mathrm{gaps}_N(a).$$

The common refinement *of sparse MRPs with supports A and B is $U(A, B) = \mathrm{cl}(A \cup B)$.*

For example, with $A = \{\rho\mathsf{LLL}\}$ and $B = \{\rho\mathsf{L}\}$, $U(A, B) = \{\rho\mathsf{LLL}, \rho\mathsf{LLR}, \rho\mathsf{LR}\}$: the common refinement of two sparse supports can be much less sparse than either [Gra22].

Proposition 6.7 (Sparse common refinement)**.** *Let N be a finite set of nodes.*

- *(i)* $\mathrm{cl}(N)$ *is an antichain; its cells cover exactly* $\bigcup_{a \in N} \mathbf{x}_a$*; every cell of* $\mathrm{cl}(N)$ *lies in some element of N; and every $a \in N$ is the union of the cells of* $\mathrm{cl}(N)$ *below it. So every cell of* $\mathrm{cl}(N)$ *lies in or is disjoint from every element of N.*
- *(ii) If A and B are the leaf sets of pavings,* $\mathrm{cl}(A \cup B)$ *is the leaf set of their union.*
- *(iii) If A and B are antichains listed in left-right order, $U(A, B)$ is computed in left-right order by one merge of the two lists (Algorithm 2), in $O(\lambda(|A| + |B| + |U(A, B)|))$ word operations.*

Proof.

- *(i), antichain.* Two elements of $\max N$ are not nested, or one would have a strict descendant in N. The gaps of one a are the maximal nodes with a property inherited by descendants, so they are pairwise disjoint. Let g be a gap of a and $m \in \max N$. If m lies strictly below a, it is disjoint from g by definition. Otherwise m is disjoint from a, since $m \sqsupseteq a$ would give m the strict descendant a (as $a \notin \max N$). Let g' be a gap of $a' \neq a$. If a, a' are disjoint, so are g, g'; if $a' \sqsupset a$, then g' is disjoint from $a \sqsupseteq g$.
- *(i), cover.* Every cell of $\mathrm{cl}(N)$ lies in an element of N. Conversely let $x \in \mathbf{x}_a$, $a \in N$, and let a^* be the deepest element of N containing x. If $a^* \in \max N$, x is covered. Otherwise follow the cells containing x below a^*, and let g be the first of them that contains no element of N strictly below a^*; it exists because cells deeper than every element of N contain none. Each element n of N strictly below a^* is disjoint from g: $n \sqsubseteq g$ is excluded by the choice of g, and $n \sqsupset g$ would put x in n, against the choice of a^*. The parent of g

is a^* or contains such an n, so g is maximal, and $g \in \mathrm{gaps}_N(a^*)$.

- *(i), each a is a union of cells.* A cell c of $\mathrm{cl}(N)$ meeting $a \in N$ is nested with it. If $c \sqsupset a$, then c is not in $\max N$ (it has the strict descendant a), so c is a gap of some $a' \sqsupseteq c \sqsupset a$ and must be disjoint from a, a contradiction. So $c \sqsubseteq a$, and by the cover property the cells below a fill it.
- *(ii).* Let $a \in A$ have a strict descendant $b \in B$. The elements of B meeting a are nested with it, and none contains a, since it would contain b. So the elements of B below a partition $\mathbf{x}_a$, and $\mathrm{gaps}(a) = \emptyset$; likewise for B. Hence $\mathrm{cl}(A \cup B) = \max(A \cup B)$, whose elements are the smaller node of each nested pair, that is, the nonempty intersections; Proposition 6.5 identifies them with the leaves of the union.
- *(iii), correctness.* Algorithm 2 keeps the heads a of A and b of B and compares them, with $O(\lambda)$ word operations (Lemma 4.28).
  - If they are disjoint, it emits the one on the left and advances its list.
  - If they are equal, it emits it and advances both.
  - If $a \sqsupset b$, the elements of B below a form a run $b = b_1 \prec \cdots \prec b_r$ (Proposition 4.27(vi)), and none of them has an element of A below it, since A is an antichain containing a. The algorithm emits $\mathrm{Gaps}(a, [b_1, \ldots, b_r])$, advances A once and B by r. The case $b \sqsupset a$ is symmetric, and when one list is empty the rest of the other is emitted.

  $\mathrm{Gaps}(a, [b_1, \ldots, b_r])$ is the leaf list, in left-right order, of the paving of $\mathbf{x}_a$ whose internal nodes are the strict ancestors of the b_k at or below a. Indeed, by Lemma 6.2, the leaves before b_1 are $\mathrm{FillLeft}(a, b_1)$ and those after b_r are $\mathrm{FillRight}(a, b_r)$. A leaf strictly between b_k and b_{k+1} lies below $w = b_k \wedge b_{k+1}$ (Proposition 4.27(vi)), after b_k in $w\mathsf{L}$ or before b_{k+1} in $w\mathsf{R}$. These are $\mathrm{FillRight}(w\mathsf{L}, b_k)$ and $\mathrm{FillLeft}(w\mathsf{R}, b_{k+1})$. The leaves other than the b_k are exactly the maximal nodes below a disjoint from $b_1, \ldots, b_r$, that is, $\mathrm{gaps}(a)$.
- *(iii), cost.* Each comparison advances a list or emits a node. The path nodes walked inside a are the internal nodes of the paving just described, together with the b_k. A path node v is walked by $\mathrm{FillLeft}(a, b_1)$ only if $v \sqsupseteq b_1$, and by $\mathrm{FillRight}(a, b_r)$ only if $v \sqsupseteq b_r$. Inside a pair, it is walked only if the run below v ends at b_k or starts at b_{k+1}. So each path node is walked at most four times. That paving has one internal node fewer than leaves, so the walks cost $O(\lambda)$ per emitted cell, joins included. □

Marks. Proved here. [LEAN: SparseUnion cl_antichain, cl_sub, cl_in_or_disj, cl_cover — built at `550077078`, for (i); Union sparseUnion_ii, sparseUnion_ii_noGap for (ii), and for (iii) at the level of sets, cl_merge ($\mathrm{cl}(A \cup B)$ is the elements nested with no other together with, for each element with a strict descendant, its run and $\mathrm{gaps}(a)$), gaps_eq_leaves, run_antichain, run_max — built at `050df0219`; and for (iii) as lists, UnionMerge run_contiguous, gapsList_spec, sparseUnion_iii (Algorithm 2 branch for branch: left-right order, no duplicates, members exactly $\mathrm{cl}(A \cup B)$) — built at `9e6d7a947`; not certified: its cost] [SIM: Algorithm 2 against Algorithm 1 and against the closure computed by brute force, on random antichains to depth 12]

Spark form. Let both operands be partitioned by one subtree partitioner, with stored leaves of depth at least its depth limit δ. Then `zipPartitions` with Algorithm 2 on each pair of partitions computes $U(A, B)$. *Law:* $U(A, B) = \bigsqcup_{|w|=\delta} U(A_w, B_w)$ (Lemma 6.4), so the result is partition-invariant. Without a common partitioner, the operands are sorted by label (an antichain, so the order is lawful, Lemma 4.28) and range-partitioned on one set of depth-δ boundaries.

Example 6.8 (Union).

- *Dense.* On $[0, 8]$, the paving s_{221} (leaves $\rho\mathsf{LL}, \rho\mathsf{LR}, \rho\mathsf{R}$) and the paving s_{122} (leaves $\rho\mathsf{L}, \rho\mathsf{RL}, \rho\mathsf{RR}$) have internal nodes $\{\rho, \rho\mathsf{L}\}$ and $\{\rho, \rho\mathsf{R}\}$. Their union has internal nodes $\{\rho, \rho\mathsf{L}, \rho\mathsf{R}\}$ and is s_{2222} [HST12].
- *Sparse.* In heap codes, let $A = [2, 7]$ ($\rho\mathsf{L}, \rho\mathsf{RR}$) and $B = [8, 10, 6]$ ($\rho\mathsf{LLL}, \rho\mathsf{LRL}, \rho\mathsf{RL}$).
  - The heads are $2 \sqsupset 8$, and the run below 2 is $[8, 10]$, with $8 \wedge 10 = 2$.

Algorithm 2: SparseUnion(A, B): the common refinement, leaf-list form [Gra22]

Input: $A = [a_1, \ldots, a_p]$ and $B = [b_1, \ldots, b_q]$, antichains of labels in left-right order.
Output: $U(A, B) = \mathrm{cl}(A \cup B)$, in left-right order.

Function Gaps$(a, [c_1, \ldots, c_r])$ `//` $c_1 \prec \cdots \prec c_r$`, all strictly below` a**:**
  $G \leftarrow \mathrm{FillLeft}(a, c_1) + [c_1]$;
  for $k \leftarrow 1$ **to** $r - 1$ **do**
    $w \leftarrow c_k \wedge c_{k+1}$;
    $G \leftarrow G + \mathrm{FillRight}(w\mathsf{L}, c_k) + \mathrm{FillLeft}(w\mathsf{R}, c_{k+1}) + [c_{k+1}]$;
  return $G + \mathrm{FillRight}(a, c_r)$;
$U \leftarrow [\,]$; $i \leftarrow 1$; $j \leftarrow 1$;
while $i \leq p$ ***and*** $j \leq q$ **do**
  if $a_i = b_j$ **then**
    $U \leftarrow U + [a_i]$; $i \leftarrow i + 1$; $j \leftarrow j + 1$;
  else if $a_i \sqsupset b_j$ **then**
    $r \leftarrow j$; **while** $r < q$ ***and*** $b_{r+1} \sqsubseteq a_i$ **do** $r \leftarrow r + 1$;
    $U \leftarrow U + \mathrm{Gaps}(a_i, [b_j, \ldots, b_r])$; $i \leftarrow i + 1$; $j \leftarrow r + 1$;
  else if $b_j \sqsupset a_i$ **then**
    $r \leftarrow i$; **while** $r < p$ ***and*** $a_{r+1} \sqsubseteq b_j$ **do** $r \leftarrow r + 1$;
    $U \leftarrow U + \mathrm{Gaps}(b_j, [a_i, \ldots, a_r])$; $j \leftarrow j + 1$; $i \leftarrow r + 1$;
  else if $a_i \prec b_j$ **then**
    $U \leftarrow U + [a_i]$; $i \leftarrow i + 1$;
  else
    $U \leftarrow U + [b_j]$; $j \leftarrow j + 1$;
return $U + [a_i, \ldots, a_p] + [b_j, \ldots, b_q]$;

– Gaps$(2, [8, 10])$ emits FillLeft$(2, 8) = [\,]$, then 8, FillRight$(4, 8) = [9]$, FillLeft$(5, 10) = [\,]$, 10, and FillRight$(2, 10) = [11]$.
– The heads are now 7 and 6, which are disjoint with $6 \prec 7$: the merge emits 6, and then the rest of A, which is 7.

So $U(A, B) = [8, 9, 10, 11, 6, 7]$: the four depth-3 cells of $\rho\mathsf{L}$, then $\rho\mathsf{RL}$ and $\rho\mathsf{RR}$. The two gaps 9 and 11 were stored in neither operand.

Implementations: MRS `SPnode::reshapeToUnion` (`spnode.cpp`), which reshapes a tree in place to the union, and `PiecewiseConstantFunction::reshapeToUnion` (`piecewise_constant_function.cpp`). SparkDensityTree, related code: `rpUnionNested` (`Truncation.scala` lines 307–426), the merge for the case in which every leaf of one operand lies in a leaf of the other, and `rpUnion` (lines 431–475).

6.3 Point location and the pointwise image

Definition. The *pointwise image* of an MRP at $x \in \mathbf{x}_\rho$ is $\bar{f}(x) = f(\eta_s(x))$, the value of the leaf whose cell contains x (§5.1). Outside the root box it is not defined.

Algorithm 3: PointWiseImage(ρ, x): point location, pointer form [HST12]

Input: the root ρ of an $\mathbb{A}$-MRP (s, f) in either form; a point $x \in \mathbf{x}_\rho$.
Output: $f(\eta_s(x))$.

$u \leftarrow \rho$;
while u is not a leaf **do**
  $c \leftarrow$ the midpoint of $\mathbf{x}_u$ along the split axis $\iota_{|u|}$;
  if $x_{\iota_{|u|}} \geq c$ **then** $u \leftarrow u\mathsf{R}$ **else** $u \leftarrow u\mathsf{L}$;
return $f(u)$;

Algorithm 4: PointWiseImage(L, x, e): point location, leaf-list form

Input: a sorted leaf list $L = [(a_1, v_1), \ldots, (a_N, v_N)]$ of a sparse MRP with background e; a point $x \in \mathbf{x}_\rho$; $D \geq \max_i |a_i|$.
Output: $\bar{f}(x)$.

$\ell \leftarrow \rho$; $\mathbf{b} \leftarrow \mathbf{x}_\rho$;
for $k \leftarrow 0$ **to** $D - 1$ **do** // $\ell = \ell_D(x)$, the depth-D cell of x
  $c \leftarrow$ the midpoint of $\mathbf{b}$ along ι_k;
  if $x_{\iota_k} \geq c$ **then** $\ell \leftarrow \ell\mathsf{R}$; $\underline{b}_{\iota_k} \leftarrow c$ **else** $\ell \leftarrow \ell\mathsf{L}$; $\overline{b}_{\iota_k} \leftarrow c$;
$lo \leftarrow 1$; $hi \leftarrow N$;
while $lo \leq hi$ **do**
  $m \leftarrow \lfloor (lo + hi)/2 \rfloor$; $t \leftarrow \ell|_{|a_m|}$;
  if $t = a_m$ **then**
    return v_m;
  else if $t \prec a_m$ **then**
    $hi \leftarrow m - 1$;
  else
    $lo \leftarrow m + 1$;
return e;

Proposition 6.9 (Point location)**.** *(i) Algorithm 3 returns $f(\eta_s(x))$ after $|\eta_s(x)|$ comparisons. (ii) Algorithm 4 returns $\bar{f}(x)$ with $O(D)$ comparisons of coordinates, D bit appends to build the*

label $\ell_D(x)$ (in all $O(D+\lambda)$ word operations when the label is built in place, $O(\lambda D)$ when each append copies it), and $O(\lambda \log N)$ word operations for the search. (iii) For Q query points sorted by their depth-D labels, one merge with L answers all of them in $O(\lambda(N+Q))$ word operations.

Proof. (i) The children of a cell partition it, and x lies in the right child exactly when $x_\iota \geq c$ (§5.1). So the loop keeps $x \in \mathbf{x}_u$, and it stops at the leaf containing x.

(ii) The first loop computes the depth-D cell ℓ of x by the same rule, and $x \in \mathbf{x}_a$ exactly when $\ell \sqsubseteq a$, that is, when $\ell|_{|a|} = a$. At most one a_m contains ℓ. For every other a_m the nodes ℓ and a_m are disjoint. Then $\ell|_{|a_m|}$ is a node of depth $|a_m|$ distinct from a_m, and $\ell \prec a_m$ exactly when $\ell|_{|a_m|} \prec a_m$, since the left-right order passes to descendants (Proposition 4.27). The indices with $a_m \prec \ell$ form a prefix of the sorted list, and those with $\ell \prec a_m$ a suffix, so the binary search is valid. If no a_m contains ℓ, the point lies in no stored cell and the value is e.

(iii) Both lists are in left-right order, and the pointer into L only advances; this is the scan of Lemma 11.11. □

Marks. Proved here. [LEAN: to write — `PointImage`; (iii) is `Scan lem_scan_ptrs`, built at `5857eebb7`] [SIM: random points against brute-force cell membership, including points on split values and on the upper faces of the root box]

Spark form. Broadcast the sorted leaf list and map each query to its value, or, for many queries, sort them by their depth-D labels and merge (iii). *Law:* the lookup is a map, so it owes nothing; the sort of the many-query form owes the sort row of Table 6.1, which labels of one depth satisfy (Lemma 4.28).

Example 6.10 (Point location)**.** Let f be the density on $[0,1]^2$ with leaves $\rho\mathsf{L}$, $\rho\mathsf{RL}$, $\rho\mathsf{RR}$ and values 0.4, 2.4, 0.8. The root is split along x_1 at 0.5, and $\rho\mathsf{R} = [0.5,1] \times [0,1]$ is split along x_2 at 0.5. Its mass is $0.4 \cdot 0.5 + 2.4 \cdot 0.25 + 0.8 \cdot 0.25 = 1$, and it serves as the running example of this section. - *Pointer form.* For $x = (0.75, 0.6)$, the root sends x right since $0.75 \geq 0.5$, and $\rho\mathsf{R}$ sends it right since $0.6 \geq 0.5$; so $\bar{f}(x) = 0.8$. - *Leaf-list form.* With heap codes, $L = [(2, 0.4), (6, 2.4), (7, 0.8)]$ and $\ell_2(x) = 7$, and the binary search finds 7.

Implementations: MRS `RealMappedSPnode::findContainingNode` (`realmappedspnode.cpp`), with `SPnode::nodeContains` (`spnode.cpp`), and `PiecewiseConstantFunction::pointwiseExtension` (`piecewise_constant_function.cpp`); the same for `IntervalMappedSPnode` and `BooleanValueMappedSPnode`. SparkDensityTree labels points with `quickDescend` and `quickDescendBox` (`SpatialTree.scala` lines 149–206) .

6.4 MinimiseLeaves

Definition. MinimiseLeaves returns the canonical form of an MRP: the paving $s^*(\bar{f})$ with the values of $\bar{f}$ on its leaves (Theorem 5.5).

Proposition 6.11 (MinimiseLeaves on a sorted leaf list)**.** *Let L be the sorted leaf list of an MRP representing ϕ: dense, or sparse with background e and no stored value e. Algorithm 6 returns, in left-right order, the leaves of $s^*(\phi)$ whose value is not e (all of them, for a dense list), with their values. It costs $O(\lambda N)$ word operations and at most $N-1$ comparisons of values, for $N = |L|$.*

Proof. Three invariants hold after each step.

Algorithm 5: MinimiseLeaves(u): pointer form [HST12]

Input: a node u of an $\mathbb{A}$-MRP in either form (initially the root); equality on $\mathbb{A}$.
Output: the subtree at u, modified in place into its canonical form.

```
1 if u is not a leaf then
2     MinimiseLeaves(uL); MinimiseLeaves(uR);
3     if uL and uR are leaves and f(uL) = f(uR) then
4         f(u) ← f(uL); Prune(uL); Prune(uR);
```

Algorithm 6: MinimiseLeaves(L): leaf-list form

Input: a sorted leaf list L of a dense MRP, or of a sparse MRP with no stored value equal to its background.
Output: the canonical form, as a sorted leaf list.

```
1 S ← an empty stack;
2 foreach (a, v) ∈ L, in order do
3     push (a, v) onto S;
4     while |S| ≥ 2 and the top two entries are (uL, y), (uR, y) for some node u and value y
        do
5         pop both; push (u, y);
6 return the entries of S, bottom to top;
```

- *(a) The stack is an antichain in left-right order, and an entry contains no unpushed input leaf.* A new input leaf is disjoint from every entry, since an entry is a union of earlier input leaves and the input is an antichain. It follows every entry in left-right order, since it follows the earlier leaves. Replacing two adjacent siblings by their parent keeps the order.
- *(b) Every reduction is a merge of two sibling cells of equal value.* So the stack, with the gaps given the value e, represents ϕ on the cells already pushed.
- *(c) No two adjacent entries are siblings of equal value.* A step pushes one entry and reduces only at the top. Each reduction replaces the top two entries by one, so every adjacent pair except the top one existed before the step, and the loop ends only when the top pair is not reducible.

Let F be the final stack, and s_F the paving whose internal nodes are the strict ancestors of the elements of F. Its leaves are F and the gaps, which carry e; by (b) it represents ϕ.

Consider a cherry u of s_F. Since u is a strict ancestor of an element of F, one of its children contains an element of F. That child is a leaf, so it is that element. If both children are in F, they are adjacent in F, since nothing lies between two siblings, and by (c) their values differ. If one child is a gap, its value e differs from every stored value.

So no cherry of s_F has equal leaf values, and $s_F = s^*(\phi)$ by Lemma 5.4.

For the cost, values are compared only when the top two entries are siblings, and each comparison either succeeds, and reduces, or fails, and ends the step. Each input pair is pushed once and each reduction removes one entry, so there are $N - |F|$ reductions. After a failed comparison of a top pair $(u\mathsf{L}, y), (u\mathsf{R}, y')$, neither entry is ever reduced, since each can be reduced only with the other and their values do not change; and the pair is never on top again, since an entry pushed above $u\mathsf{R}$ can be reduced only into an entry that is still above it. So the failed comparisons concern distinct pairs of sibling entries of F, at most $|F|/2$ of them, and none when $|F| = 1$. The number of comparisons of values is at most $N - |F| + |F|/2 \leq N - 1$ when $|F| \geq 2$, and $N - 1$ when $|F| = 1$. Each sibling test costs $O(\lambda)$. □

Marks. Proved here. [LEAN: Canonical prop_minimiseStack_sparse, prop_minimiseStack_dense, with reduce_spec, pass_spec (the invariants, (c) as "no two entries are siblings of equal value", equivalent by sib_adjacent) and stack_canonical — built at `dbf40e07f`, with the scheme's cells nonempty as a hypothesis (`Scheme.cell_nonempty`, as for Theorem 5.5(iii)); the cost sentence ($O(\lambda N)$ word operations, at most $N-1$ comparisons) is not machine-checked.] [SIM: Algorithm 6 against Algorithm 5 on random MRPs with few distinct values, dense and sparse]

Cost. Pointer form: $O(|V(s)|)$ node visits and at most $|I(s)|$ comparisons of values. Leaf-list form: as in Proposition 6.11. Values merge when they are equal under the value type's equality, which for floating-point numbers is not bitwise identity: -0 equals $+0$, and a NaN equals nothing, itself included.

Spark form. Partition by subtrees with depth limit δ, every stored leaf of depth at least δ, and the leaves sorted within each partition. - Run Algorithm 6 on each depth-δ block w within each partition. By Proposition 6.11 it returns the canonical form of ϕ on $\mathbf{x}_w$. - If ϕ is not constant on $\mathbf{x}_w$, the block's outputs lie strictly below w and are final. The internal nodes of $s^*(\phi)$ below w are the nodes of $\mathcal{N}(\phi)$ below w, and these depend only on ϕ on $\mathbf{x}_w$. - If ϕ is constant on $\mathbf{x}_w$, the block's output is w itself, or nothing when the value is the background. Collect these outputs of depth exactly δ, at most one per block, sort them, and run Algorithm 6 once more on the driver.

Law: the driver's list represents the function ϕ_δ equal to ϕ on the blocks on which ϕ is constant with a value other than the background, and equal elsewhere to the background e, or, for a dense list, to a value $\perp$ outside every stored value. Every stored entry of the driver's list has depth δ, so the canonical leaves of ϕ_δ have depth at most δ. A node v of depth at most δ on which ϕ_δ is constant with a stored value c contains only blocks on which ϕ is constant with value c, so ϕ is constant on v; conversely, if ϕ is constant on v with a stored value, so is ϕ_δ. Applied to v and to its parent, this shows that the canonical leaves of ϕ_δ with a stored value are exactly the canonical leaves of ϕ of depth at most δ with a stored value. With the final outputs of the non-constant blocks, which are the canonical leaves of ϕ strictly below them, the result is the canonical form of ϕ, equal to the single-machine result whatever the partitioning.

Example 6.12 (MinimiseLeaves)**.** The sum $f+g$ of Example 6.15 has the leaf list, in heap codes,

$$[(4,6),(5,6),(12,2),(13,4),(14,4),(15,3)].$$

- The stack pass pushes $(4,6)$ and $(5,6)$, which are siblings of equal value, and replaces them by $(2,6)$. - The pairs $(12,2),(13,4)$ and $(14,4),(15,3)$ are siblings with unequal values, and $13,14$ are not siblings.

The canonical form is $[(2,6),(12,2),(13,4),(14,4),(15,3)]$: the paving s_{13333} with values $6,2,4,4,3$. This is the minimised sum of [HST12, Example 1].

Implementations: MRS `MappedSPnode<T>::minimiseLeaves` (`mappedspnode.hpp`). SparkDensityTree: none at `c0a1837`.

6.5 The pointwise operation

Definition 6.13 (MRPOperate)**.** *Let f_1, f_2 be sparse $\mathbb{A}$-MRPs with background e and supports A, B, and let $\star : \mathbb{A}^2 \to \mathbb{A}$ satisfy $e \star e = e$. For a node u and a sparse MRP f, let $f^{\uparrow}(u) = f(a)$ if $u \sqsubseteq a \in \operatorname{supp} f$, and $f^{\uparrow}(u) = e$ if u is disjoint from every element of $\operatorname{supp} f$. Then* $\mathrm{MRPOperate}(f_1, f_2, \star, e)$ *is the sparse MRP with value $f_1^{\uparrow}(u) \star f_2^{\uparrow}(u)$ at each $u \in U(A,B)$, cells with value e dropped.*

By Proposition 6.7(i), $f^{\uparrow}$ is defined at every cell of $U(A,B)$ for $f = f_1, f_2$: an intensive value is copied down to the finer cells. For dense MRPs no background is needed, and MRPOperate gives each leaf of the union of the two pavings the value $\star$ of the values of the leaves containing it.

Algorithm 7: MRPOperate$(u_1, u_2, \star)$: pointer form [HST12]

Input: nodes u_1 of (s_1, f) and u_2 of (s_2, g), $\mathbb{A}$-MRPs of one root box in multi-resolution form, with $\mathbf{x}_{u_1} = \mathbf{x}_{u_2}$; a map $\star : \mathbb{A}^2 \to \mathbb{A}$.

Output: a new node u, the root of $h = f \star g$ on the union of the subtrees at u_1 and u_2.

Function Child(v, X) // $X \in \{\mathsf{L}, \mathsf{R}\}$**:**
  if v is not a leaf **then return** vX **else return** a temporary leaf with cell $\mathbf{x}_{vX}$ and the value of v;
$u \leftarrow$ a new node with cell $\mathbf{x}_{u_1}$ and value $h(u) \leftarrow f(u_1) \star g(u_2)$;
if u_1 or u_2 is not a leaf **then**
  $u\mathsf{L} \leftarrow$ MRPOperate(Child(u_1, L), Child(u_2, L), $\star$);
  $u\mathsf{R} \leftarrow$ MRPOperate(Child(u_1, R), Child(u_2, R), $\star$);
return u;

Algorithm 8: MRPOperate$(f_1, f_2, \star, e)$: leaf-list form

Input: f_1, f_2, sparse $\mathbb{A}$-MRPs with background e, as sorted leaf lists; $\star$ with $e \star e = e$.

Output: the sparse MRP of Definition 6.13, as a sorted leaf list.

Run Algorithm 2 on the labels of f_1 and f_2, and give each emitted cell c a value v as follows;
  heads equal, $a_i = b_j = c$: $v \leftarrow f_1(a_i) \star f_2(b_j)$;
  $a_i \sqsupset b_j$: $v \leftarrow f_1(a_i) \star f_2(c)$ for c in the run, and $v \leftarrow f_1(a_i) \star e$ for c a gap of a_i;
  $b_j \sqsupset a_i$: $v \leftarrow f_1(c) \star f_2(b_j)$ for c in the run, and $v \leftarrow e \star f_2(b_j)$ for c a gap of b_j;
  a single head or a remaining element: $v \leftarrow f_1(a_i) \star e$, or $v \leftarrow e \star f_2(b_j)$;
Emit (c, v) only when $v \neq e$;

Proposition 6.14 (Law and cost of MRPOperate)**.** *(i)* MRPOperate$(f_1, f_2, \star, e)$ *represents* $x \mapsto f_1(x) \star f_2(x)$*. Hence, as represented functions, it is commutative or associative whenever* $\star$ *is, and the constant function* e *is its identity whenever* e *is an identity of* $\star$*. (ii) It is computed in one pass of the merge of Proposition 6.7(iii), with* $O(\lambda(|A| + |B| + |U(A,B)|))$ *word operations and* $|U(A,B)|$ *evaluations of* $\star$*.*

Proof. Let x lie in a cell u of $U(A,B)$. By Proposition 6.7(i), u lies in an element a of A or is disjoint from all of them; in the first case $f_1(x) = f_1(a) = f_1^{\uparrow}(u)$, in the second $f_1(x) = e = f_1^{\uparrow}(u)$. Likewise for f_2. If x lies in no cell of $U(A,B)$, it lies in no element of $A \cup B$, both values are e, and $e \star e = e$ is the background. In the merge, the element of A or B covering an emitted cell is the current head or the cell itself, so each value costs one evaluation of $\star$. □

Marks. Proved here. [LEAN: Operate prop_operate_i ((i), for sparse MRPs whose supports are antichains, their stored leaves, and $e \star e = e$), operate_comm, operate_id, operate_assoc, with upVal_cl, upVal_point, isOperate_antichain — built at `b6ea25813`; operate_assoc compares the two sides at points deeper than the two intermediate supports as well, which is equality as represented functions since the supports are finite; not certified: (ii), the one pass and its cost] [SIM: random pairs of sparse MRPs with $\star \in \{+, \cdot, \max, \vee, \text{xor}\}$, Algorithm 8 against Algorithm 7 and against pointwise evaluation at random points]

The pointer form computes the same leaf values on the union of the two pavings. It also gives every internal node u the value $f(u_1) \star g(u_2)$ of the two nodes it pairs; for intensive values this

is the operation applied to the two coarser views. Its cost is one node and one evaluation of $\star$ per node of the union, $O(|V(s_1 \cup s_2)|)$.

Copy-down is right for intensive values only (Definition 5.19). - *Averages.* With $\star$ the midpoint on densities, MRPOperate averages two histograms. - *Sums.* With $\star = +$ it adds them. - *Sets.* On Booleans, $\vee$, $\wedge$, exclusive or and $a \wedge \neg b$ give union, intersection, symmetric difference and set difference. - *Counts.* Counts cannot be copied down on a common refinement; they must be re-read from the data. Two count tables of disjoint samples merge by adding counts label by label only when both are at a common depth, as the finest count tables of Phase 2 are, or on a common coarsening.

Spark form. As for the union: co-partition both operands by one subtree partitioner with depth limit δ, and run Algorithm 8 with `zipPartitions`. *Law:* by Lemma 6.4, the value at a cell below a depth-δ node w depends only on the entries below w, so the result is partition-invariant.

Example 6.15 (Adding two real MRPs)**.** On $[0, 8]$, let

$$f = 2 \cdot \mathbf{1}_{\mathbf{x}_{\rho\mathsf{LL}}} + 4 \cdot \mathbf{1}_{\mathbf{x}_{\rho\mathsf{LR}}} + \mathbf{1}_{\mathbf{x}_{\rho\mathsf{R}}}$$

on s_{221}. Let g take the values $4, 2, 1, 3, 3, 2$ on $\rho\mathsf{LL}, \rho\mathsf{LR}, \rho\mathsf{RLL}, \rho\mathsf{RLR}, \rho\mathsf{RRL}, \rho\mathsf{RRR}$, on s_{223333} [HST12, Example 1]. - *Leaf lists.* In heap codes, $f = [(4, 2), (5, 4), (3, 1)]$ and $g = [(4, 4), (5, 2), (12, 1), (13, 3), (14, 3), (15, 2)]$. - *The merge.* It emits $(4, 6)$ and $(5, 6)$ from equal heads. Then $3 \sqsupset 12$ with the run $[12, 13, 14, 15]$, which fills $\rho\mathsf{R}$, so Gaps emits no gap. The run gets $1+1$, $1+3$, $1+3$, $1+2$. - *The sum.* $f+g = [(4, 6), (5, 6), (12, 2), (13, 4), (14, 4), (15, 3)]$, with values $6, 6, 2, 4, 4, 3$ on s_{223333}. Its canonical form is in Example 6.12. - *Sets.* With Boolean values and $\star \in \{\vee, \text{xor}, \wedge, a \wedge \neg b\}$ the same merge gives the union, symmetric difference, intersection and set difference of two sets made of cells, as in the discs of [HST12, Example 2].

Implementations: MRS `MappedSPnode<T>::operator+=`, `-=`, `*=`, `/=` and their non-mutating forms (`mappedspnode.hpp`), which reshape both trees to the union and combine the values node by node; `PiecewiseConstantFunction::operator+=` and the others (`piecewise_constant_function.cpp`); `SpatialObjectRepresentationBV::operator+` and the others for Booleans (`spatial_object_representation_bv.hpp`). SparkDensityTree, related code: `mrpOperate` (`LeafMap.scala` lines 179–240), whose value pass assumes a union in left-right order.

6.6 Unary maps: MRPTransform

The definition and laws are Definition 5.21 and Proposition 5.22.

Algorithm 9: MRPTransform: pointer and leaf-list forms [HST12]

Input: an MRP; a map $\tau : \mathbb{A} \to \mathbb{B}$.
Output: MRPTransform(f, τ).

Function TransformTree(u) `// pointer form, from the root:`
  $f(u) \leftarrow \tau(f(u))$;
  if u is not a leaf **then** TransformTree$(u\mathsf{L})$; TransformTree$(u\mathsf{R})$;
Function TransformList(L, e) `// leaf-list form; the new background is` $\tau(e)$`:`
  return $[(a, \tau(v)) : (a, v) \in L,\ \tau(v) \neq \tau(e)]$;

Cost. $|V(s)|$ evaluations of τ in the pointer form, and N in the leaf-list form.

Spark form. `mapValues`(τ) followed by `filter`. *Law:* a map, so none is owed. It keeps the sparse background exactly when $\tau(e) = e$ (Proposition 5.22(iv)).

Example 6.16 (Normalising)**.** The canonical sum of Example 6.12 has integral $6 \cdot 4 + (2 + 4 + 4 + 3) \cdot 1 = 37$ over $[0, 8]$. MRPTransform with $\tau(y) = y/37$ gives the density with values $6/37, 2/37, 4/37, 4/37, 3/37$ on s_{13333}. MRPTransform with $\tau(y) = \mathbf{1}[y \geq 4]$ gives the Boolean MRP of the superlevel set $\{f + g \geq 4\} = \mathbf{x}_{\rho\mathsf{L}} \cup \mathbf{x}_{\rho\mathsf{RLR}} \cup \mathbf{x}_{\rho\mathsf{RRL}}$.

Implementations: MRS scalar operators `MappedSPnode<T>::operator+=(const T&)` and the others, through `_scalarAdd`, `_scalarMult` and their kin (`mappedspnode.hpp`), for $\tau(y) = y \star c$, and the visitor `acceptSPValueVisitor` (`mappedspnode.hpp`) for a general τ. SparkDensityTree `normalize` (`DensityHistogram.scala` lines 75–84), the instance $\tau(y) = y/c$.

6.7 Pushforward: relabelling, coarsening, counting and folds

Definition 6.17 (Pushforward)**.** *Let* $(\mathbb{M}, +, 0)$ *be a commutative monoid,* c *a finitely supported map from labels to* $\mathbb{M}$*, and* f *a map on labels. The* pushforward *is* $(\mathrm{relabel}_f\, c)(v) = \sum_{l:f(l)=v} c(l)$*. Coarsening to depth* j *is* $\mathrm{coarsen}_j = \mathrm{relabel}_{(\cdot)|_j}$*. The* count *of a node is* $\#_c(u) = \sum_{l \sqsubseteq u} c(l)$*. For a sample,* $\mathrm{countOf}_D(x_1, \ldots, x_n) = \sum_i \mathbf{1}_{\{\ell_D(x_i)\}}$*, where* $\ell_D(x)$ *is the depth-*D *cell containing* x*.*

Proposition 6.18 (Laws and cost of the pushforward)**.** *(i)* $\mathrm{relabel}_f$ *is a monoid homomorphism and preserves the total* $\sum_v c(v)$*. (ii)* $\mathrm{relabel}_{g \circ f} = \mathrm{relabel}_g \circ \mathrm{relabel}_f$ *and* $\mathrm{relabel}_{\mathrm{id}} = \mathrm{id}$*. (iii)* $\mathrm{coarsen}_j \circ \mathrm{coarsen}_k = \mathrm{coarsen}_{\min(j,k)}$*, and* $\mathrm{coarsen}_j\, \mathrm{countOf}_D = \mathrm{countOf}_j$ *for* $j \leq D$*. (iv) The count map of a concatenation of samples is the sum of their count maps, in any order and any grouping. (v)* $\#_c$ *is additive in* c*, and for* c *supported at depth* D *and* $|u| < D$*,* $\#_c(u) = \#_c(u\mathsf{L}) + \#_c(u\mathsf{R})$*. (vi) On a list in left-right order whose labels form an antichain, truncation to any depth is weakly monotone, so equal truncations are adjacent and* $\mathrm{coarsen}_j$ *is one pass of* $O(\lambda N)$ *word operations; a general relabelling costs* $O(\lambda N \log N)$ *by sorting.*

Proof. (i) The pushforward is a regrouping of the finite sum $\sum_l c(l)\mathbf{1}_{\{f(l)\}}$, linear in c. (ii) Regroup by $g(f(l))$. (iii) $(l|_k)|_j = l|_{\min(j,k)}$, and $\ell_j(x) = \ell_D(x)|_j$. (iv) $\mathrm{countOf}_D$ is a sum over the points, and addition in $\mathbb{Z}_{\geq 0}$ is associative and commutative. (v) Additivity is (i); the depth-D descendants of u are those of $u\mathsf{L}$ and those of $u\mathsf{R}$. (vi) Let $u \prec v$ be elements of the antichain and t a depth, and suppose $u|_t \neq v|_t$. These nodes are disjoint: if both have depth t they are distinct nodes of one depth; if both are shallower than t they are u and v; if only one is, say $u|_t = u$, then u cannot lie in $v|_t$ (its depth is smaller), and $v|_t$ cannot lie in u (then v would). Their blocks are disjoint intervals containing $\mathrm{blk}_D(u)$ and $\mathrm{blk}_D(v)$ respectively, and $\mathrm{blk}_D(u)$ precedes $\mathrm{blk}_D(v)$, so $u|_t \prec v|_t$. □

Marks. Proved here. [LEAN: CountMap merge_comm, merge_assoc, merge_zero, relabel_add, total_relabel, relabel_comp, relabel_id, coarsen_add, coarsen_coarsen, total_coarsen, coarsen_countOf, countOf_append, countOf_perm, countOf_flatten, mass_add, mass_children; MRPLift for every commutative monoid — built at `c06220d95`, for (i)–(v); Fold prop_relabel_vi, prop_relabel_vi_adjacent — built at `b6ea25813`, for (vi)] [SIM: random samples, counted at depth D and coarsened, against counting at depth j directly; the coarsening pass against a hash-map relabelling]

Folds. A pushforward to every depth at once is a *fold*: it gives each internal node the sum of the values below it. For counts it gives $\#_c(u)$ at every node, which is the multi-resolution form of an SRP. The compact intervals under the hull $\sqcup$ have no identity, so for folds they are taken with the empty interval $\emptyset$ adjoined, $\emptyset \sqcup X = X \sqcup \emptyset = X$; then $(\mathbb{IR} \cup \{\emptyset\}, \sqcup, \emptyset)$ is a commutative and idempotent monoid. With the hull in place of $+$ the fold is HullPropagate (§6.14).

Proposition 6.19 (Folds)**.** *Let* $(\mathbb{M}, \oplus, e)$ *be a monoid.*

Algorithm 10: Fold, Coarsen and Insert: pointer forms

Input: an MRP in multi-resolution form with values in a monoid $(\mathbb{M}, \oplus, e)$.

Function Fold(u) `// post-order:`
  if u is not a leaf **then**
    Fold($u\mathsf{L}$); Fold($u\mathsf{R}$); $f(u) \leftarrow f(u\mathsf{L}) \oplus f(u\mathsf{R})$;

Function Coarsen(u, j) `// on a folded MRP: every internal value is the sum of its children:`
  if u is not a leaf **then**
    if $|u| = j$ **then** Prune($u\mathsf{L}$); Prune($u\mathsf{R}$) **else** Coarsen($u\mathsf{L}, j$); Coarsen($u\mathsf{R}, j$);

Function Insert(x) `// counting one point` $x \in \mathbf{x}_\rho$ `into an SRP:`
  $u \leftarrow \rho$; $\#(u) \leftarrow \#(u) + 1$;
  while u is not a leaf **do**
    $u \leftarrow$ the child of u containing x (Algorithm 3); $\#(u) \leftarrow \#(u) + 1$;

Algorithm 11: Fold(L): leaf-list form, a path sweep

Input: a sorted leaf list $L = [(a_1, v_1), \ldots, (a_N, v_N)]$ of a sparse MRP whose background is the identity e of a monoid $(\mathbb{M}, \oplus, e)$.

Output: every strict ancestor u of a stored label, with $\bigoplus_{a_i \sqsubseteq u} v_i$ (in left-right order), in post-order.

$P \leftarrow$ an empty stack of pairs (node, running value);
for $i \leftarrow 1$ **to** N **do**
  while $P \neq [\,]$ ***and*** the top node of P is not an ancestor of a_i **do**
    pop (u, y) and emit it; **if** $P \neq [\,]$ **then** replace the running value y' of the new top by $y' \oplus y$;
  push, with value e, every node strictly above a_i and strictly below the top node of P (from ρ if P is empty), top-down;
  if $P \neq [\,]$ **then** replace the running value y' of the top by $y' \oplus v_i$ `// the parent of` a_i;
pop and emit every remaining pair as above;

- *(i) After* Fold(ρ) *of Algorithm 10, every node u carries $\bigoplus_{l\in\mathcal{L}(s),\, l\sqsubseteq u} f(l)$, in left-right order, after $|I(s)|$ applications of $\oplus$.*
- *(ii) On a folded MRP with values in a commutative monoid,* Coarsen(ρ, j) *returns the MRP of* coarsen$_j$ *of its leaf values.* Insert *keeps an SRP folded.*
- *(iii) Algorithm 11 emits every strict ancestor u of a stored label exactly once, with $\bigoplus_{a_i\sqsubseteq u} v_i$, with $O(\lambda(N+M))$ word operations, where M is the number of such ancestors.*
- *(iv) For a count map c the folded value at u is $\#_c(u)$. For intervals with $\oplus = \sqcup$, the empty interval adjoined as e, it is the hull of the leaf intervals below u.*

Proof.

- *(i).* Induction on height.
- *(ii).* At a node of depth j the folded value is the sum of the leaf values below it, which is the value of coarsen$_j$ there; leaves shallower than j are unchanged. Insert adds 1 along the path to the leaf containing x, which is 1 at every ancestor of that leaf.
- *(iii).* The stack always holds a chain of nested nodes, a path from the root downwards. A popped node u contains an earlier label but not a_i. The labels below a node are consecutive in the list (Proposition 4.27(vi)), so u contains no later label either. Hence each strict ancestor of a stored label is pushed once and popped once, after the last label below it. Its running value receives the value of each child subtree, in left-right order: a stored leaf adds its value directly, and a child that is itself an ancestor adds its total when popped. A gap child contributes e. Each push and pop costs $O(\lambda)$.
- *(iv).* This is the definition of $\#_c$, and of the hull. □

Marks. Proved here. [LEAN: Fold prop_fold_i, foldRoot_leaf ((i), in any monoid: the summed list is exactly the leaves below u, in left-right order), prop_fold_ii_coarsen, prop_fold_ii_insert ((ii)), prop_fold_iv_count, prop_fold_iv_countMap, folded_mass ((iv) for count maps, beside `CountMap mass_children` and `MRPLift mass_eq_sum`) — built at `b6ea25813`; (iv) for intervals is Proposition 6.50(i), machine-checked-structural (see its mark); not certified: (iii), the path sweep, and the costs] [SIM: the path sweep against Algorithm 10 for counts and for hulls, on random sparse MRPs]

Spark form. The pushforward is a `map` to the image label followed by `reduceByKey` in $\mathbb{M}$; Phase 2 is the case countOf$_D$. *Law:* the pushforward is a monoid homomorphism (Proposition 6.18(i)), so a reduction with a commutative combiner computes it in any grouping (Table 6.1); for the counts of samples this is Proposition 6.18(iv). A fold to every depth is local below the depth limit δ of a subtree partitioner: run Algorithm 11 per partition, then reduce the values of the nodes shallower than δ by key on the driver. *Law:* commutativity of $\oplus$, which $+$ and $\sqcup$ have, with an identity, which $\sqcup$ has once $\emptyset$ is adjoined.

Example 6.20 (Counting and coarsening)**.** On $[0, 8]$ at depth 3, the cell $[k, k+1)$, or $[7, 8]$ for $k = 7$, has heap code $8 + k$. The sample $0.5, 1.5, 2.5, 6.2, 6.7, 7.9$ gives the count map

$$\text{countOf}_3 = \{8{:}1, 9{:}1, 10{:}1, 14{:}2, 15{:}1\}.$$

- *Coarsening.* Truncating to depth 2 is a right shift, which gives $\{4{:}2, 5{:}1, 7{:}3\}$. Truncating once more gives $\{2{:}3, 3{:}3\}$, the same as coarsening to depth 1 directly. The total stays 6. - *Folding.* The path sweep over countOf$_3$ emits 4:2, 5:1, 2:3, 7:3, 3:3 and 1:6: the counts at every strict ancestor of an occupied cell. The node 6, with no point below it, is not emitted.

Implementations: MRS keeps a count at every node of `SPSnode` (`spsnode.hpp`). It counts with `AdaptiveHistogram::insertOne` and `insertFromRVec` (`adaptivehistogram.hpp`), and coarsens by `SPSnode::nodeReabsorbChildren` (`spsnode.hpp`), driven by `AdaptiveHistogram::priorityMerge` and `mergeUp` (`adaptivehistogram.hpp`). SparkDensityTree labels

and counts with `quickToLabeled` (`MergeEstimator.scala` lines 96–102). It truncates with `NodeLabel.truncate` (`NodeLabel.scala` lines 60–63), and collapses a subtree to its root with `LeafMap.mergeSubtree` (`LeafMap.scala`).

6.8 Marginals

Let $\mathbf{x}_\rho \subset \mathbb{R}^d$, let $A \subseteq \{1, \dots, d\}$ be the axes to keep, nonempty and renumbered in increasing order, and let π_A be the projection onto them, with the regular paving of $\mathbf{x}'_\rho = \pi_A \mathbf{x}_\rho$.

The depths at which the paving of $\mathbf{x}_\rho$ splits an axis of A, taken in order, split the axes of $\mathbf{x}'_\rho$ in the order of its own regular bisection (Lemma 4.24, with $K = A$): - a step that splits a kept axis splits a widest side, hence a widest kept side, with ties broken by the lowest index; - a step that splits a dropped axis leaves the kept widths unchanged.

Write $\kappa_A(l)$ for the node of the paving of $\mathbf{x}'_\rho$ whose turns are the turns of l at those depths. By Lemma 4.24, $\pi_A \mathbf{x}_l = \mathbf{x}'_{\kappa_A(l)}$.

Definition 6.21 (Marginal)**.** *Let f be a sparse real MRP with $e = 0$ (a density histogram) and support S, and $K = \kappa_A(S)$. For $k \in K$ let $g_k = \sum_{l \in S,\ \kappa_A(l) = k} f(l) \operatorname{vol}(\mathbf{x}_l)/\operatorname{vol}(\mathbf{x}'_k)$. The* marginal *of f on the axes A is the sparse MRP on $\operatorname{cl}(K)$ with value $f'(u) = \sum_{k \in K,\ u \sqsubseteq k} g_k$.*

Proposition 6.22 (Law and cost of the marginal)**.** *(i) The marginal represents $y \mapsto \int f(y, z)\, dz$ on $\mathbf{x}'_\rho$, and $\sum_u f'(u) \operatorname{vol}(\mathbf{x}'_u) = \sum_{l \in S} f(l) \operatorname{vol}(\mathbf{x}_l)$. (ii) From a list of N leaves it is computed with $O(ND)$ bit operations for the relabelling, $O(\lambda N \log N)$ word operations to sort the keys ancestors first, and $O(\lambda(|K| + |\operatorname{cl}(K)|))$ for one sweep that emits $\operatorname{cl}(K)$ with a stack of the open ancestors and their running sums.*

Proof. (i) A leaf box is the product of its projections, so $\int f(y, z)\, dz = \sum_l f(l) \operatorname{vol}(\mathbf{x}_l)/\operatorname{vol}(\pi_A \mathbf{x}_l)\, \mathbf{1}[y \in \pi_A \mathbf{x}_l] = \sum_{k \in K} g_k \mathbf{1}[y \in \mathbf{x}'_k]$. For y in a cell u of $\operatorname{cl}(K)$, $y \in \mathbf{x}'_k$ exactly when $u \sqsubseteq k$, by Proposition 6.7(i); for y outside these cells the sum is empty. Integrating, $\sum_u f'(u) \operatorname{vol}(\mathbf{x}'_u) = \sum_k g_k \operatorname{vol}(\mathbf{x}'_k) = \sum_l f(l) \operatorname{vol}(\mathbf{x}_l)$, since each $\mathbf{x}'_k$ is the union of the cells below it. (ii) The relabelling reads the $|l| \le D$ turns of each leaf. With the keys sorted by the start of their block and then by depth, the open keys containing the current key form a chain; the sweep (Algorithm 13) keeps it on a stack with the running sum of g along it, emits the gaps of each open key as in Proposition 6.7(iii), and gives each emitted cell the running sum. □

Marks. Proved here. [LEAN: to write — `Marginal`: κ_A is the regular paving of the projected box; (i) on finite cell sets] [SIM: marginals of random sparse histograms in $d \le 4$ against numerical integration on a fine grid, and Algorithm 13 against Algorithm 12]

By [DL01, Theorem 5.2], the L_1 distance between two marginal densities is at most that between the joint densities, so an exact marginal inherits the error bound of the joint estimate. The marginal of a density histogram integrates to 1 when the histogram does, by (i).

Proposition 6.23 (The pointer form of the marginal)**.** Marginalise$(\rho, 0, A)$ *returns an $\mathbb{R}$-MRP of $\mathbf{x}'_\rho$ that represents $y \mapsto \int f(y, z)\, dz$. It costs $O\big(\sum_{v \in V(s)} (|v| + 1)\big)$ node operations, that is $O(|V(s)|\, D)$ for a paving of depth D.*

Proof. By induction on height, the call at u returns an MRP of the regular paving of $\pi_A \mathbf{x}_u$ that represents $M_u(y) = \int_{\{z : (y,z) \in \mathbf{x}_u\}} f(y, z)\, dz$.

- *A leaf.* f is constant on $\mathbf{x}_u = \pi_A \mathbf{x}_u \times \pi_{A^c} \mathbf{x}_u$, so $M_u = f(u) \operatorname{vol}(\pi_{A^c} \mathbf{x}_u) = f(u) \operatorname{vol}(\mathbf{x}_u)/\operatorname{vol}(\pi_A \mathbf{x}_u)$.

Algorithm 12: Marginalise(u, h, A): pointer form [HST12]

Input: a node u of depth h of an $\mathbb{R}$-MRP (initially the root, $h = 0$); A, the axes to keep, nonempty.

Output: the root of an $\mathbb{R}$-MRP on $\pi_A \mathbf{x}_u$ representing $y \mapsto \int f(y, z)\, dz$, the integral over the fibre of $\mathbf{x}_u$.

if u is a leaf **then**
  $f(u) \leftarrow f(u) \cdot \mathrm{vol}(\mathbf{x}_u)/\mathrm{vol}(\pi_A \mathbf{x}_u)$; `// the old d-dimensional volume`
  $\mathbf{x}_u \leftarrow \pi_A \mathbf{x}_u$; **return** u;
$v_{\mathsf{L}} \leftarrow$ Marginalise$(u\mathsf{L}, h + 1, A)$; $v_{\mathsf{R}} \leftarrow$ Marginalise$(u\mathsf{R}, h + 1, A)$;
if $\iota_h \in A$ `// a kept axis: the projected cell is split` **then**
  $\mathbf{x}_u \leftarrow \pi_A \mathbf{x}_u$; make $v_{\mathsf{L}}, v_{\mathsf{R}}$ the children of u; **return** u;
else `// a dropped axis: both halves project onto` $\pi_A \mathbf{x}_u$
  return MRPOperate$(v_{\mathsf{L}}, v_{\mathsf{R}}, +)$; `// Algorithm 7`

- *A dropped axis, $\iota_h \notin A$.* The two halves have the same projection $\pi_A \mathbf{x}_u$, and $M_u = M_{u\mathsf{L}} + M_{u\mathsf{R}}$ on it. MRPOperate adds the two MRPs (Proposition 6.14).
- *A kept axis, $\iota_h \in A$.* ι_h is the first widest axis of $\mathbf{x}_u$, hence the first widest of the kept axes in increasing order. So $\pi_A \mathbf{x}_{u\mathsf{L}}$ and $\pi_A \mathbf{x}_{u\mathsf{R}}$ are the two children of $\pi_A \mathbf{x}_u$ in the regular paving of $\mathbf{x}'_\rho$ (Lemma 4.24). For y in one of them, the fibre of $\mathbf{x}_u$ over y is the fibre of that half, so M_u is $M_{u\mathsf{L}}$ on the first and $M_{u\mathsf{R}}$ on the second.

For the cost, the marginal of the subtree at v has at most as many internal nodes as that subtree, by the same induction: a union has at most the sum of the internal nodes of its operands (Proposition 6.5). So the union at u costs $O(n_u)$, with n_u the number of nodes of the subtree at u. Summing over u counts each node once per ancestor-or-self. □

Marks. Proved here. [LEAN: to write — `MarginalTree`] [SIM: as for Proposition 6.22]

Algorithm 13: Marginal(L, A): leaf-list form

Input: the sorted leaf list L of a density histogram ($e = 0$) on the regular paving of $\mathbf{x}_\rho$; A, the axes to keep, in increasing order; the split axes $\iota_0, \ldots, \iota_{D-1}$.

Output: the marginal of Definition 6.21, as a sorted leaf list on the regular paving of $\pi_A \mathbf{x}_\rho$.

foreach $(l, v) \in L$ **do** emit $\big(\kappa_A(l),\ v\,\mathrm{vol}(\mathbf{x}_l)/\mathrm{vol}(\mathbf{x}'_{\kappa_A(l)})\big)$ `//` $\kappa_A(l)$`: the turns of` l `at depths` t `with` $\iota_t \in A$;
$g \leftarrow$ the emitted pairs summed by key; $K \leftarrow$ the keys, sorted by the start of their depth-D block and then by depth, so ancestors before descendants and otherwise in left-right order;
build the forest of K under $\sqsubseteq$ with one stack pass over K ; `// the parent of a key is the deepest key strictly above it`
$Out \leftarrow [\,]$;
Function Emit(a, r) `//` r`: the running sum of` g `over the keys strictly above` a`:`
  $r \leftarrow r + g_a$; $C \leftarrow$ the children of a in the forest, in left-right order;
  if $C = [\,]$ **then**
    $Out \leftarrow Out + [(a, r)]$;
  else
    foreach $c \in$ Gaps(a, C), in order `// Algorithm 2` **do**
      if $c \in C$ **then** Emit(c, r) **else** $Out \leftarrow Out + [(c, r)]$;
foreach root r of the forest, in left-right order **do** Emit$(r, 0)$;
return Out;

In Algorithm 13 the cells emitted are $\max K$ and the gaps of the other keys, which is $\mathrm{cl}(K)$. Each cell receives the sum of g over the keys above it, accumulated down the forest.

Spark form. Fix a depth δ', split K into the shallow keys $K_s = \{k : |k| < \delta'\}$, at most $2^{\delta'} - 1$ of them, and the deep keys $K_d = \{k : |k| \geq \delta'\}$, and let $W_d = \{k|_{\delta'} : k \in K_d\}$ be the occupied blocks of depth δ'. - Relabel by κ_A with a `map`, and sum by key with `reduceByKey`. *Law:* a pushforward (Proposition 6.18(i)), with a commutative combiner. - Collect K_s with its sums g, and W_d by a `map` to $k|_{\delta'}$ and a `distinct`, and broadcast K_s. - *Driver.* Run Algorithm 13 on $K_s \cup W_d$, with $g = 0$ on the elements of W_d, and drop the emitted cells that lie in W_d. - *Partitions.* Range-partition K_d by the block $k|_{\delta'}$, sorted within each partition by the order of Algorithm 13, a total order on labels. For each occupied block w with deep keys K_w, let $r_w = \sum_{k \in K_s,\, w \sqsubseteq k} g_k$, read from the broadcast. If some shallow key lies above w, run the sweep on the forest of $K_w \cup \{w\}$, rooted at w with g_w increased by r_w; otherwise run it on the forest of K_w alone.

Law. Let T be the paving whose internal nodes are the strict ancestors of K. The cells of $\mathrm{cl}(K)$ are the leaves of T that lie in some key: the elements of $\max K$ and the gaps of a key are such leaves, and a leaf of T inside a key is in $\max K$ or is a gap of the deepest key that contains it. A leaf of T that lies in no occupied block lies in a shallow key, and it is a leaf, other than an element of W_d, of the paving whose internal nodes are the strict ancestors of $K_s \cup W_d$; these are the cells the driver emits. A leaf of T below an occupied block w lies in a key of K_w, or, when a shallow key lies above w, in w itself; these are the cells the sweep of block w emits. The value is $f'(u) = \sum_{k \sqsupseteq u,\, |k| < \delta'} g_k + \sum_{k \sqsupseteq u,\, |k| \geq \delta'} g_k$; for u below w the shallow keys above u are those above w, and the deep keys above u lie in K_w. So each cell is emitted once, with its value, from data that do not depend on the partitioning, and the result is partition-invariant. (Checked against the definition on 2000 random key sets, `scratchpad/r2_check.py`.)

Example 6.24 (Marginals of the running example)**.** Take the density f of Example 6.10, with heap codes 2, 6, 7 and values 0.4, 2.4, 0.8. The split axes are $\iota_0 = 1$ and $\iota_1 = 2$.

- *Keeping x_1, $A = \{1\}$.* The keys are $\kappa(2) = 2$, $\kappa(6) = 3$ and $\kappa(7) = 3$, with weights $0.4 \cdot 0.5/0.5 = 0.4$, $2.4 \cdot 0.25/0.5 = 1.2$ and $0.8 \cdot 0.25/0.5 = 0.4$. So $g = \{2{:}0.4, 3{:}1.6\}$, and the marginal of X_1 is 0.4 on $[0, 0.5)$ and 1.6 on $[0.5, 1]$.
- *Keeping x_2, $A = \{2\}$.* The leaf $\rho\mathsf{L}$ has no turn along x_2, so $\kappa(2) = 1$, the root, with weight $0.4 \cdot 0.5/1 = 0.2$. Also $\kappa(6) = 2$ with weight 1.2, and $\kappa(7) = 3$ with weight 0.4. The keys are nested: $\mathrm{cl}(\{1, 2, 3\}) = \{2, 3\}$, and the root's weight is added to both. The marginal of X_2 is $0.2 + 1.2 = 1.4$ on $[0, 0.5)$ and $0.2 + 0.4 = 0.6$ on $[0.5, 1]$.
- *Pointer form, keeping x_2.* The root splits the dropped axis x_1. Its left leaf becomes 0.2 on $[0, 1]$. Its right child splits the kept axis x_2 and becomes 1.2 and 0.4 on the two halves. MRPOperate adds the two, giving 1.4 and 0.6.

Both marginals integrate to 1.

Implementations: MRS `RealMappedSPnode::marginalise` (`realmappedspnode.cpp`), `PiecewiseConstantFunction::makeMarginal` (`piecewise_constant_function.cpp`), and `AdaptiveHistogramCollator::makeMarginal` (`adaptivehistogramcollator.hpp`) for collations. SparkDensityTree `marginalize` (`DensityHistogram.scala` lines 186–280), with `minimalCompletion` (`Truncation.scala` lines 219–260).

6.9 Slices and conditional densities

Let Λ be the axes to condition on, with $0 < |\Lambda| < d$, let $x' \in \pi_\Lambda \mathbf{x}_\rho$ be the values, and let $\mathbf{x}''_\rho = \pi_{\Lambda^c} \mathbf{x}_\rho$ with its regular paving. As for marginals, the depths at which the paving of $\mathbf{x}_\rho$

splits an axis outside Λ split $\mathbf{x}''_\rho$ in its own order (Lemma 4.24, with $K = \Lambda^c$). Write $\kappa_{x'}(l)$ for the node of the paving of $\mathbf{x}''_\rho$ given by the turns of l at those depths.

Definition 6.25 (Slice)**.** *A leaf l of a sparse MRP f is* kept *when $x' \in \pi_\Lambda \mathbf{x}_l$, the projection being a product of intervals $[a, b)$, or $[a, b]$ at the upper end of the root box. The* slice *of f at x' is the sparse MRP on $\mathbf{x}''_\rho$ with value $f(l)$ at $\kappa_{x'}(l)$ for every kept l. For a density, the* conditional density *given $x_\Lambda = x'$ is the slice divided by $I = \sum_u f(u) \operatorname{vol}(\mathbf{x}''_u)$, when $0 < I < \infty$.*

Proposition 6.26 (Law and cost of the slice)**.** *(i) $\kappa_{x'}$ is injective on the kept leaves, its image is an antichain, and it preserves the left-right order. (ii) The slice represents $y \mapsto f(y \oplus x')$ on $\mathbf{x}''_\rho$, where $y \oplus x'$ has coordinates y off Λ and x' on Λ; its values are unchanged, as for any intensive value. (iii) From a sorted leaf list it costs $O(ND)$ bit operations in one pass, with no sort.*

Proof. (i) Let $l_1 \neq l_2$ be kept leaves and $w = l_1 \wedge l_2$. If w splits an axis $j \in \Lambda$, the projections of $\mathbf{x}_{l_1}$ and $\mathbf{x}_{l_2}$ on j lie in the two disjoint halves, and cannot both contain x'_j. So w splits an axis outside Λ, and $\kappa_{x'}(l_1)$, $\kappa_{x'}(l_2)$ lie below the two different children of $\kappa_{x'}(w)$, on the same sides as l_1, l_2 below w; by Proposition 4.27(iv) they are disjoint and in the same order. (ii) $\mathbf{x}_l = \pi_\Lambda \mathbf{x}_l \times \mathbf{x}''_{\kappa_{x'}(l)}$, so $y \oplus x' \in \mathbf{x}_l$ exactly when l is kept and $y \in \mathbf{x}''_{\kappa_{x'}(l)}$. (iii) Each leaf costs one pass over its at most D turns, and by (i) the output comes out in order. □

Marks. Proved here. [LEAN: to write — `Slice`, (i) on `Labels` and `PropBlock`] [SIM: slices of random sparse MRPs at random points, including split values, against pointwise evaluation of $f(y \oplus x')$]

The normaliser I is the marginal density of X_Λ at x': by Proposition 6.22(i) with the kept axes Λ, that density is $\int \bar{f}(z \oplus x')\, dz$ over $z \in \mathbf{x}''_\rho$, and by Proposition 6.26(ii) this is the integral of the slice, $\sum_u f(u) \operatorname{vol}(\mathbf{x}''_u) = I$. So the conditional density of a joint histogram is exact. Conditioning is not a map of densities in the sense of [DL01, Theorem 5.2], so the slices inherit no L_1 bound from the joint estimate.

Algorithm 14: Slice(u, h, Λ, x'): pointer form [HST12]

Input: a node u of depth h of an $\mathbb{A}$-MRP with $x' \in \pi_\Lambda \mathbf{x}_u$ (initially the root, $h = 0$); Λ with $0 < |\Lambda| < d$; $x' = (x'_j)_{j \in \Lambda}$.

Output: the root of an $\mathbb{A}$-MRP on $\pi_{\Lambda^c} \mathbf{x}_u$ representing $y \mapsto \bar{f}(y \oplus x')$.

```
if u is a leaf then
    x_u ← π_{Λ^c} x_u; return u;
else if ι_h ∉ Λ then
    uL ← Slice(uL, h + 1, Λ, x'); uR ← Slice(uR, h + 1, Λ, x');
    x_u ← π_{Λ^c} x_u; return u;
else
    c ← the midpoint of x_u along ι_h;
    if x'_{ι_h} < c then v ← Slice(uL, h + 1, Λ, x') else v ← Slice(uR, h + 1, Λ, x');
    discard u and its other subtree; return v;
```

Proposition 6.27 (The pointer form of the slice)**.** *For $x' \in \pi_\Lambda \mathbf{x}_\rho$,* Slice$(\rho, 0, \Lambda, x')$ *returns an MRP of the regular paving of $\mathbf{x}''_\rho$ that represents $y \mapsto \bar{f}(y \oplus x')$. It visits only the nodes u with $x' \in \pi_\Lambda \mathbf{x}_u$.*

Proof. By induction on height, for a node u with $x' \in \pi_\Lambda \mathbf{x}_u$.

- *A leaf.* The value is constant.

- *A free axis,* $\iota_h \notin \Lambda$. Both halves have the projection $\pi_\Lambda \mathbf{x}_u \ni x'$. Their projections onto Λ^c are the two children of $\pi_{\Lambda^c}\mathbf{x}_u$ in the paving of $\mathbf{x}''_\rho$, since ι_h is the first widest of the free axes (Lemma 4.24).
- *A fixed axis,* $\iota_h \in \Lambda$. Both halves have the projection $\pi_{\Lambda^c}\mathbf{x}_u$ onto the free axes. Exactly one half contains x' in its Λ-projection, since the left half is open at the split value, and the test selects it. For $y \in \pi_{\Lambda^c}\mathbf{x}_u$, the point $y \oplus x'$ lies in that half. □

Marks. Proved here. [LEAN: to write — `SliceTree`] [SIM: Algorithm 14 against Algorithm 15]

Algorithm 15: Slice(L, Λ, x'): leaf-list form

Input: a sorted leaf list L of a sparse MRP on the regular paving of $\mathbf{x}_\rho$; Λ with $0 < |\Lambda| < d$; $x' \in \pi_\Lambda \mathbf{x}_\rho$; the split axes $\iota_0, \dots, \iota_{D-1}$.

Output: the slice of Definition 6.25, as a sorted leaf list on the regular paving of $\mathbf{x}''_\rho$.

```
Out ← [];
foreach (l, v) ∈ L, in order do
    v' ← ρ; b ← x_ρ; keep ← True;
    for t ← 0 to |l| − 1 do
        ι ← ι_t; c ← the midpoint of b along ι; θ ← the turn of l at depth t;
        if ι ∈ Λ then
            if (θ = L and x'_ι ≥ c) or (θ = R and x'_ι < c) then keep ← False; break;
        else v' ← v'θ;
        if θ = R then b_ι ← c else b̄_ι ← c;
    if keep then Out ← Out + [(v', v)] // v' = κ_{x'}(l);
return Out;
```

Spark form. A `filter` of the kept leaves followed by a `map` to $\kappa_{x'}$, with no shuffle. By Proposition 6.26(i), a partition sorted in left-right order stays sorted. The normaliser I is a sum, computed on the driver or with an exact accumulator. *Law:* a map, so none is owed; the sum is a commutative monoid.

Example 6.28 (Slices of the running example)**.** Take the density f of Example 6.10.

- *At* $x_1 = 0.75$. The leaf ρL is not kept, since $0.75 \notin [0, 0.5)$. The leaves ρRL and ρRR are kept, with $\kappa = \rho$L and ρR on the paving of $[0, 1]$ in x_2. The slice is 2.4 on $[0, 0.5)$ and 0.8 on $[0.5, 1]$, with $I = 1.6$, the marginal of X_1 at 0.75 (Example 6.24). The conditional density of X_2 given $X_1 = 0.75$ is 1.5 on $[0, 0.5)$ and 0.5 on $[0.5, 1]$.
- *At* $x_2 = 0.25$. The root splits the free axis x_1, so both halves are kept. In the right half, ρR splits the fixed axis x_2, and $0.25 < 0.5$ selects ρRL. The slice is 0.4 on $[0, 0.5)$ and 2.4 on $[0.5, 1]$, with $I = 1.4$. The conditional density of X_1 given $X_2 = 0.25$ is 2/7 and 12/7.

Implementations: MRS `MappedSPnode<T>::slice` (`mappedspnode.hpp`), with the checks of `sliceCheck`, and `PiecewiseConstantFunction::makeSlice` (`piecewise_constant_function.cpp`). SparkDensityTree `quickSlice` (`DensityHistogram.scala` lines 347–487), with the split axes from `splitOrderToDepth` (`SpatialTree.scala` lines 116–134), and `normalize` (lines 75–84).

6.10 Collation and averages

Definition 6.29 (Collation)**.** *Let* f_θ, $\theta \in \Theta$, *be sparse density MRPs with* $e = 0$ *on one root box. Their* collation *is the labelled MRP on* $\mathrm{cl}\big(\bigcup_\theta \operatorname{supp} f_\theta\big)$ *whose value at* u *is the map* $\theta \mapsto f_\theta^{\uparrow}(u)$,

with background the zero map. For two collations with disjoint key sets it is MRPOperate with the union of key maps, zero-filled, as $\star$.

Algorithms. Collation is MRPOperate with $\star = \uplus$, the union of two maps with disjoint key sets. - The pointer form is Algorithm 7: a leaf's map is copied down. - The leaf-list form is Algorithm 8, with the background the map that is zero on every key.

Collating h maps takes $h - 1$ such steps, in any order.

Law. As represented functions, collation is $x \mapsto (f_\theta(x))_{\theta\in\Theta}$, so it is commutative and associative for disjoint key sets (Proposition 6.14). The *average* $\frac{1}{|\Theta|}\sum_\theta f_\theta$ is MRPTransform of the collation with $m \mapsto \frac{1}{|\Theta|}\sum_\theta m(\theta)$. It represents the pointwise average, and it integrates to 1 when every f_θ does. Summing a collation over its keys is the pushforward that sums out the ids (Definition 6.32).

Cost. Collating h maps by $h - 1$ successive steps costs $O\big(\lambda\sum_\theta |\operatorname{supp} f_\theta| + h(\lambda + h)|U|\big)$ word operations, where U is the final refinement. Each intermediate refinement is coarser than U (Proposition 6.7(i)), and each step copies maps of at most h keys.

When the candidates are successive coarsenings of one histogram, as along the backtrack of Chapter 9, U is not the finest support A_f: a leaf of a coarser candidate also covers empty gaps between the finest cells, and these enter U. For example, with one point in each of the finest cells $A_\mathrm{f} = \{\rho\mathsf{LL}, \rho\mathsf{RR}\}$ and a candidate coarsened to $\{\rho\}$, $U = \{\rho\mathsf{LL}, \rho\mathsf{LR}, \rho\mathsf{RL}, \rho\mathsf{RR}\}$: on the two gaps $\rho\mathsf{LR}$ and $\rho\mathsf{RL}$ the coarse candidate has density $2/(n\operatorname{vol}(\mathbf{x}_\rho))$ and the finest has 0, and both values enter the Scheffé integrals. Every candidate leaf is a node at or above a finest cell, so the strict ancestors of all the supports are those of A_f, and the leaves of the coarsest candidate, with support A_c, contain those of the others. Hence $U = U(A_\mathrm{c}, A_\mathrm{f})$, the finest support together with the gaps of the coarsest candidate's leaves, emitted in one merge by Algorithm 2 in its nested case (the case of `rpUnionNested`). Collation is then a lookup: for each candidate, one pass over U with a pointer into the candidate's sorted leaf list gives each cell the value of the candidate leaf above it, or 0 when there is none, as in Proposition 6.9(iii).

Spark form. As for MRPOperate, with `zipPartitions` on co-partitioned operands (Lemma 6.4). Alternatively, the collation is a labelled sparse MRP with the candidates as ids. *Law:* Lemma 6.4; for averages, the sum over keys is a commutative monoid.

Example 6.30 (Collation and average)**.** On $[0, 1]$, let f_1 be 1.5 on $\rho\mathsf{L}$ and 0.5 on $\rho\mathsf{R}$. Let f_2 be 1 on $\rho\mathsf{L}$, 0.4 on $\rho\mathsf{RL}$ and 1.6 on $\rho\mathsf{RR}$. Both integrate to 1.

- *Collation.* The common refinement is $\{\rho\mathsf{L}, \rho\mathsf{RL}, \rho\mathsf{RR}\}$. The collation maps $\rho\mathsf{L} \mapsto (1.5, 1)$, $\rho\mathsf{RL} \mapsto (0.5, 0.4)$ and $\rho\mathsf{RR} \mapsto (0.5, 1.6)$; the value 0.5 of f_1 on $\rho\mathsf{R}$ is copied down.
- *Average.* The average is 1.25, 0.45 and 1.05 on the three cells, with integral $0.625 + 0.1125 + 0.2625 = 1$.

Implementations: MRS `CollatorSPnode` (`collatorspnode.hpp`) and `AdaptiveHistogramCollator` (`adaptivehistogramcollator.hpp`), with `addToCollation` and `makeAverage`. SparkDensityTree `collate` (`DensityHistogram.scala` lines 533–553), with `collatorOp` (lines 504–511).

6.11 Labelled arithmetic over a forest of root boxes

[TKS12] analysed radar-observed aircraft trajectories with statistical regular pavings.

- *Data.* Arrivals at a busy North American airport over periods of six hours, in two and three dimensions.

- *One SRP per flight.* Each flight was represented by its own SRP, with leaf values in $\{0, 1\}$ marking the cells the flight occupied in the period.
- *Arithmetic.* Flights were added into an aggregate co-trajectory, and the aggregates of a day with good weather and of a day with bad weather were subtracted.
- *A dynamic structure.* Each landed flight was subtracted, and empty sibling leaves were reunited, so that the structure kept only the aircraft in the air.

The paper's motivation was that data on aircraft positions can be used to estimate, and help control, the workload of air traffic controllers, together with local environmental impacts and airport or airspace throughput [TKS12, §I, §VII]. It is stated here only as that paper's motivation; the arithmetic below is what such uses need.

The same pattern recurs whenever many objects live in one space: flights in an airspace, cells in a microscopy image, candidate histograms in a selection. Each object has its own mapped tree, and the questions are about all of them at once.

Definition 6.31 (Forest of root boxes)**.** *Let $\mathbb{I}$ be a set of ids and, for each $i \in \mathbb{I}$, let $\mathbf{x}_\rho^{(i)} \subset \mathbb{R}^{d_i}$ be a root box with its regular paving.*

- *A* labelled node *is a pair (i, u) with u a node of the tree of $\mathbf{x}_\rho^{(i)}$. Its cell is $\{i\} \times \mathbf{x}_u^{(i)}$ in the disjoint union $\bigsqcup_i \mathbf{x}_\rho^{(i)}$.*
- *A* forest MRP *is a finitely supported map from labelled nodes to $\mathbb{A} \setminus \{e\}$ whose support meets each tree in an antichain.*
- *Labelled nodes are ordered first by id, then by $\prec$. A sorted forest list is the concatenation of the sorted leaf lists of its trees.*

When all root boxes are one box $\mathbf{x}_\rho$ and the values lie in a commutative monoid with background 0, a forest MRP represents, id by id, the same family of functions as a labelled sparse MRP (Definition 5.20), its collation; its node support over all ids need not be an antichain. The ids then index objects overlaid on one paving.

Two uses.

- *One root box per id.* The ids index domains of different dimensions. The envelope of the trans-dimensional Moore rejection sampler is an $\mathbb{IR}$-valued forest MRP over labelled boxes, one root box per model (§3.3), refined by bisecting the first widest side in order of priority (§3.2) [SY13].
- *Many objects on one paving.* All ids share one root box, and an id is an object: a flight, or a cell segmented in a microscopy image. The value of (i, u) is what object i does in cell u. For a flight it is its occupancy, in $\{0, 1\}$: whether the flight was observed in the cell during the period. Counts of flights appear only after the ids are summed out. For other objects the value can be a count of observations or an intensity.

Every operation of Chapter 6 acts tree by tree, and a sorted forest list is processed tree after tree. A forest MRP has finitely many records, so for infinitely many ids the forest MRPs over $\mathbb{I}$ form the restricted product (the direct sum) of the structures of the single trees, not their full product. What is new on one shared paving is the passage between objects and the space they share.

A dynamic forest. At time t the forest holds the objects of a finite set $\mathbb{I}_t$. Adding object i adds its records $\{(i, u, v)\}$; removing it deletes every record with id i. Both are local to the id, and neither touches the records of other objects.

Definition 6.32 (Summing out the ids)**.** *Let c be a finitely supported map on $\mathbb{I} \times$ (nodes) of one root box, with values in a commutative monoid $(\mathbb{M}, +, 0)$, whose support meets each id in an*

antichain. Summing out the ids is the pushforward (Definition 6.17) along $\pi(i,l) = l$:

$$(\sigma c)(l) = \sum_{i \in \mathbb{I}} c(i,l).$$

When all records lie at one depth D, σc is a sparse MRP at that depth, the summed field *of c. With records at different depths, σc is a count map on labels whose support need not be an antichain, and it is not a field.*

For example, a record of object a at $\rho\mathsf{L}$ and one of object b at $\rho\mathsf{LL}$ sum out to $\{\rho\mathsf{L}{:}a, \rho\mathsf{LL}{:}b\}$, which is not an antichain, while the pointwise sum of the two represented functions is $a + b$ on $\rho\mathsf{LL}$ and a on $\rho\mathsf{LR}$. That pointwise sum, for intensive values, is the collation of the objects (Definition 6.29) summed over its keys; for extensive values such as counts, the records are first brought to one depth by coarsening (Proposition 6.33(iii)).

Proposition 6.33 (Laws of summing out)**.** *Let c, c' be as in Definition 6.32, with values in $\mathbb{M}$.*

- *(i) $\sigma(c + c') = \sigma c + \sigma c'$ and $\sigma 0 = 0$.*
- *(ii) $\sum_l (\sigma c)(l) = \sum_{(i,l)} c(i,l)$.*
- *(iii) Summing out commutes with coarsening:* $\mathrm{coarsen}_j \circ \sigma = \sigma \circ \mathrm{coarsen}_j$*, where on labelled nodes coarsening acts on the node, $(i,l) \mapsto (i, l|_j)$.*
- *(iv) For disjoint sets of ids $\mathbb{I}_1, \mathbb{I}_2$, $\sigma(c|_{\mathbb{I}_1 \cup \mathbb{I}_2}) = \sigma(c|_{\mathbb{I}_1}) + \sigma(c|_{\mathbb{I}_2})$. So, with all records at one depth, adding an object adds its own pushforward to the summed field; when $\mathbb{M}$ is a group, as $\mathbb{Z}$ or $\mathbb{R}$, removing an object subtracts it.*
- *(v) On Spark, σ is a map $(i,l,v) \mapsto (l,v)$ followed by a reduction by key in $\mathbb{M}$, and it is partition-invariant.*

Proof. σ is the pushforward along π, so (i) and (ii) are Proposition 6.18(i). For (iii), $\pi \circ (\mathrm{id} \times (\cdot)|_j) = (\cdot)|_j \circ \pi$, and Proposition 6.18(ii) applies to both composites. (iv) is (i) applied to $c|_{\mathbb{I}_1 \cup \mathbb{I}_2} = c|_{\mathbb{I}_1} + c|_{\mathbb{I}_2}$. (v) is the reduction row of Table 6.1. □

Marks. Proved here. [LEAN: MRPLift sumOutIds_add, total_sumOutIds, coarsen_sumOutIds — built at `c06220d95`; (iv) Fold prop_sumOut_iv, prop_sumOut_iv_add, prop_sumOut_iv_remove — built at `b6ea25813`] [SIM: a dynamic forest of random trajectories with arrivals and departures; the incrementally updated field against recomputation from the ids present]

Occupancy and counts. The number of distinct objects meeting a cell is not, in general, a coarsening of the number meeting its subcells. Let $\mathrm{occ}(i,l) = 1$ when object i has an observation in the finest cell l, of depth D, and 0 otherwise. Write $\mathrm{coarsen}_j^{\vee}$ and $\mathrm{coarsen}_j^{+}$ for the pushforward to depth j (Definition 6.17) in the monoids $(\{0,1\}, \vee, 0)$ and $(\mathbb{Z}_{\geq 0}, +, 0)$, acting on the node of each record.

- *Distinct objects.* The number of distinct objects meeting a cell u of depth j is $\sigma(\mathrm{coarsen}_j^{\vee}\,\mathrm{occ})(u)$: first coarsen each object's occupancy with $\vee$, then sum out the ids.
- *Incidences.* $\mathrm{coarsen}_j^{+}(\sigma\,\mathrm{occ})(u)$ counts the pairs (object, finest cell) below u.

The two agree when no object meets two finest cells below one cell. Proposition 6.33(iii) is a statement about one monoid. Here two monoids, $\vee$ and $+$, are composed, and their order matters: occupancy is coarsened with $\vee$ before the ids are summed out.

Example 6.34 (Two flights)**.** Take the root box $[0,8]$ at depth 3. The depth-3 cells are the unit intervals $[k, k+1)$, the last one being $[7,8]$, and the cell of k has heap code $8 + k$. Flight a is observed at 1.5, 2.2 and 2.7; flight b at 2.5 and 3.4.

- *Occupancy.* Reducing by (i,l) with $\vee$ gives the records $(a,9)$, $(a,10)$, $(b,10)$, $(b,11)$, each with occupancy 1.

- *Traffic.* Summing out the ids gives the number of flights per unit cell, $\{9{:}1, 10{:}2, 11{:}1\}$.
- *Distinct flights at depth 2.* Coarsening each occupancy with $\vee$ gives $a : \{4, 5\}$ and $b : \{5\}$, and summing out gives $\{4{:}1, 5{:}2\}$.
- *Incidences at depth 2.* Coarsening the traffic with $+$ gives $\{4{:}1, 5{:}3\}$, which counts flight b twice in the cell $[2, 4)$.
- *Departure.* When flight a lands, subtracting its occupancy leaves $\{9{:}0, 10{:}1, 11{:}1\}$. Dropping the zero at cell 9 is what ReuniteEmptyNodes does (Remark 5.6). MinimiseLeaves goes further, reading the traffic as an intensive field, the number of flights present at each point: the sibling cells 10 and 11 carry the same value, and it merges them into $\{5{:}1\}$. The merged value is then the number of distinct flights in $[2, 4)$, which is 1, not the incidence count 2.

Root-labelled pavings. When all ids share one root box, the forest is a *root-labelled paving*: the label at the root says which object the tree describes. The objects can be flights, days, animals, phones or sensors. Two descriptions of it are equivalent.

- *The collation.* Take the union of the objects' trees. Each node u of the union then carries a vector $(c(i, u))_{i \in \mathbb{I}}$, with 0 where object i has no record. This is the collation of Definition 6.29 with the ids as keys, and MRS's collator, whose node summary is a vector with one entry per histogram added (`CollatorSPVnode`). Definition 6.29 is stated for densities, which are intensive, and copies a leaf's value down; for counts see Remark 6.36. In the air-traffic programs, entry i is flight i, and the collator's accumulation, the sum of a node's entries, is σc (Definition 6.32).
- *One paving of a product.* For ids written with b bits, root-labelled pavings over $\mathbf{x}_\rho$ are mapped pavings of $\{0, 1\}^b \times \mathbf{x}_\rho$ whose first b cuts split the id bits and whose later cuts bisect the first widest side. A node at depth $k < b$ is a block of ids sharing a prefix. Summing out a block is coarsening to depth k, and summing out all ids is coarsening to depth 0 in the id bits. Under this reading, Proposition 6.33(iii) is the commutation of two coarsenings, one in the id bits and one in the box.

The second form shows that nothing beyond Chapter 6 is needed for many objects on one root box. The operations are the following.

- *Per object:* every operation of Chapter 6, applied id by id.
- *Marginals over ids:* any subset of ids (a weather class, an airline, the flights of a time window) is summed out by σ restricted to it, and disjoint subsets add (Proposition 6.33(iv)).
- *Arrival and departure:* adding or deleting the records of one id (the dynamic forest).
- *Relabelling:* a pushforward along a map of ids. When the map is a bijection that may differ from cell to cell, it is a *swap* (Definition 6.37).

Proposition 6.35 (Occupancy trees)**.** *Let X be a finite set of points of $\mathbf{x}_\rho$, and let $\lambda^* = 2^{-D}\mathrm{vol}(\mathbf{x}_\rho)$. Build an SRP by inserting the points one at a time, where a leaf is split whenever a point reaches it and its children would have volume at least λ^* [TKS12, Algorithms 1 and 2]. Then:*

- *(i) the internal nodes of the result are exactly the strict ancestors of the depth-D cells that X occupies;*
- *(ii) the result does not depend on the order in which the points are inserted;*
- *(iii) the tree is determined by the set of occupied depth-D labels; the leaf values (counts, or occupancy in $\{0, 1\}$) are a sparse MRP at depth D.*

Proof. A point inserted into the tree descends from the root. At each leaf it reaches above depth D, that leaf is split, so the point descends until it reaches its depth-D cell. Every strict ancestor

of an occupied depth-D cell is therefore split.

Conversely, a node is split only when a point reaches it while it is a leaf, and that point then lies in a depth-D cell below the node. So every internal node is a strict ancestor of an occupied cell. This proves (i).

The set in (i) does not depend on the order of insertion, which gives (ii). The leaves of the tree are the occupied depth-D cells together with the siblings of their ancestors. Those siblings hold no point: a point in one of them would have split it. So the only non-zero leaves are at depth D, which gives (iii). □

Marks. Proved here. [LEAN: to write, in the heap encoding of Chapter 4: the ancestor closure of a set of depth-D codes] [SIM: random point sets, insertion in random orders, tree equality against the closure of the occupied labels]

So the trajectory SRP of [TKS12] is the sparse MRP of occupied depth-D labels, with its tree as an index. The record (i, ℓ) for flight i and depth-D label ℓ is that SRP without the tree. Because every flight is kept at the one depth D, the invariant of §7.6 holds (every leaf shallower than D holds 0), and the 2012 arithmetic is label-wise arithmetic at a common depth.

Remark 6.36 (The collator and counts)**.** MRS's collator meets a leaf of one operand against an internal node of the other by splitting the leaf and *copying* its vector to both children (`CollatorSPVnode::nodeExpand`, in the copy of MRS 1.0 on which the air-traffic programs were built). Copying is right for intensive values, such as heights, and wrong for extensive ones, such as counts. The air-traffic programs store counts: the summary of a node is its count of points (`whatSum = 2`).

The collator's `nodesReunite` merges two sibling leaves only when the sums of both vectors are 0. With non-negative counts this means both vectors are zero, so it never undoes a copy. Under the invariant of §7.6 every split leaf holds zeros, so copying copies zeros, and the counts are exact.

Without that invariant they are not. Take object a at depth 1, with count 1 at $\rho\mathsf{L}$, and object b at depth 2, with count 1 at $\rho\mathsf{LL}$. The collator then gives these leaves:

- $\rho\mathsf{LL}$ with $(1, 1)$;
- $\rho\mathsf{LR}$ with $(1, 0)$;
- $\rho\mathsf{R}$ with $(0, 0)$.

The leaf counts sum to 3, but the root's count is 2: object a's count was copied to both halves of $\rho\mathsf{L}$.

This is the combination of "different levels of precision" that the 2012 paper names as an aim [TKS12, Abstract, §IV.B]. For counts it needs the coarsening of Proposition 6.33(iii) to a common depth. For heights, copying is right.

Signed vectors are a separate case. The dynamic-airspace program subtracts one collation from another and then reunites. There a vector can sum to 0 without being zero, such as $(1, -1)$. Such a merge keeps the sum over the ids but not the entries. To keep each id's values, a pair may be merged only when every entry is zero, as Definition 6.31's per-id records require.

A second difference is between the paper and its programs. The paper's leaf values are occupancies in $\{0, 1\}$ [TKS12, §I], but the programs add counts of radar returns, so a slow or circling aircraft weighs more in the aggregate. The paper's statistic is $\sigma\,\mathrm{occ}$, which the programs approximate by σ of the counts.

Swaps. A swap exchanges the futures of objects that are in the same cell at the same time [SMT18]. For it to be a relabelling of a root-labelled paving, each id must be in exactly one

cell at each moment a swap can happen. Dahne's rule gives exactly that: an id is grouped by its last datapoint in the time interval, and the swap happens at the interval's end [Dah19, Def. 4.1, Alg. 3]. The definition below is that rule, stated for any paving (regular or not, of any root box and any set of ids). It is the rule of this monograph, since the other grouping rules lose transitions (Remark 6.40).

Definition 6.37 (Swap)**.** *Fix a partition $\mathcal{P}$ of $\mathbf{x}_\rho$ into cells (for example the depth-D cells of a regular paving) and times $t_1 < t_2 < \cdots$, which cut time into intervals $I_0 = (-\infty, t_1]$, $I_k = (t_k, t_{k+1}]$.*

- *The trajectory of an id i is a finite set of time-stamped points of $\mathbf{x}_\rho$, no two at the same time [Dah19, §2].*
- *Its segment $g_k(i)$ is its points in I_k.*
- *Its state $S_k(i)$ at t_k is the cell of its last point in I_{k-1}, the interval that ends at t_k. When $g_{k-1}(i)$ is empty, including at $k = 0$, the state is the symbol $\perp$, "no state".*
- *A swap at t_k is a family of bijections $\beta_{k,C}$, one for each cell C, each a bijection of the set of ids whose state at t_k is C. An id in state $\perp$ is never swapped.*
- *Applying it gives id i, with $S_k(i) = C$, the future of $\beta_{k,C}(i)$: its new segments g_m for $m \geq k$ are those of $\beta_{k,C}(i)$. Its past is unchanged.*
- *A sanitization applies swaps at $t_1, t_2, \ldots$ in turn.*

This is Dahne's grouping: an object takes part in the swap at the end of an interval only if it has a datapoint in that interval, and it is grouped by the last one [Dah19, Def. 4.1, Alg. 3]. A state read from an earlier interval would let an object that has been silent swap. The statistics that key a transition by its interval as well as its cell would then change (A1_0-0-1's counterexample, §7.8).

A swap at t_k is a bijection of ids on the futures after t_k, chosen cell by cell. So it is a relabelling in the sense of Definition 6.17, applied to the records after t_k, and Proposition 6.18(i) applies: every sum over ids of a quantity read from the futures is unchanged.

The swap keeps more than that. It pairs a past with a future only when both end in the same state. This is the Markov structure, and it gives the following theorem.

Theorem 6.38 (What swaps preserve)**.** *Let a sanitization (Definition 6.37) be applied to a finite set of trajectories, on a partition $\mathcal{P}$ and times $t_1 < t_2 < \cdots$. Call a statistic* one-step *if it has the form*

$$F = \sum_{k \geq 0} \sum_{i} f_k\big(S_k(i),\, g_k(i)\big),$$

where f_k is any function on $(\mathcal{P} \cup \{\perp\}) \times (\text{segments})$ into a commutative monoid with $f_k(\perp, \emptyset) = 0$. The sum is then finite, because each id has a state or a point in only finitely many intervals, and a statistic may count states. Since the state is read from the interval I_{k-1}, a one-step statistic may key its terms by the interval as well as by the cell.

- *(i) Every one-step statistic is the same before and after sanitization.*
- *(ii) The following are one-step, so they are preserved:*
  - *the count of points per (cell, interval);*
  - *the number of distinct ids with a point in a cell during one interval I_k;*
  - *the transitions between consecutive states, $S_k \to S_{k+1}$, of the ids with a point in I_k;*
  - *the transitions between consecutive points, keyed by their cells and by the intervals they fall in, including the pair that straddles t_k: the last point in I_{k-1} and the first point in I_k.*

- *(iii) Coarser in space. Let $\mathcal{Q}$ be a partition each of whose cells is a union of cells of $\mathcal{P}$ (for a regular paving, any depth $j \leq D$), and let π map a cell of $\mathcal{P}$ to the cell of $\mathcal{Q}$ that contains it. Every statistic in (ii), computed on $\mathcal{Q}$, is preserved.*
- *(iv) Coarser in time. A sum over a block of consecutive intervals of a one-step statistic is one-step, so it is preserved. In particular, counts and transitions computed at the fine step and summed over coarse intervals are preserved.*
- *(v) Not preserved in general, even with $|\mathcal{P}|$ and the number of ids as small as two:*
  - *the transitions of the chain observed at a coarser step, $S_k \to S_{k+2}$;*
  - *origin–destination pairs;*
  - *the number of distinct ids with a point in a cell over a block of two or more intervals;*
  - *any statistic of one id's whole trajectory.*

Proof. (i) Consider one swap, at t_k. Let C be a cell and i an id with $S_k(i) = C$. After the swap, id i has: - its own past, so its own segments $g_m(i)$ for $m < k$ and its own state $S_k(i) = C$; - the future of $j = \beta_{k,C}(i)$, whose state at t_k is also C.

Its state at each later time t_m, $m > k$, is read from its points in I_{m-1}, which lie in the future taken from j, so it is $S_m(j)$, $\perp$ included.

Now look at each term of F: - For $m < k$, the term $f_m(S_m(i), g_m(i))$ is unchanged. - For $m > k$, the pair (S_m, g_m) of id i after the swap is $(S_m(j), g_m(j))$. Since $\beta_{k,C}$ is a bijection of the ids in state C, the multiset of these pairs over i is unchanged. - For $m = k$, the new pair of i is $(C, g_k(j)) = (S_k(j), g_k(j))$. Again only the multiset is permuted.

So F is unchanged by one swap, and by induction by any sanitization. The sum is finite because there are finitely many trajectories, each with finitely many points.

(ii) Each statistic is a sum over ids of a function of (S_k, g_k):

- the count of points of g_k in C;
- the indicator that g_k meets C;
- the pair (S_k, S_{k+1}), where S_{k+1} is the cell of the last point of g_k (the term is 0 when g_k is empty or $S_k = \perp$);
- the cells and intervals of consecutive points within g_k, and the straddling pair formed by the last point of g_{k-1} (its cell is S_k, its interval is I_{k-1}) and the first point of g_k, counted when $S_k \neq \perp$. A pair that straddles one or more empty intervals is not a one-step statistic, but it is preserved as well (Remark 6.39).

(iii) Apply (ii) to the functions $f_k \circ (\pi \times \pi_*)$, where π_* maps a segment's points to $\mathcal{Q}$. Those statistics are still of the one-step form on $\mathcal{P}$.

(iv) A sum of one-step statistics is one-step.

(v) Each case has two ids x, y and two cells $A \neq B$.

- Coarser step and origin–destination pairs. Take states $S_1(x) = A$, $S_1(y) = B$, then $S_2(x) = S_2(y) = A$, then $S_3(x) = A$, $S_3(y) = B$, and swap at t_2, where both are in A. Before the swap, the pairs (S_1, S_3) are (A, A) and (B, B); after it, x has the future of y and y that of x, so they are (A, B) and (B, A). The same two trajectories, cut after t_3, show that the origin–destination pairs change.
- Distinct ids over two intervals. Let x have a point in B and then a last point in A during I_1, and y a point in A during I_1, so both are in state A at t_2. During I_2, let y have a point in B and x a point in A. Before the swap, the cell B over $I_1 \cup I_2$ meets two ids, x in I_1 and y in I_2. After a swap at t_2, x has its own past and y's future and meets B in both intervals, while y never meets B. So it meets one id.

In each case the statistic changes. □

Marks. Proved here. [LEAN: ProveDog.L1MDE.Swap oneStep_swap (the core: after a swap at t_k an id keeps its pair (S_m, g_m) for $m < k$ and takes that of $\beta_{k,C}(i)$ for $m \geq k$, so every sum over the ids is reindexed by a bijection), oneStep_sanitized, thm_swap_i ((i), interval by interval, and F as a finsum, which is F when finitely many terms are non-zero, as under the theorem's hypotheses; $f_k(\perp, \emptyset) = 0$ is not needed), thm_swap_ii ((ii)), thm_swap_iii, thm_swap_iii_-stats, thm_swap_countMap ((iii), any coarser partition, and for a regular paving the ids' count maps of I_k, each coarsened to depth j and summed, for every j), thm_swap_iv ((iv), any finite set of intervals), thm_swap_v_step, thm_swap_v_block ((v), the proof's examples with two ids and two cells: $S_1 \to S_3$, origin–destination pairs (also the last case: an origin–destination pair is a statistic of one id's whole trajectory), distinct ids over $I_1 \cup I_2$), with glue_spec, isSwap_glue, glue_fibrePerm (Definition 6.37's family of bijections is a permutation of the ids that keeps the state at t_k and fixes the ids in state $\perp$) — built at `9e6d7a947`; a trajectory is the sequence of its segments and a sanitization any finite sequence of swaps, in any order; Remark 6.39 is not checked.] [SIM: done for (ii) and (iii) at depths $j \leq D$, and for the coarser time step in (v): transitions kept in 200 of 200 runs with this rule and lost in 200 of 200 at a coarser step (swapmob_coarsen_check.py, swapmob_coarsen_check2.py, shared-reviews); to rerun on the T-drive data at scale]

Remark 6.39 (Pairs across empty intervals)**.** The pairs of consecutive points of one id are all preserved, keyed by their cells and their intervals, including a pair whose two points have empty intervals between them (A1_0-0-1's argument, §7.8).

Let i have its last point before a gap in I_{k-1}, in cell C, and its next point in I_m with $m > k$. Then three things hold. - At the swap at t_k, every swapped id in state C has a pair whose first end is (C, I_{k-1}). The swap permutes the futures, and with them the second ends. - An earlier swap, at t_l with $l < k$, exchanges whole futures. It permutes, over the ids, the pairs formed by the state at t_k and the first point after t_k. - A later swap, at t_l with $k < l \leq m$, does not touch the future that holds the empty intervals. That future has no point in I_{l-1}, so its state at t_l is $\perp$.

So the multiset of these pairs is the same after sanitization.

Remark 6.40 (Other grouping rules)**.** Theorem 6.38 uses only one property: an id is swapped only with an id in the same state. A rule that groups ids by anything other than their state at the swap time breaks this.

Two such rules exist: - grouping by the *first* point in the interval, as `mob-spark` does [lam]; - swapping at any shared cell within the interval, as one reading of the paper's Algorithm 1 allows [SMT+20].

Both fail. Take x at A and then B in I_k, and y at A only, so $S_{k+1}(x) = B$ and $S_{k+1}(y) = A$. Let the next points be x at C and y at D. Grouping by the first point puts both in A and swaps their futures. The transitions $B \to C$ and $A \to D$ become $B \to D$ and $A \to C$.

This holds at the swap depth itself, before any coarsening. The simulation above found the transitions lost in 200 of 200 runs under each of these rules. So the thesis's rule is the one consistent with root-labelled pavings in general, and it is the rule of this monograph.

Swaps that carry a status. The proof of Theorem 6.38 uses only two facts. Ids are swapped only when their states agree, and an id's later states are read from the future it takes. So the state may record more than the cell: anything a device knows at t_k and carries forward with its own future, such as its infection status. Swapping then exchanges identities only among devices that agree on both.

Corollary 6.41 (Swaps and a cell-structured epidemic)**.** *Fix the data of Definition 6.37 on a window of boundaries $t_1 < \cdots < t_K$, with every id having a point in every interval of the window. Assume each id carries a compartment $c_k(i) \in \{\mathrm{S}, \mathrm{I}, \mathrm{R}\}$ at each t_k, recorded with its last point in I_{k-1}. Let the augmented state be $\hat{S}_k(i) = (S_k(i), c_k(i))$, and let swaps exchange the futures of ids only within equal augmented states.*

Write $n_c(C,k)$ for the number of ids in state (C,c) at t_k. Write $a(C,k)$ for the number of ids in (C,S) at t_k that are in I at t_{k+1}, and $b(C,k)$ for the number in (C,I) at t_k that are in R at t_{k+1}. Then:

- *(i) every $n_{\mathrm{S}}(C,k)$, $n_{\mathrm{I}}(C,k)$, $n_{\mathrm{R}}(C,k)$, $a(C,k)$ and $b(C,k)$ is the same before and after sanitization;*
- *(ii) so is each of them computed on any partition $\mathcal{Q}$ coarser than $\mathcal{P}$ (each cell of $\mathcal{Q}$ a union of cells of $\mathcal{P}$);*
- *(iii) after sanitization, every id's sequence of compartments is still non-decreasing in the order $\mathrm{S} \le \mathrm{I} \le \mathrm{R}$;*
- *(iv) consider the chain binomial run cell by cell on $\mathcal{P}$, or on any coarser $\mathcal{Q}$: given the states at t_k, each susceptible in cell C is infected by t_{k+1} with probability $1-(1-p)^{n_{\mathrm{I}}(C,k)}$, and each infectious id is removed with probability γ, all independently. Its likelihood for (p,γ) is a function of the statistics in (i), on $\mathcal{P}$ or on $\mathcal{Q}$, so it is the same before and after sanitization.*

Proof. The proof of Theorem 6.38(i) applies with $\hat{S}_k$ in place of S_k. Swapped ids share $\hat{S}_k$, and after a swap at t_k an id's compartments at later times are those of the future it takes. So every sum $\sum_k \sum_i f_k(\hat{S}_k(i), g_k(i))$ is preserved.

(i) Each statistic is such a sum:

- $n_c(C,k)$ counts the ids with $\hat{S}_k = (C,c)$;
- $a(C,k)$ counts those with $\hat{S}_k = (C,\mathrm{S})$ whose compartment recorded with the last point of g_k is I;
- $b(C,k)$ is the same with (C,I) and R.

(ii) On $\mathcal{Q}$ each statistic is the sum of the same statistic over the cells of $\mathcal{P}$ that make up a cell of $\mathcal{Q}$.

(iii) An id swapped at t_k keeps its own compartments up to t_k and takes those of a future whose compartment at t_k equals its own. Each piece is non-decreasing, and they agree at t_k.

(iv) The likelihood is

$$\prod_{k<K} \prod_{C} \binom{n_{\mathrm{S}}(C,k)}{a(C,k)} \pi_{C,k}^{a(C,k)} (1-\pi_{C,k})^{n_{\mathrm{S}}(C,k)-a(C,k)} \binom{n_{\mathrm{I}}(C,k)}{b(C,k)} \gamma^{b(C,k)} (1-\gamma)^{n_{\mathrm{I}}(C,k)-b(C,k)},$$

with $\pi_{C,k} = 1-(1-p)^{n_{\mathrm{I}}(C,k)}$. On $\mathcal{Q}$ the same formula holds with the cells of $\mathcal{Q}$. □

Marks. Proved here. [LEAN: ProveDog.L1MDE.Swap, as Theorem 6.38 for the product state (the cell map to (cell, compartment)): cor_swapSIR_i ((i), and any move $c \to c'$), cor_swapSIR_ii ((ii)), cor_swapSIR_iii ((iii): a point in every interval of the window and $\mathrm{S} \le \mathrm{I} \le \mathrm{R}$ at consecutive times are kept; the window is used only here), cor_swapSIR_iv ((iv), on likelihood_congr: the likelihood written in the proof, over any finite set of steps, its product over the cells a finprod, on $\mathcal{P}$ and on $\mathcal{Q}$) — built at `9e6d7a947`; that this formula is the chain binomial's likelihood is not checked.] [SIM: an SIR chain binomial on random trajectories over a paving, sanitized with status-respecting swaps; the counts and the maximum-likelihood estimates of (p,γ) before and after, at depths $j \le D$]

A swap does not preserve what follows one id across several intervals (Theorem 6.38(v)). In particular, it does not preserve which ids one id meets at different times.

Swaps without a curator. Two devices in contact (Definition 6.42) can swap pseudonym tokens themselves, for example over a Bluetooth-mesh hop, with no trusted curator.

Theorem 6.38 then holds on the devices exactly when each one swaps only with a peer in its own cell at the same boundary, and with the same status in the setting of Corollary 6.41. A device can check this rule from its own position. Radio range alone is not enough: being within range is not transitive, and two devices in range of each other may lie in different cells.

For the lemma of §7.8, Theorem 6.38(iii) is the spatial half with the swap rule made explicit, and (iv) and (v) settle the temporal half: - statistics summed over coarser time intervals are preserved; - a chain observed at a coarser step is not.

A decision that needs the chain at step τ' needs swaps made only at multiples of τ'.

Contacts and transmissions. In this part every id keeps its identity. What passes between ids is a token, a message or an infection, and who passes it to whom is the event of interest. Swaps and transmissions read the same root-labelled paving in two ways, for different decision problems.

Each individual, with a device or without one, is in space-time a $\{0,1\}$-valued mapped paving: its occupied leaves (Proposition 6.35). Contacts are a neighbourhood relation between these occupied leaves. A neighbourhood is found by a search around each individual's leaves, a problem separate from the arithmetic of the trees (Remark 6.43).

Definition 6.42 (Contacts)**.** *Fix a root-labelled paving of ids at depth D, and the times $t_1 < t_2 < \cdots$ of Definition 6.37. Let $x_k(i)$ be the occupied leaf of id i at t_k: the depth-D cell of its last point in I_{k-1}, or $\perp$. A* neighbourhood rule $\mathcal{N}$ *assigns to each id i with $x_k(i) \neq \perp$ a set $\mathcal{N}_k(i)$ of other ids whose leaves are not $\perp$. Two examples:*

- *the* ball rule*: the ids whose leaves lie within distance r of $x_k(i)$, for a metric on the cells (for example between their centres);*
- *the* nearest-neighbour rule*: the m ids whose leaves are nearest to $x_k(i)$, with ties broken by label.*

Ids i and i' are in contact at t_k *when $i' \in \mathcal{N}_k(i)$. The* contact graph $G_{\mathcal{N}}(t_k)$ *has the ids as vertices and these pairs as directed edges. It is undirected when $\mathcal{N}$ is symmetric, as the ball rule is and the nearest-neighbour rule in general is not.*

The cell rule *at depth $j \leq D$ is the special case $\mathcal{N}_k(i) = \{i' \neq i : x_k(i')$ and $x_k(i)$ have one ancestor at depth $j\}$.*

Under the cell rule, $G_j(t_k)$ is a disjoint union of cliques, one for each occupied depth-j cell. Its number of edges is $\sum_C \binom{n(C,k)}{2}$, a function of the counts per cell, so it is computed from the summed field of the ids (Definition 6.32). Coarser cells join cliques, so $G_D(t_k)$ is a subgraph of $G_j(t_k)$ for $j \leq D$.

The cell rule is the one Theorem 6.38 needs, because a swap needs a partition by state. For transmissions any neighbourhood rule serves. The cell rule has a defect for contacts. Two ids are in contact under it only at depths no deeper than that of the common ancestor of their leaves. For two ids close together on either side of the root's first cut, that ancestor is the root itself. The ball rule has no such defect. Its computation is Proposition 6.44(iv), and Lemma 6.45 makes even the cell rule serve across cuts, by random shifts of the root box.

Over k, any of these contact graphs is a temporal graph. Its links are live at the times t_k, and its journeys run over edges at non-decreasing times, with an id holding a token between contacts (store-carry-forward). This is the communications substrate of the IAS paper, whose protocols include a Bluetooth-mesh hop, here with its links read off positions.

Remark 6.43 (Two separable computations)**.** Studying co-trajectories by tree arithmetic (summing out ids, Definition 6.32; coarsening, Proposition 6.33(iii); the union, the pointwise operation and the pushforward of §§ 6.2, 6.5 and 6.7) and finding neighbourhoods are separate computations.

- *The arithmetic* works on the keys (i, ℓ) through maps and reductions by key. It never asks which ids are near which.
- *The search* asks, for each id's occupied leaf, which other ids' leaves are near it. It needs an index over the leaves: the sorted labels themselves, since a ball around a cell meets a bounded set of cells at its depth (Proposition 6.44), or a k-d tree over the cell centres.

The two are separate because the tree bounds distances from above only. Two points in one cell of depth j are within that cell's diameter of each other (Lemma 4.23(iii)), so a deep common ancestor means nearness in space. The converse fails: two points at any positive distance can lie on either side of the root's first cut, with no common ancestor below the root, at every depth. So the depth of $u \wedge v$ is not a distance, the labels alone are no metric index, and a search needs more than the order of the labels: the neighbour labels of Proposition 6.44, the shifted pavings of Lemma 6.45, or an index over the centres.

A decision problem may need one or both. Traffic counts per cell need only the arithmetic. Who could have passed a token to whom needs only the search. The incidence of an infection per cell, given who met whom, needs both. The search's output, a contact graph, then feeds the arithmetic as a further labelled map: the edges keyed by cell and time.

Finding neighbourhoods on the labels. The search has an arithmetic of its own, on the labels. At one depth the cells are a grid, and a label is a grid coordinate written in interleaved bits.

Proposition 6.44 (Neighbour labels)**.** *Let $\mathbf{x}_\rho$ have widths $w_1, \dots, w_d$ and the regular scheme (Definition 4.22), or any midpoint scheme whose split axis depends on the depth alone (Remark 4.30). Let $\iota_1, \iota_2, \dots$ be its split axes, ι_t the axis along which every node of depth $t-1$ is cut; for $j \geq 0$ let $n_i(j) = |\{t \leq j : \iota_t = i\}|$ and $h_{j,i} = w_i 2^{-n_i(j)}$. - (i) Grid. The cells of depth j are the 2^j boxes $\prod_{i=1}^d [\underline{x}_i + c_i h_{j,i},\ \underline{x}_i + (c_i+1)h_{j,i})$, $0 \leq c_i < 2^{n_i(j)}$, closed at the upper end of $\mathbf{x}_\rho$, one for each $c \in \prod_i \{0, \dots, 2^{n_i(j)} - 1\}$. - (ii) Coordinates from the label. Let $u = \rho b_1 \cdots b_j$ with $b_t \in \{\mathsf{L}, \mathsf{R}\}$ read as the bits $0, 1$, so that b_t is bit $j - t$ of $\mathrm{lab}(u) - 2^j$ (§4.6). The cell of u is the box of (i) with $c_i(u)$ the integer whose binary digits, most significant first, are the bits b_t with $\iota_t = i$, in the order of t. The map $u \mapsto c(u)$ is a bijection from the nodes of depth j onto the grid, and its inverse interleaves the digits of $c_1, \dots, c_d$ along $\iota_1, \dots, \iota_j$. - (iii) Neighbours. The cells whose closures meet the closure of the cell of u are those of $c(u) + e$ for $e \in \{-1, 0, 1\}^d \setminus \{0\}$ with $c(u) + e$ in the grid: at most $3^d - 1$, each label computed from $\mathrm{lab}(u)$ by (ii) with $O(j)$ bit operations and no geometry. - (iv) Ball queries. Let $x \in \mathbf{x}_\rho$, let u be its cell of depth j, and let $0 < r \leq h_j = \min_i h_{j,i}$. Every point y with $\|y - x\|_\infty \leq r$, hence every y with $\|y - x\|_2 \leq r$, lies in one of the 3^d cells of $c(u) + e$, $e \in \{-1, 0, 1\}^d$. So the occupied leaves of depth $D \geq j$ (Definition 6.42) that meet the closed ball of radius r about x, those whose centres lie within r of x among them, are among the leaves below those 3^d labels, found by 3^d searches in the sorted occupied labels, each the run of Lemma 4.28(c), at most $2 \cdot 3^d(\lceil \log_2 N \rceil + 1)$ label predicates for N occupied leaves, followed by one distance per candidate. For $r > h_j$ the cells $c(u) + e$ with $|e_i| \leq \lceil r/h_{j,i} \rceil$ suffice, $\prod_i (2\lceil r/h_{j,i} \rceil + 1)$ searches.*

Proof. (i) and (ii) together, by induction on j. At $j = 0$ the one cell is $\mathbf{x}_\rho$ and $c(\rho) = 0$. Suppose the cells of depth $t-1$ are the grid of (i) with the coordinates of (ii). They all have the widths $w_i 2^{-n_i(t-1)}$, so their first widest coordinate is one axis, ι_t, and the regular scheme cuts every one of them along it at its midpoint (Definition 4.22; Lemma 4.23(ii) is this equality of widths); a scheme whose axis depends on the depth alone cuts them along its ι_t by definition. The cut halves the interval of axis ι_t: L keeps the lower half and R the upper, so c_{ι_t} gains the bit b_t as a new least significant digit and the other coordinates are unchanged; the half-open and closed ends are those of Definition 4.22. The reading of b_t in $\mathrm{lab}(u)$ is the heap encoding of §4.6. Distinct words give distinct coordinate vectors, there are $2^j = \prod_i 2^{n_i(j)}$ of each, so the map is a bijection, and the inverse reads the digits back along the axes. (iii) Two grid boxes have closures that meet exactly when their coordinate vectors differ by at most 1 on every axis, and $\{-1,0,1\}^d \setminus \{0\}$ has $3^d - 1$ elements; the label of $c(u)+e$ is (ii): extract the d coordinates, add e, interleave. (iv) $|y_i - x_i| \le r \le h_{j,i}$ puts y_i in the interval of $c_i(u)$ or in one of its two neighbours, on every axis, and $\|y-x\|_2 \le r$ gives $\|y-x\|_\infty \le r$. A leaf of depth at least j that meets the ball holds a point y within r of x; y lies in one of the 3^d cells, and the leaf lies below that cell. Depth-j labels of an antichain are in the integer order of their heap codes (Proposition 4.27(v)), and the occupied leaves below a depth-j label are a run of the sorted list that Lemma 4.28(c) finds with at most $2(\lceil \log_2 N \rceil + 1)$ predicates. The general radius is the same count with $\lceil r/h_{j,i} \rceil$ neighbours per side. □

Marks. Proved here; new. The grid of (i) is the equal-widths layout of Lemma 4.23(ii), read coordinate by coordinate. [LEAN: to write; (ii) is a statement on words and the heap code, beside `IasCert.L1MDE.Labels`.] [SIM: random points at depths $j \le 12$ and $d \le 4$: the 3^d lookups against a brute-force ball query, every true neighbour among the candidates and no candidate outside the 3^d cells; to run, L1MDE-PROOF-SIMULATION.]

The code at `c0a1837` has the pieces: `NodeLabel` holds the heap code, `truncate` and `join` give ancestors (§4.6), and `quickDescendBox` (`SpatialTree.scala` lines 149–179) descends from a point to its label along the split sequence, which is the inverse of (ii) applied to the point's coordinates. It has no neighbour-label function and no search.

The cell rule misses close pairs across cuts (Definition 6.42). Shifting the root box moves every cut, and a uniform random shift turns the miss into a probability that a second shift can repair.

Lemma 6.45 (Shifted pavings separate close points rarely)**.** *In the setting of Proposition 6.44, fix a depth j and write $h_i = h_{j,i}$. Let $U = (U_1, \dots, U_d)$ have independent coordinates, U_i uniform on $[0, h_i)$. The* shifted paving *is the paving of the shifted root box $\mathbf{x}_\rho^U = \prod_i [\underline{x}_i - U_i,\ \underline{x}_i - U_i + 2w_i)$, doubled along every axis, by the scheme whose split axes are $1, \dots, d, \iota_1, \dots, \iota_j$ (one cut per axis, then the axes of $\mathbf{x}_\rho$: a scheme whose axis depends on the depth alone, so the cells, labels and left-right order of Definition 4.22 are those of this scheme on $\mathbf{x}_\rho^U$ and Proposition 6.44 applies to them). The scheme is named rather than left to the first-widest rule, which on $\mathbf{x}_\rho^U$ would in general give another grid: for $d = 2$, $w = (4,1)$ and $j = 1$ the cells of (i) have widths $(2,1)$, while three first-widest cuts of the box of widths $(8,2)$ give widths $(1,2)$. Its cells of depth $d+j$ are the* depth-j shifted cells*: the boxes $\prod_i [\underline{x}_i - U_i + c_i h_i,\ \underline{x}_i - U_i + (c_i+1)h_i)$, $0 \le c_i < 2^{n_i(j)+1}$, the grid of (i) translated by $-U$ and continued along each axis to twice its length. Since $U_i < h_i \le w_i$, $\mathbf{x}_\rho \subseteq \mathbf{x}_\rho^U$, and every $x \in \mathbf{x}_\rho$ lies in one depth-j shifted cell for every U. Let $x, y \in \mathbf{x}_\rho$ and $\delta_i = |x_i - y_i|$. - (i) x and y lie in one depth-j shifted cell with probability $\prod_{i=1}^d (1 - \delta_i/h_i)^+$. So they are separated with probability at most $\sum_i \min(1, \delta_i/h_i) \le d\,\|x-y\|_\infty / h_j$, with $h_j = \min_i h_i$, and with probability 1 when $\delta_i \ge h_i$ for some i. - (ii) Under L independent shifts, x and y share a depth-j shifted cell in at least one shifted paving with probability at least $1 - (d\,\|x-y\|_\infty / h_j)^L$, a bound that says something for $\|x-y\|_\infty < h_j/d$, and in none when $\delta_i \ge h_i$ for some i. - (iii) So the depth-j shifted cell under a random shift is a locality-sensitive hash for the ℓ_∞ metric, and through $\|\cdot\|_\infty \le \|\cdot\|_2 \le \sqrt{d}\,\|\cdot\|_\infty$ for the Euclidean one: pairs within $\delta < h_j/d$ collide with*

probability at least $1 - d\delta/h_j$*, and pairs at* ℓ_∞ *distance* $\max_i h_i$ *or more never collide. Each shifted paving is one tree of the forest of Definition 6.31, with root box* $\mathbf{x}_\rho^U$ *and the shift as the tree's id; that tree carries a labelled sparse MRP (Definition 5.20) whose keys are the individuals, so the occupied leaves of the ids under shift* ℓ *are the records* (ℓ, i, u) *with* u *the depth-*$(d+j)$ *label of individual* i *in that tree; and the candidate neighbours of a point* x *are the ids at the* L *labels of* x*, one per shift, found by* L *lookups, each as in Proposition 6.44(iv) with* $e = 0$*.*

Proof. The first d cuts of the scheme halve each axis of $\mathbf{x}_\rho^U$ once, to the widths w_i of $\mathbf{x}_\rho$, and the next j cut along $\iota_1, \dots, \iota_j$ as in Proposition 6.44(i), so at depth $d+j$ axis i is cut into $2^{n_i(j)+1}$ intervals of width h_i with the lower end $\underline{x}_i - U_i$: the depth-j shifted cells are as stated, and $x \in \mathbf{x}_\rho$ lies in one of them since $\underline{x}_i - U_i \le x_i \le \overline{x}_i < \underline{x}_i - U_i + 2w_i$. (i) Along axis i the cuts bounding the depth-j shifted cells are at $\underline{x}_i - U_i + kh_i$, $0 \le k \le 2^{n_i(j)+1}$, which on $[\underline{x}_i, \overline{x}_i]$ is a grid of spacing h_i whose phase is uniform on $[0, h_i)$. Write $a < b$ for x_i, y_i in order. The two lie in different intervals of that grid exactly when a cut falls in $(a, b]$, since intervals are closed on the left. For $\delta_i = b - a < h_i$ there is exactly one cut in each window of length h_i, and the window $(a, b]$ holds it with probability δ_i/h_i; for $\delta_i \ge h_i$ it holds one with probability 1. The points share a depth-j cell exactly when no axis separates them, and the axes are independent, which gives the product. The bounds are $1 - \prod_i (1 - a_i) \le \sum_i a_i$ for $a_i \in [0, 1]$, and $a_i \le \|x - y\|_\infty / h_j$. (ii) The shifts are independent, and a pair with $\delta_i \ge h_i$ is separated under every shift. (iii) is (i) and (ii) read as collision probabilities; the forest reading is Definition 6.31 with $\mathbf{x}_\rho^U$ as the root box and the shift as the id of the tree, and Definition 5.20 with the individuals as the keys of that tree's values. The labels of two shifts are labels of two trees with different cells and are never merged as one tree (Remark 4.30: equal labels on different cells). □

Marks. Proved here; new. The one-axis statement is the randomly shifted grid on a line, the standard locality-sensitive hash for a line, and its product over axes is the shifted-grid hash (BIB TO ADD below). [LEAN: to write; (i) is a product of one-dimensional events.] [SIM: random pairs at ℓ_∞ distance δ in $[0, 1]^d$, $d \le 4$, $L \le 32$ shifts: the collision frequency against $\prod_i (1 - \delta_i/h_i)^+$, and the recall of the L-shift candidates against brute-force m-nearest neighbours; every point of $\mathbf{x}_\rho$, the upper faces included, hashed under every shift, with U_i near h_i as well; to run, L1MDE-PROOF-SIMULATION.]

The L shifted pavings are L root-labelled copies of one data set, L times the records with labels d bits longer, and the search per shift is one lookup. A ball rule of Definition 6.42 is then computed in two ways: exactly, by the 3^d lookups of Proposition 6.44(iv) at a depth whose cell side is at least the radius; or approximately, by the L lookups of Lemma 6.45(iii), which miss a pair at distance δ with probability at most $(d\delta/h_j)^L$ and never return a pair farther than $\max_i h_i$.

Transmission. A token held by an id at t_k may pass to the ids in contact with it at t_k, by a rule: - every contact, for a broadcast message; - each contact independently, with a probability, for an infection.

The record of who passed the token to whom is the *transmission tree* of [SW16]. It is a binary tree whose internal nodes are the transmission events, ranked by their times. Forgetting the ranks gives a Catalan binary tree (Chapter 4). The rankings of one Catalan tree are counted by its Catalan coefficient (Definition 4.6, Lemma 4.7; Example 4.10).

[SW16] grows the tree by an SI process on a fixed contact network and studies the inverse problem: what the trees say about the network. [God21] extends the process to SI, SIR, SIS and SIRS on directed networks. It adds a parameter for blocking a node's edges when it becomes infected, and it leaves time-varying networks open. On a root-labelled paving the network is $G_{\mathcal{N}}(t_k)$, which moves with the ids for any neighbourhood rule, so these processes run on a network that moves.

Two questions are open: - which law on ranked trees a given contact structure induces; - what an observed set of transmission trees says about the paving's contacts.

For the complete graph, [God21] reports SIR runs in which every ranked planar shape is equally likely.

Marks. Definition and observations, no theorem. [SIM: SI and SIR on $G_j(t_k)$ from random trajectories over a paving, at several j: the law of the ranked transmission trees, against [God21]'s complete-graph result]

Implementations: MRS keeps labelled boxes for the trans-dimensional sampler as `LabBox` and `RangedLabBox` (`SmallClasses.hpp`), refined by `MRSampler` (`MRSampler.hpp`). Its collator (`CollatorSPVnode`, with a vector summary per node) is a root-labelled paving in the collation form; the air-traffic programs are in `examples/StatsSubPav/AirTraffic` of MRS (branch gat41). SparkDensityTree at commit `c0a1837` has no labelled MRP; summing out is its Phase-2 reduction by key applied to labelled records. Swaps over Spark datasets are in `mob-spark` [lam], which groups by the first point (Remark 6.40).

6.12 Level sets, highest density regions and a centre-outward median

A nonnegative MRP is constant on its leaves, so its superlevel sets, and the highest density regions of [Hyn96], are unions of leaves, computed exactly from the leaf list. As in the rest of the monograph, R_α denotes the $(1-\alpha)$ highest density region: it holds mass at least $1-\alpha$, and it grows as α decreases.

Definition 6.46 (Highest density regions and coverage values)**.** *Let f be a nonnegative $\mathbb{R}$-MRP with leaf masses $m(u) = f(u)\,\mathrm{vol}(\mathbf{x}_u)$.*

- *For $c \geq 0$, the superlevel set $R(c) = \{x : \bar{f}(x) \geq c\}$ is a union of leaves, with mass $M(c) = \sum_{f(u)\geq c} m(u)$. Write $M_>(c) = \sum_{f(u)>c} m(u)$. The total mass is $M(0)$, assumed positive.*
- *For $0 \leq \alpha < 1$, the $(1-\alpha)$* highest density region *is $R_\alpha = R(c_\alpha)$, with $c_\alpha = \max\{f(u) : u \in \mathcal{L}(s),\ M(f(u)) \geq (1-\alpha)M(0)\}$; and $R_1 = \emptyset$.*
- *The* coverage value *of $x \in \mathbf{x}_\rho$ is $\mathrm{cov}(x) = 1 - M_>(\bar{f}(x))/M(0)$, and its* anomaly score *is $a(x) = 1 - M(\bar{f}(x))/M(0)$.*

The maximum exists, since the smallest positive leaf height h_+ has $M(h_+) = M(0)$, leaves of height 0 having no mass; so $c_\alpha \geq h_+ > 0$. For a density estimate, R_α is the highest density region of coverage $1-\alpha$; [HST12] index the same regions by their coverage. For a histogram of counts, $m(u) = \#(u)/n$. The smallest region R_α that contains a point x with $\bar{f}(x) > 0$ is $R(\bar{f}(x))$ (and no region contains a point of height 0, where $a(x) = 0$), so the anomaly score $a(x)$ is one minus the probability of the smallest highest density region containing x, the score that [STW+26] compute as an anomaly score for 10^4 samples from a two-dimensional test density, using its minimum distance estimate. The coverage value, which MRS computes, exceeds it by the mass fraction of the height class of x: $\mathrm{cov}(x) - a(x) = \sum_{f(u)=\bar{f}(x)} m(u)/M(0)$.

Proposition 6.47 (Highest density regions)**.** *Let $0 \leq \alpha < 1$.*

- *(i) $M(c_\alpha) \geq (1-\alpha)M(0)$, and $M(c) < (1-\alpha)M(0)$ for every leaf height $c > c_\alpha$. So R_α overshoots $(1-\alpha)M(0)$ by less than the mass of the leaves of height c_α.*
- *(ii) Every measurable $A \subseteq \mathbf{x}_\rho$ with $\int_A \bar{f} \geq M(c_\alpha)$ has $\mathrm{vol}(A) \geq \mathrm{vol}(R_\alpha)$.*
- *(iii) If $\alpha \leq \alpha' < 1$, then $R_{\alpha'} \subseteq R_\alpha$.*

- *(iv)* $x \in R_\alpha$ *exactly when* $\mathrm{cov}(x) > \alpha$*; this holds for* $\alpha = 1$ *too.*
- *(v) If* $M(0) = 1$ *and* p *is a density on* $\mathbf{x}_\rho$*, then* $\left| \int_{R_\alpha} p - \int_{R_\alpha} \bar{f} \right| \le \frac{1}{2} \int |p - \bar{f}|$*, so the true probability of* R_α *is at least* $1 - \alpha - \frac{1}{2} \int |p - \bar{f}|$*.*

Proof.

- *(i).* Let c' be the next leaf height above c_α, if there is one. Then $M(c') < (1-\alpha)M(0)$ by the maximality of c_α, and $M(c_\alpha) = M(c') + \sum_{f(u)=c_\alpha} m(u)$. With no higher height, $M(c') = 0$, and $M(c) = 0 < (1-\alpha)M(0)$ for every $c > c_\alpha$.
- *(ii).* Write $R = R_\alpha$; recall $c_\alpha > 0$. Then $\int_R \bar{f} - \int_A \bar{f} = \int_{R \setminus A} \bar{f} - \int_{A \setminus R} \bar{f} \le 0$. On $R \setminus A$, $\bar{f} \ge c_\alpha$, and on $A \setminus R$, $\bar{f} < c_\alpha$. So $c_\alpha \,\mathrm{vol}(R \setminus A) \le \int_{R \setminus A} \bar{f} \le \int_{A \setminus R} \bar{f} \le c_\alpha \,\mathrm{vol}(A \setminus R)$. Dividing by c_α gives $\mathrm{vol}(R) \le \mathrm{vol}(A)$.
- *(iii).* As $1 - \alpha \ge 1 - \alpha'$, the set of heights with $M \ge (1-\alpha)M(0)$ is contained in the set with $M \ge (1-\alpha')M(0)$, so $c_\alpha \le c_{\alpha'}$.
- *(iv).* If $\bar{f}(x) \ge c_\alpha$, then $M_>(\bar{f}(x)) \le M_>(c_\alpha) < (1-\alpha)M(0)$ by (i). If $\bar{f}(x) < c_\alpha$, then $M_>(\bar{f}(x)) \ge M(c_\alpha) \ge (1-\alpha)M(0)$. So $x \in R_\alpha$ exactly when $M_>(\bar{f}(x)) < (1-\alpha)M(0)$, that is, when $\mathrm{cov}(x) > \alpha$. For $\alpha = 1$, $\mathrm{cov}(x) \le 1$ and $R_1 = \emptyset$.
- *(v).* This is Scheffé's identity [DL01, Theorem 5.1], with $\int_{R_\alpha} \bar{f} = M(c_\alpha) \ge 1 - \alpha$. □

Marks. Proved here. [LEAN: Coverage prop_coverage_i ((i), with levelMass_split, levelMassGt_calpha_lt), prop_coverage_iii ((iii), calpha_mono), prop_coverage_iv ((iv), $\alpha = 1$ included, with calpha_le_iff, lt_cov_iff, cov_le_one), prop_coverage_v ((v) on the leaf masses $P(u) = \int_{\mathbf{x}_u} p$, from the finite Scheffé step CertifiedSearch two_abs_mass_sub_le_L1), prop_coverage_v_of_le ((v)'s stated form, with $\sum_u \left| \int_{\mathbf{x}_u} (p - \bar{f}) \right| \le \int |p - \bar{f}|$ a named hypothesis), and the remark after Definition 6.46, exists_hplus, calpha_pos, calpha_ge_hplus — built at `0fa5e8607`, on finite leaf sets; (ii) and the step from leaf masses to $\int |p - \bar{f}|$ are not modelled.] [SIM: highest density regions of histograms of samples from known densities, with the true coverage computed numerically, against the bound of (v)]

The minimality in (ii) is relative to the mass $M(c_\alpha)$ that the region actually holds. A set of mass exactly $(1-\alpha)M(0)$ and least volume generally contains only part of the leaves of height c_α, so it is not a union of leaves.

Algorithm 16: HighestDensityRegion(L, α): both forms [HST12]

Input: the leaves of a nonnegative $\mathbb{R}$-MRP with their heights $f(u)$ and masses $m(u)$, from a sorted leaf list or a traversal of the pointer tree, with $M(0) > 0$; $0 \le \alpha < 1$.

Output: the leaves of R_α, and c_α.

```
group the leaves of positive height into height classes H_1 > H_2 > ... > H_r, each with the
  total mass μ_j of its leaves;
a ← 0; j ← 0; C ← [ ];
while a < (1 − α)M(0) do // exact sums: integer counts, or an exact
  accumulator
    j ← j + 1; C ← C+ (the leaves of height H_j); a ← a + μ_j;
return (C, H_j);
```

The loop runs at least once, since $(1-\alpha)M(0) > 0$, and stops by $j = r$, since $\sum_j \mu_j = M(0)$. It adds whole height classes, so its output is $R(H_j)$ with $H_j = c_\alpha$, by Proposition 6.47(i). *Cost.* $O(N \log N)$ comparisons for the sort, and $O(N)$ additions. The coverage value and the anomaly score of every leaf take one more pass: $\mathrm{cov}(u) = 1 - (\text{mass of the classes above } f(u))/M(0)$, and $a(u)$ subtracts the mass of the class of u as well.

Spark form. Map each leaf to (height, mass), and reduce by height in $(\mathbb{R}_{\ge 0}, +, 0)$; for counts,

in $(\mathbb{Z}_{\geq 0}, +, 0)$. Sort the distinct heights in decreasing order, and compute prefix sums of the masses from per-partition totals: one pass on the partitions, a scan of the totals on the driver, and one more pass. Then c_α is the first height whose prefix sum reaches $(1-\alpha)M(0)$. *Law:* c_α depends only on the multiset of (height, mass) pairs, so the result is partition-invariant when the sums are exact, as they are for integer counts.

A centre-outward median. As α decreases from 1 the regions R_α expand from the modes outwards (Proposition 6.47(iii)).

- *The median region.* The boundary of $R_{1/2}$, where the expanding region first holds half the mass, splits the probability into an inner part of at least one half and an outer part. They are exactly one half each when the classes above and at $c_{1/2}$ reach $M(0)/2$ exactly.
- *A median in any dimension.* In one dimension the median splits the mass left from right, while $\partial R_{1/2}$ splits it inside from outside. The coverage value cov is a centre-outward depth: near 1 at the modes, near 0 in the tails.
- *Transformed coordinates.* Both are computed exactly from the leaf list, without numerical integration, in the coordinates of the paving. A bijection t carries these sets and their probabilities, and so the coverage guarantee (v). It does not carry the property of being highest-density, since the density picks up the Jacobian of t.

Example 6.48 (Highest density regions of the running example)**.** The density f of Example 6.10 has height classes 2.4 (mass 0.6, on $\rho\mathsf{RL}$), 0.8 (mass 0.2, on $\rho\mathsf{RR}$) and 0.4 (mass 0.2, on $\rho\mathsf{L}$), and $M(0) = 1$.

- *Regions.* $R_{1/2} = \mathbf{x}_{\rho\mathsf{RL}} = [0.5, 1] \times [0, 0.5)$, with mass 0.6. $R_{0.3} = \mathbf{x}_{\rho\mathsf{R}}$, with mass 0.8. $R_{0.1} = \mathbf{x}_\rho$.
- *Coverage values.* cov $= 1$ on $\rho\mathsf{RL}$, $1 - 0.6 = 0.4$ on $\rho\mathsf{RR}$, and $1 - 0.8 = 0.2$ on $\rho\mathsf{L}$. These reproduce the regions by (iv): $R_{0.3} = \{\text{cov} > 0.3\}$, for instance.
- *Anomaly scores.* $a = 0.4$ on $\rho\mathsf{RL}$, 0.2 on $\rho\mathsf{RR}$, and 0 on $\rho\mathsf{L}$: one minus the masses 0.6, 0.8 and 1 of the smallest regions containing them.
- *Median.* The median region $R_{1/2}$ holds 0.6, not 0.5, because the top class already exceeds one half.

Implementations: MRS `PiecewiseConstantFunction::outputCoverageRegion`, with the region found by the private `findCoverageRegion`, and `PiecewiseConstantFunction::findCoverage` for the coverage value of a point (`piecewise_constant_function.hpp`, `.cpp`). SparkDensityTree .

6.13 UniformApprox

The definition and the theorem are in §5.5 (Theorem 5.17). Here are the two forms of the algorithm.

The stack pops a left child before its sibling and finishes each subtree before the next, so *Out* comes in left-right order. The internal values, if wanted, are the Y of the split nodes.

Cost. One evaluation of G and one of g per node of the output, at most $2^{k_\epsilon+1} - 1$ from the root (Theorem 5.17). The leaf-list form uses a stack of at most $k_\epsilon + 1$ entries.

Spark form. Rounds, one per depth. The frontier is an RDD of pairs (label, cell). Each round is a `flatMap` that emits a passing cell as a leaf of the output and replaces a failing cell by its two children, which form the next frontier. There is one round for each depth $0, 1, \ldots$ up to the depth of the output, at most k_ϵ (Theorem 5.17), so at most $k_\epsilon + 1$ rounds: the cells of depth k_ϵ must still be tested and emitted. *Law:* a map; the refinement below a cell depends only on that

Algorithm 17: UniformApprox(u, g, G, ϵ): pointer form [HST12]

Input: a node u of an $\mathbb{IR}$-MRP (initially the root); g; an inclusion function G of g; $\epsilon > 0$.
Output: the subtree at u, refined in place, with $F(v) = G(\overline{\mathbf{x}}_v)$ at every node visited.

$F(u) \leftarrow G(\overline{\mathbf{x}}_u)$;
if u is not a leaf **then**
  UniformApprox$(u\mathsf{L}, g, G, \epsilon)$; UniformApprox$(u\mathsf{R}, g, G, \epsilon)$;
else if $\max\{\overline{F(u)} - g(\xi_u),\ g(\xi_u) - \underline{F(u)}\} > \epsilon$ // $\xi_u = \operatorname{mid} \mathbf{x}_u$ **then**
  if u can be split **then**
    split u; UniformApprox$(u\mathsf{L}, g, G, \epsilon)$; UniformApprox$(u\mathsf{R}, g, G, \epsilon)$;
  else report u `// Remark 5.18; in exact arithmetic every cell can be split`;

Algorithm 18: UniformApprox(g, G, ϵ): leaf-list form

Input: g; an inclusion function G of g; $\epsilon > 0$; the root box and the split axes.
Output: the sorted leaf list of the output of Algorithm 17 started from the root.

$S \leftarrow$ a stack holding $(\rho, \mathbf{x}_\rho)$; $Out \leftarrow [\,]$;
while $S \neq [\,]$ **do**
  pop $(u, \mathbf{b})$; $Y \leftarrow G(\overline{\mathbf{b}})$; $\xi \leftarrow \operatorname{mid} \mathbf{b}$;
  if $\max\{\overline{Y} - g(\xi),\ g(\xi) - \underline{Y}\} \leq \epsilon$ ***or*** $\mathbf{b}$ cannot be split **then**
    $Out \leftarrow Out + [(u, Y)]$;
  else
    push ($u\mathsf{R}$, the right half of $\mathbf{b}$); push ($u\mathsf{L}$, the left half of $\mathbf{b}$);
return Out;

cell, so the output is partition-invariant. The frontier can be repartitioned between rounds to balance the work.

Example 6.49 (Enclosing x^2)**.** Let $g(x) = x^2$ on $[0, 1]$, with $G(X) = X^2 = [\underline{X}^2, \overline{X}^2]$ on subintervals of $[0, 1]$, and $\epsilon = 1/8$.

- *The splits.* The root fails the test, since $1 - 0.25 = 0.75$, and is split. $[0, 0.5)$ fails $(0.25 - 0.0625 = 0.1875)$, $[0.5, 1]$ fails $(1 - 0.5625 = 0.4375)$, $[0.5, 0.75)$ fails $(0.5625 - 0.390625 = 0.171875)$, and $[0.75, 1]$ fails $(1 - 0.765625 = 0.234375)$.
- *The leaves.* Six leaves pass: $[0, 0.25)$, $[0.25, 0.5)$, $[0.5, 0.625)$, $[0.625, 0.75)$, $[0.75, 0.875)$ and $[0.875, 1]$, the paving s_{223333}. Their enclosures are $[0, 0.0625]$, $[0.0625, 0.25]$, $[0.25, 0.390625]$, $[0.390625, 0.5625]$, $[0.5625, 0.765625]$ and $[0.765625, 1]$.
- *The bounds.* Every width is at most $2\epsilon = 0.25$. The largest deviation from the midpoint value is $1 - 0.87890625 = 0.12109375 \leq \epsilon$, on the last leaf.
- *The depth.* G is Lipschitz with $\operatorname{Lip}(G) = 2$ on $[0, 1]$, so $w_\epsilon = 1/16$ and $k_\epsilon = 4$. The output has depth 3.

Implementations: MRS `FunctionEstimatorInterval::bruteForceEstimate` (`functionestimator_interval.cpp`), with the visitor `IntervalExpanderEstimator` (`intervalexpander_estimate.cpp`), and `FunctionEstimatorReal::bruteForceEstimate` (`functionestimator_real.cpp`), with `RealExpanderEstimator`, for the midpoint $\mathbb{R}$-MRP. The conversion of an enclosure to its midpoint MRP is `FunctionEstimatorInterval::makePiecewiseConstantFunction`. SparkDensityTree: none at `c0a1837`.

6.14 Split, propagate hulls, prune

UniformApprox fixes the tolerance and lets the tree grow. When memory fixes the number of leaves instead, a randomised priority queue spends the leaves where the enclosure is worst [HST12]. This is the refinement of the Moore rejection sampler (§3.2), now on a mapped tree. Splitting further than needed, propagating the hulls of the enclosures up the tree and pruning back tightens the enclosure on a tree of the wanted size [HST12, SHT13]. Three algorithms work in sequence:

1. **RPQEnclose**$^{\triangledown}$ splits, from the root, a leaf of largest priority, drawn uniformly among ties, until the tree has $\bar{\ell}$ leaves. Each new child gets $F \leftarrow G(\overline{\mathbf{x}})$ of its cell, so every node carries G of its cell.
2. **HullPropagate** replaces every internal value by the hull of its children's values, from the leaves up.
3. **RPQEnclose**$^{\triangle}$ prunes, back to $\bar{\ell}' < \bar{\ell}$ leaves, a cherry of smallest priority, drawn uniformly among ties. A pruned cherry keeps its hull value.

The default priority is the *volume of enclosure* of a cell, $\psi(u) = \operatorname{vol}(\mathbf{x}_u) \operatorname{wid} F(u)$. Its sum over the leaves is Ξ (Definition 5.16), so splitting by it aims at a small Ξ. The priority $\psi(u) = \operatorname{wid} F(u)$ aims instead at a small largest width.

Algorithm 19: RPQEnclose$^{\triangledown}$, HullPropagate and RPQEnclose$^{\triangle}$: pointer form [HST12, SHT13]

Function RPQEnclose$^{\triangledown}(G, \psi, \bar{\ell})$ `// from the root alone, with` $F(\rho) = G(\mathbf{x}_\rho)$**:**
  $Q \leftarrow$ a priority queue of the leaves, keyed by ψ;
  while $|\mathcal{L}(s)| < \bar{\ell}$ ***and*** Q holds a leaf that can be split **do**
    $u \leftarrow$ a leaf drawn uniformly from the leaves of largest ψ among the splittable ones in Q; remove u from Q;
    split u; $F(u\mathsf{L}) \leftarrow G(\overline{\mathbf{x}}_{u\mathsf{L}})$; $F(u\mathsf{R}) \leftarrow G(\overline{\mathbf{x}}_{u\mathsf{R}})$; insert $u\mathsf{L}, u\mathsf{R}$ into Q;
Function HullPropagate(u) `// Algorithm 10 with` $\oplus = \sqcup$**:**
  if u is not a leaf **then** HullPropagate$(u\mathsf{L})$; HullPropagate$(u\mathsf{R})$; $F(u) \leftarrow F(u\mathsf{L}) \sqcup F(u\mathsf{R})$;
Function RPQEnclose$^{\triangle}(\psi, \bar{\ell}')$**:**
  $Q \leftarrow$ a priority queue of the cherries, keyed by ψ;
  while $|\mathcal{L}(s)| > \bar{\ell}'$ ***and*** $Q \neq \emptyset$ **do**
    $u \leftarrow$ a cherry drawn uniformly from those of smallest ψ in Q; remove u from Q;
    Prune$(u\mathsf{L})$; Prune$(u\mathsf{R})$; `//` u `is now a leaf and keeps` $F(u)$
    if $u \neq \rho$ and sib(u) is a leaf **then** insert the parent of u into Q;

Leaf-list forms. The same loops run on labels. A uniform draw among the tied entries of largest (or smallest) priority needs all of them at hand, so the queue is a balanced search tree keyed by priority, or a heap from which the whole tie class is extracted; MRS uses a multiset and its `equal_range`. RPQEnclose$^{\triangledown}$ keeps such a queue of (priority, label) for the leaves and sorts the $\bar{\ell}$ output leaves at the end, with $O(\lambda(\log \bar{\ell} + t))$ word operations per step when t entries tie, $O(\lambda \bar{\ell} \log \bar{\ell})$ for the final sort, and $2(\bar{\ell} - 1)$ evaluations of G. HullPropagate is Algorithm 11 with $\oplus = \sqcup$ and the empty interval as e. RPQEnclose$^{\triangle}$ keeps the leaves in a hash set and such a queue of cherries; after a prune, the parent becomes a cherry when the sibling's label is in the leaf set. That costs $O(\lambda(|I(s)| + k \log |I(s)| + T))$ word operations for k prunes, with T the total size of the tie classes drawn from.

Proposition 6.50 (Hull propagation)**.** *Let (s, F) be an $\mathbb{IR}$-MRP in multi-resolution form, and*

run HullPropagate.

- *(i) Afterwards every node carries* $F(u) = \bigsqcup_{l \in \mathcal{L}(s),\, l \sqsubseteq u} F(l)$*, and the leaf values are unchanged.*
- *(ii) If every leaf value encloses the range of* g *over its cell, then so does every node value.*
- *(iii) If, before the call,* $F(u) = G(\overline{\mathbf{x}}_u)$ *at every node, for an inclusion isotone inclusion function* G *of* g*, then afterwards* $g(\mathbf{x}_u) \subseteq F(u) \subseteq G(\overline{\mathbf{x}}_u)$ *at every node.*
- *(iv) Under (iii), for every paving* s' *with* $I(s') \subseteq I(s)$*, the leaf-valued MRP* $(s', F|_{\mathcal{L}(s')})$ *encloses* g*, and* $\Xi(s', F) \leq \sum_{l \in \mathcal{L}(s')} \operatorname{wid} G(\overline{\mathbf{x}}_l) \operatorname{vol}(\mathbf{x}_l)$*, the volume of the direct enclosure on* s'*.*

Proof.

- *(i).* Proposition 6.19(i) with $\oplus = \sqcup$.
- *(ii).* The range over $\mathbf{x}_u$ is the union of the ranges over the leaves below u, each contained in its leaf's value, hence in their hull.
- *(iii).* For a leaf $l \sqsubseteq u$, $F(l) = G(\overline{\mathbf{x}}_l) \subseteq G(\overline{\mathbf{x}}_u)$ by isotony. $G(\overline{\mathbf{x}}_u)$ is an interval containing all of them, so it contains their hull. The lower inclusion is (ii).
- *(iv).* The leaves of s' are nodes of s; apply (ii) and (iii). □

Marks. Proved here. [LEAN: Fold prop_hull_i, prop_hull_ii, prop_hull_iii, prop_hull_-iv — built at `b6ea25813`; machine-checked-structural: proved over an abstract commutative idempotent monoid ordered by $x \oplus y = y$ (Proposition 6.19's framing of $(\mathbb{IR} \cup \{\emptyset\}, \sqcup, \emptyset)$), with the model facts as hypotheses: the points map and the width monotone in the hull order, vol ≥ 0, each cell covered by its children's cells, G isotone on nested cells; Mathlib's intervals are not instantiated] [SIM: hull propagation on enclosures of random polynomial expressions; (iii) checked node by node against G evaluated with C-XSC]

So the split–propagate–prune sequence ends with an enclosure on its final tree that is, leaf by leaf, at least as tight as G on that tree. The final tree is chosen by the prune and can differ from the tree that RPQEnclose$^{\triangledown}$ reaches at $\bar{\ell}'$ leaves. Part (iv) compares enclosures on one tree, not the two procedures; the examples below show what the sequence gains in practice.

Priority queues as threshold rules. The two queues have a structure that the sequential description hides. Call a real function ψ on nodes *monotone on* a set N of nodes when $\psi(v) \leq \psi(u)$ whenever $v \sqsubseteq u$ both lie in N; *monotone* when it is monotone on all nodes, that is, $\psi(u\mathsf{L}), \psi(u\mathsf{R}) \leq \psi(u)$ for every u; and *strict* when, in addition, $\psi(u\mathsf{L}), \psi(u\mathsf{R}) < \psi(u)$ whenever $\psi(u) > 0$.

Proposition 6.51 (Monotone priority queues are threshold rules)**.**

- *(i) Splitting. Let* ψ *be monotone. Run the queue from the root alone for* $k \geq 1$ *splits, each splitting a leaf of largest* ψ*, and let* $p_1, \ldots, p_k$ *be the priorities of the split nodes. Then* $p_1 \geq \cdots \geq p_k$*. With* $t = p_k$*, the internal-node set satisfies* $\{u : \psi(u) > t\} \subseteq I(s_k) \subseteq \{u : \psi(u) \geq t\}$*, and* t *is the* k*-th largest value of* ψ *over all nodes, counted with multiplicity.*
- *(ii) Pruning. Let* s *be a paving and* ψ *monotone on* $I(s)$*. Prune* $k \leq |I(s)|$ *cherries, each a cherry of smallest* ψ*, and let* $q_1, \ldots, q_k$ *be their priorities. Then* $q_1 \leq \cdots \leq q_k$*. With* $t = q_k$*, the set* P *of pruned nodes satisfies* $\{u \in I(s) : \psi(u) < t\} \subseteq P \subseteq \{u \in I(s) : \psi(u) \leq t\}$*,* t *is the* k*-th smallest value of* ψ *on* $I(s)$*, counted with multiplicity, and* P *contains every internal node of* s *below each of its elements.*
- *(iii) Ties in splitting. In (i), let* ψ *be strict and* $t > 0$*. Then the nodes with* $\psi = t$ *are pairwise disjoint, and each is* ρ *or has its parent in* $\{\psi > t\}$*. If the queue draws uniformly among the leaves of largest* ψ*, then* $I(s_k)$ *is* $\{\psi > t\}$ *together with a uniformly random subset, of size* $k - |\{\psi > t\}|$*, of* $\{\psi = t\}$*. Whatever the tie-breaking,* $I(s_k)$ *is* $\{\psi > t\}$ *together with a subset of that size of* $\{\psi = t\}$*, and every such subset can occur.*

- *(iv) Ties in pruning. In (ii), let every cherry of smallest ψ be drawn with positive probability. Then the possible sets of pruned nodes are exactly the sets $P \subseteq I(s)$ of k nodes with $\{u \in I(s) : \psi(u) < t\} \subseteq P \subseteq \{u \in I(s) : \psi(u) \le t\}$, for t the k-th smallest value of ψ on $I(s)$, that contain every internal node of s below each of their elements.*
- *(v) The priorities in use. If G is inclusion isotone, $\psi(u) = \operatorname{vol}(\mathbf{x}_u) \operatorname{wid} G(\overline{\mathbf{x}}_u)$ is strict, and $\psi(u) \le 2^{-|u|}\operatorname{vol}(\mathbf{x}_\rho) \operatorname{wid} G(\mathbf{x}_\rho)$; $\psi(u) = \operatorname{wid} G(\overline{\mathbf{x}}_u)$ is monotone. After HullPropagate, $\psi(u) = \operatorname{vol}(\mathbf{x}_u) \operatorname{wid} F(u)$ satisfies the strict condition on $V(s)$, and is in particular monotone on $I(s)$.*

Proof.

- *(i), order.* When p_j is split, its children have priority at most p_j, and the other leaves had priority at most p_j, which was the largest. So $p_{j+1} \le p_j$.
- *(i), upper inclusion.* A split node has priority $p_j \ge t$.
- *(i), lower inclusion.* Suppose $\psi(u) > t$ and $u \notin I(s_k)$. Let a be the shallowest node with $u \sqsubseteq a$ and $a \notin I(s_k)$. It is a leaf of s_k (Lemma 4.2(vi)), and $\psi(a) \ge \psi(u) > t$ by monotonicity. If a had become a leaf at step k, its parent would have priority $p_k = t \ge \psi(a)$, which is false. So a was a leaf when step k chose a leaf of priority $t < \psi(a)$, against the choice of a largest.
- *(i), the value of t.* $I(s_k)$ has k nodes, all with $\psi \ge t$, and contains every node with $\psi > t$. So fewer than k nodes have $\psi > t$, and at least k have $\psi \ge t$.
- *(ii), order.* When a cherry of priority q_j is pruned, the only new cherry is its parent, whose priority is at least q_j, and the other cherries had priority at least q_j. So $q_{j+1} \ge q_j$.
- *(ii), inclusions.* A pruned node has priority $q_j \le t$. Suppose $u \in I(s) \setminus P$ with $\psi(u) < t$. Let v be a deepest node of $I(s) \setminus P$ with $v \sqsubseteq u$. Then $\psi(v) \le \psi(u) < t$, and every internal node of s strictly below v is pruned, so v is a cherry after step k. If v had become a cherry at step k, the child pruned then would have priority $t > \psi(v)$, against monotonicity on $I(s)$. So v was a cherry when step k chose a cherry of priority $t > \psi(v)$, against the choice of a smallest.
- *(ii), the value of t and closure.* P has k nodes, all with $\psi \le t$, among them the node pruned at step k, of priority t, and it contains every internal node with $\psi < t$; so fewer than k internal nodes have $\psi < t$ and at least k have $\psi \le t$. A node is pruned only as a cherry, after every internal node of s below it.
- *(iii).* Under strictness, two nested nodes cannot both have $\psi = t > 0$, so the nodes with $\psi = t$ are disjoint. The parent p of such a node has $\psi(p) \ge t > 0$, hence $\psi(p) > t$. Their children have $\psi < t$ and are never split by step k. By (i), the splits of value t come after all those of value above t; at that point $I = \{\psi > t\}$, and the leaves of value t are exactly the nodes of $\{\psi = t\}$, all of them. The remaining $k - |\{\psi > t\}|$ splits each take one of them, and no new leaf has value t. A uniform draw at each of these splits gives a uniformly random subset; a choice made in advance gives any subset of that size.
- *(iv).* A possible P has these properties by (ii). Conversely, let P have them, and prune its nodes in order of increasing ψ, deepest first among equal values. At the turn of a node u, every internal node of s below u lies in P and has $\psi \le \psi(u)$, by monotonicity; those with smaller ψ, and those with equal ψ, which are deeper, have been pruned, so u is a cherry. A cherry c with $\psi(c) < \psi(u) \le t$ would lie in P and come before u; it has been pruned. So u is a cherry of smallest ψ and can be drawn.
- *(v).* Each child has half its parent's volume. By isotony, $\operatorname{wid} G(\overline{\mathbf{x}}_{u\mathsf{L}}) \le \operatorname{wid} G(\overline{\mathbf{x}}_u) \le \operatorname{wid} G(\mathbf{x}_\rho)$, so $\psi(u\mathsf{L}) \le \psi(u)/2$, which is less than $\psi(u)$ when $\psi(u) > 0$, and the bound on $\psi(u)$ follows by induction; likewise for $u\mathsf{R}$. After HullPropagate, $F(u\mathsf{L}) \subseteq F(u)$ by Proposition 6.50(i), and the same argument applies. □

Marks. Proved here; new. [LEAN: PQThreshold prop_pqThreshold_i ((i), with splitRun_or-

der, split_leaf_le, isKthLargest_unique; $I(s_k)$ has k nodes), prop_pqThreshold_ii ((ii), with pruneRun_order, pruneRun_inv, isKthSmallestOn_unique; P has k nodes), prop_pqThreshold_iv ((iv), both directions, the converse prune_reach for every t), prop_pqThreshold_iii ((iii) without its uniform law: the nodes with $\psi = t$ pairwise disjoint, each ρ or with its parent in $\{\psi > t\}$, $I(s_k)$ the set $\{\psi > t\}$ with a subset of $\{\psi = t\}$ of size $k - |\{\psi > t\}|$, and every such subset reached, by split_reach) — built at `796cfb0cd`, under every tie-break, for pavings as ancestor-closed internal-node sets and ψ in any linear order; the uniform law over ties in (iii) and (v) are not modelled.] [SIM: the sequential queues of MRS against the threshold rule, with seeded tie-breaking, on random expressions; the multiset of split priorities; for (iv), all sequential pruning outputs of small random trees with tied priorities against the characterisation (0 mismatches in 1634 cases, `scratchpad/r2_check.py`); for (iii), the uniform law over ties]

With the count as priority, which is monotone along ancestry, (i) is the structure behind Lemma 8.4: SEBTreeMC splits leaves of largest count and reaches the threshold paving of Lemma 8.3, whatever the tie-breaking. The proposition turns the queues from sequences of global steps, class (e) of §6.1, into threshold searches.

Spark forms. These are designs with proofs, under the stated hypotheses; no implementation exists.

- *Splitting.* Let ψ be strict, with $T(t_0) = \{u : \psi(u) > t_0\}$ finite for every $t_0 > 0$. This holds for $\psi = \operatorname{vol} \cdot \operatorname{wid} G$ with G isotone, where every node of $T(t_0)$ has depth below $\log_2(\operatorname{vol}(\mathbf{x}_\rho) \operatorname{wid} G(\mathbf{x}_\rho)/t_0)$ by (v). $T(t_0)$ is closed under ancestors, by monotonicity; it is computed by rounds of `flatMap` from the root, as in UniformApprox with the test $\psi(u) > t_0$, and its size by a reduction. To reach $\bar{\ell}$ leaves, take $k = \bar{\ell} - 1$, and assume $t > 0$, that is, at least k nodes have positive priority.
  - Find a $t_0 > 0$ with $|T(t_0)| \geq k$, by halving from $\psi(\rho)$. As $t > 0$, this stops once $t_0 < t$, since $T(t_0)$ then contains the k nodes of $I(s_k)$. A run caps the halving at a floor $t_{\min} > 0$; when it reaches the floor, t may be 0, ties may be nested, and the run falls back to the sequential queue on the driver.
  - Select the k-th largest priority t in $T(t_0)$, which is the k-th largest overall, by a distributed selection (class (e)).
  - Return $T(t)$ together with $k - |T(t)|$ nodes of $\{\psi = t\} \subseteq T(t_0)$, chosen by a seeded hash of their labels.

  Law: by (iii), the ties are pairwise disjoint with their parents in $T(t)$, so the result is closed under ancestors, a paving (Lemma 4.2(i)), and a possible output of the sequential queue; if the hash ranks the ties in a uniformly random order, its law is that of the queue with uniform tie-breaking. Every step is a map, a reduction or a selection of reals by exact comparison, so it is partition-invariant. For a priority that is only monotone, such as $\operatorname{wid} G$, ties can be nested and $T(t_0)$ infinite ($\operatorname{wid} G \equiv 3$ for the constant extension $G \equiv [-1, 2]$ of §5.5); no threshold form is claimed for it.
- *Hulls.* HullPropagate is a fold with the hull on $\mathbb{IR} \cup \{\emptyset\}$, a commutative and idempotent monoid. *Law:* Proposition 6.19 and class (c).
- *Pruning.* With hull-propagated values, $\psi = \operatorname{vol} \cdot \operatorname{wid} F$ is monotone on $I(s)$ (v). Select the k-th smallest priority t among the internal nodes (class (e)), and prune $\{u \in I(s) : \psi(u) < t\}$ together with the $k - |\{u \in I(s) : \psi(u) < t\}|$ deepest nodes of $I(s)$ with $\psi = t$, ties of one depth broken by a seeded hash. *Law:* the first set contains every internal node below each of its elements, by monotonicity, and taking the ties deepest first keeps this property, so by (iv) the pruned set is a possible output of the sequential queue. Its complement in $I(s)$ is closed under ancestors, so the result is a paving.

Example 6.52 (Split, propagate the hulls, prune)**.** Let $g(x) = x - x^2$ on $[0, 1]$, whose range is $[0, 0.25]$. Its natural extension is $G(X) = X - X^2 = [\underline{X} - \overline{X}^2,\ \overline{X} - \underline{X}^2]$ on subintervals of $[0, 1]$,

and it overestimates, since X appears twice. The priority is $\psi = \mathrm{vol} \cdot \mathrm{wid}\, F$.

- *Splitting to 6 leaves.* $G([0,1]) = [-1,1]$, so $\psi(\rho) = 2$. The splits, in order, are:
  - ρ: the children are $\rho\mathsf{L}$ with $[-0.25, 0.5]$, $\psi = 0.375$, and $\rho\mathsf{R}$ with $[-0.5, 0.75]$, $\psi = 0.625$;
  - $\rho\mathsf{R}$: the children are $\rho\mathsf{RL}$ with $[-0.0625, 0.5]$, $\psi = 0.140625$, and $\rho\mathsf{RR}$ with $[-0.25, 0.4375]$, $\psi = 0.171875$;
  - $\rho\mathsf{L}$: the children are $\rho\mathsf{LL}$ with $[-0.0625, 0.25]$, $\psi = 0.078125$, and $\rho\mathsf{LR}$ with $[0, 0.4375]$, $\psi = 0.109375$. At 4 leaves, $\Xi = 0.5$;
  - $\rho\mathsf{RR}$: the children are $\rho\mathsf{RRL}$ with $[-0.015625, 0.3125]$ and $\rho\mathsf{RRR}$ with $[-0.125, 0.234375]$;
  - $\rho\mathsf{RL}$: the children are $\rho\mathsf{RLL}$ with $[0.109375, 0.375]$ and $\rho\mathsf{RLR}$ with $[0.0625, 0.359375]$. At 6 leaves, $\Xi = 0.34375$.

  The split priorities $2, 0.625, 0.375, 0.171875, 0.140625$ decrease, as Proposition 6.51(i) says.
- *Propagating the hulls.* The internal values become $\rho\mathsf{L} : [-0.0625, 0.4375]$, $\rho\mathsf{RL} : [0.0625, 0.375]$, $\rho\mathsf{RR} : [-0.125, 0.3125]$, $\rho\mathsf{R} : [-0.125, 0.375]$ and $\rho : [-0.125, 0.4375]$. Each is inside the corresponding value of G; at the root, $[-0.125, 0.4375]$ against $[-1, 1]$.
- *Pruning back to 4 leaves.* The cherries have $\psi(\rho\mathsf{L}) = 0.5 \cdot 0.5 = 0.25$, $\psi(\rho\mathsf{RL}) = 0.25 \cdot 0.3125 = 0.078125$ and $\psi(\rho\mathsf{RR}) = 0.25 \cdot 0.4375 = 0.109375$. The prune takes $\rho\mathsf{RL}$, then $\rho\mathsf{RR}$. The final tree has leaves $\rho\mathsf{LL}, \rho\mathsf{LR}, \rho\mathsf{RL}, \rho\mathsf{RR}$ — the same tree as the direct split to 4 leaves — with $\Xi = 0.078125 + 0.109375 + 0.078125 + 0.109375 = 0.375$, against 0.5 for the direct enclosure.

Remark 6.53 (The reported examples)**.** Two runs with the volume-of-enclosure priority are reported in the sources.

- *An oscillating function* [HST12, SHT13]. For $g(x) = x^2 + (x+1)\sin^2(10\pi x)\cos^2(3\pi x)$ on $[0.5, 1]$, splitting to 45 leaves gave $\Xi = 0.1215$, and splitting to 50 gave 0.1102. Splitting to 50, propagating the hulls and pruning back to 45 gave 0.1159.
- *The Levy density* [HST12]. For a bivariate Levy density on $[-10, 10]^2$, splitting to 2000 leaves, propagating the hulls and pruning to 1400 gave $\Xi = 13.81$, against 18.21 for splitting directly to 1400.

The midpoint $\mathbb{R}$-MRP of the latter enclosure is the density whose highest density regions and marginals are shown in [HST12].

Implementations: MRS `FunctionEstimatorInterval::prioritySplit`, `hullPropagation` and `priorityMerge` (`functionestimator_interval.hpp`, `.cpp`), with `IntervalMappedSPnode::hullPropagation` (`intervalmappedspnode.cpp`). The volume-of-enclosure priority is `ReimannDiffMeasurer` and the width priority `IntervalWidthMeasurer` (`intervalmappedspnode_measurers.hpp`). The driver of the Levy run is `mrs-2.0/examples/MappedSP/Levy/LevyDensity2D_PaperExample.cpp`. SparkDensityTree: none at `c0a1837`; its count-priority analogues are the splitting and the backtrack of SEBTreeMC (Chapter 8, Chapter 9).

6.15 What runs where

The two forms compute the same functions. They differ in what they make cheap.

The pointer form, as in MRS, is best at: - *values at every node*: counts at every resolution, hulls, the multi-resolution views of Definition 5.1; - *refinement driven by a function*: UniformApprox and the priority queues grow the tree where an enclosure is wide, in place, with interval arithmetic and outward rounding in C-XSC [HHKR95]; - *point location* in time proportional to the depth of

the leaf; - *local moves on trees*: a split or a merge is a pointer change, which is what Markov chains on trees need, such as the chain over regular pavings of [STHL13] and SEBTreeMC (Chapter 8); - *slices and marginals by recursion*, touching only the nodes that matter (Proposition 6.27).

Its limits are those of one machine: memory for the whole tree, and recursion as deep as the tree.

The sorted leaf list, as in SparkDensityTree, is best at: - *data-driven counting at scale*: one map and one reduction by key (Phase 2); - *merges in one pass*: union, MRPOperate, collation and MinimiseLeaves are linear in the lists (Propositions 6.7, 6.11 and 6.14); - *partitioning by subtrees*: contiguous ranges of labels are whole subtrees, so refinement-based operations are local (Lemma 6.4); - *pushforwards and folds*: reductions by key in a commutative monoid (Propositions 6.18 and 6.19); - *many objects*: labelled forests with many ids (§6.11); - *fault tolerance*: recomputation from lineage, which is safe for every partition-invariant step (Definition 6.3).

Moving between them. The passage is linear in both directions.

Lemma 6.54 (Serialising a mapped tree)**.** *(i) The leaves of a paving, visited depth first with the left child before the right, come in left-right order. So a traversal of a pointer tree writes its sorted leaf list, with the heap code of each leaf computed on the way down, in $O(\lambda|V(s)|)$ word operations. (ii) A dense sorted leaf list determines its paving: the internal nodes are the strict ancestors of the leaves. The stack pass of Algorithm 6, run without the test of equal values, creates each internal node once, with its two children, and ends with the root alone. This rebuilds the pointer tree in $O(\lambda N)$ word operations. (iii) The values at internal nodes travel as a second list, of nodes in post-order. Algorithm 11 produces it for any fold; internal values that are free annotations, such as intensive heights (§5.6), are written by a post-order traversal of the pointer tree, in $O(\lambda|V(s)|)$ word operations.*

Proof. (i) A depth-first traversal visits all leaves below $u\mathsf{L}$ before those below $u\mathsf{R}$, and for disjoint leaves, $l \prec l'$ exactly when l lies in the left subtree of $l \wedge l'$ (Proposition 4.27(iv)). (ii) Run the proof of Proposition 6.11 with every pair of adjacent siblings reducible. Invariant (a) holds as before, every reduction replaces two siblings by their parent, and (c) now says that no two adjacent entries are siblings. The final stack F is the leaf set of a paving s_F with no cherry, since a dense list leaves no gaps, so s_F is the root alone. Each reduction creates the parent of two entries, once. (iii) For folds this is Proposition 6.19(iii); a post-order traversal visits every internal node once, after its children. □

Marks. Proved here. [LEAN: to write — `Serialize`, (i) on `PropBlock`] [SIM: round trips pointer tree → list → pointer tree on random pavings to depth 30]

So an algorithm can run in whichever form suits each stage, and hand its result to the next.

- **From the C++ form to Spark.**
  - *Enclosures.* An enclosure built by UniformApprox or by split, propagate and prune in MRS, with rigorous interval arithmetic, is serialised to a sorted leaf list. It can then serve as the envelope of a distributed Moore rejection sampler: the proposal chooses a leaf with probability proportional to $\mathrm{vol}(\mathbf{x}_u)\,\overline{F(u)}$, and then a uniform point in it (§3.6, [AS18]).
  - *Threshold forms.* When the tree outgrows one machine, the threshold forms of Proposition 6.51 run the queues themselves on Spark, with a possible output of the sequential queue, the same up to ties. They need a strict priority with finite search sets $T(t_0)$ and a positive threshold t for the split, and a priority monotone on $I(s)$ for the prune (§6.14); the volume of enclosure has both.
  - *Folds and canonical forms.* The multi-resolution folds become pushforwards (Proposition 6.19). MinimiseLeaves becomes a stack pass per partition and one on the driver

(§6.4).

- **From Spark to the C++ form.**
  - *After merging.* A distributed estimate is small once merged (Chapter 10). Collected and rebuilt by Lemma 6.54(ii), it supports point queries in time proportional to depth, slices and marginals by recursion, interval arithmetic on its cells, and Markov chains on its tree.
  - *Deep trees.* Conversely, the one-pass merges of the leaf lists serve the C++ form when trees are deep and unbalanced. They replace a recursion as deep as the tree by an explicit stack.
- **Both ways.** The locality lemma (Lemma 6.4) and the threshold rules (Proposition 6.51) are statements about trees, not about Spark. They partition the work among the threads of one machine as they do among the machines of a cluster.

A step that is not covered by a law of Table 6.1 does not move to Spark unchanged. A priority queue with a non-monotone priority has no threshold form here, and random tie-breaking must be seeded by label, not by arrival, for the result to be partition-invariant.

Chapter 7

Statistical regular pavings in use

Chapters 4 to 6 built regular pavings, maps on them and their arithmetic. This section shows the arithmetic at work in two published uses, from 2012–2013, each run on one computer with the C++ library MRS [SH10, SYH+18]: - **air-traffic trajectories**, where pavings of radar positions are added, subtracted and pruned as flights enter and leave [TKS12]; - **posterior-mean histograms**, where a Markov chain walks the space of pavings and its states are averaged [STHL13].

§7.6 says what the sparse labels of Chapter 4 change in each, and §7.7 lists the other uses of the arithmetic.

Notation in this section. - $\mathcal{L}(s)$ is the set of leaves of a paving s, as in Chapter 4. - k is a number of splits. - δ is a depth, and δ^* a maximal depth. - Flights are indexed by i, as in Chapter 6.

7.1 The statistical regular paving

Let $\mathbf{x}_\rho$ be a root box and s a regular paving of it. As in Chapter 4, the root box is closed and each cut leaves the left child half-open at the cut, so every point lies in exactly one leaf. A node at depth δ has volume $\mathrm{vol}(\mathbf{x}_\rho)/2^\delta$ (Chapter 4; [STHL13, §2.1]).

Caching. When n points $x_1, \ldots, x_n$ fall into $\mathbf{x}_\rho$, each node ρv keeps *recursively computable statistics* of the points in its box [STHL13, §2.2]: - the count $\#\mathbf{x}_{\rho v} = \sum_{j=1}^{n} \mathbf{1}_{\mathbf{x}_{\rho v}}(x_j)$; - the sample mean; - the sample variance–covariance matrix.

When a leaf is bisected, its points fall into the two children and their statistics are updated. When two siblings are reunited, the parent's statistics stay as they were.

The paper says the pointers from the leaves to the data "can be removed" once the paving is built, leaving "only the height statistics". It calls this "a significant memory-saving intermediary step when comparing batches of massive simulated data … or when producing a sequence of density estimates from pulses of massive real-world data" [STHL13, §2.3]. Pointers can be dropped only once no further split is needed. A chain on pavings (§7.5) keeps them, because each split must update the children's counts.

The paper says its caching is "inspired by" the cached-statistic metric trees of A. W. Moore (2000) and Gray and Moore (2003) [STHL13, Moo00, GM03, §1, §2.2]. This A. W. Moore is not R. E. Moore of Chapters 2 to 3.

The histogram. An SRP with counts is a mapped regular paving whose values are counts, in $\mathbb{Z}_{\geq 0}$. The counts are *extensive* in the sense of Chapter 5 (`Dfn-Kinds`): a parent's count is the

sum of its children's. After the scalar multiples and differences of §7.3, the values are real and may be negative. The histogram is

$$f_{n,s}(x) = \frac{\#\mathbf{x}(x)}{n \operatorname{vol}(\mathbf{x}(x))},$$

where $\mathbf{x}(x)$ is the leaf box containing x. It is zero outside the root box [STHL13, eq. (2)].

The *height* of a node is $h(\rho v) = \#\mathbf{x}_{\rho v}/(n \operatorname{vol}(\mathbf{x}_{\rho v}))$ [STHL13, §2.3]. Heights are *intensive* in the sense of Chapter 5: refining a leaf copies its height to both children. For an SRP, an internal node's height also happens to equal the mean of its two children's heights. Both uses below turn on this distinction.

Naming a paving. The paper names a paving by the depths of its leaves in left-right order, for example s_{23322}, the leaf-depth string of Chapter 4 [STHL13, §2.1]. $\mathbb{S}_k$ is the set of pavings with k splits, with C_k elements (Chapter 4), and $\mathbb{S}_{0:\infty}$ is the whole space. The root tree is s_0.

7.2 Trajectories as pavings

Radar records the positions of the aircraft in range "at least every 12 s in en-route areas and every 4.2 s around airport terminals" [TKS12, §I]. A flight's data are time-ordered points $X_1, \ldots, X_n$ in $\mathbb{R}^2$ (latitude, longitude) or in $\mathbb{R}^3$ (with altitude) [TKS12, §III.C].

The minimum volume. The paper stops bisecting at the volume

$$\lambda^* = 2^{-\delta^*} \operatorname{vol}(\mathbf{x}_\rho), \qquad \delta^* = \left\lfloor \log_2 \left(\operatorname{vol}(\mathbf{x}_\rho)/\lambda \right) \right\rfloor,$$

where λ is "an approximate volume of the aircraft" [TKS12, §III.C]. The paper writes i^* for δ^*, and prints $\operatorname{vol}(\mathbf{x}_{\rho v})$ for $\operatorname{vol}(\mathbf{x}_\rho)$. From the floor, $\lambda \le \lambda^* < 2\lambda$, so no box is smaller than an aircraft. §VI.A of the paper uses a ceiling instead, which gives $\lambda/2 < \lambda^* \le \lambda$, and calls δ^* the maximal depth.

Building. `makeSRP` starts from the root box with count zero and calls `insertData` for each point (Algorithms 1–2 of the paper). - `insertData` adds one to the count of every node whose box contains the point. - When the point reaches a leaf whose half-volume is still at least λ^*, the leaf is bisected and the point continues into the child that contains it.

So every point descends to depth δ^*. A leaf of volume greater than λ^* contains no point, as the paper states [TKS12, §III.C]. The result is the flight's *SRP trajectory*. Because the radar samples in time, the occupied boxes along a path are disconnected (Figure 2 of the paper).

What the SRP holds is the count of a flight's radar returns in each cell. An aircraft that circles returns to the same cells (aircraft B in Figure 12), so a count can exceed one. The flight's *occupancy* is the support of these counts, a $\{0, 1\}$-valued map, which is the reading of Chapter 6 (`Dfn-Forest`). The paper's leaf values are occupancies, but its programs add counts of returns, and their collator copies a leaf's counts to both children when it splits the leaf, which is right for heights but inflates counts (Chapter 6, `Rmk-CollatorSemantics`).

7.3 Arithmetic on trajectories

Addition. `AddSRP` (Algorithm 3 of the paper) adds two pavings of the same root box by recursive descent: - where both nodes have children, it recurses into both; - where one node is a leaf and the other is not, it makes temporary children for the leaf and recurses against the other's children; - at every node, the counts add.

For intensive values this is the union of Chapter 6 followed by the pointwise sum. For counts, the temporary children need a rule for their counts, and counts have none. That is the subject of the last paragraph of this subsection.

Linear combinations. For a real α, the paving $\alpha \cdot \rho^{(1)}$ multiplies every count of $s^{(1)}$ by α. Then `AddSRP`$(\alpha \cdot \rho^{(1)}, \beta \cdot \rho^{(2)})$ is the linear combination $\alpha s^{(1)} + \beta s^{(2)}$ [TKS12, §IV.B]. - With $\alpha = \beta = 1$ it is the sum of two flights. - Summing many flights gives an *aggregate trajectory*. Its smaller, darker boxes are the parts of the airspace occupied most often.

Good weather minus bad weather. The paper builds two aggregate trajectories from arrivals at a busy North American airport, over six hours each: one on a day of good weather and one on a day of bad weather. Their difference, with $\alpha = 1$ and $\beta = -1$, shows which parts of the airspace were used more on each day [TKS12, Figs. 6–7]. The paper says this "could be the first step toward classification and prediction of airspace usage based on concurrent weather data". It also notes that bad-weather days differ from one another, which makes them hard to classify [TKS12, §IV.B].

Occupancy at an instant. For a single instant, the aggregate of the flights in the air should be $\{0, 1\}$-valued: "if the value at a leaf box or cell is more than one at an instant of time, then it means that a collision has happened there" [TKS12, §II]. Read at the resolution λ^*, a value of two means two aircraft in one cell about the size of an aircraft.

When copying a leaf's value down is right. When one operand is a leaf and the other is not, `AddSRP` gives the temporary children counts that the paper writes as those of the leaf's own children, which do not exist. Two readings are possible: - Copy the leaf's count to both children. This multiplies the leaf's mass by 2^j over j levels of temporary refinement, which is wrong for counts (Chapter 5). - Split the count by volume. This is right for densities, but not what a trajectory means.

The TOMACS paper's `AddSRPHist` (§7.5) adds heights and copies the leaf's height to both temporary children [STHL13, Alg. 1]. That is correct, because heights are intensive.

For trajectories, the following invariant settles the question. It is an observation of this monograph, which A1_0-0's review confirmed by simulation: with a common λ^*, 500 of 500 random pairs kept their total, and with different λ^*, 405 of 500 did not.

> **Invariant.** Every leaf shallower than δ^* has value 0.

- `makeSRP` establishes it for one root box and one λ^*, by the guarantee of §7.2.
- It is preserved by `AddSRP`, by scalar multiples and by `ReuniteEmptyNodes` as printed.
- Under it, a leaf of one operand that meets an internal node of the other is empty, so `AddSRP` copies the count zero, which is correct.
- The invariant fails, and the choice above returns, when an operand has a nonempty leaf shallower than δ^*. Three things produce one:
  - operands built with different precisions, which the paper proposes "when different data sets or even points have different levels of precision" [TKS12, §IV.B];
  - an operand coarsened by the pushforward of Chapter 6;
  - a reunion "modified to reunite nodes with the given property" [TKS12, §V.B].
- There the choice must be made explicitly. Either coarsen the finer operand to the coarser paving and add counts there, which is a different operation from `AddSRP` (Chapter 6), or compare heights, which are intensive.

7.4 A dynamic airspace

Reuniting. `ReuniteEmptyNodes` (Algorithm 4 of the paper) works bottom-up. Wherever both children of a node are empty leaves, it deletes them and makes the parent a leaf with count zero [TKS12, §V.A]. No information is lost, and the memory of the pair is freed. In a sparse map an absent key already is the reunited node (Chapter 6, `Rmk-Reunite` and the example of two flights).

Entering and landing. To track only the flights in the air: - a flight entering the airspace is added; - a flight that has landed is subtracted, its now-empty leaves are reunited, and the aggregate keeps only the flights still flying (Figure 10 of the paper).

The paper separates three settings [TKS12, §I]: - *n-presensed*: all data present; - *$n_{1:m}$-presensed*: m bursts read from external memory, at times one may choose. The paper's runs use bursts of 30 minutes (Figs. 11c, 13c). - *$n_{1:m:\infty}$-sensing*: data arriving from the radar.

With radar bursts of 10 s, the CPU time per burst stays below about 5 s against the 10 s line, by inspection of Fig. 14. The machine had two six-core Intel X5670 processors and 48 GB of memory [TKS12, §VI.B, Fig. 14].

Complexity. On a unit cube with cells of side h (so $\lambda = h^d$, and $q = 1/h$ cells along each axis), a regular grid has $q^d = 1/\lambda$ cells, exponential in d [TKS12, §VI.A]. - *Insertion.* An SRP spends $\delta^* = O(d \log_2 q)$ node visits per point, linear in d, against one for a grid [TKS12, §VI.B]. - *Memory.* The SRP spends nodes only where flights have been. By Figs. 11a and 13a, an individual trajectory needs about 10^{-3} as many nodes as the grid has cells. By Fig. 13b, the aggregates reach about 0.08 on the good-weather day and 0.16 on the bad-weather day. These values are read from the figures; the paper's text attributes 0.08 to individual trajectories, which its figures contradict. From $d = 2$ to $d = 3$ the ratio falls by three orders of magnitude (Fig. 11). An aircraft circling over one region adds few nodes, because its points fall into boxes already split (aircraft B in Fig. 12).

The node bound, restated. Write $|s|$ for the number of nodes of s. The paper writes $2j + 1$ nodes for j leaves, and bounds the count by $\min\{n\,\delta^*, 2q^d - 1\}$ (printed as $nd \log_2(1/h)^d$, which counts splits) [TKS12, §VI.A]. A full binary tree with j leaves has $2j - 1$ nodes, which agrees with the paper's own worst case $2q^d - 1$. Each point causes at most δ^* bisections, and each bisection adds two nodes, so

$$|s| \le \min\{1 + 2n\,\delta^*,\ 2q^d - 1\}.$$

The paper's $n\,\delta^*$ fails already at $n = 1$, where $|s| = 1 + 2\delta^*$. In A1_0-0's simulation of 2000 random builds, it was exceeded in 1264. The orders of growth, linear in n and in d, are unchanged.

Other moving objects. The same arithmetic tracks animals. Figure 8 of the paper sums the paths of one animal over three days, with data courtesy of Paul Sagar [TKS12, §IV.B]. The paper's stated purpose for air traffic is quoted in Chapter 6 (`Dfn-Forest`). Velocity, fuel and other measures can be further coordinates of the root box [TKS12, §VII].

7.5 Posterior-mean histograms

The second use asks which paving a histogram should use, and answers: average over pavings, weighting each by its posterior probability given the data [STHL13].

The posterior. The likelihood of the data given s is approximated by its maximum value from

the histogram on $\mathcal{L}(s)$ [STHL13, §2.3]:

$$\pi(s) \;\propto\; \prod_{\mathbf{x}_{\rho v}\in\mathcal{L}(s)} \Big(\frac{\#\mathbf{x}_{\rho v}}{n\,\mathrm{vol}(\mathbf{x}_{\rho v})}\Big)^{\#\mathbf{x}_{\rho v}} \cdot \Pr\{s\},$$

with $0^0 = 1$ for empty leaves. The paper names the approximation "a current limitation". The fully Bayesian alternative it proposes is a Dirichlet prior over the heights, as in the Pólya trees of Lavine [STHL13, Lav92, Lav94, §5].

The Catalan prior. Take any summable non-increasing sequence $a_k > 0$ with $\sum_k a_k = a$. The *$\{a_k\}$-penalised uninformative proper Catalan prior* gives the class $\mathbb{S}_k$ the mass a_k/a and spreads it uniformly over its C_k pavings:

$$\Pr\{s\} = \sum_k \mathbf{1}_{\mathbb{S}_k}(s)\frac{a_k}{a\,C_k}.$$

The paper fixes $a_k = 1/C_k$, the *natural Catalan prior* [STHL13, eqs. (3)–(4)]:

$$\Pr\{s\} = \frac{1}{a\,C_k^2} \quad (s\in\mathbb{S}_k), \qquad a = \sum_{k\geq 0}\frac{1}{C_k} = 2 + \frac{4\pi}{3^{5/2}} \approx 2.8061.$$

The paper cites the value of a to OEIS A000108 [MC05]. - *The index.* The paper indexes a_k from $k = 1$, but its value of a is the sum from $k = 0$; the sum from $k = 1$ is $1 + 4\pi/3^{5/2}$. The index must start at 0 for the prior to cover s_0. From $k = 0$, $a_0 = a_1 = 1$, which is why the sequence is non-increasing rather than decreasing. - *The induced prior on size.* Under this prior, the number of splits has the law $\Pr\{k\} = 1/(a\,C_k)$, which decays like $4^{-k}k^{3/2}$. The prior penalises large pavings geometrically.

Averaging. Two histograms with the same root box are added by `AddSRPHist` [STHL13, Alg. 1, Fig. 4]. It is `AddSRP` on heights: where a leaf meets an internal node, the leaf's height is copied to both temporary children, which is correct because heights are intensive (§7.3). The average of r histograms is $((\cdots((f_{n,s^{(1)}} + f_{n,s^{(2)}}) + f_{n,s^{(3)}}) + \cdots) + f_{n,s^{(r)}})/r$, on the union of their pavings [STHL13, §2.4]. In Chapter 6's terms, it is the average of the collation (`Dfn-Collation`).

A finite state space, set by the machine and the data. On a machine with double-precision numbers, the space of pavings is necessarily finite. A box whose widest side is already two adjacent floating-point numbers cannot be bisected [STHL13, §3]. This is the limit of numerical resolution of §2.3 and §3.4 again: the machine bounds the state space. The paper bounds the space $\widetilde{\mathbb{S}}_n$ in three ways: 1. at most $\overline{m}_n = n^\beta$ splits, with $\beta < 1$; 2. a minimum leaf volume $\underline{v}_n$, equivalent to a maximal depth $\overline{\delta}_n = \lfloor \log_2(\mathrm{vol}(\mathbf{x}_\rho)/\underline{v}_n)\rfloor$, so that $\widetilde{\mathbb{S}}_n \subseteq \mathbb{S}_{0:\tau}$ with $\tau = 2^{\overline{\delta}_n} - 1$; 3. the data-driven rule: split a leaf only if each child receives at least $\#_n$ points, or if one child is empty and the other receives all of $\#_n$ or more points.

The splittable leaves under rule 3 are $\widehat{\mathcal{L}}(s) \subseteq \mathcal{L}(s)$, and the paper sometimes combines rule 3 with rules 1 and 2. On $\#_n$ it makes two statements: "In all the simulations carried out here, we use this $\#_n$-specified splitting rule and fix $\#_n = 1$", and that it observed no significant change in integrated absolute error for $\#_n \in \{0, 1, 2, 3\}$, since the Catalan prior makes boundary states rare [STHL13, §3].

The chain. The *stay-split-merge* base chain on $\widetilde{\mathbb{S}}_n$ [STHL13, eq. (6)]: - stays with probability $\sigma > 0$; - otherwise proposes a split or a merge, with probability $(1-\sigma)/2$ each; - for a split, picks a splittable leaf uniformly; - for a merge, picks a cherry uniformly. A *cherry* is a node whose two children are leaves; its set is $c(s)$.

So

$$Q(s, s') = \frac{1-\sigma}{2|\widehat{\mathcal{L}}(s)|} \text{ (one split)}, \qquad \frac{1-\sigma}{2|c(s)|} \text{ (one merge)}, \qquad \sigma \ (s' = s).$$

The chain is irreducible, since any paving reaches the root by merges and returns by splits, and aperiodic, since it can stay [STHL13, §3].

Boundary states. At s_0, which has no cherry, and at any state with no splittable leaf, the row of Q sums to $\sigma + (1-\sigma)/2$. The missing mass must go somewhere. MRS adds it to staying: `proposeChangeMCMCState` in `adaptivehistogram.cpp` proposes no move with the remaining probability, and its proposal ratio $\log|\widehat{\mathcal{L}}(s)| - \log|c(s')|$ is consistent with that choice [SYH+18]. This changes no off-diagonal entry of Q, so no acceptance probability changes. Moving the mass to the possible move instead would also be consistent, but it changes the proposal ratio at boundary states.

The Metropolis–Hastings chain accepts a proposal with probability

$$a(s, s') = \min\left\{1, \frac{\pi(s')Q(s', s)}{\pi(s)Q(s, s')}\right\},$$

and its stationary law is π truncated to $\widetilde{\mathbb{S}}_n$ [STHL13, §3].

When chains dive. The acceptance ratio of a split of leaf $\mathbf{x}$, in a paving with k splits, factors into three parts, as the MRS implementation computes them [SYH+18]: - **Proposal:** $|\widehat{\mathcal{L}}(s)|/|c(s')|$. Under rule 3 with $\#_n = 1$, away from the machine's resolution, this is at least one, by an injection from $c(s')$ into $\widehat{\mathcal{L}}(s)$. Every internal node was split while it held a point, so every cherry of s' other than the new one has a nonempty leaf child, which is splittable and is not $\mathbf{x}$; send that cherry to that child, and the new cherry to $\mathbf{x}$. Under the caps of rules 1–2, or with $\#_n \geq 2$, the factor can be below one. One example has a depth cap of 3 and one splittable leaf at depth 2: splitting it creates the fourth cherry, so the factor is 1/4. - **Prior:** $(C_k/C_{k+1})^2 = \left(\frac{k+2}{2(2k+1)}\right)^2$. This is 1 for the first split and 1/4 for the second, and it decreases to 1/16 from above. - **Likelihood:** $2^{n_{\mathbf{x}}}(n_{\mathsf{L}}/n_{\mathbf{x}})^{n_{\mathsf{L}}}(n_{\mathsf{R}}/n_{\mathbf{x}})^{n_{\mathsf{R}}}$ for children holding $n_{\mathsf{L}} + n_{\mathsf{R}} = n_{\mathbf{x}}$ points. This is 1 for an even split and $2^{n_{\mathbf{x}}}$ when all the points go to one side.

What follows from the three factors: - **Coincident points always split.** A leaf holding four or more coincident points, or points closer together than the machine's resolution, splits them all to one side. Its prior times likelihood is at least $16\left(\frac{k+2}{2(2k+1)}\right)^2 > 1$ for every k (1.03 at $k = 100$). With the proposal factor at least one, the split is accepted with probability one at every depth, down to the floating-point floor. Three such points suffice while $k \leq 3$. Rounded radar or measurement data produce exactly such leaves. Neither rule 3 with $\#_n = 1$ nor the prior stops this. Only the depth and split caps of rules 1–2, or $\#_n \geq 2$, do. - **Distinct points stop diving.** Once distinct points are separated, a one-sided split of a single point has likelihood factor 2. After the first split, the prior's factor of at most 1/4 outweighs it, unless the proposal factor exceeds 2.

Starting and stopping. For structured densities, the chain is started from `InitSEBPQ` [STHL13, Alg. 2], a precursor of the randomised priority queue of Chapter 8. It repeatedly bisects a leaf drawn uniformly from those with the most points, a rule from the statistically-equivalent-blocks principle [And66, Ges70]. It records the states it visits and returns the one of highest log-posterior. For the uniform densities, the paper starts two chains, one from the root and one from a state with many leaves, and notes that `InitSEBPQ` "is not necessary for simpler unstructured densities" [STHL13, §3, §4.1].

Samples are collected once the Gelman–Rubin statistic for the number of leaves falls below 1.1 [GR92]. The paper adds trace plots of the log-likelihood (Chapter 4) or log-posterior (Chapter 5.2, Fig. 9) and of the number of leaves, since the statistic alone "can be misleading" [STHL13, §3].

What the simulations show. Each MIAE is over 25 replicate data sets, and each estimate averages 1000 to 10000 states, thinned by 10 to 50 [STHL13, §4]. - **Uniform densities on**

$[0,1]^d$, for $d \in \{1, 2, 10, 100, 1000\}$. Sample sizes go up to 10^8 in 1 and 2 dimensions, 10^7 in 10, and 10^6 in 100 and 1000, limited by memory. The MIAE falls by at least a factor of two for each tenfold increase in n, and does not depend on d, because the target is s_0 itself (Table I). The paper sets this against A. W. Moore's view that uniform data allow "little or no acceleration in high dimensions no matter what we do". - **Gaussian densities in 1, 2 and 5 dimensions, and Rosenbrock densities in 2 and 5.** Here the curse of dimensionality returns, through the number of leaves Λ needed to approximate the density (Table II).

The red thread. For the Gaussian and Rosenbrock tests, the target is replaced by a simple-function approximation with Λ leaves [STHL13, §4.2]: 1. the leaves are refined by the method of Chapter 3, bisecting the leaf of largest $\mathrm{vol}(\mathbf{x}) \times (\overline{y} - \underline{y})$, where $[\underline{y}, \overline{y}]$ is an interval enclosure of the density over the box; 2. each leaf's height is the density at the leaf's midpoint, normalised; 3. data are sampled exactly from that approximation [SY13].

Because both the truth and the estimate are pavings, the integrated absolute error is computed exactly, by the arithmetic of Chapter 6, even in high dimensions [STHL13, HST12, §4.2]. The simulation side of 2005 thus supplies the estimation side with exact errors, as §9.6 does again with `CheckBounds`.

The paper states who did what in its section "Authors' contributions" [STHL13].

Marginals. The average is defined on one paving, so its marginal densities come from the marginal algorithm of Chapter 6 (`Dfn-Marginal`, `Prop-Marginal`) [HST12]. A node split on a removed coordinate is replaced by the sum of its two marginalised children. A leaf drops the removed coordinates and scales its height by the ratio of volumes, so that height times volume is preserved. This is MRS's `AdaptiveHistogramCollator::makeMarginal` [SYH+18].

7.6 What sparse labels change

Every algorithm above descends pointer trees recursively. That works on one machine and does not scale to massive data or many dimensions (Chapter 10). With the sparse labels of Chapter 4, each use becomes a map over keys.

Trajectories. These are Chapter 6's forest of flights (`Dfn-Forest`, `Dfn-SumOut`, `Prop-SumOut`). With every flight on one root box, it is a root-labelled paving (Chapter 6). A point x of flight i becomes the key $(i, \ell_{\delta^*}(x))$. By Chapter 6's `Prop-Occupancy`, the set of these keys determines the 2012 trajectory SRP, whatever the order in which the points were inserted; the tree is only an index. Flights, animals and phone traces (§7.8) are the same object, with a different set of ids and a different depth. - *Occupancy* of flight i is a reduction by key with "or". - *Traffic*, the number of distinct flights in a cell, sums out the flight ids. At a coarser depth, each flight's occupancy must first be coarsened with "or", and only then the ids summed out. Coarsening the summed traffic with + would count incidences instead of flights. - *Linear combinations* are label-wise $\alpha a + \beta b$ at a common depth. With different precisions, one operand is first coarsened (Chapter 6). That is a different operation from `AddSRP` (§7.3). - *A landing* is a filter on i (`Prop-SumOut`). - `ReuniteEmptyNodes` is not needed (§7.4).

Posterior means. A chain's state is a set of leaf labels. A split replaces label u by $u\mathsf{L}, u\mathsf{R}$, and a merge does the reverse. The running average is a sparse map from labels to summed heights. Each new state is added label-wise on the common refinement, every label taking the height of the leaf above it in each operand (the copy-down of Chapter 5).

This is a design, not code. The code of Chapter 10 (commit `c0a1837`) has neither labelled trajectories nor chains on pavings.

7.7 Other uses of the arithmetic

The papers of §§ 7.1 to 7.5 are two of several uses. The others follow. Each is a use only: Chapter 5 defines the value sets and Chapter 6 proves the laws, and this subsection does not restate either. - **Spatial objects as Boolean maps.** A set built from cells is a map with Boolean values (Chapter 5, `Dfn-MRP`). Union, intersection, symmetric difference and set difference are the pointwise operations of Chapter 6 (`Prop-Operate`). The uses: - enclosures of spatial objects, such as the intersection of two hollow spheres in Figure 1b of [HST12]; - the regular subpavings of set inversion [JKDW01], which are Boolean maps. - **Colour.** Colours are one of Chapter 5's value sets. The use is the arithmetic of images: adding and subtracting colour maps cell by cell, and taking slices of a colour cube as colour maps of lower dimension, as pictured in [HST12]. - **Sampler envelopes.** The envelope of the Moore rejection sampler of Chapter 3 is a map with interval values, built by Chapter 6's enclosure algorithms (`UniformApprox`, `RPQEnclose`), with the guarantee of Chapter 5's `Thm-Enclose`. It is used by the sampler of §3.2–§3.3, one paving per labelled domain [SY13].

7.8 Co-trajectories, privacy and decision problems

A *trajectory* is a set of time-stamped locations, no two at the same time, with finitely many in every finite interval of time. A *co-trajectory* is a finite multiset of trajectories, one per object [Dah19, §2, Definitions 2.1–2.3].

On pavings, a co-trajectory is a root-labelled paving (Chapter 6, `Dfn-Forest` and its root-labelled form): - trajectory i gives the records $(i, (\ell, t))$, where ℓ is a depth-D cell and t a time interval, with value the number of its datapoints there, an extensive value; - summing out the ids gives the counts per (cell, interval); - transitions form a second count map, on ordered pairs of such keys, and coarsen by the pushforward on both coordinates.

Occupancy is a different statistic, and it does not add across cells (Chapter 6, "Occupancy and counts").

A sufficient sanitizer [SMT+20, §2]. - *Definition.* A sanitizer is a randomised map from the data to sanitized data that carries an added privacy guarantee. It is *sufficient* for a statistical experiment $\{P_\theta\}$ when it preserves that experiment's sufficient statistic. Then "any decision based on the estimated probability model of the data will be the same as without sanitization" [SMT+20, §1]. Here *preserves* means pathwise: for every realisation of the sanitizer's randomness. - *SwapMob.* SwapMob originates in [SMT18]. It cuts space into squares of side χ and time into intervals of length τ, and swaps the futures of objects that are in the same cell at the same time [SMT+20, §3], [Dah19, §4.1–§4.2]. This monograph uses Dahne's rule, which is `Dfn-Swap` of Chapter 6: - an object's state at the end of an interval is the cell of its last datapoint *in that interval*; an object with no datapoint in the interval has no state there and is not swapped; - the swap happens at the interval's end, among the objects in one state; - the swap is a uniformly random permutation of that group, so it can be the identity.

The other readings of the swap lose transitions (Remark 6.40), so this rule is the one to use. - *What it preserves.* The paper argues, in prose ("It is easy to see …"), that SwapMob preserves the sufficient statistics of an i.i.d. model over the cells and of a Markov chain over them, time-inhomogeneous as well as homogeneous [SMT+20, §2.1–§2.2, §3.2]. The precise statement is Dahne's [Dah19, §4.4], and `Thm-Swap` of Chapter 6 proves it for the rule above. Under that rule the following are preserved: - the number of datapoints per (cell, interval); - the initial-state counts, since each new trajectory keeps its own beginning; - the transitions between consecutive datapoints, each end keyed by its (cell, interval); - the number of distinct objects per (cell, interval).

Other grouping rules break the transitions even at the swap cells themselves (Chapter 6, `Rmk-SwapRules`). Grouping by the first datapoint, as the Scala package does, is one of them. - *Origin–destination matrices.* Unconstrained, SwapMob preserves none (`Thm-Swap`(v)), as the thesis states [Dah19, §4.4]. A sentence of the paper's Chapter 4.2 says otherwise, and its Chapter 5 says not [SMT+20, §4].

The paper's constrained variant swaps only among trajectories that share their origin cell and their destination cell at a chosen grid g [SMT+20, §4.3]. Every new trajectory then keeps the origin of one and the destination of another, and the two share both cells at g. So the multiset of (origin, destination) pairs at g is unchanged. At every grid coarser than g and nested with it, the matrix is a pushforward of the one at g, and it is unchanged too.

The question the paper leaves open is "sequences of ODMs across spatial and temporal resolutions" [SMT+20, §4.3]. - *The trade-off.* A finer grid makes the matrix more precise but leaves fewer swaps, and so less privacy [SMT+20, §4.3]. Privacy is measured as the *adversary information gain*: the length of the longest segment without swapping, as a fraction of the whole trajectory [SMT+20, §4.2]. - *The T-drive data* [YZZ+10, YZXS11]. These are the paper's numbers for the taxi data of Beijing. Dahne's thesis cleans the data differently and reports different counts, so the two are not combined. - After cleaning: 10 280 trajectories and 16 906 423 measurements. - With $\chi = 0.001°$ (about 111 m north–south and 85 m east–west at 40°N) and $\tau = 60$ s: 641 262 co-location groups, the possible swaps. A trajectory takes part in 137 of them on average. - Without an origin–destination constraint, the adversary information gain was below 0.2 for more than 75% of the trajectories and below 0.4 for 90% [SMT+20, §4.1–§4.2].

What pavings add. The pushforward is taken on a finite key set: (cell, interval) for counts, and pairs of these for transitions. The proof of Chapter 6's `Prop-Relabel`(i)–(ii) applies to it verbatim, so the lemma below holds for any nested coarsening, not only for pavings. Pavings add three things: - a canonical nested hierarchy; - coarsening by truncating labels (`Prop-Relabel`(iii), (vi)); - a distributed pushforward (`Prop-SumOut`(v)).

Lemma (sufficiency coarsens). Let a sanitizer, pathwise, preserve both of the following on a spatial partition $\mathcal{P}$ and a time partition $\mathcal{T}$: - $N_{(\ell,t)}$, the number of datapoints in each (cell, interval); - $N_{(\ell,t)\to(\ell',t')}$, the number of ordered pairs of consecutive datapoints of one trajectory with the first in (ℓ, t) and the second in (ℓ', t').

Let $\mathcal{Q}$ be a spatial partition whose cells are unions of cells of $\mathcal{P}$, and $\mathcal{T}'$ a time partition whose intervals are unions of intervals of $\mathcal{T}$. Write π for the map $(\ell, t) \mapsto (u, s)$ to the cells that contain them. Then the sanitizer preserves both statistics on $\mathcal{Q}$ and $\mathcal{T}'$. For a regular paving, $\mathcal{Q}$ can be the cells at any depth $j \le D$, and π truncates the label (Chapter 4).

Proof. - The count of a coarse key (u, s) is the sum of $N_{(\ell,t)}$ over $\pi(\ell, t) = (u, s)$, a sum of preserved counts. - The number of transitions from (u, s) to (u', s') is the sum of $N_{(\ell,t)\to(\ell',t')}$ over $\pi(\ell, t) = (u, s)$ and $\pi(\ell', t') = (u', s')$, again a sum of preserved counts. A fine transition whose two ends lie in one coarse cell becomes the transition $(u, s) \to (u, s')$; it is a self-transition when $s = s'$. □

What the lemma does not cover: - *Jump-chain counts* record a transition only when the cell changes, as Dahne's estimate of the chain does [Dah19, §5.1]. A fine jump inside one coarse cell then disappears, and the coarse jump counts are the off-diagonal part of the pushforward: still a function of the fine counts, so still preserved. - *Distinct objects* are not sums over sub-cells: an object that meets two fine cells of one coarse cell is counted twice. Per (cell, interval) they are preserved by SwapMob directly (`Thm-Swap`(ii)–(iii)), but not over a window of several intervals (`Thm-Swap`(v)). - *Finer resolutions* are never covered. - *A coarser step* of a discrete-time chain, $S_k \to S_{k+2}$, is a different statistic, and SwapMob does not preserve it (`Thm-Swap`(v)). A decision that needs the chain at step τ' needs swaps made only at multiples of τ'.

Marks: proved here. Lean: to write, from the lemmas of `Prop-Relabel`. Simulation: the conclusion holds by construction; what is at risk is the hypothesis. A1_0-0's check, with 30 random one-dimensional trajectories at $D = 5$ and 200 trials per line, found the transitions kept under the thesis's rule in 200 of 200 trials at depths 5, 3 and 1, and lost under the other two rules in 200 of 200. It is to be rerun on the T-drive data, with a root box of side $2^{11}\chi$ aligned to the grid, $[115, 117.048] \times [39, 41.048]$, where depth $D = 22$ gives squares of side $0.001°$.

For SwapMob, the hypotheses are: - the swap cells are all the depth-D cells over the cleaned support; - SwapMob's time partition equals or refines $\mathcal{T}$; - the swaps follow `Dfn-Swap`: at the end of an interval, only the objects with a datapoint in that interval are grouped, each by its last one, as Dahne's rule does.

Under them the lemma's hypothesis holds at depth D. The datapoint counts are `Thm-Swap`(ii). For the transitions, `Thm-Swap`(ii) covers the pairs inside one interval and the pairs across a swap time with no empty interval between them. The remaining pairs straddle an empty interval, and they need one more argument (this monograph's).

Claim. Under `Dfn-Swap`, the multiset of consecutive pairs of datapoints, each end keyed by its (cell, interval), is the same before and after sanitization.

Proof. Consider one swap, at t_k. - An object in state $\perp$ is not swapped, and its pairs are unchanged. - An object i in state C has its last point before t_k in (C, I_{k-1}). Its pairs are of three kinds: - those wholly before t_k, which it keeps; - those wholly after, which move with its future; - one straddling pair, from (C, I_{k-1}) to the first point of its future, wherever that lies, possibly after several empty intervals. If the future is empty there is no such pair. - Within the group in state C, the swap permutes the futures. So it permutes the second ends of the straddling pairs, while every first end is (C, I_{k-1}). - The multiset is unchanged by one swap, and by induction by any sanitization. □

The rule matters. Suppose an object's state could come from an earlier interval. Take x last seen in cell A during I_0, and y in A during I_1, both in A at t_2, with next datapoints in B and C during I_2. A swap at t_2 turns $(A, I_0) \to (B, I_2)$ and $(A, I_1) \to (C, I_2)$ into $(A, I_0) \to (C, I_2)$ and $(A, I_1) \to (B, I_2)$. A count by cells alone would not see the change.

The lemma then carries the conclusion to every coarser resolution. For the statistics of its part (ii), `Thm-Swap`(iii)–(iv) also gives it directly.

Two cautions about grids: - The paper's grid is not a regular paving of its box. The box is $[115, 117] \times [39, 41]$ with $\chi = 0.001°$, so its side is 2000χ, not a power of two times χ [SMT+20, §4.1]. A nested non-dyadic grid such as $0.001° \subset 0.01° \subset 1°$ still satisfies the lemma, without a paving. - The Scala package computes a cell index by truncating toward zero, so for negative coordinates its cell 0 is $(-\chi, \chi)$, twice as wide as the others [lam].

The privacy is that of the depth-D swaps. So one sanitization serves every coarser resolution, at the price of the fewer swaps that the finest cells allow.

This answers one part of the question the paper ends on: sufficiency for counts and transition counts at every coarser resolution, spatial and (nested) temporal. The question reads: "A formal privacy-preserving decision-theoretic framework based on probabilistic models and statistical experiments for co-trajectories that can be integrated across multiple spatial and temporal resolutions in a distributed computational setting to handle massive mobility data needs further investigations" [SMT+20, §6]. Privacy guarantees, sequences of origin–destination matrices and a distributed algorithm are not answered here.

Distances between co-trajectories at scale [Tor23]. - *The data.* Kristoffer Torp aggregated address-based co-trajectories of the greater Stockholm area from mobile network data: about

3000 antennas, five-minute samples aggregated by the hour. There is one co-trajectory per date, for all Thursdays of 2022 and every day of February 2023. - *The method.* Each date's co-trajectory is a vector of summed device counts per (hour, antenna), with about 24×3000 entries. The dates were compared by the L_1 norm and by the weighted Jaccard index, with a MinHash approximation of the latter, to build distance matrices for clustering. The computation ran in PySpark on an on-premise cluster of ten computers. - *Caveats.* The thesis does not define its L_1 distance explicitly. Its code computes one global weighted Jaccard index rather than the per-time average of its definition (A1_0-0's reading of the thesis and code). The thesis uses neither SwapMob nor the Scala package.

Its locations are antenna addresses, without coordinates, so its keys are flat (hour, antenna) tokens, not paving cells. On that key set the L_1 distance is Chapter 6's pointwise $|a - b|$ followed by a sum. Placing the antennas in a paving by their coordinates would let it coarsen and distribute as in Chapter 10; the thesis does not do this.

The code. Dahne's Scala and Spark package for co-trajectories continues as `mob-spark` [lam]. It computes SwapMob's swap groups over Spark datasets and its swap graph in GraphX for the privacy analysis. It does not write out a swapped co-trajectory, and it groups by the first datapoint per interval (Chapter 6, `Rmk-SwapRules`). The theses of this line are collected in the repository `mob` [lam26]. Among them is Zhang's master's thesis on distributed trajectories from mobile phone network data (2022), which this monograph has not yet read.

Chapter 8

Estimation I: consistent histograms

Estimation is the inverse problem of simulation (Chapter 1). The Moore rejection sampler of Chapter 3 refines a partition where the enclosure of a density is worst; the estimator of this section refines a partition where the data are densest. Both are priority queues on the Catalan binary trees of Chapter 4, and both are threshold rules (Proposition 6.51). This section:

- defines the priority queue SEBTreeMC, which grows a histogram from the root by splitting a cell that holds the most points (§8.1);
- shows that its states are threshold pavings up to ties, and that merging upwards from a fine grid reaches the same pavings (§8.2);
- states the theorem of Lugosi and Nobel (§8.3), and derives from it the L_1 consistency of SEBTreeMC (§8.4) and of the rules that coarsen a fine grid under a budget of leaves and a condition on the shallow leaves, with a root box fixed in advance or chosen from the data, and with the depth fixed in advance or chosen from the data (§8.5);
- treats data that arrive as counts on a grid (§8.6);
- studies the queue under the uniform law, where its trees braid around the complete binary trees, and states the simulation that measures the braiding (§8.7).

What is machine-checked here, said once: Theorem 8.8 and Corollary 8.11 are (the module `VC.CorCoarsen` proves the corollary from the theorem; §15.1), but Theorem 8.9 is not, since its three conditions for SEBTreeMC's partitions are argued in the text only (§15.4). So the consistency of the bottom-up estimator rests on a certificate, and that of the top-down chain on the argument below.

Notation. - The sample $x_1, \dots, x_n$ lies in a root box $\mathbf{x}_\rho$, which is closed, with the left child half-open at each cut (Chapter 4). So every point lies in exactly one cell of each depth. Until §8.5 says otherwise, the root box is fixed before the data are seen. - For a node u, $\#(u)$ is the number of sample points in $\mathbf{x}_u$. Counts are monotone along ancestry (Chapter 5): $\#(w) \le \#(u)$ when $w \sqsubseteq u$. - The histogram of a paving s is $f_{n,s}(x) = \#(u)/(n \operatorname{vol}(\mathbf{x}_u))$ for x in the leaf $\mathbf{x}_u$ of s, and 0 outside $\mathbf{x}_\rho$ (§7.1). - μ is the law of the sample and μ_n its empirical measure, so $\mu_n(\mathbf{x}_u) = \#(u)/n$. - N_h is the set of the 2^h nodes of depth h, and s_m^{c} is the *complete tree of depth* m (the tree s_m of Lemma 4.25), whose internal nodes are the $2^m - 1$ nodes of depth less than m and whose leaves are N_m. - A *box* is a product of d intervals, each of them open, closed or half-open, possibly empty or a single point.

8.1 The priority queue SEBTreeMC

A partition into *statistically equivalent blocks* puts the same number of points into each block [Tuk47]. Anderson built multivariate procedures on such blocks [And66], and Gessaman proved

a density estimator on them consistent [Ges70]. A regular paving cannot cut where it likes, since its cuts are the fixed midpoints of Chapter 4. It can, however, keep splitting the cell that holds the most points, and so push the counts of its leaves toward one another. The randomised queue built on this rule, and its L_1 consistency, are [THS14]; the published form is the algorithm SEBTreeMC of [ST19, Algorithm 1]. A precursor, `InitSEBPQ`, started the Markov chains of §7.5 [STHL13, Alg. 2]. In MRS the queue is `AdaptiveHistogram::prioritySplit` with the node comparison `CompCount`; it draws uniformly among the leaves of equal largest count [SYH+18].

Definition 8.1 (SEBTreeMC)**.** *Fix a sample, a count limit* $\overline{\#} \geq 0$*, a leaf cap* $\overline{m} \geq 1$ *and a depth cap* $D \in \{1, 2, \dots\} \cup \{\infty\}$*. A leaf is* splittable *when its depth is less than* D*. Start from the root tree, and repeatedly split a splittable leaf of largest count, drawn uniformly among the splittable leaves of that count. Stop when no leaf is splittable, or the paving has* $\overline{m}$ *leaves, or every splittable leaf holds at most* $\overline{\#}$ *points. The chain* stops by its count criterion *when, at the stop, every splittable leaf holds at most* $\overline{\#}$ *points; this includes a stop with no splittable leaf, and a stop at which the leaf cap is met at the same time. The states* $S_0, S_1, \dots, S_T$ *form the* SEBTreeMC path*;* S_k *has* k *splits, and* $\dot{s} = S_T$ *is the* terminal state*.*

SEBTreeMC is a Markov chain on the trees of Chapter 4 that moves along the arrows of Proposition 4.4. It is not the SEB splitting model of Proposition 4.16(ii). That model draws the leaf to split at random, with probability its mass. The queue takes a leaf of largest mass, as the counts estimate it, and randomises only among ties.

Three remarks place the definition. - *The stopping rule* is that of [ST19, Algorithm 1]: the loop runs while a splittable leaf exists, the paving has fewer than $\overline{m}$ leaves, and the largest count of a splittable leaf exceeds $\overline{\#}$. The paper calls a run *successful* when it stops by its count criterion within the leaf cap [ST19, §2]. - *Splittable leaves.* A machine cannot split a leaf when it cannot represent both halves of its box (Chapter 2). A floor on the volume of a cell makes the limit explicit, and for a regular paving a volume floor is a depth cap, since a node of depth h has volume $2^{-h}\mathrm{vol}(\mathbf{x}_\rho)$. A leaf that holds no data has count $0 \leq \overline{\#}$ and is never split. $D = \infty$ is the chain without a limit on resolution, which no machine has (Chapter 12). - *The root box.* Algorithm 1 takes the root box from domain knowledge or historical data when there is some, and otherwise makes it contain the data [ST19, Algorithm 1]. §8.4 needs it fixed before the data are seen; §8.5 also treats the box hull of the data.

8.2 Threshold pavings, from above and from below

Every state of an SEBTreeMC path is a threshold paving, up to ties. To apply Proposition 6.51, give each node of depth less than D its count as priority, and each deeper node the priority -1. This priority ψ_D is monotone. As long as a splittable leaf exists, a splittable leaf of largest count is a leaf of largest ψ_D. So, by Proposition 6.51(i), the counts of the split nodes do not increase along the path, and for every $1 \leq k \leq T$

$$\{u : |u| < D,\ \#(u) > t_k\} \subseteq I(S_k) \subseteq \{u : |u| < D,\ \#(u) \geq t_k\},$$

where t_k is the k-th largest count among the nodes of depth less than D, counted with multiplicity. Counts are monotone but not strict: a child holds as many points as its parent when its sibling is empty. So ties can be nested, and Proposition 6.51(iii), the law of the ties, does not apply. §8.7 returns to it for a strict priority.

Definition 8.2 (Threshold paving)**.** *Fix a root box, a sample of* n *points in it, a count limit* $\overline{\#}$ *and a depth* D*. Let* $S_{\overline{\#}} = \{u : \#(u) > \overline{\#}\}$ *and* $S_{\overline{\#},D} = \{u \in S_{\overline{\#}} : |u| < D\}$*. The* threshold

paving at depth D, s^*_D, *is the RP whose internal nodes are exactly* $S_{\overline{\#},D}$ *(the root alone when it is empty; for* $D = \infty$*, it is defined when* $S_{\overline{\#}}$ *is finite). The sample satisfies* condition (G) at depth D *when every depth-*D *cell holds at most* $\overline{\#}$ *points; then the* threshold paving *is* $s^* = s^*_D$*. The* (G)-depth D^{G} *is the least depth at which (G) holds.*

The root box may be fixed in advance or computed from the sample, such as the box hull of the training points of Chapter 10; all that the definition and the lemma below use is that the n points lie in it.

Lemma 8.3 (Threshold paving)**.** *Let the* n *sample points lie in the root box, fixed or computed from the sample. - (i)* s^*_D *is a well-defined RP of depth at most* D*, its leaves of depth less than* D *hold at most* $\overline{\#}$ *points each, and*

$$|\mathcal{L}(s^*_D)| = 1 + |S_{\overline{\#},D}| \le 1 + D\left\lfloor \frac{n}{\overline{\#}+1} \right\rfloor.$$

- (ii) Assume (G) at D*. Then* $S_{\overline{\#},D} = S_{\overline{\#}}$*, every leaf of* s^* *holds at most* $\overline{\#}$ *points, and the leaves of* s^* *are the root when* $\#(\rho) \le \overline{\#}$*, and otherwise exactly the nodes* u *with* $\#(u) \le \overline{\#} < \#(\mathrm{parent}(u))$*. - (iii) If (G) holds at* D*, it holds at every* $D' \ge D$*, and* $s^*_{D'} = s^*$*. If* $\overline{\#} \ge 1$ *and the points are distinct,* D^{G} *is finite; it is* 0 *when* $\#(\rho) \le \overline{\#}$*, and otherwise* $1 + \max\{|u| : \#(u) > \overline{\#}\}$*, the depth of* s^**. Whenever* D^{G} *is finite and* $\overline{\#} \ge 1$*,* $2^{D^{\mathrm{G}}} \ge n/\overline{\#}$*, so* $D^{\mathrm{G}} \ge \log_2(n/\overline{\#})$*, and* $|\mathcal{L}(s^*)| \le 1 + D^{\mathrm{G}}\lfloor n/(\overline{\#}+1)\rfloor$*.*

Proof. - (i), well-defined. The parent of a node of $S_{\overline{\#},D}$ has smaller depth and at least as many points, so $S_{\overline{\#},D}$ contains the parent of each of its elements other than ρ. By Lemma 4.2(i) it is the internal-node set of a Catalan binary tree, with no internal node of depth D or more. *- (i), leaves.* A leaf of depth less than D is not in $S_{\overline{\#},D}$, so it holds at most $\overline{\#}$ points. *- (i), leaf count.* $|\mathcal{L}(s^*_D)| = 1 + |S_{\overline{\#},D}|$ by Lemma 4.2(ii). The nodes of $S_{\overline{\#},D}$ of one depth are disjoint cells, each holding at least $\overline{\#}+1$ of the n points, so there are at most $\lfloor n/(\overline{\#}+1)\rfloor$ of them at each of the depths $0, \dots, D-1$. *- (ii).* A node of depth at least D lies in a depth-D cell, so under (G) at D it holds at most $\overline{\#}$ points; hence $S_{\overline{\#}} = S_{\overline{\#},D}$, and the leaves of depth D hold at most $\overline{\#}$ points too. A leaf other than ρ is a child of an internal node that is not itself internal (Lemma 4.2(i)), which is the stated description. *- (iii).* A depth-D' cell lies in a depth-D cell, so (G) passes to deeper depths, and (ii) at D and at D' gives $s^*_{D'} = s^*_D$. By monotonicity, (G) holds at D exactly when no node of depth D or more lies in $S_{\overline{\#}}$, that is, when $D > \max\{|u| : u \in S_{\overline{\#}}\}$; and distinct points with $\overline{\#} \ge 1$ are separated at some finite depth, after which no cell holds more than one of them. The $2^{D^{\mathrm{G}}}$ cells of depth D^{G} partition the root box, which holds the n points, so one of them holds at least $n2^{-D^{\mathrm{G}}}$ points, and (G) gives $n2^{-D^{\mathrm{G}}} \le \overline{\#}$. The leaf count is (i) at $D = D^{\mathrm{G}}$. □

Marks. Proved above. [LEAN: (i) and (ii) certified, `IasCert.L1MDE.ThresholdPaving`, built at `c06220d95` (Chapter 15): under (G) `internal_parent`, `internal_depth_lt`, `leaf_mass_le`, `isLeaf_root_iff`, `isLeaf_cons_iff`, `leaf_cover`, `card_internalSet_le`, `card_leavesOf`; without (G), for s^*_D, `leafD_cover`, `card_leafSet_le`, `shallow_leaves_mass_le`. (iii) certified from the count map at a finest depth M with (G) at M, which stands for "D^{G} is finite" when M is the deepest resolved depth, `ProveDog.ProveDog.DecisionTheory.L1MDE.GDepthBound`, built at `b5b02097d`: `lem_threshold_iii`; the value $1 + \max\{|u| : \#(u) > \overline{\#}\}$, `gDepth_eq_one_add_max`; $n \le 2^{D^{\mathrm{G}}}\overline{\#}$, $n/\overline{\#} \le 2^{D^{\mathrm{G}}}$ and $\log_2(n/\overline{\#}) \le D^{\mathrm{G}}$, `total_le_two_pow_mul`, `total_le_two_pow_gDepth_mul`, `div_cap_le_two_pow_gDepth`, `logb_div_cap_le_gDepth`; the leaf count, `card_leafSet_le_gDepth`; and the count map's form of "distinct points give a finite D^{G}", every depth-M cell holding at most one point with $\overline{\#} \ge 1$ giving (G) at M, `condG_of_cells_le_one`; the first sentence and the least depth are `ProveDog.L1MDE.Depth`'s `condG_mono`,

`condG_iff_gDepth_le` and `leafSet_coarsen`. To write: the finiteness of D^{G} for distinct points of $\mathbb{R}^d$ (proposed module `GDepthFinite`, beside `GDepthBound`, on the diameters of `Aspect`'s (iv)).] [SIM: s^*_D and D^{G} computed from the counts of random samples, in random fixed root boxes and in their box hulls, against (i)–(iii); to run, L1MDE-PROOF-SIMULATION.]

Lemma 8.4 (SEBTreeMC reaches the threshold paving)**.** *SEBTreeMC stops by its count criterion exactly when $\overline{m} > |S_{\overline{\#},D}|$, and its terminal state is then s^*_D, whatever the tie-breaking. In particular this happens when $\overline{m} > D\lfloor n/(\overline{\#}+1)\rfloor$, and under (G) at D the terminal state is then the threshold paving s^*.*

Proof. - *The split nodes lie in $S_{\overline{\#},D}$.* A leaf is split only while the largest count of a splittable leaf exceeds $\overline{\#}$, and the leaf split has that count and depth less than D. - *No early stop.* While the set of split nodes is a proper subset of $S_{\overline{\#},D}$, take an unsplit $u \in S_{\overline{\#},D}$ of least depth. Its parent is split, or $u = \rho$, so u is a current leaf; it is splittable and holds more than $\overline{\#}$ points. So the count criterion fails, and a splittable leaf exists. - *Exactly when.* So the chain stops by its count criterion only when its split set is $S_{\overline{\#},D}$, after $|S_{\overline{\#},D}|$ splits. It then does stop, since every splittable leaf lies outside $S_{\overline{\#},D}$. If $\overline{m} > |S_{\overline{\#},D}|$, the leaf cap cannot stop it earlier, since until then it has at most $|S_{\overline{\#},D}| < \overline{m}$ leaves. If $\overline{m} \le |S_{\overline{\#},D}|$, the leaf cap stops it after $\overline{m} - 1$ splits, while its split set is still a proper subset. - *Two constructions.* The case under (G) and the general case use different constructions of the chain. Under (G) at D, while the count criterion fails, a leaf of largest count among all the leaves holds more than $\overline{\#}$ points, so it has depth less than D and is splittable: the chain splits a leaf of largest count among all the leaves. Without (G), a leaf of depth D may hold the largest count, and that construction would stop early; the general case uses the priority ψ_D of §8.2, under which, while a splittable leaf exists, the leaves of largest priority are the splittable leaves of largest count, and the steps above hold for it with any tie-breaking. - *In particular.* Lemma 8.3. □

Marks. Proved above. [LEAN: the case under (G) with $\overline{m} > D\lfloor n/(\overline{\#}+1)\rfloor$ certified (`IasCert.L1MDE.Exact`: `reach_inv`, `seb_step_exists`, `seb_no_step_at_threshold`, `lem_seb`, built at `c06220d95`); the equivalence and the case without (G) certified for a finite depth cap D, `ProveDog.ProveDog.DecisionTheory.L1MDE.SEB`, built at `cae430874`: the proof's steps, `countCrit_iff` (no early stop), `maxLeaf_mem_internalSet` and `sebRun_inv` (the split nodes lie in $S_{\overline{\#},D}$), `sebRun_card_le`, `seb_stopped` (exactly when); the chain as the splitting queue of ψ_D on `PQThreshold`'s runs, with `isMaxLeaf_psiD_iff` (the sentence of §8.2, both ways); the lemma without (G), `lem_seb_noG`, `lem_seb_noG_bound`, and under (G) (on the queue of ψ_D), `lem_seb_noG_threshold`, `lem_seb_noG_threshold_bound`; on `Exact`'s chain, `reach_countCrit` and `reach_lt_mbar`, and under (G) `sebStep_iff_psiD`, `seb_stop_iff` and `lem_seb_iff` (step by step it is the chain of Definition 8.1, and the equivalence holds at the exact threshold $|S_{\overline{\#},D}|$). `Exact`'s chain is SEBTreeMC under (G) only: without (G), an over-full leaf of depth D and largest count stops it early, so the case without (G) is stated on the queue of ψ_D (the proof's two constructions). $D = \infty$, the law of the ties and the construction of a run are not in the module.] [SIM: checked exactly over every positive-probability run, for 400 random count fields at $D \le 4$, $\overline{\#} \le 4$ and every $\overline{m} \le 2^D + 1$, 3430 cases, no mismatch (the review's `rev7_check.py lemseb`); MRS `prioritySplit` with `CompCount`, and a re-implementation on labels, against s^*_D on random samples with seeded ties, to run, L1MDE-PROOF-SIMULATION.]

The implementation builds the same paving from below (Chapter 10). Every point is labelled by its cell at the finest depth D, and cells are merged upwards [Gra22]. The merge queue below is the dense form of that construction, on the complete tree of depth D. Under (G), Chapter 11 computes the nonempty leaves of the same paving from the occupied cells alone, by a walk over the sorted labels (Lemma 11.4).

Lemma 8.5 (The merge queue reaches the threshold paving)**.** *Fix a sample, a depth $D \geq 1$ and a count limit $\overline{\#}$. Start from s_D^{c}, and repeatedly merge a cherry of least count, ties broken in any way, while that count is at most $\overline{\#}$. The queue stops at s_D^*.*

Proof. Let $N = \{u : |u| < D,\ \#(u) \leq \overline{\#}\}$, the internal nodes of s_D^{c} outside $S_{\overline{\#},D}$. A merged cherry holds at most $\overline{\#}$ points, so every merged node lies in N. Suppose the queue stops while a node of N is unmerged, and let u be such a node of greatest depth. Every internal node of s_D^{c} strictly below u holds at most $\#(u) \leq \overline{\#}$ points, so lies in N; it is deeper than u, so it has been merged. Then u is a cherry holding at most $\overline{\#}$ points, and the queue has not stopped. So the queue stops exactly when all of N has been merged, at the paving whose internal nodes are $S_{\overline{\#},D}$. □

Marks. Proved above. [LEAN: to write — `BottomUp`, on `PQThreshold` (ii) and `Threshold-Paving`.] [SIM: every tie path, 300 random count fields at $D \leq 4$, no mismatch (the review's `rev7_check.py bottomup`); to rerun at larger depths, L1MDE-PROOF-SIMULATION.]

The two directions agree beyond their stopping points too. Run without the count criterion, the queue from the root and the merge queue from the complete tree reach the same trees with positive probability, and the merge paths are split paths read backwards.

Proposition 8.6 (Splitting down and merging up)**.** *Fix a sample and a depth $D \geq 1$, let $K = 2^D - 1$, and for $1 \leq k \leq K$ let t_k be the k-th largest count among the K nodes of depth less than D, counted with multiplicity. Let $\mathcal{I}_k$ be the family of the sets I of k nodes of depth less than D that contain the parent of each of their elements other than ρ and satisfy*

$$\{u : |u| < D,\ \#(u) > t_k\} \subseteq I \subseteq \{u : |u| < D,\ \#(u) \geq t_k\}.$$

- (i) The internal-node sets with k splits that SEBTreeMC with depth cap D, run without its other stopping rules, reaches with positive probability are exactly the elements of $\mathcal{I}_k$. So are those that the merge queue from s_D^{c}, which merges a cherry of least count drawn uniformly among ties, reaches. The laws of the two queues on $\mathcal{I}_k$ differ in general (Example 8.7). - (ii) If the K counts are distinct, $\mathcal{I}_k$ has one element, the k nodes of largest count, and the merge path from s_D^{c} is the SEBTreeMC path read backwards. - (iii) Every merge path from s_D^{c} that has positive probability, read backwards, is a path that SEBTreeMC with depth cap D, run without its other stopping rules, follows with positive probability.

Proof. - Splitting reaches only $\mathcal{I}_k$. Proposition 6.51(i) with ψ_D, as above; an internal-node set contains the parent of each of its elements other than ρ (Lemma 4.2(i)). *- Splitting reaches all of $\mathcal{I}_k$.* Let $I \in \mathcal{I}_k$, and split its nodes in order of decreasing count, shallower first among equal counts. At the turn of u, its parent lies in I, is shallower and holds at least as many points, so it has been split, and u is a leaf. A current leaf ℓ with $\#(\ell) > \#(u) \geq t_k$ would lie in I and would have been split before u. So u is a splittable leaf of largest count, and each step has positive probability. *- Merging reaches only $\mathcal{I}_k$.* For $k = K$ there is nothing to prove. Otherwise the queue prunes $K - k$ cherries. The $(K-k)$-th smallest count among the K nodes is $t' = t_{k+1}$. By Proposition 6.51(ii), the pruned set P contains every internal node below each of its elements, and $\{\# < t'\} \subseteq P \subseteq \{\# \leq t'\}$ among the nodes of depth less than D. The remaining set R has k elements and contains the parent of each of its elements other than ρ, since an ancestor in P would put its descendants in P; and $\{\# > t'\} \subseteq R \subseteq \{\# \geq t'\}$. If $t_k = t'$, this is the condition of $\mathcal{I}_k$. If $t_k > t'$, exactly k nodes hold at least t_k points, all of them more than t'; so R is this set, which is then the only element of $\mathcal{I}_k$. *- Merging reaches all of $\mathcal{I}_k$.* For $I \in \mathcal{I}_k$, its complement P among the internal nodes of s_D^{c} has $K - k$ elements and contains every internal node below each of its elements. From $I \subseteq \{\# \geq t_k\}$ and $t' \leq t_k$, $P \supseteq \{\# < t'\}$. If $t_k = t'$, $I \supseteq \{\# > t'\}$ gives $P \subseteq \{\# \leq t'\}$; if $t_k > t'$, I is the set of the k nodes holding more than t' points, and again

$P \subseteq \{\# \le t'\}$. By Proposition 6.51(iv), P is a possible pruned set. - *(ii).* With distinct counts, $t_k > t_{k+1}$ for every k, so $\mathcal{I}_k$ has one element, and both queues are deterministic. - *(iii).* Let the merge path, read backwards, be $R_0 \subset R_1 \subset \cdots \subset R_K$, with $R_{k+1} = R_k \cup \{u\}$; by the third step, $R_j \in \mathcal{I}_j$ for every j. The parent of u lies in R_{k+1} and is not u, so it lies in R_k (or $u = \rho$ and $k = 0$), and u is a leaf of the tree with internal nodes R_k; it has depth less than D, so it is splittable. A splittable leaf ℓ of that tree with $\#(\ell) > \#(u)$ would satisfy $\#(\ell) > t_{k+1}$, since $u \in R_{k+1} \subseteq \{\# \ge t_{k+1}\}$, so $\ell \in R_{k+1}$; as $\ell \ne u$, $\ell \in R_k$, which a leaf is not. So u is a splittable leaf of largest count, and the step has positive probability. □

Marks. Proved above. [LEAN: to write — `TwoWays`, on `PQThreshold` (i), (ii), (iv).] [SIM: the families reached by each queue compared with $\mathcal{I}_k$ computed from its definition, 2888 (field, k) cases at $D = 2, 3, 4$ with heavy ties, no mismatch; for (iii), 400 random merge paths at $D = 3, 4$, each reversed and tested step by step as SEBTreeMC steps, no failure (the review's `rev7_check.py twoways`); to rerun at larger depths, L1MDE-PROOF-SIMULATION.]

Example 8.7 (Threshold pavings)**.** Take the six points $0.5, 1.5, 2.5, 6.2, 6.7, 7.9$ of Example 6.20, in the root box $[0, 8]$, with the depth-3 count map $\{8{:}1, 9{:}1, 10{:}1, 14{:}2, 15{:}1\}$ in heap codes. The counts at the shallower nodes are $\#(\rho) = 6$, $\#(\rho\mathsf{L}) = \#(\rho\mathsf{R}) = \#(\rho\mathsf{RR}) = 3$, $\#(\rho\mathsf{LL}) = 2$, $\#(\rho\mathsf{LR}) = 1$ and $\#(\rho\mathsf{RL}) = 0$. Let $\overline{\#} = 2$ and $D = 3$. - *The threshold paving.* $S_{2,3} = \{\rho, \rho\mathsf{L}, \rho\mathsf{R}, \rho\mathsf{RR}\}$, and (G) holds, so s^* has the leaves $\rho\mathsf{LL}, \rho\mathsf{LR}, \rho\mathsf{RL}, \rho\mathsf{RRL}, \rho\mathsf{RRR}$, with leaf-depth string 22233. Its five leaves are within the bound $1 + 3\lfloor 6/3 \rfloor = 7$ of Lemma 8.3, and $D^{\mathrm{G}} = 3 \ge \log_2(6/2)$. - *From above.* SEBTreeMC with $\overline{m} \ge 5$ splits ρ, then one of the tied $\rho\mathsf{L}, \rho\mathsf{R}$. After $\rho\mathsf{L}$ it must split $\rho\mathsf{R}$, then $\rho\mathsf{RR}$. After $\rho\mathsf{R}$, the leaves $\rho\mathsf{L}$ and $\rho\mathsf{RR}$ tie at 3, and either order follows. So it has three paths, with probabilities $\frac{1}{2}, \frac{1}{4}, \frac{1}{4}$, and all end at s^* (Lemma 8.4). They are all three split paths to 22233, whose Catalan coefficient is $B = 4!/(4 \cdot 2 \cdot 1 \cdot 1) = 3$ (Lemma 4.9), but the queue does not weight them equally. With $\overline{m} = 4 \le |S_{2,3}|$, the leaf cap stops it after three splits, short of the count criterion. - *From below.* The merge queue from s_3^{c} merges $\rho\mathsf{RL}$ (count 0), $\rho\mathsf{LR}$ (1) and $\rho\mathsf{LL}$ (2). Its least cherries, $\rho\mathsf{L}$ and $\rho\mathsf{RR}$, then hold 3 points, more than $\overline{\#}$, and it stops at s^* (Lemma 8.5). - *Both ways, two laws.* At $k = 3$, $t_3 = 3$ and $\mathcal{I}_3 = \{\{\rho, \rho\mathsf{L}, \rho\mathsf{R}\}, \{\rho, \rho\mathsf{R}, \rho\mathsf{RR}\}\}$. Splitting reaches the first with probability $\frac{3}{4}$, by two of its paths, and the second with probability $\frac{1}{4}$, by the third. The merge queue, continued past s^*, breaks a tie between $\rho\mathsf{L}$ and $\rho\mathsf{RR}$ and reaches each with probability $\frac{1}{2}$ (Proposition 8.6).

The backtrack of Chapters 9 to 10 merges a cherry of least count from the merged histogram toward the root. It continues the merge queue of Lemma 8.5 below the count limit, and by Proposition 8.6(iii) its paths are SEBTreeMC paths read backwards. Lemma 11.9 (Chapter 11) proves this for the backtrack as coded, under the leaf cap. So the candidates of the selection of Chapter 9 are the states of one queue, read in either direction.

8.3 The theorem of Lugosi and Nobel

Consistency of histograms on partitions chosen from the data rests on one theorem. Lugosi and Nobel gave three conditions under which such a histogram converges to the density in L_1 with probability one [LN96].

Theorem 8.8 (Lugosi and Nobel [LN96])**.** *Let $X_1, X_2, \ldots$ be i.i.d. in $\mathbb{R}^d$ with distribution μ having a density f, and let μ_n be the empirical distribution of $X_1, \ldots, X_n$. For each n let $\pi_n = \pi_n(X_1, \ldots, X_n)$ be a partition of $\mathbb{R}^d$ chosen by a fixed rule, let $\mathcal{A}_n$ be the family of all partitions the rule can produce, let $m(\mathcal{A}_n)$ be the largest number of cells of a member of $\mathcal{A}_n$, and let $\Delta_n^*(\mathcal{A}_n)$ be the largest number of distinct partitions of an n-point set induced by members of $\mathcal{A}_n$. Write $\pi_n[x]$ for the cell containing x. If (a) $m(\mathcal{A}_n)/n \to 0$, (b) $\log \Delta_n^*(\mathcal{A}_n)/n \to 0$, and (c) $\mu\{x : \mathrm{diam}(\pi_n[x]) > \gamma\} \to 0$ with probability one for every $\gamma > 0$, then the histogram $f_n(x) =$*

$\mu_n(\pi_n[x])/\mathrm{vol}(\pi_n[x])$, *with* $f_n(x) = 0$ *when* $\pi_n[x]$ *has infinite volume, satisfies* $\int |f - f_n| \to 0$ *with probability one.*

The growth function Δ_n^* keeps the notation of [LN96]; it is unrelated to the deviations Δ of Chapter 9. Condition (c) is condition (c) of [LN96, Theorem 1] as stated there. Their Theorem 2, for classification, weakens it to an infimum over the sets S with $\mu(S) \geq 1 - \delta$ [LN96, Theorem 2 and the Remark after its proof]. The convention $f_n = 0$ on cells of infinite volume is theirs [LN96, eq. (8)]; Chapter 8 uses it for the cell $\mathbb{R}^d \setminus \mathbf{x}_\rho$. For an uncountable family $\mathcal{A}_n$, they assume that the functions assigning points to cells are pointwise limits of a countable subfamily, so that the supremum below is measurable [LN96, Remark before Lemma 1]. The families of §8.4 are finite, and Lemma 8.12 treats the data-chosen root boxes of §8.5 without this condition.

How the proof goes. Let $\bar{f}_n(x) = \mu(\pi_n[x])/\mathrm{vol}(\pi_n[x])$, and 0 on cells of infinite volume, be the histogram of μ itself on π_n. - *The estimation error.* $\int |f_n - \bar{f}_n|$ is at most $\sum_{A \in \pi_n} |\mu_n(A) - \mu(A)|$, and so at most the deviation $\sup_{\pi \in \mathcal{A}_n} \sum_{A \in \pi} |\mu_n(A) - \mu(A)|$. - A Vapnik–Chervonenkis inequality for families of partitions bounds the probability that this deviation exceeds ϵ by $4\Delta_{2n}^*(\mathcal{A}_n)\, 2^{m(\mathcal{A}_n)} e^{-n\epsilon^2/32}$ [LN96, Lemma 1]. - Its proof passes through the class $\mathcal{B}(\mathcal{A})$ of unions of cells of members of $\mathcal{A}$, bounding the deviation by twice the supremum over $\mathcal{B}(\mathcal{A})$ [LN96, eq. (6)], with shatter coefficient $S_n(\mathcal{B}(\mathcal{A})) \leq 2^{m(\mathcal{A})} \Delta_n^*(\mathcal{A})$. - For a class of sets, the trace on $2n$ points is determined by the traces on the two halves, so $S_{2n}(\mathcal{B}) \leq S_n(\mathcal{B})^2 \leq 4^{m(\mathcal{A})} \Delta_n^*(\mathcal{A})^2$. With the Vapnik–Chervonenkis inequality for sets [LN96, eq. (4)], this gives $\mathbb{P}\{\sup_{\pi \in \mathcal{A}_n} \sum_{A \in \pi} |\mu_n(A) - \mu(A)| > \epsilon\} \leq 4 \cdot 4^{m(\mathcal{A}_n)} \Delta_n^*(\mathcal{A}_n)^2 e^{-n\epsilon^2/32}$, which is summable under (a) and (b). The Borel–Cantelli lemma then makes the deviation tend to 0 with probability one [LN96, Corollary 1]. - *The approximation error* $\int |f - \bar{f}_n|$ tends to 0 under (c). The proof bounds it for whatever partition is used, given (c) [LN96, proof of Theorem 1, (10)–(15)].

Sandstedt's thesis, on the distributed form of this estimator, restates Lemma 1, Corollary 1 and Theorem 1 [San23, §1.4], and the consistency theorem of [ST19] [San23, §2.3].

Ties broken at random. This extension is not in [LN96], whose theorem is for a fixed rule $\pi_n(X_1, \ldots, X_n)$. Their proof carries over to a partition that also depends on randomness independent of the sample, such as the draws that break the ties of a queue, provided it lies in $\mathcal{A}_n$ and satisfies (c): the estimation term is bounded by the supremum over $\mathcal{A}_n$, and the approximation term is bounded pathwise for whatever partition is used. The machine-checked form states the theorem in this way.

Marks. Cited, with the derivation of Corollary 1 above. [LEAN: certified, `ProveDog.DecisionTheory.VC.HistogramConsistency` (`lugosi_nobel_theorem1`), on `VC.LugosiNobel`, `Summability`, `HistogramDev`, `HistogramOp`, `HistogramApprox` and `HistogramDensity`, built at `647131dc4` (Chapter 15). It is stated for a countable family of partitions, each a finite family of measurable, pairwise disjoint cells of finite reference measure (not a partition of $\mathbb{R}^d$), with the partition any function of the outcome. Lemma 1 and Corollary 1 of [LN96] are proved on the way; Corollary 1 (`lugosi_nobel_corollary1_ab`) bounds the traces of labelled cells, which the traces on the two halves determine.] [SIM: not needed; each use below carries its own check.]

8.4 SEBTreeMC is consistent

Theorem 8.9 (L_1-consistency of SEBTreeMC [ST19])**.** *Let* $X_1, X_2, \ldots$ *be i.i.d. with a density* f *on a root box* $\mathbf{x}_\rho$ *fixed before the data are seen. For each* n*, run SEBTreeMC on* $X_1, \ldots, X_n$ *with count limit* $\overline{\#}_n$*, leaf cap* $\overline{m}_n$ *and depth cap* $D_n \leq \infty$*, with any tie-breaking, and let* $\dot{s}_n$ *be its terminal state. If, as* $n \to \infty$*,* $\overline{\#}_n/n \to 0$*,* $\overline{m}_n \to \infty$*,* $\overline{m}_n/n \to 0$ *and* $D_n \to \infty$*, then* $\int |f - f_{n,\dot{s}_n}| \to 0$ *with probability one, whichever rule stops the chain.*

Proof. The proof checks the three conditions of Theorem 8.8. Let $\mathcal{A}_n$ be the family of the partitions of $\mathbb{R}^d$ formed by the leaves of an RP of $\mathbf{x}_\rho$ with at most $\overline{m}_n$ leaves, together with the cell $\mathbb{R}^d \setminus \mathbf{x}_\rho$; that cell has μ-measure zero, and the histogram is zero on it. Every terminal state lies in $\mathcal{A}_n$, whatever the tie-breaking, and ties broken at random are allowed (§8.3). - *(a)* $m(\mathcal{A}_n) \le \overline{m}_n + 1$, so $m(\mathcal{A}_n)/n \to 0$. - *(b)* Each Catalan binary tree gives one RP of the fixed root box (Definition 4.22), so the RPs with k splits give at most $C_k \le 4^k$ partitions (Proposition 4.33(ii), Proposition 4.5). So $|\mathcal{A}_n| \le \sum_{k<\overline{m}_n} C_k < 4^{\overline{m}_n}$. Since $\Delta_n^*(\mathcal{A}_n) \le |\mathcal{A}_n|$, $\log \Delta_n^*(\mathcal{A}_n)/n < \overline{m}_n \log 4/n \to 0$. - *(c), the shallow leaves.* Fix $\gamma > 0$. By Lemma 4.23(iii), the cells of depth k have diameter at most diam_k, which tends to 0; fix k with $\mathrm{diam}_k < \gamma$. Cells along a path are nested, so every leaf of depth at least k has diameter less than γ. A point of $\mathbf{x}_\rho$ whose leaf is wider than γ therefore lies in U_n, the union of the leaves of $\dot{s}_n$ of depth less than k. - *(c), how many points a shallow leaf holds.* For $\overline{m}_n \ge 2$, let c_n be the $(\overline{m}_n - 1)$-th largest count among the nodes of depth less than D_n, counted with multiplicity. Take n with $D_n > k$ and $\overline{m}_n \ge 2$. A leaf of $\dot{s}_n$ of depth less than k is splittable. - If the chain stopped by its count criterion, the leaf holds at most $\overline{\#}_n$ points. - Otherwise the chain stopped by its leaf cap, after $\overline{m}_n - 1$ splits: it cannot stop for want of a splittable leaf while this one exists. By Proposition 6.51(i) with ψ_{D_n}, the leaf, which is not internal, holds at most the $(\overline{m}_n - 1)$-th largest count among the nodes of depth less than D_n, that is, at most c_n points.

These leaves are pairwise disjoint nodes of depth less than k, so there are at most 2^{k-1} of them, and $\mu_n(U_n) \le 2^{k-1} \max(\overline{\#}_n, c_n)/n$. - *(c),* $c_n/n \to 0$ *with probability one.* Given $\delta > 0$, choose H with $\max_{|u|=H} \mu(\mathbf{x}_u) < \delta/2$; this is possible by the absolute continuity of the integral: since f is integrable there is $\eta > 0$ with $\int_A f < \delta/2$ for every A of volume less than η, and the cells of depth H have volume $2^{-H}\mathrm{vol}(\mathbf{x}_\rho) < \eta$ once H is large. By the strong law on the 2^H cells of depth H, with probability one every node of depth at least H eventually holds fewer than δn points. The nodes holding δn points or more then have depth less than H, and at most $1/\delta$ of them are disjoint at each depth, so there are at most H/δ of them. Once $\overline{m}_n - 1 > H/\delta$, $c_n < \delta n$. Taking $\delta = 1/j$ for $j = 1, 2, \ldots$ gives $c_n/n \to 0$ with probability one, and so $\mu_n(U_n) \to 0$. - *(c), conclusion.* U_n is a disjoint union of cells from the finite class $\mathcal{C}_k$ of the $2^k - 1$ nodes of depth less than k, so

$$\mu(U_n) \le \mu_n(U_n) + \sum_{C \in \mathcal{C}_k} |\mu(C) - \mu_n(C)|,$$

and the sum tends to 0 with probability one by the strong law of large numbers. Hence $\mu\{x : \mathrm{diam}(\pi_n[x]) > \gamma\} \le \mu(U_n) + \mu(\mathbb{R}^d \setminus \mathbf{x}_\rho) = \mu(U_n) \to 0$ with probability one. □

Marks. Proved above, from Theorem 8.8. [LEAN: to write — the three conditions for the partitions of SEBTreeMC, from `VC.Coarsen`, `Exact` and `PQThreshold` (i); the theorem is not machine-checked.] [SIM: the bound $\max(\overline{\#}, c)$ on the splittable leaves at the stop, checked over every positive-probability run for 400 random count fields at $D \le 4$, every $\overline{\#} \le 3$ and every $\overline{m} \le 2^D + 1$, 18521 cases, no violation (scratchpad `r2_consistency_check.py`, made while revising); the L_1 error of the terminal histogram along growing n, for densities whose integrals over cells are exact (uniform densities, and mixtures given as mapped pavings as in §9.6), with the stopping cause of each run recorded, to run, L1MDE-PROOF-SIMULATION.]

On the hypotheses. - *The published conditions.* [ST19, Theorem 1] states the theorem without a depth cap, under $\overline{\#} \to \infty$, $\overline{\#}/n \to 0$, $\overline{m} \ge n/\overline{\#}$ and $\overline{m}/n \to 0$. These imply the hypotheses above, since $\overline{m}_n \ge n/\overline{\#}_n \to \infty$; the condition $\overline{\#}_n \to \infty$ is not needed. - *The two stops.* By Lemma 8.4, the chain stops by its count criterion exactly when $\overline{m}_n > |S_{\overline{\#}_n, D_n}|$, and its terminal state is then $s^*_{D_n}$, whatever the ties. When the leaf cap stops it, the terminal state depends on the ties; the proof covers every tie-breaking. - *Why the leaf cap matters.* The proof in [ST19] bounds the mass of every leaf by $\overline{\#}/n$ at its step (c), which holds when the run stops by its count criterion. The condition $\overline{m} \ge n/\overline{\#}$ does not ensure such a stop. For uniform data with $\overline{m} = n/\overline{\#}$ and $\overline{\#}$ large, the nodes of depth less than $\log_2(n/\overline{\#})$ have expected counts of at least $2\overline{\#}$ and

number about $n/\overline{\#}$, and about half the nodes of the next depth, whose expected count is $\overline{\#}$, hold more than $\overline{\#}$ points as well. So about $\frac{3}{2}n/\overline{\#}$ nodes hold more than $\overline{\#}$ points, more than the leaf cap allows, and the leaf cap stops the run. The bound $\max(\overline{\#}_n, c_n)$ above covers every stop. - *The root box.* The count of partitions by Catalan numbers in (b) needs a root box fixed before the data. §8.5 treats the box hull of the data; a version of this theorem with the hull is not written out here. - *A density on all of* $\mathbb{R}^d$. The coordinatewise map $t(x) = x/(1+|x|)$ carries it into the open cube $(-1,1)^d$ and leaves the L_1 error unchanged (Chapter 4, after Lemma 4.25); the root box is then $[-1,1]^d$.

8.5 Consistency by coarsening

Theorem 8.9 is about a chain that refines from the root. The implementation of Chapter 10 runs the other way: every point becomes a sparse leaf at the finest depth, and cells are merged upwards [Gra22, San23]. This subsection proves consistency in that direction directly from Theorem 8.8, for the rules that coarsen a fine grid under a budget of leaves and a condition on the shallow leaves, and without condition (G). It then gives the versions that Chapter 10's pipeline needs: the root box chosen as the box hull of the training points, and the finest depth chosen by (G).

A *merge rule at depth* D maps the finest count table of a sample, its depth-D count map $\mathrm{countOf}_D$ (Definition 6.17), to an RP of $\mathbf{x}_\rho$ of depth at most D. Every such RP coarsens the depth-D partition: each of its leaves is the union of the depth-D cells below it. Merge rules at depth D in this monograph: - the merge queue of Lemma 8.5; - SEBTreeMC with depth cap D, which breaks its ties at random; - the paving whose nonempty leaves the walk of Chapter 11 returns (Lemma 11.4); without (G), Lemma 11.7 must show that it is an RP of depth at most D.

They read only the counts of the nodes of depth at most D, which are sums over the table (Definition 6.17, the count $\#_c$).

Theorem 8.10 (Consistency by coarsening)**.** *Let $X_1, X_2, \ldots$ be i.i.d. with a density f on a root box $\mathbf{x}_\rho$ fixed before the data are seen. For each n let π_n be the RP returned by a merge rule at depth D_n applied to the depth-D_n count table of $X_1, \ldots, X_n$. Suppose that - (M1) there are numbers $\overline{m}_n$ with $\overline{m}_n/n \to 0$ such that every RP the rule can return has at most $\overline{m}_n$ leaves, and - (M2) for every fixed k, the empirical mass $\mu_n\big(\bigcup\{\mathbf{x}_u : u \in \mathcal{L}(\pi_n),\ |u| < k\}\big)$ of the leaves of depth less than k tends to 0 with probability one.*

Then $\int |f - f_{n,\pi_n}| \to 0$ with probability one.

Proof. The proof checks the three conditions of Theorem 8.8. The partition of $\mathbb{R}^d$ is formed by the leaves of π_n together with $\mathbb{R}^d \setminus \mathbf{x}_\rho$, a cell of μ-measure zero on which the histogram is zero; $\mathcal{A}_n$ is the family of these partitions for the RPs that the rule can return. - *(a)* $m(\mathcal{A}_n) \le \overline{m}_n + 1$, so $m(\mathcal{A}_n)/n \to 0$ by (M1). - *(b)* The bisection rule is fixed, so each Catalan binary tree gives one RP of $\mathbf{x}_\rho$ (Definition 4.22), and the RPs with k splits give at most $C_k \le 4^k$ partitions (Proposition 4.33(ii), Proposition 4.5). Hence $|\mathcal{A}_n| \le \sum_{k<\overline{m}_n} C_k < 4^{\overline{m}_n}$, $\Delta_n^*(\mathcal{A}_n) \le |\mathcal{A}_n|$, and $\log \Delta_n^*(\mathcal{A}_n)/n < \overline{m}_n \log 4/n \to 0$. - *(c)* By Lemma 4.23(iii), a cell of depth k has diameter at most $\mathrm{diam}_k = \sqrt{d}\, a\, (2^{-k}\mathrm{vol}(\mathbf{x}_\rho))^{1/d}$, where $a = \max(2, a_\rho)$ and a_ρ is the aspect ratio of the root box; $\mathrm{diam}_k \to 0$. In the cube $[-1,1]^d$, Lemma 4.23(iv) gives the sharper bound $2\sqrt{d}\, 2^{-\lfloor k/d \rfloor}$. Given $\gamma > 0$, choose k with $\mathrm{diam}_k < \gamma$. A point of $\mathbf{x}_\rho$ whose cell is wider than γ then lies in $U_n = \bigcup\{\mathbf{x}_u : u \in \mathcal{L}(\pi_n),\ |u| < k\}$. U_n is a disjoint union of cells from the finite class $\mathcal{C}_k$ of the

$2^k - 1$ nodes of depth less than k, so

$$\mu(U_n) \le \mu_n(U_n) + \sum_{C \in \mathcal{C}_k} |\mu(C) - \mu_n(C)|.$$

The sum tends to 0 with probability one by the strong law of large numbers, and $\mu_n(U_n) \to 0$ by (M2). □

Condition (M2) forces $D_n \to \infty$: if $D_n < k$, every leaf has depth less than k, and the leaves carry all the mass.

The proof uses of the rule only that its output is an RP of the fixed root box with at most $\overline{m}_n$ leaves, and (M2). It does not use the depth bound, and it allows ties broken at random (§8.3). The proof of Theorem 8.9 is the same argument, with (M2) supplied by the bound at the stop. Part (b) counts Catalan binary trees, so it holds for any fixed binary scheme; part (c) uses the diameter bound of regular bisection, which any scheme with shrinking cells replaces (Remark 4.31).

Marks. Proved above, from Theorem 8.8. [LEAN: (b), the count for a fixed number of internal nodes, certified (`IasCert.L1MDE.PavingCount`: `card_family_le_catalan`, `catalan_le_four_pow`, `card_family_le_four_pow`, built at `c06220d95`); (c) in its abstract form, for a hierarchy whose deep cells are small, and for the cells of the cube $[-1,1)^d$, certified (`ProveDog.DecisionTheory.VC.Coarsen`: `coarse_tendsto_zero`, `consistency_of_coarsening`; `VC.EmpiricalSLLN`; `VC.RegularPaving`: `small_of_deep`; built at `647131dc4`); the diameter bound for a general root box (Lemma 4.23(iii)) not yet authored.] [SIM: not needed for the theorem; its corollaries carry the checks.]

Corollary 8.11 (The threshold merge is consistent)**.** *Let $s_n^* = s_{D_n}^*$ be the RP whose internal nodes are the nodes u with $|u| < D_n$ and $\#(u) > \overline{\#}_n$, the threshold paving cut off at depth D_n (Definition 8.2). If $D_n \to \infty$, $D_n/\overline{\#}_n \to 0$ and $\overline{\#}_n/n \to 0$, then $\int |f - f_{n,s_n^*}| \to 0$ with probability one, for every density f on $\mathbf{x}_\rho$. Condition (G) is not assumed. When (G) holds at D_n, s_n^* is the threshold paving s^* of Lemma 8.3.*

Proof. s_n^* is an RP of depth at most D_n for every sample (Lemma 8.3(i)), and a function of the depth-D_n count table, so it is the output of a merge rule at depth D_n. - *(M1).* By Lemma 8.3(i), $|\mathcal{L}(s_n^*)| \le 1 + D_n\lfloor n/(\overline{\#}_n + 1)\rfloor =: \overline{m}_n$, and $\overline{m}_n/n \le 1/n + D_n/\overline{\#}_n \to 0$. - *(M2).* Fix k, and let n be large enough that $D_n \ge k$. A leaf of depth less than k is not internal and has depth less than D_n, so it holds at most $\overline{\#}_n$ points. The leaves of depth less than k are pairwise disjoint nodes of depth less than k, so there are at most 2^{k-1} of them, and their empirical mass is at most $2^{k-1}\overline{\#}_n/n \to 0$. - *Under (G)* no node of depth D_n holds more than $\overline{\#}_n$ points, so the cut-off removes nothing (Lemma 8.3(ii)). Theorem 8.10 gives the claim. □

Marks. Proved above, from Theorem 8.10. [LEAN: certified from Theorem 8.8 (`ProveDog.DecisionTheory.VC.CorCoarsen`, `cor_coarsen`, built at `647131dc4`; Chapter 15): (a) from `card_leafSet_le`, (b) by `card_cellTraces_le`, (M2) from `shallow_leaves_mass_le` through point location (`mass_cmap`); with the half-open cube $[-1,1)^d$ as root box, any regular reference measure, $D_n/(\overline{\#}_n + 1)$ in place of $D_n/\overline{\#}_n$, and $f \ge 0$ everywhere.] [SIM: the L_1 error of f_{n,s_n^*} along n, with $\overline{\#}_n = n^\alpha$ and $D_n = \lceil b \log_2 n \rceil$, for densities with exact cell integrals, with (G) holding and failing; to run, L1MDE-PROOF-SIMULATION.]

The two directions are consistent together. By Lemma 8.4, s_n^* is also the terminal state of SEBTreeMC with depth cap D_n and leaf cap $\overline{m}_n = 1 + D_n\lfloor n/(\overline{\#}_n + 1)\rfloor$, whatever the ties. By Lemma 8.5, the merge queue reaches it from below. Condition (G) is needed only to identify s_n^* with the threshold paving s^*, through which Chapter 11 identifies the distributed merge with the chain.

When (G) holds. For a bounded density, take $\overline{\#}_n = n^\alpha$ and $D_n = \lceil b \log_2 n \rceil$, with $0 < \alpha < 1$ and $b > 1 - \alpha$. Then $D_n/\overline{\#}_n \to 0$. A depth-$D_n$ cell has volume proportional to n^{-b}, so its expected count is $O(n^{1-b}) = o(n^\alpha)$. A Chernoff bound, a union bound over the 2^{D_n} cells and the Borel–Cantelli lemma give (G) at D_n for all large n with probability one.

A root box chosen from the data. Chapter 10 takes as root box the box hull of the training points. The cells then move with the sample, and the family of partitions in Theorem 8.8 contains the regular pavings of every box: it is uncountable, and the count of trees in (b) no longer bounds its growth function. The following lemma bounds the deviation directly, through unions of boxes.

Lemma 8.12 (Deviation over unions of boxes)**.** *Let $\mathcal{B}_m$ be the class of unions of at most m boxes, and let $X_1, \dots, X_n$ be i.i.d. with law μ. - (i) If π is a partition of $\mathbb{R}^d$ into at most m boxes and the complement of their union, then $\sum_{A \in \pi} |\mu_n(A) - \mu(A)| \le 2 \sup_{B \in \mathcal{B}_m} |\mu_n(B) - \mu(B)|$. - (ii) The supremum is measurable, and $\mathbb{P}\{\sup_{B \in \mathcal{B}_m} |\mu_n(B) - \mu(B)| > \epsilon\} \le 4(2n+1)^{2dm} e^{-n\epsilon^2/8}$ for every $\epsilon > 0$. - (iii) If $m_n \log n/n \to 0$, then $\sup_{B \in \mathcal{B}_{m_n}} |\mu_n(B) - \mu(B)| \to 0$ with probability one.*

Proof. - (i). Both μ_n and μ give the cells of π total mass one, so $\sum_{A \in \pi} |\mu_n(A) - \mu(A)| = 2(\mu_n(\tilde{A}) - \mu(\tilde{A}))$, where $\tilde{A}$ is the union of the cells with $\mu_n \ge \mu$ [LN96, proof of Lemma 1]. Either $\tilde{A}$ is a union of at most m of the boxes, or it contains the complement cell, and then its complement is such a union, with the same deviation up to sign. - *(ii), traces.* Let T be a set of N points. The trace on T of a box C is the trace of the closed box spanned by $C \cap T$, since an interval containing two values contains the segment between them. The trace of a closed box is fixed by the positions of its two ends on each axis among the sorted coordinates of T, so boxes have at most $(N+1)^{2d}$ traces on T, the empty one included. A union of at most m boxes has its trace among the unions of m such traces, so $S_N(\mathcal{B}_m) \le (N+1)^{2dm}$. - *(ii), measurability.* A box is the pointwise limit of boxes with rational ends, of the same kind at each end, approaching each closed end from outside and each open end from inside. Under pointwise convergence of indicators, μ_n and μ of the boxes converge (by dominated convergence for μ). So the supremum over $\mathcal{B}_m$ equals the supremum over the countable subclass of unions of boxes with rational ends. - *(ii), the bound.* The Vapnik–Chervonenkis inequality for a class of sets [LN96, eq. (4)] with $S_{2n}(\mathcal{B}_m) \le (2n+1)^{2dm}$. - *(iii).* The bound of (ii) is $\exp\big(\log 4 + 2dm_n \log(2n+1) - n\epsilon^2/8\big)$, which is at most $4e^{-n\epsilon^2/16}$ for large n when $m_n \log n/n \to 0$, so it is summable; the Borel–Cantelli lemma gives the claim. □

Marks. Proved above; new as stated. [LEAN: the inequality of (ii) for boxes with rational corners and the VC dimension at most $2d$ are built, `ProveDog.DecisionTheory.VC.Boxes` (`vcDimLE_boxes`, `vc_inequality_ratBoxes`; Chapter 15), with the constants of the VC inequality for sets rather than the $(2n+1)^{2dm}$ trace count above; to write — (i), the unions of m boxes and the reduction of the supremum to the rational subclass, on `VC.Boxes`.] [SIM: not needed; it is a bound.]

Theorem 8.13 (Consistency with the training hull)**.** *Let $X_1, X_2, \dots$ be i.i.d. with a density f whose support is bounded. For $n \ge 2$ let H_n be the closed box hull of $X_1, \dots, X_n$, and let π_n be an RP of the regular scheme on H_n, chosen from the sample, possibly with independent randomness. Suppose that - (M1′) there are numbers $\overline{m}_n$ with $\overline{m}_n \log n/n \to 0$ such that, with probability one, π_n has at most $\overline{m}_n$ leaves for all large n, and - (M2) for every fixed k, the empirical mass of the leaves of π_n of depth less than k tends to 0 with probability one.*

Then $\int |f - f_{n,\pi_n}| \to 0$ with probability one, where f_{n,π_n} is the histogram on the leaves of π_n and is zero outside H_n.

Proof. For $n \geq 2$, two points share a coordinate with probability zero, so H_n has positive widths. Let $\hat{\pi}_n$ be the partition of $\mathbb{R}^d$ formed by the leaves of π_n and $\mathbb{R}^d \setminus H_n$; the latter has infinite volume, so f_{n,π_n} is the histogram of Theorem 8.8 on $\hat{\pi}_n$. The proof of [LN96, Theorem 1] bounds $\int |f - f_{n,\pi_n}|$, for whatever partition is used, by an arbitrarily small constant plus two terms [LN96, (10)–(15)]: the estimation term $\sum_{A \in \hat{\pi}_n} |\mu_n(A) - \mu(A)|$, and a multiple of $\mu\{x : \operatorname{diam}(\hat{\pi}_n[x]) \geq \gamma\}$ for a $\gamma > 0$ that depends on f and the constant only, not on n or the partition. The set is written with $\geq$, as in the paper's (11); over all $\gamma > 0$ this is the same requirement as condition (c) of Theorem 8.8 with $>$, since $\{\operatorname{diam} > \gamma\} \subseteq \{\operatorname{diam} \geq \gamma\} \subseteq \{\operatorname{diam} > \gamma/2\}$. It is enough to show that both terms tend to 0 with probability one, the second for every $\gamma > 0$. - *The estimation term.* The leaves of π_n are disjoint boxes with union H_n. So, for all large n, $\hat{\pi}_n$ consists of at most $\overline{m}_n$ boxes and the complement of their union, and Lemma 8.12(i), (iii) gives the claim. - *The cells.* Let K be a closed box with $\mu(K) = 1$, and W_K its widest width. With probability one all points lie in K, so $H_n \subseteq K$. By Lemma 4.23(iv), every cell of depth k of the scheme on H_n has diameter at most $\sqrt{d}\, W_K 2^{-\lfloor k/d \rfloor}$; fix k with this bound below γ. Then $\{x : \operatorname{diam}(\hat{\pi}_n[x]) \geq \gamma\} \subseteq (\mathbb{R}^d \setminus H_n) \cup U_n$, where U_n is the union of the leaves of π_n of depth less than k. - $\mu_n(H_n) = 1$, so $\mu(\mathbb{R}^d \setminus H_n) = \mu_n(H_n) - \mu(H_n) \leq \sup_{B \in \mathcal{B}_1} |\mu_n(B) - \mu(B)| \to 0$ (Lemma 8.12(iii)). - U_n is a union of at most 2^{k-1} boxes, so $\mu(U_n) \leq \mu_n(U_n) + \sup_{B \in \mathcal{B}_{2^{k-1}}} |\mu_n(B) - \mu(B)|$. The first term tends to 0 by (M2), and the second by Lemma 8.12(iii). □

Marks. Proved above; new. [LEAN: to write — on `VC.HistogramApprox` for the approximation terms and Lemma 8.12.] [SIM: the L_1 error along n with the training hull, for bounded densities with exact cell integrals; to run, L1MDE-PROOF-SIMULATION.]

Corollary 8.14 (The threshold merge on the training hull)**.** *In the setting of Theorem 8.13, let s_n^* be the threshold paving at depth D_n of the regular scheme on H_n. If $D_n \to \infty$, $D_n \log n / \overline{\#}_n \to 0$ and $\overline{\#}_n / n \to 0$, then $\int |f - f_{n,s_n^*}| \to 0$ with probability one.*

Proof. Lemma 8.3 holds for any root box that contains the sample. (M1′) holds with $\overline{m}_n = 1 + D_n \lfloor n/(\overline{\#}_n + 1) \rfloor$, since $\overline{m}_n \log n / n \leq \log n / n + D_n \log n / \overline{\#}_n \to 0$. (M2) holds as in the proof of Corollary 8.11. □

Marks. Proved above; new. [LEAN: to write.] [SIM: with the training hull, $\overline{\#}_n = n^\alpha$ and $D_n = \lceil b \log_2 n \rceil$; to run, L1MDE-PROOF-SIMULATION.]

The depth chosen by (G). Chapter 10 chooses the finest depth from the training points, as the (G)-depth D_n^{G}. By Lemma 8.3(iii), the paving it merges to is then the threshold paving $s_n^* = s_D^*$ for every $D \geq D_n^{\mathrm{G}}$, with no cut-off, and every leaf of it holds at most $\overline{\#}_n$ points.

Corollary 8.15 (The depth chosen by (G))**.** *Let $X_1, X_2, \ldots$ be i.i.d. with a density f. For each n let D_n^{G} be the (G)-depth of $X_1, \ldots, X_n$ with count limit $\overline{\#}_n \geq 1$, on a root box that contains the sample, and let s_n^* be the threshold paving. Suppose that $\overline{\#}_n / n \to 0$, and that there are deterministic depths d_n with $D_n^{\mathrm{G}} \leq d_n$ for all large n, with probability one. - (i) If the root box is fixed before the data are seen and $d_n / \overline{\#}_n \to 0$, then $\int |f - f_{n,s_n^*}| \to 0$ with probability one. - (ii) If the support of f is bounded, the root box is the training hull H_n, and $d_n \log n / \overline{\#}_n \to 0$, the same holds. - (iii) If f is bounded and its support lies in a bounded box, $\overline{\#}_n = n^\alpha$ with $0 < \alpha < 1$, and $d_n = \lceil b \log_2 n \rceil$ with $b > 1 - \alpha$, then $D_n^{\mathrm{G}} \leq d_n$ for all large n with probability one, both for a fixed root box that contains the support and for the training hull; so (i) and (ii) apply. - (iv) No cap on the depth is compatible with these conditions: $D_n^{\mathrm{G}} \geq \log_2(n/\overline{\#}_n) \to \infty$ for every sample. With a cap $D_{\max}$, (G) fails at every depth up to $D_{\max}$ as soon as $n/\overline{\#}_n > 2^{D_{\max}}$.*

Proof. - *(i).* $D_n^{\mathrm{G}} \geq \log_2(n/\overline{\#}_n)$ by Lemma 8.3(iii), so $\log_2(n/\overline{\#}_n)/\overline{\#}_n \leq d_n/\overline{\#}_n \to 0$ for all large n, which forces $\overline{\#}_n \to \infty$. Let $d_n' = \max(d_n, \lceil \sqrt{\overline{\#}_n} \rceil)$: it is deterministic, $d_n' \to \infty$ and

$d'_n/\overline{\#}_n \to 0$. For all large n, with probability one, $D_n^{\mathrm{G}} \le d_n \le d'_n$, so (G) holds at d'_n and $s_n^* = s_{d'_n}^*$ (Lemma 8.3(iii)). Corollary 8.11 with the depths d'_n gives $\int |f - f_{n,s_{d'_n}^*}| \to 0$ with probability one, and the two histograms agree for all large n. - *(ii).* Theorem 8.13 with $\pi_n = s_n^*$. (M1′): for all large n, with probability one, $|\mathcal{L}(s_n^*)| \le 1 + d_n \lfloor n/(\overline{\#}_n + 1) \rfloor =: \overline{m}_n$ (Lemma 8.3(iii)), and $\overline{m}_n \log n/n \le \log n/n + d_n \log n/\overline{\#}_n \to 0$. (M2): every leaf of s_n^* holds at most $\overline{\#}_n$ points (Lemma 8.3(ii)), so the leaves of depth less than k, at most 2^{k-1} of them, carry empirical mass at most $2^{k-1}\overline{\#}_n/n \to 0$. - *(iii), a fixed root box.* As under "When (G) holds" above: every depth-d_n cell has mass $O(n^{-b})$, so its expected count is $o(n^\alpha)$, and a Chernoff bound, a union bound over the 2^{d_n} cells and the Borel–Cantelli lemma give (G) at d_n for all large n, with probability one. - *(iii), the training hull.* Let K^* be the smallest closed box with $\mu(K^*) = 1$: the product over the axes of the intervals from the essential infimum to the essential supremum of the marginals of μ; it has positive widths, since μ has a density, and $f \le B$. With probability one, $H_n \subseteq K^*$ for every n, since every point lies in K^* almost surely, and the widths of H_n tend to those of K^*, since a strip of any positive width inside K^* along one of its faces has positive mass, by the definition of the essential bounds, and is therefore hit by some X_i with probability one; so for all large n the aspect ratio of H_n is at most $2a^*$, where a^* is that of K^*. Let $a = \max(2, 2a^*)$. A cell of depth h of the scheme on H_n has volume at most $2^{-h}\mathrm{vol}(K^*)$ and aspect ratio at most a (Lemma 4.23(i)), so its widest width is at most $\sigma_h = a\,(2^{-h}\mathrm{vol}(K^*))^{1/d}$, where d is the dimension. Tile K^* by the fixed grid of cubes of side σ_h. Each such cell meets at most 2^d cubes of the grid, so it holds at most 2^d times the largest count of a cube. A cube has mass at most $B\sigma_h^d = Ba^d\mathrm{vol}(K^*)2^{-h}$, which is $O(n^{-b})$ at $h = d_n$, and there are $O(2^{d_n}) = O(n^b)$ cubes. A Chernoff bound for the count $2^{-d}n^\alpha$, a union bound over the cubes and the Borel–Cantelli lemma show that, for all large n, every cube holds at most $2^{-d}n^\alpha$ points, so every depth-d_n cell of the scheme on H_n holds at most $\overline{\#}_n$ points. - *(iv).* Lemma 8.3(iii). □

Marks. Proved above; new. (iii) for the hull is written compactly: the Chernoff step is as under "When (G) holds", with the grid of cubes in place of the cells. [LEAN: to write.] [SIM: D_n^{G} along n for bounded densities, against $\lceil b \log_2 n \rceil$ and $\log_2(n/\overline{\#}_n)$, for fixed root boxes and for training hulls; to run, L1MDE-PROOF-SIMULATION.]

So the pipeline of Chapter 10, with the training hull and the (G)-depth, is consistent when the density is bounded with bounded support and the count limit grows like n^α; the depth it chooses then grows like $\log n$. A representable cap on the depth stops this: by (iv), once $n/\overline{\#}_n$ exceeds $2^{D_{\max}}$, the run uses the deepest representable depth, and its estimate is of the cell integrals at that resolution, not of f (Chapter 12).

Remark 8.16 (Samples that are not one sequence)**.** The results of this subsection, and Theorem 8.9, are stated for the first n points of one i.i.d. sequence. They hold also when, for each n, the sample is any i.i.d. sample of size n from μ, with no relation between the samples of different sizes. This covers training samples drawn afresh for each n, as by a random split of each data set. Every almost-sure step above is of one of three kinds: - a Borel–Cantelli step on probabilities that are summable whatever the joint law of the samples: [LN96, Corollary 1], Lemma 8.12, and the Chernoff steps of Corollary 8.15; - a pathwise bound: the approximation terms of [LN96, Theorem 1], and the counts of leaves; - the strong law on finitely many cells, in Theorem 8.10 and Theorem 8.9, which Hoeffding's inequality and the Borel–Cantelli lemma replace for each cell.

The widths of the hull in Corollary 8.15(iii) are covered too: the probability that a sample of size n leaves a strip of positive mass empty is exponentially small in n.

Marks. Proved by the argument above, which lists each step. [LEAN: to write.] [SIM: not needed.]

Other merges and selection. Under (G), the walk of Chapter 11 returns the nonempty leaves of s_n^* (Lemma 11.4). Without (G) its output can differ from s_n^*; whether it is consistent is Corollary 11.8, stated in Chapter 11 with the walk. Smoothing by going back along the queue keeps consistency: Chapter 10 proves this for its search, from Corollary 8.11 (Corollary 10.5). For the training hull and the (G)-depth, that proof needs Corollaries 8.14 and 8.15 in place of Corollary 8.11, and the leaf bound of Lemma 8.3(iii) for the length of the backtrack path; this adaptation is Chapter 10's.

8.6 Counts already on a grid

Data sometimes arrive as counts on a dyadic grid, not as points: for example, counts of events indexed by pairs of positions, binned into $2^k \times 2^k$ cells of the unit square. Such a table is a depth-D count map with $D = 2k$, since the first-widest rule alternates the two coordinates of a square (Definition 4.22). Everything in this section that reads only the count map applies to it unchanged: - the threshold paving and its leaf bound (Lemma 8.3); - SEBTreeMC with depth cap D, and the merge queue (Lemmas 8.4 and 8.5 and Proposition 8.6); - every merge rule at depth D.

So do the merge and selection phases of Chapter 10, with their exactness (Chapter 11), and the marginals and slices of Chapter 6. The phases of Chapter 10 that locate points are skipped.

Two things change. - *Holding out.* The selection of Chapter 9 needs a held-out sample, so the table must be split into a training table and a validation table. For i.i.d. events, holding out each event independently with probability φ gives this split. On a table already binned, the same law is obtained by thinning each cell's count: the validation count of a cell u is Binomial$(\#(u), \varphi)$, drawn independently over the cells, and the rest of the count is for training. The validation size is then random, and the guarantees of Chapter 9 hold conditionally on it. - *Resolution.* The grid fixes D. Corollary 8.11 needs $D_n \to \infty$, so on a fixed grid it does not apply as n grows. The estimate is then an estimate of the cell integrals at the resolution of the grid, the density f_D of Chapter 12, whose resolution bias does not shrink with more data. The grid should be as fine as the measurement supports, so that (G) can hold at the grid's depth for the count limit in use.

The guarantees need the counted events to be an i.i.d. sample from the density being estimated. Whether a given measurement process meets this has to be argued for that process.

8.7 The braiding: the queue on the uniform law

Under the uniform law, the priority of the queue is the mass of a cell, which halves at each split, and the queue grows the complete binary trees one level at a time. With finitely many points the counts fluctuate around the masses, and the trees that the queue visits wind around the complete trees. The author calls this *braiding.* It is the author's observation that the braiding and the Catalan coefficients are intimately connected, that the braiding law is the stationary law of an SEB priority-queue chain under the uniform law, on a hypercube or on the unit interval, and that it holds as well when the queue merges back. The word is older. A note of 2012 in the MRS group records "the 'braiding' effect" for the observation that more split paths lead to the more balanced trees [Har12]. For example, the complete tree with 7 splits is reached by 80 of the $7! = 5040$ split paths (Example 4.10), so the Yule model of Proposition 4.16(i) gives it probability $1/63$.

This subsection makes the observation precise in the population limit, where the law of the queue follows from the threshold law of Chapter 6 (Proposition 6.51). By the author's ruling, both readings of "the braiding law" are stated, the visit law first: Theorem 8.18 gives the law of the trees visited at each number of splits, going down by splitting and coming back by merging,

and Proposition 8.19 then gives a split–merge chain whose stationary law is that visit law. At a finite sample size the merge-back law is not the split law, for example $\frac{19}{192}$ apart in total variation at $D = 3$, $n = 4$, $k = 4$ (Remark 8.23); so the merge-back statement is a statement about the limit. The subsection then states what is proved for n points, and the simulation that measures the rest.

The population queues. Let μ be the uniform law on $\mathbf{x}_\rho$. Every cell of depth h has mass $\mu(\mathbf{x}_u) = 2^{-|u|}$, whatever the dimension and the root box (Chapter 4). The *population split queue* starts from the root tree and splits a leaf of largest mass, drawn uniformly among ties. The *population merge queue* merges a cherry of smallest mass, drawn uniformly among ties. They are SEBTreeMC and the merge queue of Lemma 8.5, run without stopping, with each count $\#(u)$ replaced by $n\mu(\mathbf{x}_u)$; Corollary 8.22 makes this limit precise. The mass $\psi(u) = 2^{-|u|}$ is a strict priority in the sense of Chapter 6: a child has half the mass of its parent.

Splitting needs only Proposition 6.51(iii). Merging needs its counterpart for the ties in pruning.

Lemma 8.17 (Uniform ties in pruning)**.** *In Proposition 6.51(ii), let $\psi(v) < \psi(u)$ whenever $v \sqsubset u$ both lie in $I(s)$, and let each prune draw uniformly among the cherries of smallest ψ. Let t be the k-th smallest value of ψ on $I(s)$. Then the nodes of $I(s)$ with $\psi = t$ are pairwise disjoint. The pruned set is $\{u \in I(s) : \psi(u) < t\}$ together with a uniformly random subset, of size $k - |\{u \in I(s) : \psi(u) < t\}|$, of $\{u \in I(s) : \psi(u) = t\}$. Moreover, at each prune from the first of value t on, the cherries of smallest ψ are exactly the nodes of value t not yet pruned.*

Proof. Two nested nodes of $I(s)$ have different values, so the nodes of value t are disjoint. By Proposition 6.51(ii), whose monotonicity the hypothesis gives, the pruned values do not decrease, and the pruned set contains $\{u \in I(s) : \psi(u) < t\}$. So all prunes of value less than t come first. At the first prune of value t, let $u \in I(s)$ have $\psi(u) = t$. Every internal node of s strictly below u has a smaller value, so it has been pruned, and u is a cherry. Every remaining cherry has value at least t, since one of smaller value would lie in $\{\psi < t\}$ and would have been pruned. So the cherries of smallest ψ are the nodes of value t not yet pruned. Pruning one of them can make its parent a cherry, but the parent lies in $I(s)$ and has a larger value. So the remaining prunes of value t draw uniformly without replacement from this one set, and the nodes they prune form a uniformly random subset of the stated size. □

Marks. Proved above; new. It completes Proposition 6.51(iv) as part (iii) completes (i). [LEAN: PQThreshold lem_pruneTies, without the uniform law: the nodes of $I(s)$ with $\psi = t$ pairwise disjoint, the pruned set $\{u \in I(s) : \psi(u) < t\}$ with a subset of $\{u \in I(s) : \psi(u) = t\}$ of size $k - |\{u \in I(s) : \psi(u) < t\}|$, the prunes of value t last, and at each of them the cherries of smallest ψ exactly the nodes of value t not yet pruned — built at `796cfb0cd`, under every tie-break; the uniform law of the subset is not modelled.] [SIM: the exact law of the pruned set over all pruning sequences, for 150 random trees with strict integer priorities and many ties, 871 (tree, k) cases, no mismatch (the review's `rev7_check.py prune`); to rerun, L1MDE-PROOF-SIMULATION.]

The trees of the limit law. For $k \geq 0$ write $k = 2^m - 1 + j$, with $m \geq 0$ and $0 \leq j < 2^m$; so $m = \lfloor \log_2(k+1) \rfloor$. Let $\mathbb{T}_k$ be the set of the $\binom{2^m}{j}$ trees whose internal nodes are those of s^{c}_m together with j nodes of depth m, and let q_k be the uniform law on $\mathbb{T}_k$. In particular $\mathbb{T}_{2^m-1} = \{s^{\mathrm{c}}_m\}$. A split path is *by levels* when, for every h, it splits all 2^h nodes of depth h before any node of depth $h+1$. Its trees then lie in $\mathbb{T}_0, \mathbb{T}_1, \ldots$ in turn. A tree $t \in \mathbb{T}_k$ contains every node of depth less than m, so the split paths by levels to t are the rankings of t (Definition 4.6 and Lemma 4.7) that rank every node of depth h before every node of depth $h+1$. These order each level freely, so there are

$$A(t) = j! \prod_{h<m} (2^h)!$$

of them.

Theorem 8.18 (The population SEB queue on the uniform law: the visit law)**.** *Let μ be the uniform law on $\mathbf{x}_\rho$. - (i) Splitting down. Let $S_k^{\uparrow}$ be the tree after k steps of the population split queue. Then $S_k^{\uparrow}$ has the law q_k. In particular $S_{2^m-1}^{\uparrow} = s_m^{\mathrm{c}}$ for every m. - (ii) Merging back. Let $M \geq 1$, $K = 2^M - 1$, $S_K^{\downarrow} = s_M^{\mathrm{c}}$, and let $S_k^{\downarrow}$ be the tree with k splits that the population merge queue reaches from s_M^{c}. Then $S_k^{\downarrow}$ has the law q_k for every $k \leq K$. Moreover the merge path read backwards, $S_0^{\downarrow}, S_1^{\downarrow}, \dots, S_K^{\downarrow}$, has the law of $S_0^{\uparrow}, \dots, S_K^{\uparrow}$. - (iii) Split paths. The split queue follows a split path with positive probability exactly when the path is by levels. To each $t \in \mathbb{T}_k$ there are $A(t)$ such paths, among the $B(t)$ split paths to t (Lemma 4.9), and each has the probability*

$$p_k = \frac{(2^m - j)!}{(2^m)! \prod_{h<m} (2^h)!}, \qquad \text{so that} \qquad \Pr\{S_k^{\uparrow} = t\} = A(t)\, p_k = \binom{2^m}{j}^{-1}.$$

Proof. - (i). For $k = 0$ there is nothing to prove. Let $k \geq 1$. The priority $\psi(u) = 2^{-|u|}$ is strict and positive, and the 2^h nodes of depth h have $\psi = 2^{-h}$. If $j \geq 1$, exactly $2^m - 1$ nodes have $\psi > 2^{-m}$ and $2^{m+1} - 1$ have $\psi \geq 2^{-m}$, so the k-th largest value of ψ is 2^{-m}. By Proposition 6.51(iii), $I(S_k^{\uparrow})$ is the set of the nodes of depth less than m together with a uniformly random subset of N_m of size $k - (2^m - 1) = j$. If $j = 0$ and $m \geq 1$, the k-th largest value is $2^{-(m-1)}$. The same part gives the $2^{m-1} - 1$ nodes of depth less than $m - 1$ together with a subset of N_{m-1} of size 2^{m-1}, which is all of N_{m-1}; so $I(S_k^{\uparrow}) = I(s_m^{\mathrm{c}})$. - *(iii).* By (i), $S_i^{\uparrow} \in \mathbb{T}_i$ for every i. A tree of $\mathbb{T}_i$ with an internal node of depth $h+1$ has all nodes of depth h internal, so every path the queue follows is by levels. Conversely, let a path be by levels, and consider a step that splits a node of depth h. All nodes of depth less than h have been split, so all 2^h nodes of depth h are present, and no node of depth $h+1$ has been split. The current leaves are the r nodes of depth h not yet split and the children of those split, of depth $h+1$. The leaves of largest mass are the r unsplit nodes of depth h, so the step has probability $1/r > 0$. Multiplying over the steps gives $1/(2^h)!$ for each full level $h < m$, and $(2^m - j)!/(2^m)!$ for the j splits at level m: the product is p_k. There are $A(t)$ paths by levels to t, and $A(t)p_k = j!\,(2^m - j)!/(2^m)!$. - *(ii).* For $k = K$ there is nothing to prove; let $k < K$, so $m < M$. The merge queue prunes $K - k = 2^M - 2^m - j$ cherries from s_M^{c}. Among the internal nodes of s_M^{c}, the $2^M - 2^{m+1}$ nodes of depth greater than m have $\psi < 2^{-m}$; with the 2^m nodes of depth m, they are the $2^M - 2^m$ nodes with $\psi \leq 2^{-m}$. As $0 \leq j < 2^m$, the $(K-k)$-th smallest value of ψ on $I(s_M^{\mathrm{c}})$ is 2^{-m}. By Lemma 8.17, the pruned set is the set of the nodes of depth greater than m together with a uniformly random subset of N_m of size $2^m - j$. Its complement in $I(s_M^{\mathrm{c}})$ is $I(s_m^{\mathrm{c}})$ together with a uniformly random j-subset of N_m, so $S_k^{\downarrow}$ has the law q_k. For the paths: by Proposition 6.51(ii) the pruned values do not decrease, so the queue prunes s_M^{c} level by level from the deepest, and read backwards its path is a split path to s_M^{c} by levels. By the last part of Lemma 8.17, when r nodes of the level being pruned remain, the queue draws among exactly these r. So each merge path has probability $\prod_{h<M} 1/(2^h)!$, and the reversed merge path is uniform over the $A(s_M^{\mathrm{c}}) = \prod_{h<M} (2^h)!$ split paths by levels to s_M^{c}. By (iii) with $k = K$, so is $S_0^{\uparrow}, \dots, S_K^{\uparrow}$. □

Marks. Proved above; new. [LEAN: to write — `Braid`, on `PQThreshold` (i)–(iii) and Lemma 8.17.] [SIM: exact for $M \leq 4$: the laws of (i) and (ii) and $A(t)p_k$ (scratchpad `braid_check.py`); the path statements of (ii) and (iii) by full enumeration for $M \leq 3$ and by streaming enumeration for $M = 4$, 1,935,360 split paths and as many merge paths, each by levels with probability 1/1935360, every prefix as (iii) says (the review's `rev7_check.py braid3` and `rev7_braid4.py`). At $M = 5$ the paths number $A(s_5^{\mathrm{c}}) = 1!\,2!\,4!\,8!\,16! \approx 4.05 \times 10^{19}$, so the laws on trees and the one-step rule (by levels, uniform within a level), which together imply the path statements, are to be checked instead; to run, L1MDE-PROOF-SIMULATION.]

Table 8.1 lists the limit law up to the complete tree of depth 3.

Table 8.1: The law q_k of the population SEB queue on the uniform law, for $k \le 7$ splits (Theorem 8.18). Trees are named by their leaf-depth strings. A counts the split paths by levels to each tree, and B is its Catalan coefficient, the number of all its split paths (Lemma 4.9).

k	trees of $\mathbb{T}_k$	q_k each	A	B
0	0	1	1	1
1	11	1	1	1
2	221, 122	$\frac{1}{2}$	1	1
3	2222	1	2	2
4	33222, 23322, 22332, 22233	$\frac{1}{4}$	2	3
5	333322, 223333	$\frac{1}{6}$	4	8
5	332332, 332233, 233332, 233233	$\frac{1}{6}$	4	6
6	3333332, 3333233, 3323333, 2333333	$\frac{1}{4}$	12	20
7	33333333	1	48	80

How the queue weights the split orders. A split-exchangeable model gives every split path to t the same probability, so the probability of t is $B(t)$ times that of one path (Theorem 4.14). The population queue is not split-exchangeable: it never follows a path that is not by levels. On the paths by levels it is exchangeable: it gives each of them the probability p_k, so the probability of $t \in \mathbb{T}_k$ is $A(t)\,p_k$, with $A(t) \le B(t)$ in the role of the Catalan coefficient. The complete tree with 7 splits shows the contrast. It has $B = 80$ split paths (Example 4.10), of which $A = 48$ are by levels. Its probability after 7 splits is: - $1/429$ under the uniform law on $\mathbb{S}_7$ (Proposition 4.5); - $80/7! = 1/63$ under the Yule model, which splits a leaf drawn uniformly (Proposition 4.16(i)); - $80 \cdot 2^{-10} = 5/64$ under the uniform SEB model, which splits a leaf drawn with probability its mass (Proposition 4.16(ii)); - 1 under the queue, which splits a leaf of largest mass and reaches the tree by its 48 paths by levels, each with probability $1/48$.

Down the last three entries the choice of leaf favours the shallow leaves more and more, and the mass moves to the complete tree: the braiding of the 2012 note, carried to its end by the greedy queue. The queue uses a shrinking share of the split orders. For the complete tree of depth 4, with 15 splits, Lemma 4.9(i) gives $B = 21964800$, while $A = 1!\,2!\,4!\,8! = 1935360$: fewer than one split path in eleven.

The law is constant on shapes. For example, the six trees of $\mathbb{T}_5$ form two shapes. In two of them the split nodes of depth 2 are siblings, and in four they are cousins, as Lemma 4.13(iii) counts: 2^{5-4} and 2^{5-3} trees. So q_5 gives the two shapes the probabilities $\frac{1}{3}$ and $\frac{2}{3}$.

The second reading: a stationary law. A queue that only grows has no stationary law. One that splits and merges in turn has one, and it is the visit law of Theorem 8.18.

Proposition 8.19 (The stationary law of the split–merge queue)**.** *Fix $M \ge 1$, let $K = 2^M - 1$, and let $\Omega_M = \bigcup_{k \le K} \mathbb{T}_k$. Let $(Z_r)_{r \ge 0}$ be the Markov chain that, from a tree with k splits, takes one step of the population split queue with probability $\frac{1}{2}$ and one step of the population merge queue with probability $\frac{1}{2}$ when $0 < k < K$, a split step when $k = 0$, and a merge step when $k = K$. - (i) The chain stays in Ω_M and is irreducible on it. - (ii) Its unique stationary law is $\pi(t) = w_k\, q_k(t)$ for $t \in \mathbb{T}_k$, with $w_0 = w_K = 1/(2K)$ and $w_k = 1/K$ for $0 < k < K$. - (iii) The chain that alternates, splitting from the root to s_M^{c} and merging back to the root, spends in the long run the fraction $\pi(t)$ of its steps at each tree t, with probability one; so does (Z_r). - (iv) (Z_r) has period 2, since its number of splits changes by one at each step, so the law of Z_r does not converge to π. The lazy chain, which stays where it is with probability $\frac{1}{2}$ and otherwise moves as (Z_r), has the same stationary law and converges to it in law.*

Proof. - The steps carry the laws. $S^{\uparrow}$ is a Markov chain, so by Theorem 8.18(i) one split step

applied to a tree drawn from q_k gives a tree drawn from q_{k+1}. One merge step applied to a tree drawn from q_k gives a tree drawn from q_{k-1}, directly. A tree of $\mathbb{T}_k$ with $k = 2^m - 1 + j$ has as internal nodes every node of depth less than m and j nodes of depth m. If $j \geq 1$, its cherries of smallest mass are exactly those j nodes: each is a cherry, of mass 2^{-m}, and every other cherry has depth $m-1$ and mass $2^{-(m-1)}$. The population merge queue prunes one of them uniformly (Lemma 8.17, its last part), and removing a uniform element from a uniform j-subset of N_m leaves a uniform $(j-1)$-subset: the law q_{k-1}. If $j = 0$, the tree is s^{c}_m, its cherries are all of N_{m-1}, and one uniform merge leaves the nodes of depth less than $m-1$ with a uniform $(2^{m-1}-1)$-subset of N_{m-1}, which is again q_{k-1}. In particular the steps map $\mathbb{T}_k$ into $\mathbb{T}_{k\pm1}$, so the chain stays in Ω_M. - *(ii), stationarity.* The number of splits of Z_r is itself a Markov chain on $\{0, \dots, K\}$: it moves up or down with probability $\frac{1}{2}$ inside, and is reflected at the ends. Detailed balance shows that w is its stationary law. If Z_r is drawn from π, then given that its number of splits moves from k to k', Z_{r+1} is drawn from $q_{k'}$, by the first step. So Z_{r+1} is drawn from $\sum_{k'} w_{k'} q_{k'} = \pi$. - *(i), irreducibility, and uniqueness.* From a tree with k splits, k merge steps reach the root with positive probability. From the root, split steps reach every tree of Ω_M with positive probability, by Theorem 8.18(i). So the chain is irreducible on the finite set Ω_M, and its stationary law is unique. - *(iii).* The alternating chain returns to the root every $2K$ steps, and its cycles are independent. Each cycle visits a tree with 0 splits and one with K splits once, and a tree with k splits twice for $0 < k < K$, drawn each time from q_k (Theorem 8.18(i), (ii)). The renewal reward theorem gives the long-run fraction $\pi(t)$. For (Z_r), the ergodic theorem for an irreducible chain on a finite set gives the same. - *(iv).* The lazy chain has the transition matrix $\frac{1}{2}(I + P)$, with the stationary laws of P; it is irreducible and aperiodic on a finite set, so it converges to π. □

Marks. Proved above; new. [LEAN: to write.] [SIM: exact for $M \leq 4$: the push-forwards $q_k \mapsto q_{k\pm1}$, closure, stationarity, total mass one and irreducibility (the review's `rev7_stationary.py`; the stationarity check at $M = 4$ also in scratchpad `braid_check.py`); the occupation frequencies of the alternating chain against π for $M \leq 5$, to run, L1MDE-PROOF-SIMULATION.]

So the author's description "the stationary distribution of a SEB PQ Markov chain" has an exact meaning in the population limit: the long-run frequencies of (iii), which equal the visit law of Theorem 8.18 weighted by the time spent at each number of splits.

With n points. Three facts hold at every sample size, or in the limit, and the simulation measures the rest.

Remark 8.20 (The dimension does not matter)**.** Let $X_1, \dots, X_n$ be i.i.d. uniform on $\mathbf{x}_\rho$. Each cell of depth h has mass 2^{-h}, so the depth-h cells of the points are i.i.d. uniform on N_h, for every h. They are consistent along ancestry: the cell of a point at depth h is the parent of its cell at depth $h+1$. So the field of counts $\{\#(u)\}$ has the same law in every dimension and for every root box, and so has every path of SEBTreeMC, of its backtrack and of the merge queue, since these read only the counts. A simulation on the unit interval, as the author proposed, therefore loses nothing. It may as well draw the points as uniform random binary words.

Proposition 8.21 (Plane invariance on uniform data)**.** *Let $X_1, \dots, X_n$ be i.i.d. uniform on $\mathbf{x}_\rho$. Run SEBTreeMC with any count limit, leaf cap and depth cap, breaking ties uniformly; or run the merge queue by least count, with uniform ties, from s^{c}_M. For every tree s, the probability that the path visits s is the same for all trees of the shape of s. So, if s has k splits, the probability that the path visits a tree of the shape of s is $2^{k-\mathrm{sym}(s)}$ times the probability that it visits s (Lemma 4.13(iii)). The same holds for the terminal state.*

Proof. Let u be a node, ι the split axis of its depth, and w half the width of $\mathbf{x}_u$ along ι. Let T_u translate $\mathbf{x}_{u\mathsf{L}}$ by $+w$ along ι, translate $\mathbf{x}_{u\mathsf{R}}$ by $-w$, and fix the rest of $\mathbf{x}_\rho$. All cells of one

depth have the same widths and the same split axis (Lemma 4.23(ii)). So T_u maps the cell of each node v onto the cell of $\sigma_u(v)$, where σ_u is the swap at u of §4.3, up to the faces of the cells, a set of volume zero. T_u preserves volume, so $T_uX_1, \dots, T_uX_n$ are again i.i.d. uniform. With probability one no point lies on a face, and then $\#_{T_uX}(\sigma_u(v)) = \#_X(v)$ for every node v. Both queues choose their moves from the counts and the tree alone. σ_u maps leaves to leaves and cherries to cherries, and keeps depths and the number of leaves, so it preserves every comparison the queues make; their ties are broken uniformly. Hence the path on T_uX visits $\sigma_u(s)$ with the probability that the path on X visits s. Since T_uX and X have one law, the path on X visits $\sigma_u(s)$ with that probability too. The root tree and s^{c}_M are fixed by every swap, and swaps generate the trees of one shape (§4.3). Lemma 4.13(iii) gives the count. □

Marks. Proved above; new. [LEAN: to write.] [SIM: exact over every tuple of uniform labels, for both queues and for the terminal state, in seven settings $(n, \overline{\#}, \overline{m}, D)$ with $n \le 5$ and $D \le 3$: visit probabilities are equal within every shape (the review's `rev7_check.py sym`); visit frequencies within shapes at every n of Remark 8.24, to run, L1MDE-PROOF-SIMULATION.]

Corollary 8.22 (The braiding law is the limit)**.** *Let $X_1, X_2, \dots$ be i.i.d. uniform on $\mathbf{x}_\rho$. Fix $M \ge 1$, $K = 2^M - 1$, a count limit $\overline{\#}$ and a leaf cap $\overline{m} > K$. For each n and $k \le K$, let $S_k^{\uparrow,n}$ be the state with k splits of SEBTreeMC on $X_1, \dots, X_n$, with uniform ties and no depth cap, set to a cemetery state ∂ when the chain stops before k splits; and let $S_k^{\downarrow,n}$ be the state with k splits of the merge queue by least count from s^{c}_M, with uniform ties. As $n \to \infty$: - (i) the probability that SEBTreeMC makes at least K splits tends to 1, and for each $k \le K$ the laws of $S_k^{\uparrow,n}$ and of $S_k^{\downarrow,n}$ converge to q_k in total variation; - (ii) the probability that $S_k^{\uparrow,n} = S_k^{\downarrow,n}$ for every $k \le K$ tends to 1, however the two queues break their ties.*

Proof. - The event. Let E_n be the event that every node of depth less than M holds more than $\overline{\#}$ points, and that for every $h < M$ every node of depth h holds more points than every node of depth $h+1$. By the strong law of large numbers, $\#(u)/n \to 2^{-|u|}$ with probability one for each of the finitely many nodes of depth at most M. So E_n holds for all large n, and $\Pr(E_n) \to 1$. - *On E_n both queues go by levels.* Split direction: as in the proof of Theorem 8.18(iii), the current leaves have depth h or $h+1$ with $h < M$, and those of depth h hold more points. SEBTreeMC therefore splits a node of depth h of largest count; that node has depth less than M, so it holds more than $\overline{\#}$ points, and the count criterion does not stop the chain. With $\overline{m} > K$, it makes at least K splits. Merge direction: the current cherries lie on the level being pruned, or one level above it, and on E_n the deeper ones hold fewer points; so the queue prunes the deepest level first, a cherry of least count at a time. - *(i).* So on E_n, for $k = 2^m - 1 + j \le K$, both $I(S_k^{\uparrow,n})$ and $I(S_k^{\downarrow,n})$ are $I(s^{\mathrm{c}}_m)$ together with j nodes of N_m. For the split queue these are the j nodes of largest count, ties broken uniformly; for the merge queue, the j left after removing $2^m - j$ of least count, ties broken uniformly. The counts of the 2^m nodes of depth m are multinomial with equal cell probabilities, so their law is invariant under every permutation of N_m. Both rules commute with these permutations, so each, applied to these counts on every sample, gives a uniformly random j-subset of N_m. The queues agree with these rules on E_n, so the total variation distance from q_k is at most $\Pr(E_n^{\mathrm{c}}) \to 0$. - *(ii).* If, moreover, the counts of the nodes of each depth less than M are distinct, both queues are deterministic on E_n and take the same j nodes at every k. For two disjoint cells of equal mass p, the difference of their counts divided by $\sqrt{n}$ converges in law to a centred normal law with variance $2p > 0$, so the probability that the counts are equal tends to 0. There are finitely many pairs. □

Marks. Proved above; new. [LEAN: to write.] [SIM: a seeded review simulation at $M = 3$ shows the predicted trend in both (i) and (ii) (the review's `rev7_limit_sim.py`); the distance of the visit law from q_k, and the fraction of replicates in which both directions agree at every k, as functions of n (Remark 8.24), to run, L1MDE-PROOF-SIMULATION.]

Remark 8.23 (At a finite sample size the two directions differ)**.** The merge-back half of Theorem 8.18 holds in the population limit and, by Corollary 8.22, as $n \to \infty$. At a finite n it fails. Take n i.i.d. uniform points, the depth cap $D = 3$, no count criterion and uniform ties, and compare the law of the SEBTreeMC state with k splits with the law of the state with k splits of the merge queue from s_3^{c}. An exact enumeration over all 8^n tuples of depth-3 cells gives total variation distances of $\frac{3}{64}$, $\frac{19}{192}$ and $\frac{19}{192}$ at $k = 3, 4, 5$ for $n = 4$; of $\frac{5}{48}$, $\frac{5}{48}$ and $\frac{1}{8}$ for $n = 3$; and of $\frac{15}{256}$, $\frac{31}{768}$ and $\frac{31}{768}$ for $n = 5$ (the review's `rev7_updown.py`, rerun while revising). Example 8.7 shows the same on one sample: at $k = 3$ the split queue gives its two trees $\frac{3}{4}$ and $\frac{1}{4}$, and the merge queue $\frac{1}{2}$ and $\frac{1}{2}$. The two directions reach the same trees (Proposition 8.6), with different laws.

So, for large n, the queue retraces in both directions the path of the population limit: it passes through each complete tree s_m^{c} and spreads uniformly over $\mathbb{T}_k$ in between. For moderate n the counts at the level being filled, of order $n2^{-m}$, fluctuate on the scale of their square root. Then a deeper cell can outnumber a shallower one, the queue leaves $\mathbb{T}_k$, and its trees braid around the complete trees, more widely as m grows at fixed n. How widely, and how the split orders the queue then uses spread over the $B(t)$ split paths of each tree, is what the simulation measures.

Remark 8.24 (The braiding simulation)**.** The simulation this monograph runs:

1. **Programs.** MRS already counts visits. Its example `HistReport` draws uniform points in $[0, 1]$, grows a histogram with `prioritySplit` and the comparison `CompCount`, and tabulates how often each tree, named by its leaf-depth string, is visited at each number of leaves [SYH+18]. The rerun extends it in four ways:
   - it runs both directions, the merge direction with `priorityMerge` and the same comparison, from s_M^{c} built by `splitToShape`;
   - it records a seed for each replicate;
   - it records the split path of each run;
   - it runs to every $k \le K$, not only to four leaves.

   MRS selects a tie by adding $1/r$ in floating point over the r tied nodes until the sum exceeds a uniform draw; if rounding leaves the sum below the draw, the loop runs past the last tied node. The rerun selects the tie by an integer index instead. A re-implementation on labels (Chapter 4) is checked against it on the same seeds. Both programs are also run in their population form, with the masses in place of the counts: an exact enumeration of that form must reproduce q_k and π (Theorem 8.18 and Proposition 8.19), and a randomised run must reproduce them within its Monte Carlo error.
2. **Design.**
   - $d = 1$ suffices (Remark 8.20); a run with $d = 2$ checks the programs.
   - Fix M, so $K = 2^M - 1$, and a range of n over which the expected count $n2^{-M}$ of a deepest cell runs from below one to many.
   - Run R independent replicates for each n.
   - The split queue runs without its count criterion, for exactly K splits, with ties drawn uniformly. The merge queue runs from s_M^{c} to the root on the same counts.
3. **Records.** For each replicate, each $k \le K$ and each direction: the tree, as its leaf-depth string; its shape; whether it lies in $\mathbb{T}_k$; and the path that reached it, as a ranking (Lemma 4.7).
4. **Comparisons.**
   - Visits per tree at each k, in each direction, against q_k (Theorem 8.18): the total variation distance with its Monte Carlo error, and the mass outside $\mathbb{T}_k$, as functions of n and k. Corollary 8.22(i) predicts that both tend to 0 as n grows at fixed k.
   - The two directions against each other, at each n and k; for $D = 3$ and $n \le 5$, against the exact distances of Remark 8.23.

- Visits to the trees of one shape against one another. Proposition 8.21 predicts equal frequencies, up to sampling error, at every n.
- The split paths used to reach each visited tree t: how many of its $B(t)$ split paths appear, with what frequencies, and what share of them are by levels, against $A(t)$ and p_k (Theorem 8.18(iii)). $B(t)$ comes from the recursion of the MRS example `CatalanCoeff` (Lemma 4.9(ii)).
- The fraction of replicates in which the two directions visit the same tree at every k, against Corollary 8.22(ii).

5. **Report.** Tables and plots of these comparisons, with the seeds and the program versions, and no claim beyond what they show.

[SIM: to run, L1MDE-PROOF-SIMULATION.]

Chapter 9

Estimation II: minimum distance

Chapter 8 builds a path of histograms, the states of the randomised priority queue SEBTreeMC, and proves that its terminal state is consistent in L_1 [ST19, Theorem 1]. Consistency is a statement about $n \to \infty$. For the sample in hand it does not say how far along the path to stop: too few leaves smooth too much, and too many follow the noise. (Figure 3 of the paper contrasts two values of the count limit $\overline{\#}$.) This section chooses a state by minimum distance. Its guarantee, the expected bound of §9.3, holds at every n and for every density, without assumptions on the density's smoothness [ST19].

9.1 The distance

An estimate f_n of a density f on $\mathbb{R}^d$ is judged by how well it gives the probabilities of sets. When f_n integrates to one, Scheffé's identity says that the total variation distance equals half the L_1 distance:

$$\mathrm{TV}(f_n, f) = \sup_B \left| \int_B f_n - \int_B f \right| = \frac{1}{2} \int |f_n - f|,$$

with the supremum over Borel sets [DL01, Theorem 5.1]. So the L_1 error, on the scale $[0, 2]$, is a distance between probabilities.

A density estimator has *universal performance guarantees* when its bounds hold for every density $f \in L_1$ [DL01, p.1]. The classical rates of convergence assume $f \in L_2$ or $f \in C^k$ in order to choose the smoothing for a given n. Universality drops these assumptions [ST19, §1].

9.2 Where the method comes from

Four ideas meet in the selection. - **Likelihood.** Fisher defined the likelihood of a parameter value as proportional to the probability of the observations under it, compared two values by the ratio of their likelihoods, and chose the value of greatest likelihood [Fis22]. Between two densities f and g, one observation x favours f exactly when $f(x) > g(x)$. - **Convergence.** By Scheffé's theorem [Sch47], if densities converge almost everywhere to a density, their probabilities converge uniformly over Borel sets. By a remark of A. P. Morse recorded there, the densities also converge in L_1. - **Minimum distance.** The principle of choosing the model closest to the empirical measure is Wolfowitz's [Wol57]. - **Yatracos's sets.** Yatracos took the supremum over the sets $\{f_i > f_j\}$, $i < j$, of the centres of a finite L_1 cover of a model, instead of over all Borel sets, on which the empirical measure is far from every density. He proved that the minimum-distance estimate over these sets comes within 3ϵ, plus terms of the order of the cover

radius and of the deviation of the empirical measure over the sets, when the truth lies within ϵ of the model [Yat85, Lemma 2, Theorem 1, Proposition 2].

In his own account, Yatracos chose these sets motivated by maximum likelihood: the set $\{x : f(x) > g(x)\}$ determines f as the maximum-likelihood estimate of the two (Y. G. Yatracos, personal communication).

Devroye and Lugosi then took arbitrary candidate estimates, built on part of the data and selected on the rest, and bounded the complexity of the class of sets by Vapnik–Chervonenkis methods. Their book gives the deterministic form for finitely many candidates used below [DL01, Theorem 6.3]. It calls $\{f > g\}$ a *Scheffé set* (§7.1) and credits the selection by such sets to Yatracos (Chapter 7.9, §8.2, §8.7).

The JJSDS paper records its own beginning: "This research began from a conversation RS had with Luc Devroye at the World Congress in Probability and Statistics in 2008" [ST19, Acknowledgements].

9.3 The minimum distance estimate and its guarantees

Hold-out. Fix $0 < \varphi < 1/2$. Use $n - \varphi n$ points for training and the remaining φn for validation, where φn means $\lfloor \varphi n \rfloor$ [ST19, §3]. Let Θ index finitely many candidate densities $f_{n-\varphi n,\theta}$, built from the training points, each integrating to one.

Yatracos sets. For an ordered pair $\theta \neq \vartheta$, the Scheffé set is

$$A_{\theta,\vartheta} = \{x : f_{n-\varphi n,\theta}(x) > f_{n-\varphi n,\vartheta}(x)\}.$$

The paper's *Yatracos class* $\mathcal{A}_\Theta$ is the collection of these over ordered pairs [ST19, p.516]. Yatracos's own class takes one set per unordered pair [Yat85, p.770]. That suffices, because for two densities of mass one the deviation over A equals the deviation over its complement.

The estimate. With $\mu_{\varphi n}$ the empirical measure of the validation points, the *minimum distance estimate* (MDE) is the candidate with the smallest index θ^* that minimises

$$\Delta_\theta = \sup_{A \in \mathcal{A}_\Theta} \left| \int_A f_{n-\varphi n,\theta} - \mu_{\varphi n}(A) \right| \qquad \text{[ST19, eq. (3)]}.$$

For a class $\mathcal{A}$ of sets, write $\Delta(\mathcal{A}) = \max_{A \in \mathcal{A}} \left| \int_A f - \mu_{\varphi n}(A) \right|$.

Theorem 9.1 (Oracle inequality)**.** *For any finite set Θ of densities of mass one, in particular the training histograms $f_{n-\varphi n,\theta}$ below, and θ^* the minimum distance estimate over Θ with the validation points: Theorem 2 of [ST19], deterministic,*

$$\int \left| f_{n-\varphi n,\theta^*} - f \right| \leq 3 \min_{\theta \in \Theta} \int \left| f_{n-\varphi n,\theta} - f \right| + 4\Delta(\mathcal{A}_\Theta).$$

Its first term is the dyadic approximation number of f at the state's resolution, up to the sampling noise on the state's cells (Proposition 16.5). *Why 3 and 4* (the proof of [DL01, Theorem 6.3], in four lines). Fix $\theta \neq \theta^*$ and let $A = A_{\theta^*,\theta} \in \mathcal{A}_\Theta$. 1. By the triangle inequality, $\int |f_{\theta^*} - f| \leq \int |f_{\theta^*} - f_\theta| + \int |f_\theta - f|$. 2. Both candidates integrate to one, so Scheffé's identity gives $\int |f_{\theta^*} - f_\theta| = 2 \int_A (f_{\theta^*} - f_\theta) \leq 2\Delta_{\theta^*} + 2\Delta_\theta \leq 4\Delta_\theta$, since θ^* minimises. 3. Also $\Delta_\theta \leq \sup_A \left| \int_A f_\theta - \int_A f \right| + \Delta(\mathcal{A}_\Theta) \leq \frac{1}{2} \int |f_\theta - f| + \Delta(\mathcal{A}_\Theta)$. 4. Altogether, $\int |f_{\theta^*} - f| \leq 3 \int |f_\theta - f| + 4\Delta(\mathcal{A}_\Theta)$.

Nothing in the argument is random. So the inequality holds for any finite Θ, however chosen, provided θ^* minimises over the Yatracos class of that same Θ. With the finitely many candidates

the minimum is attained, so no additive term appears; the paper notes this against the book's $3/n$ [ST19, p.518]. Theorem 2 alone is not yet a guarantee: when Θ is chosen with the validation data, $\Delta(\mathcal{A}_\Theta)$ is not controlled.

Theorem 3 of the paper (in expectation) [ST19, Theorem 3, eq. (7)]. For all n, φn, finite Θ and $f \in L_1$,

$$\mathbb{E}\int |f_{n-\varphi n,\theta^*} - f| \le 3 \min_\theta \mathbb{E}\int |f_{n,\theta} - f| \Big(1 + \frac{2\varphi}{1-\varphi} + 8\sqrt{\varphi}\Big) + 8\sqrt{\frac{\log 2|\Theta|(|\Theta|+1)}{\varphi n}}.$$

The proof: - takes expectations in Theorem 2; - relates candidates built on $n - \varphi n$ points to those built on n [DL01, Theorem 10.2]; - bounds $\mathbb{E}\Delta$ by the shatter coefficient of $\mathcal{A}_\Theta$ [DL01, Theorem 3.1]. That coefficient is at most the number of sets in the class, which the paper bounds by $|\Theta|(|\Theta|+1)$ [ST19, eq. (6)]. - The $+1$ comes from Algorithms 2 and 4, which index Θ by the number of splits i, so that $i+1$ states give an $(i+1) \times (i+1)$ matrix. - The exact count of ordered pairs of $|\Theta|$ states is $|\Theta|(|\Theta|-1)$, and Yatracos's unordered class halves it. Every form needs $|\Theta| \ge 2$.

Where eq. (7) applies. The theorem needs candidates that are regular additive: each of the form $n^{-1}\sum_i K_\theta(x, X_i)$ for a fixed function K_θ (Theorem 10.2 of the book). The paper asserts that SRP histograms form such a class [ST19, p.518]. That holds when θ names a fixed tree, its leaf-depth string (p.516). It fails when θ names a position along a path grown from the same sample, as SEBTreeMC grows it. There the tree, and with it $\min_\theta \mathbb{E}$, depends on the sample. This monograph therefore does not claim eq. (7) for the candidates of Chapter 8, and states the following bound instead.

The bound this monograph uses. Let $P \supseteq \Theta$ be a set of states, with $|P| \le N_P$ for a bound N_P that is fixed given the training points (deterministic, or a function of them), such that the class $\mathcal{A}_P$ is fixed given the training points. Then: - $\mathcal{A}_\Theta \subseteq \mathcal{A}_P$, so $\Delta(\mathcal{A}_\Theta) \le \Delta(\mathcal{A}_P)$, deterministically; - given the training points, Hoeffding's inequality and a union bound over the $N_P(N_P-1)$ sets give $\mathbb{E}\,\Delta(\mathcal{A}_P) \le \sqrt{\log\big(2N_P(N_P-1)\big)/(2\varphi n)}$; - with Theorem 2, taking expectations:

$$\mathbb{E}\int |f_{n-\varphi n,\theta^*} - f| \le 3\,\mathbb{E}\min_{\theta\in\Theta}\int |f_{n-\varphi n,\theta} - f| + 2\sqrt{2}\sqrt{\frac{\log\big(2N_P(N_P-1)\big)}{\varphi n}}.$$

Here Θ is the set over which θ^* is finally selected. It needs no regular additivity, and it drops eq. (7)'s factor $1 + 2\varphi/(1-\varphi) + 8\sqrt{\varphi}$ by comparing with candidates built on $n - \varphi n$ points rather than n. The dominating class and its Hoeffding bound are v1's (`Prop-OneDelta` and `Rmk-Ties` of the 2026-11-04 preprint, to be carried into Chapters 10 to 11). Writing the bound in expectation, as above, is this monograph's.

Its hypotheses, case by case: - **The root box.** It must be fixed, or computed from the training points alone; validation points outside it count only through the total mass of $\mu_{\varphi n}$. If it is the hull of all n points, as in the MRS program `MDETest` and in the first phase of the Spark code at `c0a1837`, every state depends on the validation points through the box, and the conditional step fails. The same gap sits under the paper's Theorem 3. `CheckBounds` meets the hypothesis, since its root box is the fixed $[0,1]^d$. - **The path's end.** The paper chooses it with the validation data: the path "ends at a state with a large number of leaves and a significantly higher Δ_θ score than its preceding states" [ST19, p.520]. So P must be the whole path up to its deterministic cap, not the stretch the search used. - **The MRS search.** Each zoom iteration rebuilds the path from the root with the same random-number seed (the GSL default, unless `GSL_RNG_SEED` is changed), so the path is retraced. P is that path, with $N_P \le$ `CRITLEAVES`, and Θ is every

state scored over all iterations. - **The Spark search.** It keeps only its last round. So Θ is that round's candidates, and Theorem 2 holds round by round. Under v1's tie-breaking condition (T), P is one backtrack path. Without it, the code at `c0a1837` does not guarantee (T), and P is v1's set Q of all states the search can score, with $|Q| \leq (2k-2)^{R+1}$ over R rounds.

9.4 Computing it on pavings

Every quantity in Δ_θ is a sum over boxes, because the candidates are SRP histograms on one root box.

Collation. `SRPCollate` (Algorithm 3 of the paper) overlays the candidates' pavings on a *collator regular paving* (CRP). It is the union of Chapter 6, and each node holds the vector of all candidates' heights there [ST19, §3, Fig. 5]. Where a candidate's leaf meets a finer part of the CRP, its height is copied down to the temporary children. That is correct, because heights are intensive (Chapter 5).

Counts. Each node caches two counts as the data fall through: training $\#\mathbf{x}_{\rho v}$ and validation $\dot{\#}\mathbf{x}_{\rho v}$ (the paper's $\check{\#}$) [ST19, Fig. 4]. Then: - a candidate's height is $\#\mathbf{x}_{\rho v}/((n-\varphi n)\operatorname{vol}(\mathbf{x}_{\rho v}))$; - the validation measure is $\mu_{\varphi n}(\mathbf{x}_{\rho v}) = \dot{\#}\mathbf{x}_{\rho v}/\varphi n$.

The Yatracos matrix, updated one split at a time. `GetYatracos` (Algorithm 4) keeps a matrix whose entry (j, i) is the Yatracos set $\{f_j > f_i\}$ as a list of boxes. When the path splits box $\mathbf{x}_{\rho v^*}$ to make state i from state $i-1$, it updates row and column i from row and column $i-1$. For each earlier state j: 1. remove $\mathbf{x}_{\rho v^*}$ from the entry; 2. add whichever of its two children satisfy the comparison.

Why this is exact, explained here and not in the paper: - The states on one path only refine, so $f_i = f_{i-1}$ outside $\mathbf{x}_{\rho v^*}$. - Every earlier, coarser f_j is constant on $\mathbf{x}_{\rho v^*}$, a leaf of state $i-1$. - By induction, every entry of column $i-1$ is a set of leaves of state $i-1$, so $\mathbf{x}_{\rho v^*}$ is either an element of the entry or disjoint from it.

The paper confines the selection to states along one path "to take advantage of the structure of consecutive refinements" [ST19, §5].

In MRS this update is `getYatracosClassAll`, used only by the variants that score every state of the path. They put ties ($f_j = f_i$) into the column set. That is harmless, since Scheffé's identity holds for any set between $\{g > h\}$ and $\{g \geq h\}$, but it is not the strict class of eq. (3). The zoom-in and `CheckBounds` runs that produced the paper's results instead recompute every Scheffé set pairwise on the collation of the chosen states (`getMinDistEst`) [SYH+18]. What the single path buys them is nesting: every coarser candidate is constant on each leaf of the finest.

The objective. `GetDelta` (Algorithm 2) computes each Δ_θ as a maximum over the sets in the matrix of $\left|\sum_{\mathbf{x}\in A}\left(f_\theta(\mathbf{x})\operatorname{vol}(\mathbf{x}) - \mu_{\varphi n}(\mathbf{x})\right)\right|$. When a new candidate joins, it updates the old Δ_θ with the new row and column only.

The matrix grows quadratically with $|\Theta|$, which is why Θ is kept small [ST19, §4.1].

9.5 The zoom-in search

Taking every state of the path as a candidate would make $|\Theta|$ as large as $\overline{m}$. Instead the paper searches adaptively, with $10 \leq k \leq 20$ candidates per iteration [ST19, §4.1]: 1. Start with k states spaced uniformly along a long SEBTreeMC path, from the root to a state chosen as quoted in §9.3. 2. Zoom in around the states with the lowest Δ_θ, adding k more states between them. 3. Repeat until no further zoom is possible.

The paper reports near-optimal states within five iterations or fewer, and $|\Theta|$ typically below 100.

In the C++ code, `mrs2/companions/mrs-1.0-YatracosThis` ("gat41's thesis branch from mrs-1", its README), the parameters are named [SYH+18]: - `NUM_CHECKS`: candidates per iteration; - `NUM_ITERS`: the maximum number of iterations.

The first iteration collates `NUM_CHECKS` + 1 states, including the root. The range spanned by the three lowest Δ_θ is then split into `NUM_CHECKS` equal steps. The search stops after `NUM_ITERS` iterations or when the step is one leaf. Candidates of earlier iterations stay in Θ, and the final selection reruns the MDE over all of them (README, Chapter 3; `MDETest`).

`MDETest` then refits: it grows a new path on all n points and stops it at the selected number of leaves. The guarantees of §9.3 cover the training-sample candidate $f_{n-\varphi n,\theta^*}$, not this refit.

The Spark port, Graner's `getMDE`, keeps the idea but changes the bookkeeping [Gra22]. - Each round takes about k equally spaced states (at most $2k-2$, from integer division of the step) and keeps the winner and its next larger neighbour. - It zooms while the gap between them exceeds $k/2$. - If the gap then exceeds one, a last round scores the $2 \cdot \text{gap} + 1$ consecutive states. - Each round's Θ is fresh, and the estimate is the last round's winner.

Chapter 10 gives the search with its distribution over machines and its round bound.

9.6 Simulation and the theorems: `CheckBounds`

The paper does not stop at proving Theorems 2 and 3. It checks both on simulated data [ST19, §4.2, Fig. 6]. The paper points to the program `MDETest`; the program that computes the two sides is `CheckBounds`, in the same C++ package, called by `RunCheckBounds.sh` [SYH+18]. Its procedure: 1. **A known density.** Build a known density as a piecewise-constant function on a paving, by the method of Chapter 3: interval enclosures, a priority-queue split, hull propagation, and a priority merge to at most `MAXLEAVESEST` leaves. The package's set-up uses the uniform density on the fixed root box $[0,1]^d$; another density takes two lines (its README, Chapter 4). 2. **Data.** Simulate data from it. 3. **The MDE.** Run the MDE once, without zooming, over the `NUM_CHECKS` + 1 states with $1, c, 2c, \ldots,$ `CRITLEAVES` leaves, where $c = \lfloor$`CRITLEAVES`$/$`NUM_CHECKS`$\rfloor$: eleven states with the defaults. The code holds out the first round(φn) points, against the paper's $\lfloor \varphi n \rfloor$. 4. **Values.** The integrated absolute errors are computed by paving arithmetic (Chapter 6). The program returns: - `IAEforMinDelta`: the error of the MDE; - `minIAE`: the best error among the candidates built on $n - \varphi n$ points; - `maxDelta`: meant to be $\Delta(\mathcal{A}_\Theta)$ of Theorem 2; - `minIAEAllPoints`: the best error along a separate path grown from all n points, at the same leaf counts. Its mean over replicates estimates $\mathbb{E}\min$, not eq. (7)'s $\min \mathbb{E}$; that is the conservative direction. 5. **Theorem 2** is checked replicate by replicate: `IAEforMinDelta <= 3*minIAE + 4maxDelta`, written as a Boolean column. 6. **Theorem 3** is checked on means over replicates. The package's R scripts read five replicates and plot each right-hand side against its left-hand side.

`maxDelta` is not Δ. The function that computes it, `getMappedFunctionDelta`, sets its two accumulators, the integral of f and the validation mass, once before the loop over the Yatracos sets and never resets them (`adaptivehistogramvalidation.cpp` l.4929–4948). So its k-th value is the absolute running sum of the deviations of the first k sets, and `maxDelta` is the largest such sum. That can exceed $\Delta(\mathcal{A}_\Theta)$ or fall below it.

The published Theorem 2 panel is consistent with this. For the uniform target the root candidate is exact, so `minIAE` $= 0$. Since $\Delta \le 1$, the right-hand side minus the left-hand side is then at most 4, yet Fig. 6 (left) shows values above 5 near $n = 10^2$. The README's own sample output

implies `maxDelta` ≈ 1.35.

So the published check of Theorem 2 does not test Theorem 2. The theorem and its proof are untouched. Resetting the two accumulators for each set repairs the code, and the check is to be rerun with a per-set Δ; the simulation node of this monograph does so before Theorem 2 carries its simulation mark. The Theorem 3 panel uses only `IAEforMinDelta` and `minIAEAllPoints`, so the fault does not reach it.

Δ involves $\int_A f$ for the true f over every Yatracos set. It is computable only because both f and the sets are pavings, so the integral is the arithmetic of Chapter 6.

In the paper's runs, on uniform densities in 1, 10, 100 and 1000 dimensions (the package's scripts use $n = 1.5 \cdot 10^k$, $k = 2, \ldots, 7$, plotted as 10^k), the paper reports the right-hand side minus the left-hand side as positive and decreasing with n. It calls the bounds "not sharp", because "they are extremely general by construction and based merely on the cardinality of the set of candidate densities" [ST19, §4.2].

This is the habit described in Chapter 1, of checking each theorem by simulation. The final algorithm's theorems carry the same mark (Chapter 10, Chapter 14), each from a check that has itself been checked.

9.7 What the estimator does

Uniform densities on $[0,1]^d$, for $d \in \{1, 10, 100, 1000\}$ and n from 10^2 to 10^7 (1000 dimensions up to 10^6, limited by memory). The MIAE over five replicates is about the same in every dimension, because the true density is the root paving, the first candidate [ST19, Table 1].

Gaussian densities in 1, 2 and 5 dimensions, and Rosenbrock densities in 2 and 5 [ST19, Table 2]: - The paper calls the MDE's errors comparable to those of the posterior-mean histogram of §7.5. The posterior mean is lower in 17 of the 20 cells of the table, by a factor of up to 1.84. - The paper states, without reporting MDE timings, that the MDE is "at least an order of magnitude faster". The MDE also carries the guarantee. - For highly structured densities beyond about ten dimensions, no reasonably sized candidate on a regular paving is close to f. The paper proposes reducing the dimension first, by an auto-encoder for instance.

Bot flows [ST19, §4.3]. The data are scenario 11 of the CTU-13 data set: 8164 botnet flows mixed with 99087 normal and background flows [GGSZ14]. - The flows were embedded in 80 dimensions and reduced to 8 by an auto-encoder trained on normal flows [Ram19]. - The MDE histogram was built from the 99087 normal flows. Whether the histogram used was `MDETest`'s refit (§9.5) is to be checked. - Each of the 107251 flows was then scored by its tail probability: one minus the mass of the leaves whose height exceeds the height at the flow [HST12, Algorithm 9].

Of the 8164 botnet flows, 7115 (87%) fell within the lowest 7.6% of tail probabilities, and 8090 (99.1%) within the lowest 10%.

The paper states who did what, and that the work was funded in part by the Marie Curie Actions project CORCON (FP7, grant 612638) [ST19, Acknowledgements, Funding].

The next step, named in 2019. The paper's last sentence reads: "the pure tree structure allows one to possibly extend this MDE to a distributed fault-tolerant computational setting such as Zaharia et al. (2016) as the sample size becomes too large for the memory of a single machine" [ST19, §5]. Zaharia et al. (2016) is the Apache Spark paper [ZXW+16]. Chapter 10 is that step.

Chapter 10

The final distributed algorithm

This section specifies, once, the distributed L_1 minimum distance estimator this monograph stands behind. Each refinement is a design choice of the algorithm, and Chapter 11 gives its exactness lemma; there is no separate list of improvements. Chapter 6 gave each operation's Spark form and the law it owes. Chapter 14 runs the algorithm.

10.1 Where it comes from

The distributed line has three steps, each in its own paper or thesis. - Sainudiin, Tucker and Wiklund built top-down histograms on Apache Spark over integer node labels, the heap code of Chapter 4 [STW20]. - Graner, starting from that work and its code, built the bottom-up merge, the sparse arithmetic, and the Spark port of the minimum distance search [Gra22]. - Sandstedt added subtree partitioning, longest-processing-time assignment and partition-local counting of the validation points in the Yatracos sets, and ran the terabyte runs [San23].

The library is [SGWS23a], at commit `c0a1837`. The final algorithm below keeps its structure and changes the choices that the analysis of Chapters 11 to 13 shows to matter.

Throughout, $n' = n - \varphi n$ is the training size, $\overline{\#}$ the count limit, and μ the law of the data, with density f.

10.2 The pipeline

Phase 0: the data. - The data are read or generated once and persisted, so that no pass regenerates them (Chapter 13, layer 3). - They are split into n' training points and φn validation points before anything else is computed. - The split must not depend on how the data are partitioned. Spark's `randomSplit` assigns points by partition and by position within it, so its result depends on the partitioning, and it is stable across recomputation only on a persisted or deterministically ordered parent. Two splits are partition-invariant: - a function of a stable point id, such as a hash of it. Then φn is random, and the guarantees hold conditionally on it; - generating the two samples separately, with separate seeds, as the terabyte runs did.

Phase 1: the root box. - The root box is the box hull of the *training* points, a reduction by the hull, which is associative, commutative and idempotent. - Validation points outside the box get no label. They count only through φn, the total mass of $\mu_{\varphi n}$. Point location must not clamp them into a boundary cell. - This is what lets the guarantees of §9.3 condition on the training points. The code at `c0a1837` takes the hull of all points, validation included, and so does not meet the hypothesis.

Phase 2: depth, labels and counts. Everything in this phase is computed from the training points. - *The depth.* The finest depth is the shallowest D satisfying (G) (Chapter 12): every depth-D cell holds at most $\overline{\#}$ training points. 1. A first pass counts the training points at a coarse depth D_0. 2. If D_0 satisfies (G), the shallowest $D \le D_0$ is found by coarsening these counts, with no new pass. 3. Otherwise a second pass labels the points of the over-full cells at the deepest representable depth. For each depth d, the largest count of a depth-d cell is then a scan of the sorted labels, and the shallowest D follows, since "(G) holds at d" is monotone in d (Lemma 8.3(iii); §12.1). 4. If no representable depth satisfies (G), the deepest representable one is used. Its over-full cells are kept as atoms, the merged paving is then π^*_D of Lemma 11.7(iii), a function of the counts alone, and the report says so in the words of §12.3 (Chapter 2). - *The labels.* A label is a fixed-width integer: one 64-bit word for $D \le 62$, two for $63 \le D \le 126$. It replaces the arbitrary-precision label, with the same labels in the same order (Chapter 11). A (label, count) record then takes about 16 B in a primitive or columnar layout, about 24 B as a Spark row, and about 144 B with an arbitrary-precision label on the heap (Chapter 13, layer 1). - *The counts.* Each training point maps to its depth-D label (Chapter 6, point location). Each map partition combines its counts per label, and the reduce side adds the partial counts. The code at `c0a1837` does the same; the gain is the shallow depth, at which far fewer keys are distinct.

Law: the pushforward is a monoid homomorphism (Chapter 6, `Prop-Relabel`), so the counts do not depend on the partitioning. The shuffle carries at most one record per map task and occupied cell. - *The validation points* are labelled at depth D and not combined, so that each keeps its own record.

Phase 3: the merge. - *Partitioning.* Either of two partitioners; each makes a depth limit δ safe (Chapter 11, `Cor-Safe`), and each partition's records are sorted in left-right order. - *Range partitioning*, when the count map is small enough to rank exactly by a sort and an index. Spark's own range partitioner chooses its bounds from a sample, so it does not give exact ranks. Each partition is then a contiguous run of labels (Chapter 4, `Prop-Block`). For boundary i, let c_i be the deepest common ancestor of the last label of partition i and the first of partition $i+1$. Set $\delta = 1 + \max_i |c_i|$, or $\delta = 0$ for one partition.

```
*Why $\delta$ is safe.* A node deeper than every $c_i$ contains no boundary pair.
  So its labels form a contiguous run inside one partition (`Prop-Block`(vi)).

With exact ranks on dense data, the labels across a boundary are often siblings
  at depth $D$. Then $\delta$ is $D$ or close to it, the local walks merge
  nothing, and the work falls on the driver. Snapping each boundary to the
  nearest boundary of a node at a chosen depth $\delta_0$ gives
  $\delta\le\delta_0$, at the price of balance at the granularity of
  depth-$\delta_0$ nodes. If the whole count map fits on the driver, the driver
  walk alone suffices.
```

- *Subtree partitioning*, otherwise (§10.3).
- *The number of partitions* is sized to bytes:

$$\left\lceil \beta \times \text{records} \times \text{bytes per record} \times \text{working-space factor} \,/\, \text{memory per concurrent task} \right\rceil$$

  Here β is the balance factor of the partitioner: 1 for exact range partitioning, which then leaves a task at most one record over its memory (Proposition 13.4), and the factor of Proposition B.9 for subtree partitioning (Chapter 13, layer 2).
- *The walk.* Each partition runs the merge walk $W(\delta)$ of Chapter 11 (`Lem-Merge`) on its labels in left-right order. It *streams* over the sorted iterator instead of holding the partition

in memory, so the sort before it can spill.

To decide whether to extend the current subtree, the walk needs the leaves of its next ancestor that lie ahead of the iterator. With counts at least 1, at most $\overline{\#}+1$ of them decide it. So the streaming walk keeps:

- a look-ahead of at most $\overline{\#}+1$ records;
- the label of the last emitted subtree, for the overlap test;
- a one-subtree delay, for raising.

Chapter 11 proves that it emits exactly what $W(\delta)$ emits on the materialised list. A partition never relabels its output as the root.

- *The driver.* It collects the local outputs, sorts them, and *always* finishes with the walk $W(0)$ on them. This costs one linear walk over records that are collected anyway.

  The code at `c0a1837` runs the driver walk only when some partition's depth limit refused a join (`MergeEstimator.scala` l.270, l.320). That trigger misses cases. Take four depth-2 cells with one point each, count limit 10, and two subtree partitions $\rho\mathsf{L}$ and $\rho\mathsf{R}$, so $\delta = 1$. Each partition merges its two cells and emits its root at depth 1, and no join is refused. The driver walk does not run, and the output is $\{\rho\mathsf{L}, \rho\mathsf{R}\}$ where the threshold paving is the root.

- *Exactness:* `Lem-Merge` and `Cor-Safe` of Chapter 11, with δ as above. Without (G), it is `Lem-MergeNoG`.

Phase 4: the selection. - *The candidates* are states of the backtrack path P from the merged histogram toward the root. - Each backtrack step merges the cherry of least count, ties broken by the heap code of its label. The heap-code integer order is total on all nodes, so this gives condition (T) (§10.4): every scored state lies on the one path P. As an option, the step merges instead the cherry of least $|a-b|$, the difference of its children's counts (the arc length of the step, Lemma 10.3(i)), with the same tie-break: (T) holds as before, the path is still a function of the training counts, and for a density that is itself a paving this path reaches it where the count rule need not (Theorem 16.7, Example 16.4). The runs of §14.1 record which rule was used. - *The search* is the gap search of §10.4: - a first round spaced by arc length on P, containing P's two ends; - then bisection, at the arc-length midpoint, of every gap that is not settled; - one final selection over all the scored states. - *What a round scores.* It scores its new states together with the end-points of their gaps, over the Yatracos class of that set. A round with one new state therefore still has a non-empty class. - It collates its states on their common refinement (Chapter 6, `Dfn-Collation`). It skips the leaves common to the coarsest and the finest of them: under (T), such a leaf is a leaf of every state in between, with one count, so it lies in no Yatracos set. It then minimises the leaves of the union (Chapter 6, MinimiseLeaves). - *The records it scans.* Round 1 contains s^* and the root, so nothing is skipped in it unless s^* is the root paving, and its scan returns each validation record at the leaf of s^* that contains it, *before* MinimiseLeaves (Theorem 11.12(iii)). From those records, one distributed pass computes the leaf counts κ^*, the number of validation points in each leaf of s^* (the pushforward of Theorem 11.12(iv), at most $|\mathcal{L}(s^*)|$ entries), and every later round is scored on the driver from κ^* and the refinement, with no further scan. The alternative, the round-1 records persisted and reused unchanged, scans the validation data once per round and gives the same counts (Theorem 11.12(iii)). Either way no leaf of s^* strictly contains a cell of a refinement of states of P, so the record condition (R) of Theorem 11.12 holds in every round. Handing on records merged to the previous round's refinement, as the code at `c0a1837` does for its zoom, would break (R) for rounds that jump along P (Remark 11.13(a)). - *The counts.* Each partition counts the validation points in every Yatracos set into long integers and returns its count matrix as data. The matrices are combined by `treeAggregate`, so each partition's counts enter once, whatever Spark re-executes.

Accumulators updated in a transformation do not have this property. The total passed is φn, including the validation points outside the root box. Chapter 11 (`Thm-DSSC`) states when the counts are exact. - The sorts on the validation path must spill too; they replace the code's in-memory `toArray` sorts. - *The report:* - the estimate; - the parameters ξ, ϵ_0 and $\overline{\Delta}$; - $|P|$ and the failure bound $2|P|^2 e^{-2\varphi n \overline{\Delta}^2}$; - the depth D, the representable depth $D_{\max}$, and whether (G) held; when it held, D^{G}; when it failed, the number of over-full depth-D cells and the largest count, the warning of §12.3 verbatim, and the number of leaves of π^*_D raised over empty siblings; - whether the winner is the finest state (§12.5).

Storage. Labels are stored sorted, in a columnar fixed-width integer column, so that a reload reads them in order. Whether this shortens the merge's sort is to be measured: Spark sorts again on the shuffle, though its sort is fast on sorted runs.

10.3 Subtree partitioning

Subtree partitioning chooses the partitioner of Phase 3 so that whole subtrees share a partition [San23]. It has four steps, all in `SubtreePartitioner.scala` at `c0a1837`. The proofs are in Appendix B, except for Conjecture B.10, which is a conjecture. Below, λ is the cost of one label operation (Chapter 6, `Dfn-Cost`). 1. **Sample.** Each input partition is sampled by reservoir sampling. A sampled label gets the weight (records of its partition)/(its sample size); a label from a resampled, imbalanced partition gets 1/(its sampling fraction) (l.109–115). The weights estimate numbers of occupied finest cells, not of points. 2. **Maximal-weight subtrees.** Let W be the total sampled weight and m the requested number of partitions. The merge walk $W(0)$ of Definition 11.1, run with the weights in place of counts and the limit $\overline{w} = W/m$, emits subtrees. Each is raised to its highest ancestor that overlaps no other emitted subtree. A sampled label whose weight alone exceeds $\overline{w}$ is an over-full atom; the walk keeps it as a subtree of its own (Lemma 11.7(ii)), and such a subtree may carry more than $\overline{w}$.

The output is an antichain of subtree roots in left-right order. Each root carries the weight of its sampled cells, which is at most $\overline{w}$ unless the subtree is a single label.

On N sampled labels the walk costs $O(\lambda N \log N + N \min(D, N))$ word operations, and prefix sums would make it $O(\lambda N \log N)$ (Theorem B.3). Whether to use them is left to the measurements of Chapter 13. 3. **Longest-processing-time assignment.** Let the subtree weights be $p_1 \geq p_2 \geq \cdots$, with total p_{tot}, and let C^* be the least possible largest load over m partitions. The subtrees are taken in decreasing order of weight, and each is given to a partition of least current load. Whatever the tie-breaking (Theorem B.6): - the largest load is at most $(\frac{4}{3} - \frac{1}{3m})C^*$, the bound for this order [Gra69]; - it is at most $\frac{p_{\mathrm{tot}}}{m} + (1 - \frac{1}{m}) \max_k p_k$, the bound of list scheduling in any order [Gra66].

When every sampled weight is at most $\overline{w}$, there are at least m subtrees, and the largest sampled load is at most $\min\{(\frac{4}{3} - \frac{1}{3m})C^*, (2 - \frac{1}{m})W/m\}$ (Corollary B.7).

These bound the *sampled* weights. The transfer to the cells a partition actually receives is Proposition B.9, which rests on a conjecture about the sample (Conjecture B.10). 4. **Lookup.** Each record goes to the partition of `findSubtree` of its label, a binary search over the subtree roots. It makes at most $\lfloor \log_2 S \rfloor + 1$ probes for S subtrees. It returns the subtree containing the label when there is one, and otherwise the neighbour whose common ancestor with the label is deepest. Its result depends on the label only through its ancestor at depth δ, the largest depth of a subtree root (Theorem B.8).

So, with δ the largest depth of a subtree root and the records of each partition sorted in left-right order, subtree partitioning makes its depth limit safe (Chapter 11, `Cor-Safe`).

The sample is capped. The code takes $\min(\text{hint} \cdot p, 10^6)$ labels for p partitions and then oversamples three-fold, so about $3 \cdot 10^6$ labels are collected in all (`SubtreePartitioner.scala` l.86–88 at `c0a1837`). That is about 183 per partition at 16 384 partitions (the two-dimensional terabyte runs) and about 366 at 8192 (the ten-dimensional ones). With a hint of 1000, the uncapped figure would be 1000–3000 per partition; the hint's value in the terabyte notebooks is only partly verified. Range partitioning with exact ranks replaces the sample by the exact order, and removes the conjecture from the balance argument.

Marks: Theorem B.3, Theorem B.6, Corollary B.7, Theorem B.8 and Proposition B.9 (given Conjecture B.10) are proved in Appendix B. [LEAN: Lem-Raise and Thm-Find(iv) built, Appendix B] [SIM: not yet run; a randomised check of Theorem B.8 and of the LPT bounds on random weights is planned.] Conjecture B.10 is open.

10.4 The search and its guarantees

The search selects among states of the backtrack path. The guarantee of §9.3 holds for any set of scored states. This subsection gives the stronger guarantee of the search the final algorithm runs.

Write $s^*_{n'}$ for the merged histogram, and $P = (\theta_0, \dots, \theta_H)$ for the backtrack path from $\theta_0 = s^*_{n'}$ to the root θ_H, so $|P| = H + 1 = |\mathcal{L}(s^*_{n'})|$.

(T) is the condition that each backtrack step merges the least cherry in a fixed total order. Ordering by the pair (count, heap code) gives it.

The conditions on the run. The results below assume: - (T); - the root box, the depth D and the count limit are computed from the training points, or fixed in advance; - the split does not depend on the values of the points, and any randomness of the merge is independent of the validation points; - the collation is fixed: every collation step returns the common refinement of its operands (Chapter 11); - the validation counts are exact (Chapter 11, `Thm-DSSC`), with the total φn; - the integrals and scores are exact. The specification computes them in integer arithmetic on the driver: every integral is an integer over $n'2^D$ and every empirical measure an integer over φn (Corollary 11.17, "the scores can be exact"), so $\varepsilon_{\text{fp}} = 0$. If they are computed in floating point and the computed winner minimises Δ_θ only up to ε_{fp}, add $2\varepsilon_{\text{fp}}$ to Theorem 9.1 and Proposition 10.1, and $(2 + 3\xi)\varepsilon_{\text{fp}}$ to Theorem 10.4 (Corollary 11.17).

Proposition 10.1 (One deviation for every round)**.** *Assume the conditions on the run. Let: - $\mathcal{A}_P$ be the Yatracos class of all the states of P, at most $|P|(|P| - 1) \le |P|^2$ sets, and $\Delta_P = \max_{A \in \mathcal{A}_P} |\mu(A) - \mu_{\varphi n}(A)|$; - $\Theta_r \subseteq P$ be the states scored in round r, chosen by any rule, possibly using the validation points, and θ_r the minimum distance estimate over Θ_r.*

Then for every round r,

$$\int |f_{n',\theta_r} - f| \le 3 \min_{\theta \in \Theta_r} \int |f_{n',\theta} - f| + 4\Delta_P.$$

Given the training points, $\mathbb{P}\{\Delta_P > t\} \le 2|P|^2 e^{-2\varphi n t^2}$ for every $t > 0$, and $\mathbb{E}\,\Delta_P \le \sqrt{\log(2|P|^2)/(2\varphi n)}$.

Proof. 1. **One path.** Under (T) the next backtrack state is a function of the current one, so every scored state lies on P. 2. **A fixed class.** By the conditions on the run, P is determined by the training points and by randomness independent of the validation points. So is $\mathcal{A}_P$. 3. **The inequality.** With the collation fixed, each round scores its states on their common refinement. Theorem 2 of §9.3 (`Thm-Oracle`) then holds with the deviation over the Yatracos class of the

states compared. That class lies in $\mathcal{A}_P$, so its deviation is at most Δ_P, whatever rule chose Θ_r. 4. **The bounds.** Given the training points, the validation points are i.i.d. from μ. Hoeffding's inequality and the union bound over at most $|P|^2$ sets give the tail. The maximal inequality for at most $2|P|^2$ sub-Gaussian variables with variance proxy $1/(4\varphi n)$ gives the expectation. □

Remark 10.2 (Ties in the backtrack)**.** The code at `c0a1837` does not guarantee (T). - *Why.* `spacedBacktrack` computes each candidate from the previous one by a fresh call of `backtrackNumSteps` (`MinimumDistanceEstimation.scala` l.142). Each call builds a new priority queue from the cherries of its current state, ordered by count alone (`Histogram.scala` l.184–190). Ties between cherries of equal count are therefore broken by the layout of a heap rebuilt at each call, and a later round can leave P. Counts tie often on sparse data. - *The fix.* Ordering by (count, heap code) gives (T). - *Without (T).* The bounds hold with P replaced by the set Q of all states the code's zoom can score, over all possible sequences of winners. Given the training points the code is deterministic, so Q is determined by the training points, *provided the root box is*, which the code at `c0a1837` does not ensure. With k candidates per round, $k \geq 4$ and at least k stored leaves, each round scores at most $2k-2$ candidates and there are at most R rounds, a bound depending only on k and the leaves of $s^*_{n'}$ (Chapter 11, `Lem-Rounds`); with $k = 3$ the zoom can loop forever. So $|Q| \leq (2k-2)^{R+1}$, and $\log|Q|$ is still logarithmic in $|P|$.

`Prop-OneDelta` makes the concentration term independent of how the rounds were chosen. The approximation term is another matter. A round's winner is compared only with that round's best state. A bound against earlier rounds would need each round to contain the previous winner, which the code at `c0a1837` does not ensure. Its rounds start at the winner's finer neighbour and step by $\lfloor 2\,\mathrm{gap}/(k-1)\rfloor$ merges (`MinimumDistanceEstimation.scala` l.419–422, l.602–603). With $k = 10$ and 1000 stored leaves, round 1 steps by 111 merges and round 2 by 24, so the old winner is not a candidate. The code's estimate is therefore compared only with its last round's candidates. The geometry of the path gives a bound against every state of P. This is the skeleton estimate of [DL01, Chapter 7], with the path as the model.

Write $f_t = f_{n',\theta_t}$ and $\mathrm{err}(t) = \int |f_t - f|$. For a class $\mathcal{A}$, write $\Delta(\mathcal{A}) = \sup_{A\in\mathcal{A}} |\mu(A) - \mu_{\varphi n}(A)|$, and $\Delta_t(\mathcal{A}) = \sup_{A\in\mathcal{A}} |\int_A f_t - \mu_{\varphi n}(A)|$ for the score of θ_t.

Lemma 10.3 (The path in L_1)**.**

- *(i) A backtrack step that merges a cherry whose children hold a and b training points changes the histogram only on the cherry, and* $\int |f_t - f_{t+1}| = |a-b|/n'$ *($b = 0$ for a lone child).*
- *(ii) With* $\mathrm{arc}(t,t') = \sum_{t\leq i<t'} \int |f_i - f_{i+1}|$*, we have* $|\mathrm{err}(t) - \mathrm{err}(t')| \leq \mathrm{arc}(t,t')$ *and* $\mathrm{arc}(0,H) \leq D$*.*
- *(iii) If t lies between u and u' on P, then* $\mathrm{err}(t) \geq \frac{1}{2}\big(\mathrm{err}(u) + \mathrm{err}(u') - \mathrm{arc}(u,u')\big)$*.*
- *(iv) For every state θ_u and every class $\mathcal{A}$,* $\mathrm{err}(u) \geq 2\big(\Delta_u(\mathcal{A}) - \Delta(\mathcal{A})\big)$*.*

Proof. - (i) The children are the two halves of the cherry c. Before the step their densities are $2a/(n'\mathrm{vol}(c))$ and $2b/(n'\mathrm{vol}(c))$; after it, $(a+b)/(n'\mathrm{vol}(c))$ on both. Each half contributes $|a-b|/(2n')$. - (ii) The first claim is the triangle inequality. Each internal node of $s^*_{n'}$ is merged once and contributes at most its count over n', and the internal nodes at one depth are disjoint. That gives at most D. - (iii) Average $\mathrm{err}(t) \geq \mathrm{err}(u) - \mathrm{arc}(u,t)$ and $\mathrm{err}(t) \geq \mathrm{err}(u') - \mathrm{arc}(t,u')$, using $\mathrm{arc}(u,t) + \mathrm{arc}(t,u') = \mathrm{arc}(u,u')$. - (iv) f_u and f integrate to one, so for $A \in \mathcal{A}$, $\int |f_u - f| \geq 2|\int_A f_u - \mu(A)| \geq 2\big(|\int_A f_u - \mu_{\varphi n}(A)| - |\mu_{\varphi n}(A) - \mu(A)|\big)$. □

By (iv), the scores the search computes are lower bounds on the true errors, up to the deviation. There is no bound on the error that tends to zero uniformly over densities, since L_1 density estimation has no universal rates of convergence [DG85]. So the search uses lower bounds only.

The gap search. - *Gaps.* Let $U \subseteq P$ be the states scored in all rounds, containing θ_0 and

θ_H. A *gap* J is the stretch of P strictly between two consecutive states u, u' of U, of arc length $\mathrm{arc}_J = \mathrm{arc}(u, u')$. - *The event.* Fix $\overline{\Delta} > 0$ given the training points, before the validation points are seen, and let $\mathcal{E} = \{\Delta(\mathcal{A}_P) \le \overline{\Delta}\}$. Then $\mathbb{P}(\mathcal{E}^c) \le 2|P|^2 e^{-2\varphi n \overline{\Delta}^2}$. - *A lower bound in each gap.* Let $\Delta_u = \Delta_u(\mathcal{A})$ for any class $\mathcal{A} \subseteq \mathcal{A}_P$ over which θ_u has been scored; a larger class gives a larger score and a tighter bound. By `Lem-Arc` (iii) and (iv), on $\mathcal{E}$ every state of J has error at least

$$\underline{\mathrm{err}}_J = \Delta_u + \Delta_{u'} - 2\overline{\Delta} - \tfrac{1}{2}\mathrm{arc}_J.$$

- *Settled gaps.* Fix $\xi > 0$ and $\epsilon_0 \ge 0$; unlike $\overline{\Delta}$, these may be chosen after the scores are seen. A gap J is *settled* when it holds no state, or when $\frac{1}{2}\mathrm{arc}_J \le \max(\xi\, \underline{\mathrm{err}}_J, \epsilon_0)$.

Theorem 10.4 (The gap search)**.** *Assume the conditions on the run. Let every gap of U be settled, and let $\hat{\theta}$ be the minimum distance estimate over U with the Yatracos class of U. Then on $\mathcal{E}$, so with probability at least $1 - 2|P|^2 e^{-2\varphi n\overline{\Delta}^2}$ conditionally on the training points, for every density f,*

$$\int |f_{n',\hat{\theta}} - f| \le 3(1+\xi) \min_{\theta \in P} \int |f_{n',\theta} - f| + 3\epsilon_0 + 4\overline{\Delta}.$$

Proof. 1. $U \subseteq P$, so Theorem 2 of §9.3 gives $\int |f_{n',\hat{\theta}} - f| \le 3 \min_{t \in U} \mathrm{err}(t) + 4\overline{\Delta}$ on $\mathcal{E}$, whatever rule chose U. 2. Let t^* minimise err on P. If $\theta_{t^*} \in U$, then $\min_U \mathrm{err} = \mathrm{err}(t^*)$. 3. Otherwise t^* lies in a gap J, whose nearer end is within arc length $\frac{1}{2}\mathrm{arc}_J$. So $\min_U \mathrm{err} \le \mathrm{err}(t^*) + \max(\xi\, \underline{\mathrm{err}}_J, \epsilon_0)$, and $\underline{\mathrm{err}}_J \le \mathrm{err}(t^*)$ on $\mathcal{E}$. 4. Hence $\min_U \mathrm{err} \le (1+\xi)\mathrm{err}(t^*) + \epsilon_0$, and step 1 gives the claim. □

What the theorem gives: on $\mathcal{E}$, the estimate's error is at most $3(1+\xi)$ times that of the best state of the whole path, plus $3\epsilon_0 + 4\overline{\Delta}$. It does not bound the error itself, and $\min_P \mathrm{err}$ is unknown.

Corollary 10.5 (The gap search is consistent)**.** *Assume the conditions on the run, for the pipeline of §10.2 as specified, with no representable cap on the depth. For each n let the training sample be any i.i.d. sample of size $n' = n - \varphi n$ from μ (the samples of different n need not be nested, Remark 8.16), with a count limit $\overline{\#}_{n'} \ge 1$ such that $\overline{\#}_{n'}/n' \to 0$, and let one of the following hold: - (a) the root box is fixed and contains the support of f; the finest depth is the (G)-depth $D^{\mathrm{G}}_{n'}$ of the training sample (Phase 2); and there are deterministic d_n with $D^{\mathrm{G}}_{n'} \le d_n$ for all large n with probability one and $d_n/\overline{\#}_{n'} \to 0$; - (b) the support of f is bounded, the root box is the training hull (Phase 1), the finest depth is $D^{\mathrm{G}}_{n'}$, and d_n is as in (a) with $d_n \log n/\overline{\#}_{n'} \to 0$; - (c) the finest depth is a deterministic $D_{n'} \to \infty$ with $D_{n'} \le \overline{\#}_{n'}$, satisfying the growth conditions of Corollary 8.11 for a fixed root box containing the support, or those of Corollary 8.14 for the training hull and a bounded support.*

Let ξ be fixed, $\epsilon_{0,n} \to 0$, and $\overline{\Delta}_n = \sqrt{\log(2(n+1)^2 n^2)/(2\varphi n)}$. Then the gap search's estimate satisfies $\int |f_{n',\hat{\theta}} - f| \to 0$ with probability one. By Corollary 8.15(iii), (a) and (b) hold for every bounded density with bounded support when $\overline{\#}_{n'} = n'^{\alpha}$ with $0 < \alpha < 1$ and $d_n = \lceil b \log_2 n \rceil$ with $b > 1 - \alpha$.

Proof. 1. *The size of P.* $|P| = |\mathcal{L}(\theta_0)|$, the number of leaves of the merged histogram. In (a) and (b), (G) holds at the (G)-depth by definition (it is finite with probability one, the points being distinct), so $\theta_0 = s^*$ and $|P| \le 1 + D^{\mathrm{G}}_{n'}\lfloor n'/(\overline{\#}_{n'}+1)\rfloor \le 1 + d_n\lfloor n'/(\overline{\#}_{n'}+1)\rfloor$ for all large n (Lemma 8.3(iii)); in (c), θ_0 is $s^*_{D_{n'}}$ or $\pi^*_{D_{n'}}$, and $|P| \le 1 + D_{n'}\lfloor n'/(\overline{\#}_{n'}+1)\rfloor$ (Lemma 8.3(i), Lemma 11.7(i)). Since $d_n/\overline{\#}_{n'} \to 0$, or $D_{n'} \le \overline{\#}_{n'}$, $|P| \le n+1$ for all large n, with probability one. On that event, given the training points, Proposition 10.1 gives $\mathbb{P}(\mathcal{E}_n^c) \le 2(n+1)^2 e^{-2\varphi n\overline{\Delta}_n^2} = n^{-2}$, so by the Borel–Cantelli lemma $\mathcal{E}_n$ holds for all large n with probability one. 2. *The bound on $\mathcal{E}_n$.* Theorem 10.4 bounds the error by $3(1+\xi)\int |f_{n',\theta_0} - f| + 3\epsilon_{0,n} + 4\overline{\Delta}_n$, since θ_0 lies on P. 3. *The merged histogram is consistent.* In (a), $\theta_0 = s^*$ at the (G)-depth and Corollary 8.15(i) applies; in

(b), Corollary 8.15(ii). In (c), on each sample θ_0 is $s^*_{D_{n'}}$ when (G) holds at $D_{n'}$ (Lemma 11.4) and $\pi^*_{D_{n'}}$ otherwise (Lemma 11.7(iii)), so $\int |f_{n',\theta_0} - f| \le \max\big(\int |f_{n',s^*_{D_{n'}}} - f|, \int |f_{n',\pi^*_{D_{n'}}} - f|\big)$, and both tend to 0 with probability one: by Corollary 8.11 and Corollary 11.8(i) for a fixed root box, by Corollary 8.14 and Corollary 11.8(ii) for the training hull. Remark 8.16 covers training samples that are not nested. With $\overline{\Delta}_n \to 0$ and $\epsilon_{0,n} \to 0$, the claim follows. □

The corollary covers the pipeline as §10.2 specifies it, the training hull and the (G)-depth included, with one exception: the representable cap. Under a cap, (G) eventually fails at every representable depth, since $\overline{\#}_n/n \to 0$ (Corollary 8.15(iv)); the run then takes the fallback of Phase 2, and its estimate is of the resolution-D density f_D, the quantised representation that Chapter 12 names, not of f: the oracle chain holds against f_D with the resolution bias added (Lemma 12.4(ii)), and on a fixed root box the estimate of the cell integrals converges (Lemma 12.4(iv)). The (G)-depth, computed from the training counts, keeps P training-determined (Lemma 11.18), as the conditions on the run require.

The search the theorem licenses. - The first round scores a skeleton spaced by arc length, not by number of merges. With k states it is an ϵ-cover of P in arc length, with $\epsilon \le \mathrm{arc}(0,H)/(2(k-1))$ plus half the largest single step, and by `Lem-Arc`(ii) the errors along P vary by at most the arc length. - It then bisects, at the arc-length midpoint, every gap that is not settled. It terminates with at most $|P|$ scored states, since a gap that holds no state is settled. - With $\epsilon_0 > 0$, every gap of arc length at most $2\epsilon_0$ is settled. Through ξ alone, a gap settles only if its arc length is at most $4\xi/(1+\xi)$, since scores are at most 1 and so $\underline{\mathrm{err}}_J \le 2 - \frac{1}{2}\mathrm{arc}_J$. Concentrated densities can have $\mathrm{arc}(0,H)$ close to D, so first-round gaps need not settle at once, however far they lie from the minimum. A bound on the number of scored states in terms of the error profile along P is open (§16.9). - It ends with one selection over all scored states: $|U|(|U|-1)$ Yatracos sets. As the code at `c0a1837` computes the integrals on the driver, that costs $O(|U|^3 M)$ for M leaves of the collation (`MinimumDistanceEstimation.scala` l.448–465).

The factor 3 cannot be lowered in general for an estimate that is one of the candidates: for every $\alpha < 3$ there is a class of two distributions that is not properly α-learnable [BKM19, Theorem 2]. For the nested states of one path the question is open. Their factor 2 needs an improper estimate, such as a mixture of the candidates, which for path states is again a mapped paving that the arithmetic of Chapter 6 computes exactly; their factor-2 learners need more samples. Whether their guarantee survives a validation-dependent U is open (§16.9).

Marks: - `Prop-OneDelta`, `Lem-Arc`, `Thm-Certified` and `Cor-Certified` are proved here. [LEAN: built, `IasCert.L1MDE.OneDelta` (the deterministic part and the tail of Proposition 10.1) and `IasCert.L1MDE.CertifiedSearch` (Lemma 10.3(i)–(iv), the gap lower bound, Theorem 10.4 with Theorem 9.1 over U as a hypothesis, the tail of Corollary 10.5); the Borel–Cantelli step and the limit argued in the text; Chapter 15.] - [SIM: the gap search on the refined code at $n = 10^7$ (E1c, Theorem 11.16's [SIM] mark): $k = 16$, $\xi = 0.1$, $\epsilon_0 = 10^{-3}$, tail 0.05, $\overline{\Delta} = 0.0018363$ from $|P| = 3915$ and φn. Under least count it scored $|U| = 95$ states in 9 rounds and chose the state with 3461 leaves; under least $|a-b|$ it scored $|U| = 104$ states in 9 rounds and chose the state with 2693 leaves. Every gap was settled, and the rounds, the scored states and $\hat{\theta}$ were the same under 1, 2, 7 and 64 validation partitions. The floating-point margin was 1.1×10^{-16}, and the floating-point winner equalled the exact winner in every round under both rules (Corollary 11.17). On this smooth density the arc-length rule chose a coarser estimate than least count from the same s^*. Not yet run: on known densities, the gap search's error against $3(1+\xi)\min_P \mathrm{err} + 3\epsilon_0 + 4\overline{\Delta}$, with the per-set Δ of §9.6; and a check of (T) by repeated backtracks under permuted ties.] - `Rmk-Ties`'s code claims were checked by reading the code at `c0a1837`. v1's regression tests for them are to be carried into the simulation checks.

10.5 The code at `c0a1837`, against the final algorithm

Choices that change cost.

Choice	Final algorithm	Code at `c0a1837`	Effect (Chapters 11 to 14)
data	read or generated once, persisted; split partition-invariant	regenerated by each pass	one generation pass per run saved
root box	hull of the training points	hull of all points	the conditioning of §9.3 holds
depth	the shallowest satisfying (G), from training counts	fixed at 60 (44 for one run)	for the 2D uniform, about 10^5 times fewer records after Phase 2: about $4 \cdot 10^{10}$ at depth 60 against about $5 \cdot 10^5$ at $D \approx 19$ with count limit 10^5 (study I-1)
labels	fixed-width integers	arbitrary-precision integers	about 9 times less memory per record in a primitive layout, about 6 as Spark rows
counts	combined on the map side	combined on the map side, but at depth 60 nearly every key is distinct	the shuffle shrinks with the occupied cells
partitions	sized to bytes with the balance factor; range-partitioned when the count map can be ranked exactly	a fixed power of two; subtree partitioning always	no out-of-memory failure in the merge when the bytes per record are as measured and the partitions balanced: exactly for range partitioning, under Conjecture B.10 for subtree partitioning
merge	streams over the sorted iterator	holds the partition in memory (`leavesIter.toVector`)	can spill

Choices that change the result.

Choice	Final algorithm	Code at `c0a1837`	Effect
driver walk	always	only when a depth limit refused a join (`MergeEstimator.scala` l.320); a partition may relabel its subtree as the root (`SubtreePartitioner.scala` l.240–241)	`Lem-Merge` holds; the estimate does not depend on the partitioner
(G) on every record	checked	the first record of a partition is never checked (`MergeEstimator.scala` l.241, l.278)	over-full cells are found
empty partitions	allowed	`require(len > 0)` fails (l.238)	no failure on an empty partition
collation	the common refinement	the operands can be swapped (`LeafMap.scala` l.195–197: `coarser = finer` where `coarser = tmp` was meant)	the Yatracos class is the intended one
validation counts	count matrices combined by `treeAggregate`	accumulators updated in a transformation (`MinimumDistanceEstimation.scala` l.481)	counts independent of re-execution
backtrack ties search	broken by heap code: (T) the gap search, from both ends of P, final selection over all scored states	heap layout spaced by merges over the first $L_3 - 1$ stored leaves, so with lone-child merges the states nearest the root are never scored; needs $k \geq 4$; keeps only the last round	one path P, so `Prop-OneDelta` $3(1 + \xi)$ against every state of P, in place of Theorem 2 against the last round's candidates only

Chapter 11 states and proves the exactness of each choice in the second table. The as-run record of the code at `c0a1837` is kept in the research record of this monograph, not here.

Chapter 11

Exactness of the distributed computation

Chapter 9 and Chapter 10 prove guarantees for the estimate that the final algorithm *specifies*. This section proves that the distributed computation *returns* that estimate. For every safe partitioning, every number of partitions and every execution history that Spark allows, the merged histogram, the backtrack path, every validation count, every score and the selected estimate are those of the sequential specification below. The section also records, result by result, where the code at `c0a1837` departs from the specification and what each departure can change. The departures were found by reading the code; none has been observed in a run, and the earlier runs may carry them.

The sequential specification. Fix the training and validation samples, the root box, the depth D, the count limit $\overline{\#}$, and the parameters k, ξ, ϵ_0 and $\overline{\Delta}$ of the gap search. On one machine, the specification computes: 1. the merged paving: the threshold paving s^* (Definition 8.2 in §8.2) when (G) holds, and otherwise the paving π_D^* of Lemma 11.7; its stored leaves are those that hold training points, with their counts; 2. the backtrack path P from the merged paving to the root, under (T) (§10.4); 3. the gap search of §10.4 (Appendix A.10), each round scoring its states with exact integer validation counts over the Yatracos class of the round, and with integrals and scores computed by one fixed sequence of operations; 4. the estimate and its report.

Exact means that the distributed pipeline returns the same labels and the same integers, and the same floating-point numbers bit for bit, because they are computed by the same sequence of operations from the same inputs.

Notation. In this section, counts refer to the training sample, of size n': $\#(u)$ is the number of training points in the cell of u, and a node *holds* those points. The count limit is $\overline{\#} \geq 0$, and condition (G) says that every depth-D cell holds at most $\overline{\#}$ training points (Definition 8.2). Where a statement divides by $\overline{\#}$, it assumes $\overline{\#} \geq 1$. The *stored labels* are the depth-D labels of positive count, the support of the count map $\mathrm{countOf}_D$ (Definition 6.17); they are distinct and form an antichain. A grid of counts (§8.6) enters as its positive entries: a zero-count cell is not a stored label and never reaches the walk. Labels, $u|_t$, $u \wedge v$ and the left-right order $\prec$ are those of §4.6 (Proposition 4.27 and Lemma 4.28).

11.1 The merge walk

Definition 11.1 (The merge walk)**.** *Let $a_1 \prec a_2 \prec \cdots \prec a_N$, $N \geq 1$, be an antichain of nodes of depth at most D in left-right order, the* atoms*, each with a count $\kappa(a_i) \geq 1$. For a node u write $\#_W(u) = \sum_{a_i \sqsubseteq u} \kappa(a_i)$. Fix a depth limit $\delta \leq \min_i |a_i|$. The* merge walk *$W(\delta)$ keeps a current*

subtree t and the last emitted subtree e°, none at first. 1. *Set $t \leftarrow a_1$.* 2. *For $i = 2, \dots, N$, let $A = t \wedge a_i$. If $|A| \geq \delta$, $\#_W(A) \leq \overline{\#}$ and A does not contain e°, set $t \leftarrow A$ (an* extension*). Otherwise emit t, and set $e^\circ \leftarrow t$ and $t \leftarrow a_i$ (a* refusal*).* 3. *Emit t.*

Let $e_1, \dots, e_S$ be the emitted subtrees, in the order emitted. The raising *replaces each e_j by $r_j = e_j|_k$ for the least $k \geq \delta$ such that $e_j|_k$ contains no e_l with $l \neq j$. The* output *of $W(\delta)$ is $r_1, \dots, r_S$, each with the count $\#_W(r_j)$.*

On a partition the atoms are its stored labels with their counts; on the driver they are the outputs of the partitions with their counts. With one emitted subtree the raising gives $e_1|_\delta$, which is the root only when $\delta = 0$. So a partition never relabels its output as the root unless it is the only partition, whose depth limit is then 0 (Corollary 11.6). An atom that lies below the current subtree is absorbed by an extension that leaves t unchanged; the code instead jumps past the whole run of atoms below A (`MergeEstimator.scala` l.261–263 at `c0a1837`), with the same effect.

Lemma 11.2 (The walk's properties)**.** *In the setting of Definition 11.1: - (a) The atoms below each e_j are consecutive, $a_{s_j}, \dots, a_{s_{j+1}-1}$, with $1 = s_1 < s_2 < \cdots < s_{S+1} = N + 1$. The e_j are pairwise disjoint, $e_1 \prec \cdots \prec e_S$, $|e_j| \geq \delta$, and each e_j is a single atom or has $\#_W(e_j) \leq \overline{\#}$. - (b) For $j < l$ with $|e_j \wedge e_l| \geq \delta$, $\#_W(e_j \wedge e_l) > \overline{\#}$. For $\delta = 0$: every common ancestor of two distinct emitted subtrees has $\#_W > \overline{\#}$. - (c) The r_j are pairwise disjoint, $r_1 \prec \cdots \prec r_S$, $e_j \sqsubseteq r_j$ and $|r_j| \geq \delta$. The atoms below r_j are exactly those below e_j, so $\#_W(r_j) = \#_W(e_j)$. If $|r_j| > \delta$, the parent of r_j contains some e_l with $l \neq j$, and $|e_j \wedge e_l| \geq |r_j| - 1 \geq \delta$. - (d) For $S \geq 2$, r_j is the deepest of $(e_{j-1} \wedge e_j)\mathsf{R}$, $(e_j \wedge e_{j+1})\mathsf{L}$ (those that exist) and $e_j|_\delta$. So r_j depends only on e_{j-1}, e_j and e_{j+1}.*

Proof. - (a). Let s be the index at which the current subtree was started. At the head of step i four invariants hold: (I1) the atoms below t are $a_s, \dots, a_{u-1}$ for some $u \geq i$; (I2) either e° is none and $s = 1$, or $e^\circ \prec t$ are disjoint and the atoms below e° end at a_{s-1}; (I3) $t = a_s$ or $\#_W(t) \leq \overline{\#}$; (I4) $|t| \geq \delta$. They hold at the start. At step i: - If $a_i \sqsubseteq t$, then t is not a single atom, so $\#_W(t) \leq \overline{\#}$; $A = t$; and t does not contain e° by (I2). The step extends and leaves t unchanged. - Otherwise $i = u$, and $t \not\sqsubseteq a_i$ (else $a_s \sqsubseteq a_i$), so $A \sqsupset t$. If A holds an atom of index below s, it holds a_{s-1}, since the atoms below A are consecutive and include a_s (Proposition 4.27(vi)). Then A meets e°, does not lie inside it (it holds a_s), and so contains it: the step refuses. So after an extension the atoms below A are consecutive from a_s, and (I1), (I3), (I4) hold by the test. After a refusal, t is emitted holding $a_s, \dots, a_{i-1}$, and the new $t = a_i$ satisfies the invariants with $s = i$, since $\delta \leq |a_i|$.

At the end the last t holds $a_s, \dots, a_N$. Distinct emitted subtrees hold disjoint nonempty blocks of atoms, so neither contains the other, and disjoint nodes holding consecutive blocks are in the order of the blocks. - *(b).* Let e_j be emitted by a refusal at a_i, $i = s_{j+1}$, and let $A = e_j \wedge a_i$. Then $A = e_j \wedge e_{j+1}$: e_{j+1} and A both contain a_i, so they are nested; $e_{j+1} \sqsupset A$ would put e_j inside e_{j+1}; so $e_{j+1} \sqsubseteq A$, and $A \sqsupseteq e_j \wedge e_{j+1} \sqsupseteq e_j \wedge a_i = A$. The refusal had one of three causes: $|A| < \delta$; $\#_W(A) > \overline{\#}$; or A contains e_{j-1}. In the third case $A \sqsupseteq e_{j-1} \wedge e_j$, so $|A| \leq |e_{j-1} \wedge e_j|$ and $\#_W(A) \geq \#_W(e_{j-1} \wedge e_j)$. By induction on j (for $j = 1$ the third cause is absent), $|e_j \wedge e_{j+1}| < \delta$ or $\#_W(e_j \wedge e_{j+1}) > \overline{\#}$. For $j < l$, $e_j \wedge e_l$ contains e_j and e_l, hence e_{j+1} (Proposition 4.27(vi) on the antichain of emitted subtrees), so $e_j \wedge e_l \sqsupseteq e_j \wedge e_{j+1}$. If $|e_j \wedge e_l| \geq \delta$, then $|e_j \wedge e_{j+1}| \geq \delta$, so $\#_W(e_j \wedge e_{j+1}) > \overline{\#}$, and $\#_W(e_j \wedge e_l) > \overline{\#}$ by monotonicity. - *(c).* $k = |e_j|$ qualifies, so r_j exists and $|r_j| \geq \delta$. If $r_j \sqsupseteq r_l$ with $l \neq j$, then $r_j \sqsupseteq e_l$, which the raising excludes. An atom below r_j lies below some e_l by (a); e_l and r_j are then nested; $r_j \sqsupseteq e_l$ with $l \neq j$ is excluded, and $e_l \sqsupset r_j \sqsupseteq e_j$ contradicts (a); so $l = j$. The order follows from $e_j \sqsubseteq r_j$. If $|r_j| > \delta$, the minimality of k makes $e_j|_{k-1}$, the parent of r_j, contain some e_l with $l \neq j$; it also contains e_j, hence $e_j \wedge e_l$,

which therefore has depth at least $k-1 \geq \delta$. - *(d).* By (c) and Proposition 4.27(vi), $e_j|_k$ contains another emitted subtree exactly when it contains e_{j-1} or e_{j+1}, that is, when $k \leq |e_{j-1} \wedge e_j|$ or $k \leq |e_j \wedge e_{j+1}|$. So $r_j = e_j|_{k^*}$ with $k^* = \max(\delta, |e_{j-1} \wedge e_j| + 1, |e_j \wedge e_{j+1}| + 1)$. Since e_j lies in the right subtree of $e_{j-1} \wedge e_j$ and in the left subtree of $e_j \wedge e_{j+1}$ (Proposition 4.27(iv)), these candidates are $(e_{j-1} \wedge e_j)\mathsf{R}$, $(e_j \wedge e_{j+1})\mathsf{L}$ and $e_j|_\delta$. □

Marks. Proved here. [LEAN: MergeWalk walk_spec ((a), (b) and coverage for $\delta = 0$), walkD_-spec ((a) for any δ), raise_props, raiseD_props, mass_raiseD, raiseD_not_nested ((c)); Raise decreaseDepth_eq_raiseD, lem_raise ((d) for $S \geq 2$, with the code's neighbour-join raising) — built at `a5a2cee59`; to write — (b) and the last clause of (c) for $\delta > 0$, used only by Remark 11.5.] [SIM: random antichains of nodes of depth at most 6 with random counts and random $\delta \leq \min |a_i|$, the walk against brute-force checks of (a)–(d), (d) also against the transliterated `maximalSubtreesDecreaseDepth` for $S \geq 2$: 4000 cases, no failure (the review's `rev10_walk.py`); to rerun at larger depths, L1MDE-PROOF-SIMULATION.]

Definition 11.3 (Safe depth limit)**.** *A partitioner $\mathcal{P}$ sends each stored label to one of $q \geq 1$ partitions. A depth limit $\delta \in \{0, \ldots, D\}$ is* safe for $\mathcal{P}$ *when: - (Safe-1) for every node u with $|u| \geq \delta$, all stored labels below u are sent to one partition; - (Safe-2) each partition presents its stored labels in left-right order.*

(Safe-1) is monotone: a depth limit deeper than a safe one is safe. $\delta = D$ always meets (Safe-1), since a depth-D node is its own only depth-D descendant; under hashing it is in general the only safe choice, and then all merging falls on the driver. (Safe-2) is a sort of distinct depth-D labels, an antichain, so `leftRightOrd` is lawful on it, and so is the integer order of the heap codes (Lemma 4.28(b), Proposition 4.27(v)).

Lemma 11.4 (The merge computes the threshold paving)**.**

- *(i) One list. Let every atom of Definition 11.1 hold between 1 and $\overline{\#}$ training points, with $\kappa(a) = \#(a)$, and let every training point lie in some atom. Then (G) holds, and $W(0)$ returns exactly the leaves of s^* that hold points, each with its count. They are the stored leaves of the output of the merge queue (Lemma 8.5) and of the terminal state of every run of SEBTreeMC with depth cap D and leaf cap $\overline{m} > D\lfloor n'/(\overline{\#}+1)\rfloor$, whatever its tie-breaking (Lemma 8.4). In particular this holds when the atoms are the stored labels and (G) holds.*
- *(ii) Distributed. Assume (G), and let δ be safe for $\mathcal{P}$. Each partition runs $W(\delta)$ on its stored labels, with its own counts. The driver collects the outputs of all partitions, sorts them in left-right order, and runs $W(0)$ on them with their counts, always. Then the collected outputs form an antichain, so the driver's sort is a sort of an antichain; the count of each on its partition is its global count, between 1 and $\overline{\#}$; between them they hold every training point; and the driver returns exactly the leaves of s^* that hold points, with their counts. This holds for every safe $(\mathcal{P}, \delta)$, every number of partitions and every grouping of the labels, empty partitions included.*
- *(iii) Streaming. Let a partition's atoms arrive one at a time in left-right order, each with a count at least 1. Consider the walk that keeps a first-in first-out buffer of records read ahead, the current subtree with its count, the label of the last emitted subtree, and the last two emitted subtrees, in their emitted form. At the next atom a it forms $A = t \wedge a$. It refuses if $|A| < \delta$ or A contains e°. Otherwise it reads records into the buffer while they lie below A and the running sum $\#_W(t) + \sum \kappa$ is at most $\overline{\#}$, and extends exactly when the run of records below A ends first. It raises e_j when e_{j+1} is emitted, or at the end. This walk emits and raises exactly what $W(\delta)$ emits and raises on the materialised list, and its buffer never holds more than $\overline{\#}+1$ records.*

Proof. - *(i).* Apply Lemma 11.2 with $\delta = 0$. For every node u that contains an atom,

$\#_W(u) = \#(u)$: each point lies in exactly one atom, that atom is nested with u, and an atom strictly containing u would contain another atom. (G) holds: a depth-D cell holding a point meets the atom holding it, and atoms have depth at most D, so the cell lies in the atom. Each output r_j holds $\#(r_j) = \#(e_j)$ points, between 1 and $\overline{\#}$, since e_j is an atom or an extension. - If $\#(\rho) \le \overline{\#}$, no node holds more than $\overline{\#}$ points, so by (b) there is one emitted subtree, and the output is $e_1|_0 = \rho$, the only leaf of s^*. - If $\#(\rho) > \overline{\#}$, no output is ρ. By (c), the parent of r_j contains $e_j \wedge e_l$ for some $l \neq j$, which holds more than $\overline{\#}$ points by (b). So $\#(r_j) \le \overline{\#} < \#(\text{parent}(r_j))$, and r_j is a leaf of s^* by Lemma 8.3(ii). - Conversely, a leaf ℓ of s^* holding a point x: x lies in some atom, below some e_j, below r_j. Then ℓ and r_j are leaves of one paving sharing x, so $\ell = r_j$.

Under (G), $s^*_D = s^*$, so Lemma 8.5 and Lemma 8.4 give the last claims. - *(ii).* On partition p the atoms are its stored labels in left-right order (Safe-2), each with $1 \le c \le \overline{\#}$ by (G). Write $\#_p(u)$ for the count of p's labels below u. By Lemma 11.2(a) and (c), each output r of p has $|r| \ge \delta$ and $\#_p(r) = \#_p(e) \in [1, \overline{\#}]$. - *Local counts are global.* r holds a label of p and $|r| \ge \delta$, so by (Safe-1) every stored label below r is on p, and $\#(r) = \#_p(r)$. - *Antichain.* Two outputs of p are disjoint by Lemma 11.2(c). If outputs r of p and r' of $p' \neq p$ were nested, say $r \sqsupseteq r'$, then r, of depth at least δ, would hold labels of p and of p', against (Safe-1). - *Cover.* Every stored label is an atom of its partition, below some e, below its r.

So the driver's atoms satisfy the hypotheses of (i), which gives the claim. An empty partition emits nothing. Safety enters in the raising: without it, a raised node of one partition can hold labels of another, as the example after Corollary 11.6 shows. - *(iii).* Keep t, s and e° as in the proof of Lemma 11.2(a). If $|A| < \delta$ or A contains e°, both walks refuse without counting. Otherwise, by that proof, the atoms below A are those below t, namely $a_s, \dots, a_{i-1}$, and a run $a_i, a_{i+1}, \dots$ of consecutive atoms starting at the current one. So $\#_W(A) = \#_W(t) + \sum_{\text{run}} \kappa$. The partial sums increase strictly, since every count is at least 1, so stopping at the first record outside A, or as soon as the sum exceeds $\overline{\#}$, decides $\#_W(A) \le \overline{\#}$ exactly. After an extension the buffered records below A are dropped. $W(\delta)$ would absorb each of them by an extension that leaves $t = A$ unchanged (the first case of the proof of Lemma 11.2(a)), so both walks reach the first record outside A in the same state. The raising is (d) of Lemma 11.2.

The buffer. A decision at atom a_{i_0} reads records beyond a_{i_0} into the buffer while they lie below A and the running sum is at most $\overline{\#}$. The sum starts at $\#_W(t) + \kappa(a_{i_0}) \ge 2$ and each record adds at least 1. So either at most $\max(\overline{\#} - 1, 0)$ records below A are read, the sum then exceeding $\overline{\#}$, or fewer and one record outside A. Either way it reads no record beyond index $i_0 + \max(\overline{\#} - 1, 0)$. Every buffered record was read by a decision at some $i_0 \le i$, the current index, and has not been consumed, so its index lies in $[i, i + \max(\overline{\#} - 1, 0)]$: at most $\max(\overline{\#}, 1) \le \overline{\#} + 1$ records. □

Marks. (i) and (ii) proved here; (iii) proved here, new. [LEAN: (i) MergeWalk lem_merge, raised_eq_leaves, output_eq_leaves; (ii) MergeWalk local_walk_props, mass_localCounts, driver_atom_props, driver_atoms_cover, driver_atoms_antichain, local_outputs_not_nested, and the composition with SEBTreeMC for $\overline{m} = 1 + \lceil Dn'/\overline{\#} \rceil$ with $\overline{\#} \ge 1$ (`hcap`), Pipeline thm_exact_pipeline, mbar_bound and Exact thm_exact_distributed — built at `a5a2cee59`. The Lean walk absorbs atoms one at a time, and (iii)'s argument shows that the code's jump past the run of A is the same. (iii) Stream lem_merge_iii (the streaming walk, with its look-ahead, the drop of the run of A (the code's jump) and the raising from the neighbours' joins one subtree late, emits exactly $W(\delta)$ and raises as the materialised walk does, for atoms in left-right order carrying their counts and covering the stored labels; no count, count-limit or depth hypothesis), stream_eq_walkD, runSum_eq_mass, delayRaise_eq, lem_merge_iii_buffer (at most $\max(\overline{\#}, 1) \le \overline{\#} + 1$ records held, counts ≥ 1; a record of count 0 voids the buffer bound, not the output, lookReads_zero_counts) and lem_merge_iii_partition — built at `8afd4d5f2`; the emitted counts, StreamCounts streamWalkC_fst (the counting walk emits the streaming walk's

labels), streamWalkC_eq, streamWalkC_count (each emitted count is the mass of its subtree) and lem_merge_iii_counts (under lem_merge_iii's hypotheses it emits $W(\delta)$'s subtrees with their counts), with lookSum_eq_runSum and lookAhead_ends (the look-ahead's running sum is the run's) — built at `15a5577ef`; the raised output's counts, which need $\delta \le |e_j|$, are not.] [SIM: (i) $W(0)$ on the stored labels of 600 random samples meeting (G), against the nonempty leaves of s^* with their counts; (ii) 600 runs each under range partitioning at $\delta = 1 + \max_i |c_i|$, subtree partitioning with random roots of depth at most δ, and hashing at $\delta = D$, with 1–5 partitions and 300 cases of empty partitions, against s^*; (iii) the streaming walk with a first-in first-out look-ahead on 600 lists at random δ against the materialised walk, the buffer's high-water mark (head included) never above $\overline{\#}$; the example of §10.2 and the example after Corollary 11.6 as regression tests: no failure (the review's `rev10_merge.py`); the cluster runs, L1MDE-PROOF-SIMULATION.]

Remark 11.5 (When the driver walk changes nothing)**.** §10.2 runs the driver walk always, at the cost of one linear walk over records that are collected anyway. For the record, the walk can change the collected outputs only if one of them has depth exactly δ. Suppose every output r of every partition has $|r| > \delta$. By Lemma 11.2(c) the parent of r contains $e \wedge e'$ for two emitted subtrees of r's partition p, with $|e \wedge e'| \ge \delta$. By (b), $\#_p(e \wedge e') > \overline{\#}$, and by (Safe-1) this is the global count. So $\#(r) \le \overline{\#} < \#(\text{parent}(r))$, every output is a leaf of s^* (Lemma 8.3(ii)), and the outputs hold every point: they are already the answer. The converse fails in general, so the test is a sufficient one. A partition with one emitted subtree outputs it at depth exactly δ, as do the two partitions of the example in §10.2. The code's trigger at `c0a1837`, "some depth limit refused a join", is a different condition, and that example shows that it misses.

Marks. Proved here; not used by Chapter 10. [LEAN: to write, with (b) of Lemma 11.2 for $\delta > 0$.] [SIM: in 600 distributed runs, whenever no collected output had depth exactly δ, the driver walk returned the collected outputs unchanged, no failure of the sufficiency claim (the review's `rev10_merge.py`). On the refined code at $n = 10^7$ (E1a, Theorem 11.16's [SIM] mark), the converse case occurred and the always-run walk was needed: under subtree partitioning at 64 partitions most runs had outputs at depth exactly δ, and the driver merged the 4808 collected outputs to the 4656 leaves of s^*. A conditional walk keyed to that test would have been right there, but the code's trigger at `c0a1837` is a different test, and §10.2's example shows it can miss.]

The code at `c0a1837`. `maximalCountSubtreeGeneration` (`MergeEstimator.scala` l.233–288) is $W(\delta)$, with its overlap test written as `oldMaxIndex < ancestorInterval.lower` (l.261). By the proof of Lemma 11.2(a), that test is "A does not contain e°". The raising is right: for $S \ge 2$ the code's neighbour-join form (`maximalSubtreesDecreaseDepth`, `SubtreePartitioner.scala` l.236–287) is the raising of Definition 11.1 (Lemma 11.2(d)). The code departs from the specification in four places. 1. *The driver walk is conditional.* It runs only when a partition's depth limit refused a join (the flag of l.270, tested at l.320). §10.2 gives a case the trigger misses. 2. *A partition can relabel its output as the root.* A walk that emits one subtree relabels it as the root (`SubtreePartitioner.scala` l.240–241, reached from `MergeEstimator.scala` l.287 with the depth limit δ). This is the raising of $W(0)$ applied on a partition. Together with 1: take four depth-2 cells with one point each, count limit 10, and the subtree partitions $\rho\mathsf{L}$ and $\rho\mathsf{R}$, so $\delta = 1$. Each partition emits $(\rho, 2)$, the flag stays zero, and `mergeLeavesHistogram` builds its map with `toMap` (l.346), so one $(\rho, 2)$ overwrites the other. The result is the root with count 2, a density that integrates to $\frac{1}{2}$, where s^* is the root with count 4. 3. *The first record of a partition is never compared with the count limit* (l.241; the test of l.278 applies only to records that start a later subtree). Under (G) this is harmless. Without (G), whether the run stops or returns an over-full leaf depends on the partitioner (the code paragraph after Corollary 11.8). 4. *The partition is held in memory* (`leavesIter.toVector`, l.236), and `require(len > 0)` (l.238)

fails on an empty partition. Neither changes an estimate.

Items 1 and 2 are the driver-pass condition (X3) of §11.7; item 3 is (X6).

11.2 Safe depth limits

Corollary 11.6 (Safe depth limits)**.** *Let each partition present its stored labels in left-right order (Safe-2). - (a) Partitioners that read only an ancestor. If $\mathcal{P}(x) = g(x|_\delta)$ for every stored label x and some map g, then δ is safe. In particular, let $u_0, \dots, u_{S-1}$ be any subtree roots of depth at most $\delta \le D$, and let $\mathcal{P}(x)$ be the partition assigned to* `findSubtree`(x)*. Then δ is safe; so is the code's $\delta =$* `maxSubtreeDepth` $= \max_k |u_k|$*; and with one partition $\delta = 0$ is safe. - (b) Contiguous runs. Let each partition's labels form a run of consecutive labels of the sorted list of all stored labels. For each boundary between consecutive nonempty runs, let c_i be the deepest common ancestor of the last label before it and the first label after it. Then δ is safe exactly when $\delta > \max_i |c_i|$, every δ being safe when there is one nonempty run. So $\delta = 1 + \max_i |c_i|$, the choice of §10.2, is the least safe depth limit of range partitioning. - (c) Snapped runs. If moreover each boundary falls between the blocks of two nodes of a depth δ_0, then $1 + \max_i |c_i| \le \delta_0$, and δ_0 is safe by (a) as well as by (b).*

Proof. - (a). If $x \sqsubseteq u$ and $|u| \ge \delta$, then $x|_\delta = u|_\delta$, so all stored labels below u have one image under g. For subtree partitioning, `findSubtree`(x) depends on x only through $x|_\delta$ whenever every root has depth at most δ: this is Theorem B.8(iv), a forward reference to Appendix B, and meanwhile the Lean theorem `FindSubtree.findSubtree_truncate` (hypothesis: every root has depth at most δ; no sortedness assumed), built at `a5a2cee59`, so that this proof stands before Appendix B lands. With one partition every label goes to it. - *(b), if.* This is the argument of §10.2. Let $|u| \ge \delta > |c_i|$ for every i. The stored labels below u are consecutive in the sorted list (Proposition 4.27(vi)). If they met two runs, they would include the last label before some boundary and the first after it. Then $u \sqsupseteq c_i$ and $|u| \le |c_i| < \delta$, a contradiction. - *(b), only if.* c_i holds stored labels of two partitions, so (Safe-1) fails at every $\delta \le |c_i|$. - *(c).* The two labels across a boundary lie in different depth-δ_0 nodes, so their deepest common ancestor is shallower than δ_0. And $\mathcal{P}(x)$ depends only on the depth-δ_0 node containing x. □

The bound is sharp (asked by the review of Chapter 10, M2). On $[0, 1]$ with $D = 3$, put one training point in each of the eight cells (heap codes 8–15), let $\overline{\#} = 1$, and use the runs $\{8, 9, 10\}$ and $\{11, \dots, 15\}$. Then $c = 10 \wedge 11 = 5 = \rho\mathsf{LR}$, of depth 2. With $\delta = 2$: - the first partition emits $8, 9, 10$ unmerged and raises 10 to $(9 \wedge 10)\mathsf{R} = \rho\mathsf{LR}$; - the second raises 11 to $11|_2 = \rho\mathsf{LR}$, since its other candidate $(11 \wedge 12)\mathsf{L} = \rho\mathsf{L}$ has depth $1 < \delta$.

Two copies of $\rho\mathsf{LR}$, each with count 1, reach the driver, where the true $\rho\mathsf{LR}$ holds 2 points and must stay split. The driver's list is not an antichain. At `c0a1837` the driver walk (both partitions raise the flag at l.270: the first refuses $9 \wedge 10 = \rho\mathsf{L}$, of depth $1 < \delta$, and the second refuses $11 \wedge 12 = \rho$) then throws at `MergeEstimator.scala` l.264–265. With $\delta = 3 = 1 + |c|$ both raisings stop at depth 3, and Lemma 11.4(ii) applies.

Exact-rank cuts give a deep δ. A boundary placed at an exact rank falls between two neighbouring stored labels, which in dense regions share all but their last few turns, so $|c_i|$ is close to D and the least safe depth limit $1 + \max_i |c_i|$ is close to D as well: correct, but the local walks then merge almost nothing and the driver receives nearly every stored label. On 10^7 points of a two-dimensional Gaussian mixture (Theorem 11.16's [SIM] mark), exact-rank range partitioning gave $1 + \max_i |c_i|$ of 16–17 at $D = 17$ and 22 at $D = 22$, and the driver received all 631 866 stored labels at $D = 22$. The parallel merge comes from snapped runs, (c), whose boundaries fall between the blocks of nodes of a chosen depth δ_0, and from subtree partitioning, (a). On the same data with δ_0 the shallowest depth with at least $4p$ nonempty nodes, snapped runs at

$D = 22$ and $p = 64$ gave $\delta = \delta_0 = 10$ (exact rank: 22), and the driver received 4808 outputs (exact rank: 631 866), against $|s^*| = 4656$. The largest local input was 1296 labels, the same as subtree partitioning by largest-first assignment. All 24 runs (range, snapped and subtree at $p = 1, 2, 7, 64$, at $D = 17$ and $D = 22$) returned the same s^*. The price is balance: a snapped run cannot split a depth-δ_0 block, so the runs are balanced only to within the heaviest block. Here 55 of the 64 partitions received labels, where subtree partitioning filled all 64.

Marks. (a) proved here, from Theorem B.8(iv) and `FindSubtree.findSubtree_truncate`; (b) "if" is §10.2's, "only if" and (c) new. [LEAN: (a) FindSubtree findSubtree_truncate, cor_safe, cor_safe_maxSubtreeDepth, pairwise_strict_of_sorted, and MergeWalk safeDepth_of_roots — built at `a5a2cee59`; (a) in its general form, RangeSafe safe_of_factor; (b) RangeSafe safe_iff (with safe_of_cutDepth_le, cutDepth_le_of_safe, part_eq_of_below on Labels subtree_contiguous), safe_cutDepth and not_safe_of_lt (the least safe depth limit), safe_of_no_boundary; (c) RangeSafe cutDepth_le_of_snapped — built at `bdeef562e`; and RangeSafe safe_of_snapped (the snapped δ_0 is safe through (b)), cutDepth_le_D and cutDepth_le_of_atDepth ($\delta = 1 + \max_i |c_i| \le D$ on labels of depth at most D) — built at `fcb47e5df`. The Lean proves (b) for any partitioner read along the sorted list, with c_i taken at each adjacent pair the partitioner separates; for runs these pairs are the boundaries, so contiguity is not needed.] [SIM: (Safe-1) checked by brute force over all nodes of depth at least δ under range partitioning at $\delta = 1 + \max_i |c_i|$, subtree partitioning and hashing, no failure; (b) "only if": at $\delta = \max_i |c_i|$ (Safe-1) failed in all 383 cases with two or more nonempty runs, and the driver received a non-antichain or returned the wrong paving in 376 of them; the example above reproduced, with the throw at l.264–265 (the review's `rev10_merge.py`); snapped runs, to run, L1MDE-PROOF-SIMULATION.]

The code at `c0a1837`. The code has subtree partitioning only, with $\delta =$ `maxSubtreeDepth` (`SubtreePartitioner.scala` l.126–135) and `getPartition` through `findSubtree` (l.149–152). By (a) this δ is safe for every set of sampled roots, including labels that fall in the gaps between roots. (Safe-2) holds when the repartition sorts by `leftRightOrd`, as the library's tests do. With one requested partition the partitioner has the root as its only subtree, and $\delta = 0$. Range partitioning, exact or snapped, is not in the code; Spark's own range partitioner takes its bounds from a sample, so its runs are contiguous but its balance is not exact (§10.2).

11.3 Without condition (G)

When (G) fails, some depth-D cell holds more than $\overline{\#}$ training points. §10.2 keeps such a cell as an atom and says so in the report. The walk then still returns a paving determined by the counts alone (Lemma 11.7), which Chapter 12 names in the report. As a merge rule at depth D_n it is consistent under the growth conditions of Corollary 8.11, $D_n \to \infty$ included (Corollary 11.8); at the fixed representable depth of §10.2's fallback no consistency statement applies (Corollary 8.15(iv), Chapter 8), and what the estimate then estimates is Chapter 12's.

Lemma 11.7 (The merge without (G))**.** *Let the atoms of Definition 11.1 be an antichain of nodes of depth at most D, each holding at least one training point, with $\kappa(a) = \#(a)$, and holding every training point between them. An atom may hold more than $\overline{\#}$ points; (G) is not assumed. Let π be the paving whose internal nodes are the strict ancestors of the outputs of $W(0)$. - (i) The outputs are pairwise disjoint and hold every point; they are leaves of π, and the other leaves of π hold no point. Every internal node of π has depth less than D and holds more than $\overline{\#}$ points, so $|\mathcal{L}(\pi)| \le 1 + D\lfloor n'/(\overline{\#} + 1)\rfloor$. - (ii) Every output holds at most $\overline{\#}$ points, or exactly the points of one atom below it. - (iii) If every atom holds at most $\overline{\#}$ points or exactly the points of one depth-D cell, then $\pi = \pi^*_D$, the paving with internal nodes*

$$I^* = \{u : \ |u| < D, \ \#(u) > \overline{\#}, \text{ and the points of } u \text{ do not all lie in one depth-}D \text{ cell}\}.$$

So π depends on the counts alone, not on how they were grouped into atoms. Under (G), $\pi^*_D = s^*$. - (iv) The distributed execution of Lemma 11.4(ii), with each local walk keeping an over-full stored label as an atom, returns π^*_D for every safe $(\mathcal{P}, \delta)$.

Proof. - (i). Disjointness and cover are Lemma 11.2(a) and (c) with $\delta = 0$; neither uses a cap on the atoms. The strict ancestors of the outputs are closed under taking ancestors, so they are the internal nodes of a paving (Lemma 4.2(i)). No output lies below another, so each output is a leaf. Every other leaf is disjoint from every output: a leaf containing an output would be internal, and a leaf strictly inside an output r would have its parent, an internal node, at or below r, where no other output lies. So the other leaves hold no point. An internal node u is a strict ancestor of an output r, so $u \sqsupseteq \text{parent}(r) \sqsupseteq e \wedge e'$ for two emitted subtrees (Lemma 11.2(c)), and $\#(u) > \overline{\#}$ by (b). Outputs have depth at most D, so $|u| < D$. Internal nodes of one depth are disjoint and each holds at least $\overline{\#} + 1$ of the n' points, so there are at most $\lfloor n'/(\overline{\#} + 1) \rfloor$ of them at each depth $0, \ldots, D-1$ (the count of Lemma 8.3(i)), and a paving has one more leaf than internal nodes (Lemma 4.2(ii)). - *(ii).* An emitted subtree is an atom, or an extension, which holds at most $\overline{\#}$ points. Raising keeps its atoms (Lemma 11.2(c)). - *(iii).* $I(\pi) \subseteq I^*$. An internal node u has $|u| < D$ and $\#(u) > \overline{\#}$ by (i), and contains two disjoint outputs. A depth-D cell holding a point of each would meet both; as both have depth at most D, it would lie below both, which are disjoint. $I^* \subseteq I(\pi)$. Let $u \in I^*$ and suppose $u \notin I(\pi)$. Then u lies at or below a leaf of π (Lemma 4.2(vi)), which holds u's points, so it is an output r, with $\#(r) \geq \#(u) > \overline{\#}$. By (ii) and the hypothesis, r holds exactly the points of one depth-D cell, and so does u, a contradiction. A paving is determined by its internal nodes (Lemma 4.2(i)). Under (G), a node holding more than $\overline{\#}$ points cannot have them all in one depth-D cell, so I^* is the internal-node set of s^* (Definition 8.2). - *(iv).* The proof of Lemma 11.4(ii) used (G) only to bound the atoms; its antichain, cover and local-equals-global steps use (Safe-1) alone. By (ii) applied on each partition, each partition's output holds at most $\overline{\#}$ points or exactly the points of one over-full stored label, globally by (Safe-1). So the driver's atoms meet the hypothesis of (iii). □

This is the paving the report names without (G) (Chapter 12): an RP of the root box of depth at most D (by (i)), a function of the counts alone (by (iii)), with the counts of its stored leaves as the quantised representation.

π^*_D can differ from s^*_D, the threshold paving cut off at depth D (Definition 8.2). Take $D = 1$, $\overline{\#} = 2$, five points in $\rho\mathsf{L}$ and none in $\rho\mathsf{R}$: then s^*_1 has the leaves $\rho\mathsf{L}$ and $\rho\mathsf{R}$, while π^*_1 is the root, because the walk raises the over-full atom to the highest ancestor that holds no other point. Both are consistent along $D_n \to \infty$ (Corollaries 8.11 and 11.8).

Marks. Proved here; (iii) and (iv) new. [LEAN: (i) and (ii) for one list, MergeNoG walk_spec_noG, mass_raise, raise_parent_heavy, raise_not_nested, mem_outFinset, mem_internalOf, ancClosed_internalOf, lem_mergeNoG — built at `279d904f6`; to write — the empty leaves, (iii) and (iv).] [SIM: 600 random samples with ties forced above the count limit, 497 of them without (G): (i)–(ii) no failure, $I(\pi) = I^*$ in every case, and the distributed execution under range, subtree and hash partitioning (1800 runs) returned π^*_D in every case, the driver's atoms meeting (iii)'s hypothesis in all of them; the example above reproduced (the review's `rev10_nog.py`); to rerun at larger depths, L1MDE-PROOF-SIMULATION.]

Corollary 11.8 (The merge is consistent without (G))**.** *Let $X_1, X_2, \ldots$ be i.i.d. with a density f, and let $\pi_n = \pi^*_{D_n}$ be the paving of Lemma 11.7(iii) for $X_1, \ldots, X_n$ with count limit $\overline{\#}_n$, computed on one machine or distributed, the sequences D_n and $\overline{\#}_n$ deterministic. - (i) A fixed root box. If the root box is fixed before the data are seen, $D_n \to \infty$, $D_n/\overline{\#}_n \to 0$ and $\overline{\#}_n/n \to 0$, then $\int |f - f_{n,\pi_n}| \to 0$ with probability one. - (ii) The training hull. If the support of f is bounded, the root box is the closed box hull H_n of $X_1, \ldots, X_n$ $(n \geq 2)$, $D_n \to \infty$, $D_n \log n/\overline{\#}_n \to 0$ and $\overline{\#}_n/n \to 0$, then $\int |f - f_{n,\pi_n}| \to 0$ with probability one, where f_{n,π_n} is zero outside H_n.*

Proof. By Lemma 11.7(i) and (iii), π_n is an RP of the root box of depth at most D_n and a function of the depth-D_n count table: a merge rule at depth D_n (§8.5), with the convention that an empty table gives the root paving, since the walk needs an atom. - *(i), (M1).* By Lemma 11.7(i), $|\mathcal{L}(\pi_n)| \le \overline{m}_n = 1 + D_n\lfloor n/(\overline{\#}_n + 1)\rfloor$, and $\overline{m}_n/n \le 1/n + D_n/\overline{\#}_n \to 0$. - *(i), (M2).* Fix k, and let n be large enough that $D_n \ge k$. The leaves of π_n of depth less than k are pairwise disjoint, so there are at most 2^{k-1} of them. By Lemma 11.7(ii) each holds at most $\overline{\#}_n$ points or the points of one depth-D_n cell. Their empirical mass is therefore at most $2^{k-1}\max(\overline{\#}_n/n, M_n)$, where $M_n = \max_{|v|=D_n}\mu_n(\mathbf{x}_v)$. - *(i), $M_n \to 0$ with probability one.* $M_n \le \max_{|v|=D_n}\mu(\mathbf{x}_v) + \max_{|v|=D_n}|\mu_n(\mathbf{x}_v) - \mu(\mathbf{x}_v)|$. - The first term tends to 0. The cells of depth D_n have volume $2^{-D_n}\mathrm{vol}(\mathbf{x}_\rho) \to 0$, and f is integrable, so for every $\epsilon > 0$ there is $\eta > 0$ with $\int_B f < \epsilon$ whenever $\mathrm{vol}(B) < \eta$. - For the second, Hoeffding's inequality and a union bound over the 2^{D_n} cells give $\mathbb{P}\{\max_v|\mu_n(\mathbf{x}_v) - \mu(\mathbf{x}_v)| > \epsilon\} \le 2^{D_n+1}e^{-2n\epsilon^2}$. Now $D_n/n = (D_n/\overline{\#}_n)(\overline{\#}_n/n) \to 0$, so for large n the bound is at most $e^{-n\epsilon^2}$, which is summable. The Borel–Cantelli lemma gives the claim.

Theorem 8.10 concludes. - *(ii).* Check the conditions of Theorem 8.13; π_n is an RP of the regular scheme on H_n, chosen from the sample. *(M1′).* By Lemma 11.7(i), $|\mathcal{L}(\pi_n)| \le \overline{m}_n = 1 + D_n\lfloor n/(\overline{\#}_n + 1)\rfloor$ for every sample, and $\overline{m}_n \log n/n \le \log n/n + D_n \log n/\overline{\#}_n \to 0$. *(M2).* As in (i), the leaves of depth less than k have empirical mass at most $2^{k-1}\max(\overline{\#}_n/n, M_n)$, where M_n is now the largest empirical mass of a depth-D_n cell of the scheme on H_n. These cells move with the sample, so the union bound over fixed cells is replaced by a uniform one: each cell is a box, so

$$M_n \le \max_{|v|=D_n}\mu(\mathbf{x}_v) + \sup_{B\in\mathcal{B}_1}|\mu_n(B) - \mu(B)|,$$

and the second term tends to 0 with probability one by Lemma 8.12(iii) with $m_n = 1$. For the first, let K be a closed box with $\mu(K) = 1$, as in the proof of Theorem 8.13; with probability one $H_n \subseteq K$ for every n, so every depth-D_n cell of the scheme on H_n has volume $2^{-D_n}\mathrm{vol}(H_n) \le 2^{-D_n}\mathrm{vol}(K) \to 0$, and the absolute continuity of μ gives $\max_v \mu(\mathbf{x}_v) \to 0$ as in (i). Theorem 8.13 concludes. □

Marks. (i) proved here; (ii) proved here, new: the training-hull version that §10.2 needs, from Theorem 8.13 and Lemma 8.12. The 2026-11-04 preprint's proof used the Vapnik–Chervonenkis inequality for boxes; the union bound over the depth-D_n cells used in (i) is shorter and needs no citation. [LEAN: to write — (i) on `VC.Coarsen` consistency_of_coarsening and `VC.EmpiricalSLLN`, (ii) on Theorem 8.13's module when it exists, both with MergeNoG lem_mergeNoG's leaf bound.] [SIM: the review's seeded exact simulation (a density with a spike of width 2^{-14}, $\overline{\#}_n = n^{1/2}$, $D_n = \lceil 0.4\log_2 n\rceil$, $n \le 2^{16}$) has (G) failing at every n and the L_1 error decreasing from 1.07 to 1.02, with $\pi^*_{D_n} = s^*_{D_n}$ in every run: consistent with (i) but too short to show the limit (the review's `rev10_nog.py`); the L_1 error of $\pi^*_{D_n}$ along n, with $\overline{\#}_n = n^\alpha$ and $D_n = \lceil b\log_2 n\rceil$, $b < 1 - \alpha$, so that (G) fails for densities with a peak, against $s^*_{D_n}$ on the same samples, on a density that separates the two pavings, and with the training hull for (ii); to run, L1MDE-PROOF-SIMULATION.]

The code at `c0a1837`. The code has no walk that keeps an over-full atom. An over-full record that starts a later subtree stops the run with an error (`MergeEstimator.scala` l.278–279; also l.264–265). An over-full first record of a partition is emitted silently (l.241). So whether a run without (G) stops, or returns an over-full leaf, depends on the partitioner and on the cell's position. With `getCountLimit` (l.82–85) called first, as in the documented workflow and in the terabyte runs, the count limit becomes the larger of the requested limit and the largest depth-D count. Then (G) holds by construction, and the count limit depends on the data.

11.4 The backtrack

Lemma 11.9 (The candidates are states of the chain)**.** *Assume (G) and $\overline{m} > D\lfloor n'/(\overline{\#}+1)\rfloor$, for example $\overline{m} = 1 + \lceil Dn'/\overline{\#}\rceil$ with $\overline{\#} \geq 1$. Start from s^*, and merge a cherry of least count at each step, ties broken in any way, until the root. - (i) The merged nodes $u_1, \ldots, u_K$, $K = |I(s^*)|$, have non-decreasing counts, and each is merged after every internal node below it. Read backwards, the states form a run of SEBTreeMC with depth cap D and leaf cap $\overline{m}$ from the root to s^*. So every state of every backtrack from s^* is a state of an SEBTreeMC path that ends at s^*, and every candidate any search scores is one. - (ii) Under (T) the backtrack from s^* is one path P. A search whose scored states are each reached from s^* by backtrack steps, in one call or in several that restart from earlier scored states, scores only states of P.*

Proof. - (i). $I(s^*) = \{u : \#(u) > \overline{\#}\}$ under (G) (Lemma 8.3). Each step removes a cherry, so K steps reach the root (Proposition 4.4(iii)), and a node is merged only as a cherry, after the internal nodes below it. *Counts.* u_{j+1} was a cherry before step j, and then $\#(u_{j+1}) \geq \#(u_j)$ by the choice at step j; or it became one by the merge of u_j, and then $u_{j+1} = \text{parent}(u_j)$ and $\#(u_{j+1}) \geq \#(u_j)$ by monotonicity. *Backwards.* This is the argument of Proposition 8.6(iii) for the merge queue, with the stopping rules added. Consider the state with internal nodes $\{u_K, \ldots, u_{j+1}\}$, and split u_j. - u_j is a leaf of this state: its parent, if any, is some u_l with $l > j$. - It is a leaf of largest count: the current leaves inside $I(s^*)$ are among $u_1, \ldots, u_j$, with counts at most $\#(u_j)$, and the other leaves hold at most $\overline{\#} < \#(u_j)$ points. - It is splittable: $|u_j| < D$ (Lemma 8.3(i)). - No stopping rule applies: the count criterion fails since $\#(u_j) > \overline{\#}$, and the state has $K - j + 1 \leq K \leq D\lfloor n'/(\overline{\#}+1)\rfloor < \overline{m}$ leaves (Lemma 8.3(i)).

So each backward step is one that SEBTreeMC takes with positive probability, and after the last the chain stops at s^* (Lemma 8.4). For $\overline{m} = 1 + \lceil Dn'/\overline{\#}\rceil$, the inequality is `Pipeline.mbar_bound`. - *(ii).* Under (T) each step is a function of the current state. By induction, each restart is from a state of P, and continues along P. □

Marks. Proved here. [LEAN: (i) Exact back_inv, back_step_is_seb, reach_of_back, seb_reverse_unique and Pipeline mbar_bound; (ii) OneDelta candidates_onPath — built at `a5a2cee59`.] [SIM: 500 random samples, three random tie orders each, 5970 reversed steps: merged counts non-decreasing, each reversed step splits a splittable leaf of largest count, no stopping rule applies before s^*, the chain stops at s^*, and under (T) every restart followed P, no failure (the review's `rev10_backtrack.py`); to rerun at larger depths, L1MDE-PROOF-SIMULATION.]

(G) is needed in (i). Without it the merged paving is π^*_D, which can have an over-full leaf of depth less than D, a leaf that SEBTreeMC would split. The statistical results of §10.4 do not use (i), only the path.

The code at `c0a1837`. `backtrackNumSteps` (`Histogram.scala` l.180–274) merges a cherry of least count, with ties broken by the layout of a queue rebuilt at each call (Remark 10.2). Its cherries include a stored leaf whose sibling holds no point (`Truncation.scala` l.151–182). In the dense paving that is a cherry with an empty child, whose count is that of the stored leaf. So (i) holds for every candidate the code scores, and (ii) does not hold at `c0a1837`.

11.5 The Yatracos counts

The selection compares candidate densities over Yatracos sets, and needs the number of validation points in each set. This subsection proves those counts exact for the rounds of the gap search, and says which validation records each round may scan. Its key fact is that a refinement of states of P is never finer than s^*.

Lemma 11.10 (Refinements of path states)**.** *Let s be a paving each of whose internal nodes holds a point, such as s^* or π_D^*, whose internal nodes hold more than $\overline{\#} \geq 0$ points. Let Θ be a set of pavings that s refines ($I(\theta) \subseteq I(s)$), such as any set of states of a backtrack from s. For $\theta \in \Theta$ let $\operatorname{supp}\theta$ be its leaves that hold points, the stored leaves, and let $N = \bigcup_{\theta\in\Theta} \operatorname{supp}\theta$. - (i) Every leaf of every $\theta \in \Theta$, and every cell of the common refinement $\operatorname{cl}(N)$ (Definition 6.6), is a union of leaves of s: no leaf of s strictly contains it. - (ii) If the sample is nonempty and Θ contains s and the root paving, then $\operatorname{cl}(N) = \mathcal{L}(s)$, all leaves of s, holding points or not. - (iii) Every $\theta \in \Theta$ is constant on every leaf of s. So every Yatracos set of Θ is a union of leaves of s. Merging sibling cells, as MinimiseLeaves does, and dropping cells keep (i).*

Proof. - (i), leaves. Let λ be a leaf of θ. If $\lambda \in I(s)$, the leaves of s below λ partition it. Otherwise λ lies at or below a unique leaf ℓ of s, and ℓ lies at or below a leaf λ' of θ (Lemma 4.2(vi), as $I(\theta) \subseteq I(s)$). Then $\lambda' \sqsupseteq \ell \sqsupseteq \lambda$ are leaves of θ, so $\lambda' = \lambda = \ell$. - *(i), cells.* A cell of $\operatorname{cl}(N)$ in $\max N$ is a leaf of some θ. Let g be a gap of $a \in N \setminus \max N$, and suppose $g \sqsubset \ell$ for a leaf ℓ of s. By maximality of the gap, either $\operatorname{parent}(g) = a$, or $\operatorname{parent}(g)$ meets some $n \in N$ with $n \sqsubset a$. - If $\operatorname{parent}(g) = a$, then $a \sqsubseteq \ell$, and a, a leaf of some θ, is not strictly inside ℓ; so $a = \ell$. Then no element of N lies strictly below a, so $a \in \max N$, a contradiction. - Otherwise n and $\operatorname{parent}(g)$ are nested. $n \sqsupseteq \operatorname{parent}(g)$ would put g inside n, against the definition of a gap. $n \sqsubseteq \operatorname{parent}(g) \sqsubseteq \ell$ forces $n = \ell$, since n is not strictly inside ℓ; then $g \sqsubset n$, again against the definition. - *(ii).* $\max N = \operatorname{supp} s$: a stored leaf of a coarser state is a union of leaves of s by (i) and holds a point, so it contains a stored leaf of s, and no element of N lies strictly inside a stored leaf of s. The other cells of $\operatorname{cl}(N)$ are gaps. A gap is disjoint from $\max N$, so it holds no point and is not internal in s; by (i) it is not strictly inside a leaf of s; so it is a leaf of s (Lemma 4.2(vi)). Conversely, let g be a leaf of s holding no point, $g \neq \rho$. Let a be the least element of N strictly containing g; it exists since the root, which holds a point, is in N. Every element of N strictly below a is disjoint from g: one containing g would contradict the choice of a, and none lies inside g, which holds no point. And g is maximal: $\operatorname{parent}(g) = a$, or $\operatorname{parent}(g) \sqsubset a$ is an internal node of s, which holds a point and so contains a stored leaf of s, an element of N strictly below a. So $g \in \operatorname{gaps}_N(a)$. - *(iii).* Each leaf of s lies inside one leaf of θ by (i), where f_θ is constant. Unions of unions of leaves of s are unions of leaves of s. □

Marks. Proved here; new. [LEAN: PathRefine lem_pathRefine_i ((i), leaves and cells), lem_pathRefine_ii ((ii), with the nonempty sample as stated), lem_pathRefine_iii, lem_pathRefine_iii_keep ((iii), with the dropped cells and MinimiseLeaves's sibling merges), with onLeaves_node, onLeaves_up, onLeaves_leaf, onLeaves_cl, exists_stored_below, cl_not_mem — built at `133fe3210`, on SparseUnion's closure and Canonical's stack form; the states are histograms and the points labels below the leaves of s, the model the statements use.] [SIM: 900 (sample, Θ) cases, with $s = s^*$ and $s = \pi_D^*$ and random sets of states of the backtrack path, $\operatorname{cl}(N)$ computed by brute force from Definition 6.6: no failure of (i) for leaves and cells, of (ii) when s and the root are in Θ, and of (iii) (the review's `rev10_pathrefine.py`); the common refinement by Algorithm 2 against the leaves of s^*, to run, L1MDE-PROOF-SIMULATION.]

One round. Fix a round of the gap search scoring the states $\Theta \subseteq P$, $|\Theta| = h \geq 2$. Under (T), Θ is a chain, from its finest state θ_{f} to its coarsest θ_{c}. Its refinement is formed in three steps (§10.2, Phase 4): 1. the common refinement $C = \operatorname{cl}(N)$, with N the stored leaves of the states of Θ, computed by the sparse union of Proposition 6.7(iii) as the collation of Definition 6.29; 2. the cells of C that are leaves of both θ_{f} and θ_{c} are skipped; 3. MinimiseLeaves (Proposition 6.11) merges sibling cells on which the vector $(f_\theta)_{\theta\in\Theta}$ takes equal values.

Call the result $c_1 \prec \cdots \prec c_M$. Write $A_{\theta,\vartheta} = \{f_\theta > f_\vartheta\}$. The validation data are records (v, κ), $\kappa \geq 1$, spread over q partitions in any way; a partition may hold several records with one label, which (S) below allows and the scan handles.

Lemma 11.11 (The scan)**.** *Let a partition's records satisfy (S) and (R) of Theorem 11.12. When the scan* `scheffeSetsValidationCount` *(*`MinimumDistanceEstimation.scala` *l.319–390 at* `c0a1837`*) processes a record* (v, κ)*, its pointer equals* $|\{l : c_l \prec v\}|$*. It adds* κ *to the entry* (θ, ϑ) *exactly when* $v \sqsubseteq c_l$ *for some* l *with* $f_\theta(c_l) > f_\vartheta(c_l)$*. For* V_j *records it costs* $O(\lambda(M + V_j) + V_j h^2)$ *word operations (Definition 6.1).*

Proof. By (R), v and each c_l are disjoint, or $v \sqsubseteq c_l$. The loop at l.332 advances past c_l while `isLeftOf(v.truncate(|c_l|), c_l)` fails, where `isLeftOf` is "equal or strictly left of" (`NodeLabel.scala` l.137). If $v \sqsubseteq c_l$, the truncation is c_l, and the test holds. Otherwise the truncation (which is v itself when $|v| < |c_l|$) is disjoint from c_l on the side of v, so by Lemma 4.28(a) the test fails exactly when $c_l \prec v$. The set $\{l : c_l \prec v\}$ is an initial segment, since the c_l are sorted, and it can only grow along the partition by (S). So the pointer stops at its size p. If $v \sqsubseteq c_l$, then every $c_{l'}$ with $l' < l$ precedes v while c_l does not, so $l = p$. The test at l.342, $v|_{|c_p|} = c_p$, recognises exactly the records that lie in a cell, and l.357–365 add κ to (θ, ϑ) exactly when $f_\theta(c_p) > f_\vartheta(c_p)$. The pointer moves at most M times, and each record costs $O(\lambda)$ label work and $h(h-1)/2$ passes of the comparison loop (each a three-way comparison of f_θ and f_ϑ). □

Marks. Proved here. [LEAN: Scan scanTest_iff, lem_scan_ptrs, inLeafAt_iff, codeAdds_iff_inSet, scanCount_eq_yCount — built at `a5a2cee59`.] [SIM: with Theorem 11.12's check below.]

Theorem 11.12 (Exactness of the distributed Yatracos-set count)**.** *In one round, as above, assume: - (C) every validation point inside the root box is represented by exactly one record* (v, κ)*, and lies in* $\mathbf{x}_v$*; each record represents* κ *points; points outside the root box are represented by none; - (S) within each partition the records are in left-right order, and their distinct labels form an antichain; - (R) no record label strictly contains a cell* c_l*; - (E) each partition's count matrix enters the total exactly once; - (N) the total passed to the scores is* φn*, the number of all validation points, including those outside the root box, and* $\varphi n < 2^{63}$*.*

Then: - (i) The total for each ordered pair is $N_{\theta,\vartheta}$*, the number of validation points in* $A_{\theta,\vartheta}$*. It does not depend on* q *or on how the records are spread over the partitions. - (ii) The driver's* $\Delta_\tau = \max_{\theta \neq \vartheta} |\int_{A_{\theta,\vartheta}} f_\tau - N_{\theta,\vartheta}/\varphi n|$ *is the score of* τ *over the Yatracos class of* Θ*, with* $\mu_{\varphi n}$ *the empirical measure of all validation points. In exact arithmetic it is unchanged by the skipping and by MinimiseLeaves. It is computed on the driver by one fixed sequence of floating-point operations, so the round's winner is that of a run with one partition. - (iii) Records for every round. If every record label lies at or below a leaf of* s^**, then (R) holds for every* $\Theta \subseteq P$*, with or without skipping and MinimiseLeaves. (S) holds after one sort per partition when each partition's distinct labels form an antichain, as they do for each of the records below. (C) holds for each of the records below. Such records are: the depth-D validation labels; their pushforward to the leaves of* s^**; and the records that the first round of the gap search returns, if it returns them before MinimiseLeaves. For that round contains* $\theta_0 = s^*$ *and the root, so* $C = \mathcal{L}(s^*)$*, nothing is skipped unless* s^* *is the root paving, and the scan returns each record at the leaf of* s^* *that contains it. - (iv) The counts are a function of the leaf counts. Let* $\kappa^*(\ell)$ *be the number of validation points in the leaf* ℓ *of* s^**. Then* $N_{\theta,\vartheta} = \sum\{\kappa^*(\ell) : \ell \in \mathcal{L}(s^*),\ f_\theta(\ell) > f_\vartheta(\ell)\}$ *for every round. The map* κ^* *is the pushforward of the depth-D validation labels by "the leaf of* s^* *containing it", partition-invariant (Proposition 6.18), with at most* $|\mathcal{L}(s^*)| \leq 1 + D\lfloor n'/(\overline{\#} + 1)\rfloor$ *entries. So after one distributed pass every round can be scored on the driver.*

All of this holds with π_D^* *in place of* s^* *when (G) fails.*

Proof. - (i). By Lemma 11.11, partition j's entry (θ, ϑ) is the sum of κ over its records that lie in a cell c_l with $f_\theta(c_l) > f_\vartheta(c_l)$. By (E) the total is the sum of these entries over the partitions,

once each. It is a sum of nonnegative integers below 2^{63} by (N), so it is exact in any order and grouping. A point represented by (v, κ) lies in $\mathbf{x}_{c_l}$ exactly when $v \sqsubseteq c_l$: if $v \sqsubseteq c_l$ this is clear; if the point lies in $\mathbf{x}_{c_l}$, then v and c_l meet, so they are nested, and $v \sqsupset c_l$ is excluded by (R). And $A_{\theta,\vartheta}$ is exactly the union of the cells c_l with $f_\theta(c_l) > f_\vartheta(c_l)$: - outside C every state is 0; - a skipped cell is a leaf of both θ_f and θ_c, hence of every state between them (if $c \neq \rho$, $c \notin I(\theta_\mathrm{f}) \supseteq I(\theta)$ and $\mathrm{parent}(c) \in I(\theta_\mathrm{c}) \subseteq I(\theta)$). With one count and one volume, every state has the same value there, so the cell lies in no $A_{\theta,\vartheta}$; - MinimiseLeaves merges only cells with equal vectors.

By (C), each validation point in the root box is counted once, so $N_{\theta,\vartheta}$ is the number of validation points in $A_{\theta,\vartheta}$. - *(ii).* $\int_{A_{\theta,\vartheta}} f_\tau = \sum_{l:f_\theta(c_l)>f_\vartheta(c_l)} \mathrm{vol}(c_l) f_\tau(c_l)$. A skipped cell lies in no set, and merging two siblings with equal vectors replaces two terms by one of equal value. With (i) and (N), $N_{\theta,\vartheta}/\varphi n = \mu_{\varphi n}(A_{\theta,\vartheta})$. The integrals, the maximum and the first minimiser are computed on the driver from the refinement and the counts, in a fixed order, and by (i) the counts are the same for every partitioning. - *(iii).* Each c_l is a union of leaves of s^*, by Lemma 11.10(i) and (iii). A record label $v \sqsubseteq \ell$, ℓ a leaf of s^*, strictly containing c_l would make $\ell \sqsupset c_l$; so (R) holds. (C) holds for the depth-D labels by Phase 2, and is kept by any map that sends a label to an ancestor. The distinct depth-D labels, and the leaves of s^*, are antichains, and $\prec$ is total on an antichain (Proposition 4.27(iii)), so one sort per partition gives (S); two records at or below one leaf may be nested, and then no order gives (S). For the first round: $C = \mathcal{L}(s^*)$ by Lemma 11.10(ii); a cell common to s^* and the root paving would be ρ, so nothing is skipped unless s^* is the root; every depth-D label lies in a leaf of s^*, and the scan returns a record that lies in a cell c as c (l.342–356). - *(iv).* By Lemma 11.10(iii), each f_θ is constant on each leaf of s^*, and $A_{\theta,\vartheta}$ is the union of the leaves on which $f_\theta > f_\vartheta$. The leaf of s^* containing a depth-D label x is found by a binary search over the stored leaves (Proposition 6.9). If x lies in none, it is $x|_{m+1}$, where m is the larger of $|x \wedge a^-|$ and $|x \wedge a^+|$ for the stored leaves $a^- \prec x \prec a^+$ next to x (those that exist). Indeed, the internal nodes of s^* are the strict ancestors of its stored leaves (each internal node holds a point). The deepest of them above x is $x \wedge a$ for a stored leaf a maximising $|x \wedge a|$, and the maximum is attained at a neighbour: for stored leaves $a \prec a^- \prec x$, $x \wedge a$ contains a and x, hence a^- (Proposition 4.27(vi)), so $x \wedge a \sqsupseteq x \wedge a^-$ and $|x \wedge a| \leq |x \wedge a^-|$; symmetrically on the right. The leaf is its child towards x.

With π^*_D in place of s^*, the same proofs apply, since its internal nodes hold more than $\overline{\#}$ points (Lemma 11.7(i)). □

Marks. (i)–(ii) prove the counting of [San23] exact under (S), (R), (E), (N); hypothesis (C), (N) as φn with points outside the root box, and (iii)–(iv) are new. [LEAN: (i) and (ii) in exact arithmetic, DSSC yCount_append, yCount_flatten, yCount_perm, dssc_partition_invariant, dssc_deviation_invariant, with Lemma 11.11 from Scan scanCount_eq_yCount — built at `a5a2cee59`; the skipping and MinimiseLeaves in (i)–(ii), PathRefine thm_dssc_i, thm_dssc_ii (exact arithmetic, histogram states), with yatracos_cl, skip_flat, inRows_minimise, inSet_record, and (iii)–(iv), thm_dssc_iii_R, thm_dssc_iii_first (a nonempty sample), thm_dssc_iv, kappa_perm, mem_iff_below_stored, point_location — built at `133fe3210`; (iii)'s (C) over the points of the root box and (S) after one sort per partition, DSSCConditions thm_dssc_iii, with condC_depth, condC_map, condC_leaves, depth_records, leaf_records, sortRecs_sorted, thm_dssc_iii_S, hR_of_records, and in (i) count_points (it counts the points in DSSC's InSet; their identification with $A_{\theta,\vartheta}$ is PathRefine's thm_dssc_i) — built at `d5dd60df2`; (S) is proved for partitions whose distinct labels form an antichain, as the families listed are (the first round's records modelled as the pushforward to the leaves), and that nested records can break it is argued in the text, not certified; the aggregated κ^* records (merging the records of one label) are a later row, and (iv)'s binary search (Proposition 6.9) is not modelled; (E) is a hypothesis about Spark and floating point is not modelled.] [SIM: 530 rounds (195 without (G)) with random chains $\Theta \subseteq P$ jumping along P, skipping and MinimiseLeaves, validation records spread

over 1–4 partitions and 10% of the validation points outside the root box: the counts from the depth-D labels, from the round-1 records returned before MinimiseLeaves and from κ^* equalled a brute-force count per point in every round, (S) and (R) held for all three, Δ_τ in exact arithmetic was unchanged by skipping and by MinimiseLeaves, and the point-location formula of (iv) agreed with the leaf of s^* in all 1674 labels; with records handed on between rounds as the code does, (R) broke in 64 rounds and the counts were wrong in exactly those (the review's `rev10_dssc.py`, exhibiting Remark 11.13(a)); on the refined code at $n = 10^7$ (E1c, Theorem 11.16's [SIM] mark): κ^* computed by (iv) (each validation point located in s^*, one pass over each partition) is the same under 1, 2, 7 and 64 validation partitions spread by a hash of the point's id, 3913 leaves holding the 3 000 250 validation points in the root box, and every scored state's Δ in exact integer arithmetic is the same under all four; to rerun at larger depths and at 10^8, L1MDE-PROOF-SIMULATION.]

Remark 11.13 (Which records a round may scan). • *(a) Records handed on between rounds break (R) when rounds jump along P* (asked by the review of Chapter 10, M7). The code at `c0a1837` returns each round's records merged to that round's cells, and a record that lies in no cell is truncated to the deepest cell depth (l.366–380). Take $\overline{\#} = 2$, an s^* with the leaves $\rho\mathsf{LLL}$ (2 points), $\rho\mathsf{LLR}$ (1), $\rho\mathsf{LR}$ (1) and $\rho\mathsf{R}$ (2), and $D \geq 3$. Then P merges $\rho\mathsf{LL}$, then $\rho\mathsf{L}$, then ρ: $\theta_1, \theta_2, \theta_3$.

- A round with $\Theta = \{\theta_2, \theta_3\}$ has the cells $\rho\mathsf{L}, \rho\mathsf{R}$, and returns a validation point of $\rho\mathsf{LLL}$ as the record $\rho\mathsf{L}$.
- A later round with $\Theta = \{\theta_0, \theta_1\}$ skips $\rho\mathsf{LR}$ and $\rho\mathsf{R}$, and keeps $\rho\mathsf{LLL}$ and $\rho\mathsf{LLR}$, whose vectors differ ($\frac{2}{n'v}, \frac{1.5}{n'v}$ against $\frac{1}{n'v}, \frac{1.5}{n'v}$, for v their volume). The record $\rho\mathsf{L}$ strictly contains $\rho\mathsf{LLL}$, so (R) fails, and the scan counts the point in no set, although $\rho\mathsf{LLL} \subseteq A_{\theta_0,\theta_1}$.

Skipping alone breaks (R) as well, even when the windows are nested. With $\overline{\#} = 2$ and the leaves $\rho\mathsf{LL}$ (2), $\rho\mathsf{LR}$ (1), $\rho\mathsf{RLL}$ (2), $\rho\mathsf{RLR}$ (1), $\rho\mathsf{RR}$ (1), (T) merges $\rho\mathsf{L}$ first, then $\rho\mathsf{RL}$. A round with $\{\theta_0, \theta_1\}$ skips everything right of $\rho\mathsf{L}$, keeps cells of depth 2, and returns a point of $\rho\mathsf{RLL}$ as $\rho\mathsf{RL}$, which strictly contains the cell $\rho\mathsf{RLL}$ of a later round with $\{\theta_0, \theta_2\}$. Hence the scheme of §10.2 and Theorem 11.12(iii): records at or below the leaves of s^*, reused unchanged.

- *(b) First-round records returned after MinimiseLeaves.* If the first round minimises before it returns its records, a record sits at a cell m of that round's canonical form, which may strictly contain a later round's cell, so (R) can fail. The counts are still exact. s^* is constant on m, and so is every state of P: by Lemma 11.10(i), a leaf of a state either contains m or is a union of leaves of s^* inside m, with the density of s^* there. So a later round's vector is constant on m.
  - If no cell inside m is skipped, the cells inside m cover m (or none exist), and MinimiseLeaves leaves no cell strictly inside m (Theorem 5.5).
  - If a cell inside m is skipped, the vector is flat there (all states share it as a leaf), hence flat on m, and m lies in no Yatracos set. The scan's pointer passes the cells inside m (`isLeftOf` fails on a strict ancestor) and counts the record nowhere, which is right.

  This proves the counts exact for minimised first-round records, a behaviour read from the code; returning the records before MinimiseLeaves avoids the failure of (R) altogether.
- *(c) The code's zoom.* For the zoom search at `c0a1837`, with the collation fixed and nothing skipped, the records a round returns satisfy (S) and (R) for the next round. The next round's finest state is a candidate of the current round, and its cells are its stored leaves or empty siblings on paths below the current cells, so no returned record strictly contains one of them. (Proof sketch; the full argument, from the 2026-11-04 preprint's Thm-DSSC(iii), is owed to Appendix B.) This is the antichain that the audit of the sorts in §4.6 (Definition 4.29) uses at `MinimumDistanceEstimation.scala` l.482.

Marks. (a) worked examples, checked by hand; (b) proved here; (c) a proof sketch for the code's zoom, which the final algorithm does not run; not pursued further. [LEAN: not authored.] [SIM: (a) both examples reproduced (example 1: the record $\rho\mathsf{L}$ strictly contains $\rho\mathsf{LLL}$, the vectors there are $(8/3, 2)$ and $(4/3, 2)$, and the scan counts the point nowhere while $N_{\theta_0,\theta_1} = 1$; example 2: the point of $\rho\mathsf{RLL}$ returns as $\rho\mathsf{RL}$); (b) a constructed case (count limit 1, s^* with the leaves $8, 9, 10, 11$ of one point each and $\rho\mathsf{R}$; round 1 merges 8–11 to $\rho\mathsf{L}$; the round $\{\theta_0, \theta_1\}$ keeps $8, 9$, minimised to 4, so (R) fails for the record at $\rho\mathsf{L}$ while the counts are exact) and 530 random rounds with minimised round-1 records, no wrong count (the review's `rev10_dssc.py`); both as regression tests in Theorem 11.12's rerun.]

The code at `c0a1837`. `mdeStep` (`MinimumDistanceEstimation.scala` l.406–534) departs from Theorem 11.12's hypotheses in four places. 1. *(E).* In the first round the counts are added to accumulators on the map side of a `reduceByKey` (l.481), read after a `count` (l.487). If an executor is lost before the reduce reads its output, its map tasks run again and add their counts again [The]. The final algorithm returns each partition's count matrix as data and combines the matrices with `treeAggregate` (§10.2), condition (X2) below. 2. *Records handed on between rounds* (l.480–484, l.603). This is exact for the code's zoom when the collation is fixed (Remark 11.13(c)), and not for the gap search (Remark 11.13(a)). 3. *The collation can drop cells.* The nested union guesses the finer operand by the number of stored leaves and corrects a wrong guess with a swap that assigns the same operand to both roles (`LeafMap.scala` l.195–197: `coarser = finer` where `coarser = tmp` was meant). After a wrong guess the common refinement loses the cells that only the coarser operand covers. Then $A_{\theta,\vartheta}$ is computed on too few cells, and Theorem 11.12 does not apply. This is condition (X4) below. 4. *(N).* The total is supplied by the caller (`validationCount`, l.494). The code's root box is the hull of all points, so no validation point lies outside it.

The scan itself is Lemma 11.11, and the driver takes the absolute value after adding the counts of all partitions (l.513), as (ii) requires.

11.6 The rounds of the search

Lemma 11.14 (Rounds of the search)**.**

- *(a) The code's zoom. Let $k \geq 4$, and let the merged histogram have $L_3 \geq k$ stored leaves. Then `getMDE` (`MinimumDistanceEstimation.scala` l.575–621 at `c0a1837`) stops after*

$$R \leq 2 + \left\lceil \log_{(k-1)/2} \frac{2(L_3 - 1)}{k(k-1)} \right\rceil^+$$

rounds, and every round scores at most $2k - 2$ candidates. For $k = 3$ the zoom need not stop: when the middle candidate wins each round and each backtrack step between the candidates removes a stored leaf, it repeats the same round forever.

- *(b) The gap search. Let $k \geq 2$, $A = \operatorname{arc}(0, H) \leq D$ (Lemma 10.3(ii)), and let σ be the largest arc length of one step of P, the largest $|\#(u\mathsf{L}) - \#(u\mathsf{R})|/n'$ over the internal nodes u of s^* (Lemma 10.3(i)). Run the gap search of Appendix A.10. Its first round takes, for $j = 0, \ldots, k-1$, a state nearest in arc length to $jA/(k-1)$. Each bisection takes an interior state of the gap nearest in arc length to the gap's midpoint. A state's score, once computed, is never lowered. Then:*
  - *(i) every round after the first adds at least one state, except the last, which adds none. So the search stops after at most $|P| - |U_1| + 1$ rounds after the first, U_1 being the first round's states, with $|U| \leq |P| = |\mathcal{L}(s^*)| \leq 1 + D\lfloor n'/(\overline{\#} + 1) \rfloor$ scored states;*
  - *(ii) every gap of the first round has arc length at most $A/(k-1) + \sigma$, and a bisection splits a gap of arc length g into two of arc length at most $g/2 + \sigma$. So a gap made by r bisections has arc length at most $(A/(k-1) + \sigma)2^{-r} + 2\sigma$;*

– *(iii) if $\epsilon_0 > \sigma$, no gap is bisected more than $r^* = \lceil \log_2 \frac{A/(k-1)+\sigma}{2(\epsilon_0-\sigma)} \rceil^+$ times. The search then stops after at most r^*+1 rounds after the first, and*

$$|U| \le k + \frac{A+(k-1)\sigma}{\epsilon_0-\sigma} \quad \text{(strictly when } \sigma > 0\text{)}, \qquad |U| < k + \frac{D+(k-1)\sigma}{\epsilon_0-\sigma}.$$

Proof. - *(a)* (as in the 2026-11-04 preprint, with the Lean module's corrections). - *Candidates per round.* Round r steps by $s_r = \lfloor X_r/(k-1) \rfloor$ merges (l.420). Its span is $X_1 = L_3 - 1$ in the first round and $X_r = 2\,\mathrm{sizeDiff}_{r-1}$ in the loop (l.602). It scores $h_r \le 1 + X_r/s_r$ candidates (l.133–137). Since $X_r < (s_r+1)(k-1)$, $h_r \le 2k-2$. The final round (l.613) scores $2\,\mathrm{sizeDiff} + 1 \le k+1$. - *Shrinking.* A backtracking step removes at most one stored leaf, and the winner and its finer neighbour are s_r steps apart, so $\mathrm{sizeDiff}_r \le s_r \le 2\,\mathrm{sizeDiff}_{r-1}/(k-1)$. Hence $\mathrm{sizeDiff}_r \le (2/(k-1))^{r-1} s_1$, with $s_1 \le (L_3-1)/(k-1)$. - *Counting.* Round $r \ge 2$ of the loop runs only if $\mathrm{sizeDiff}_{r-1} > k/2$ (l.600), that is, only if $((k-1)/2)^{r-2} < 2s_1/k$. Add the final round. - $k = 3$. Then $s_r = \mathrm{sizeDiff}_{r-1}$. If the middle candidate wins and each of the s_r steps between the candidates removes a stored leaf, the next round starts from the same finest histogram with the same span; its refinement is the same and, by Remark 11.13(c), its counts are the same, so the same candidate wins, and so on. - *(b)(i).* An unsettled gap holds a state, and its bisection state lies strictly inside, so it is new. Gaps are only split, and a settled gap stays settled: the settling test $\frac{1}{2}\mathrm{arc}_J \le \max(\xi\,\underline{\mathrm{err}}_J, \epsilon_0)$ (§10.4) is monotone in the scores of the gap's two ends, which are never lowered. The search stops when a round adds nothing. At most $|P|$ states exist (Lemma 8.3(i) for the bound on $|P|$). - *(b)(ii).* The arc positions $x_t = \mathrm{arc}(0,t)$ increase from 0 to A in steps of at most σ. So the chosen state for a target lies within $\sigma/2$ of every target that chose it, and consecutive distinct chosen states for targets $p < p'$, with $p' - p = A/(k-1)$, are at most $A/(k-1)+\sigma$ apart. For a gap from u to u' with arc length g, let i_0 be the first index with $x_{i_0} - x_u \ge g/2$. If $i_0 < u'$, then $|x_{i_0} - x_u - g/2| \le \sigma$. If $i_0 = u'$, the state $u'-1$ has $g/2 > x_{u'-1} - x_u \ge g - \sigma$. Either way the nearest interior state is within σ of the midpoint, and both halves have arc length at most $g/2+\sigma$. Iterating, $g_r \le g_0/2^r + \sigma(1 + \frac{1}{2} + \cdots) \le g_0/2^r + 2\sigma$. - *(b)(iii).* A gap of arc length at most $2\epsilon_0$ is settled. A gap made by r^* bisections has arc length at most $2(\epsilon_0-\sigma)+2\sigma = 2\epsilon_0$, so it is settled, and only gaps made by fewer bisections are bisected, in rounds $1, \ldots, r^*$. Round r adds at most the number of gaps, at most $(k-1)2^{r-1}$. So $|U| \le k + (k-1)(2^{r^*}-1)$. If $r^* \ge 1$, $2^{r^*} < (A/(k-1)+\sigma)/(\epsilon_0-\sigma)$, and both bounds are strict. If $r^* = 0$, $|U| \le k$: this is strictly below the D-form, since $D \ge 1$ (and at $D = 0$, $|P| = 1 < k$), but the A-form can be an equality when $\sigma = 0$: the root split into two leaves of $n'/2$ points each gives $H = 1$, $\sigma = A = 0$, and for $k = 2$ the run that scores $\{\theta_0, \theta_H\}$ stops with $|U| = k$; when $\sigma > 0$, $r^* = 0$ gives $|U| \le k < k + (A+(k-1)\sigma)/(\epsilon_0-\sigma)$. □

When $\epsilon_0 \le \sigma$, (b)(i) is the only bound proved here. A big step of P, typically near the root, ends up as a gap with no interior state, which is settled, so it does not stop the search. A bound in terms of the error profile along P is open (§16.9).

Marks. (a) proved in the 2026-11-04 preprint and here; (b) proved here, new. [LEAN: (a) LemRounds cands_le, final_after_loop, shrink, rounds, stops, k3_runs_forever (the constant run `runK3`, one leaf per step), witness_k4 — built at `a5a2cee59`; the clauses that the Scala realises the recursion are argued in the text and not checked; (b) GapRounds `Run.adds_iff`, `Run.stops`, `Run.card_le` ((i)), `Run.pick_within`, `Run.first_gap`, `Run.bisect_halves`, `iterate_halves`, `Run.gap_inv` ((ii), in the division-free form $2^r g \le A/(k-1) + \sigma + 2^r \cdot 2\sigma$), `settled_of_small`, `Run.settled_after`, `Run.card_le_pow`, `Run.lem_rounds_b_iii`, `Run.lem_rounds_b_iii_strict` ((iii), with r^* the least r with $A/(k-1)+\sigma \le 2^r \cdot 2(\epsilon_0-\sigma)$ and the count as $(|U|-k)(\epsilon_0-\sigma) \le A + (k-1)\sigma$, strictly when $\sigma > 0$) — built at `166f3514c`; the search is the structure `Run`, whose fields are the statement's hypotheses; $|P| \le 1 + D\lfloor n'/(\overline{\#}+1) \rfloor$ is Lemma 8.3(i), and $A \le D$ enters as a hypothesis (Lemma 10.3(ii)). The Lean found the A-form's strict count false at $\sigma = 0$; the statement above is corrected there only. The D-form's strict

count without $\sigma > 0$, `Run.lem_rounds_b_iii_strict_D`; the round at which the search stops, `Run.stopRound`, `Run.stopRound_spec`, and the bounds for the set it stops with, `Run.lem_rounds_b_iii_stop`; r^*'s existence, `exists_rstar`; a witness run, `trivialRun`, `run_nonempty`; the first round's ends θ_0, θ_H in every round, `Run.ends_mem` — built at `050df0219`.] [SIM: (a) 1016 simulations of the Lean `Run` clauses for $k = 4, \ldots, 12$ with random and adversarial winners, no bound violation and $h_r \leq 2k-2$ always, and 191 real backtrack paths with lone-child merges and random winners, $\text{sizeDiff}_r \leq s_r$ and the bound in every case; the $k = 3$ loop exhibited with the adversarial winner and one leaf per step; (b) 400 random paths with random arc steps and fixed scores, $k \in [2, 6]$, 189 of them with $\epsilon_0 > \sigma$: no failure of (i)–(iii) (the review's `rev10_rounds.py`); the gap search on real paths at larger depths, to run, L1MDE-PROOF-SIMULATION.]

The code at `c0a1837`. (a) is about the code's zoom; Remark 10.2 uses it for the set Q. With fewer stored leaves than k the step is 0, and the loop of `spacedBacktrack` (l.135) never ends; $k = 1$ divides by zero (l.420). The first round spans $L_3 - 1$ merges, counted against the stored leaves (l.134, l.419). A merge of a lone child keeps the number of stored leaves, so when s^* has empty leaves the zoom never reaches the states nearest the root. The code has no gap search.

11.7 The conditions on the execution, and the theorem

Definition 11.15 (The execution conditions)**.** *A run of the final algorithm meets the execution conditions when: - (X1) Deterministic tasks. Each task's output is a function of the content and order of its input partition and of broadcast values. Every random choice (the split, the partitioner's sample) is seeded by stable keys, such as a run seed, point ids, labels or partition indices of a persisted parent, never by arrival order, executor or time. - (X2) Values, not side effects. Every number that enters the estimate is the value of a job: a reduction by key, `treeAggregate`, `reduce` or `collect`. None is a sum of accumulator updates made inside a transformation. - (X3) The driver-pass condition. No partition relabels its output as the root: a partition whose walk emits one subtree raises it to its ancestor at depth δ. The driver runs $W(0)$ on the sorted union of the partitions' outputs, always. - (X4) The collation condition. Every collation step returns the common refinement of its operands, with the finer operand decided by containment, not by the number of stored leaves. - (X5) The record conditions. Each round's counts come from records that satisfy (C), (S) and (R) of Theorem 11.12, such as those of Theorem 11.12(iii) or the leaf counts κ^* of (iv), with the total φn. - (X6) Every record is checked. The walks compare every record with the count limit, the first record of each partition included, and keep an over-full record as an atom. - (X7) Floating point only on the driver. The integrals, scores and arc lengths are computed on the driver from the merged histogram, the refinement and the counts, in left-right order of the cells and in a fixed order of pairs and states.*

(X1) and (X2) are what Spark must provide. Spark recomputes a lost partition from its lineage and runs failed or slow tasks again, possibly more than once. Under (X1) a task run again reproduces its output. Under (X2) one output per partition enters the value of a job, the output of a repeated task replacing that of the failed attempt. An accumulator update made inside a transformation, by contrast, may be applied once per attempt [The]. So (X1) and (X2) extend the partition invariance of Definition 6.3 and Table 6.1 to every execution history.

Each condition leans on a row of Table 6.1. (X1) is what the `map` and `mapPartitions` rows assume of a task. (X2) keeps every number on the reduction row (`reduceByKey`, `aggregate`, `reduce`), whose combiner is a commutative monoid, and off the accumulator row, whose once-only guarantee holds only inside actions. (X3) and (X6) concern the walk, a `mapPartitions` step that is not a homomorphism for concatenation, so its row asks for an exactness theorem: Lemma 11.4. The driver's walk over the union of the outputs uses the `collect` row's "or a sort first", with the sort row's lawful order on labels (Lemma 4.28). (X4) and (X7) are the `collect`

row: a consumer on the driver made independent of the order of the records, by containment for the collation and by a fixed left-right order for the sums. (X5) puts the counts on the reduction row, as key-wise sums of finitely supported maps.

Theorem 11.16 (Exactness under distribution)**.** *Fix the training and validation samples, the root box, D, $\overline{\#}$, and the parameters of the gap search. Assume (X1)–(X7), and let $\delta \le D$ be safe for the partitioner (Corollary 11.6; range partitioning's $\delta = 1 + \max_i |c_i|$ is at most D). Then, for every such partitioner and δ, every number of partitions, every grouping of the validation records and every execution history: - (i) the merged histogram is the sequential one. Under (G) it is s^*, with its stored leaves and their counts (Lemma 11.4), and it is the terminal state of every run of SEBTreeMC with depth cap D and leaf cap $\overline{m} > D\lfloor n'/(\overline{\#}+1)\rfloor$, for example $1 + \lceil Dn'/\overline{\#}\rceil$ when $\overline{\#} \ge 1$ (Lemma 8.4). Without (G) it is π^*_D (Lemma 11.7); - (ii) the path P, and every arc length along it, are the sequential ones; - (iii) every round scores the same states, with the same validation counts and bit-for-bit the same scores, and the final selection returns the same estimate and the same report; - (iv) so every statement that Chapters 9 to 10 make about the sequential estimate holds for the distributed one: Theorem 9.1 for each round, Proposition 10.1, Theorem 10.4 and Corollary 10.5. Under (G), Lemma 11.9 places every scored state on an SEBTreeMC path that ends at s^*.*

Proof. Phase by phase. 1. *Phases 1 and 2.* The box hull and the counts are reductions in commutative monoids: the counts by Proposition 6.18(iv), the hull being the idempotent monoid of Proposition 6.19(iv). By (X1) and (X2) the value of each job is a function of its input's content. The depth is computed from training counts by such reductions and by driver computations (Lemma 11.18), and fixed-width labels change no label (Lemma 11.19). 2. *Phase 3.* By Lemma 11.4(ii)–(iii) under (G), and Lemma 11.7(iv) without it, using (X3) and (X6). The driver sorts an antichain, so the order is unique. 3. *The path.* P is computed on the driver from the merged histogram's leaves and counts, deterministically under (T). The arc lengths are $|a-b|/n'$ with integer a and b (Lemma 10.3(i)), computed on the driver (X7). 4. *The rounds,* by induction. The states of a round are a function of the arc lengths and of the scores of earlier rounds (Appendix A.10), which by induction are the sequential ones. The collation is computed on the driver (X4). By Theorem 11.12(i), with (X5), and (E) from (X2), every count is the number of validation points in its set. By (X7) and Theorem 11.12(ii), each score is computed by the same sequence of operations from the same inputs, so it is the same bit for bit. The settling tests and the final selection are functions of these. If the rounds after the first are scored on the driver from the leaf counts κ^* (Theorem 11.12(iv)), (E) and (X2) concern only the one pass that computes κ^*. 5. *(iv).* Those statements concern the sequential output for the given samples, and by (i)–(iii) the distributed output is that output. □

Exactness holds sample by sample. The conditions on the run of §10.4, that the root box, the depth and the count limit come from the training points, concern how the samples and parameters arise, and are needed for the probability bounds, not for exactness.

Marks. Proved here. [LEAN: under (G), the two halves composed, ThmExact thm_exact (the merged histogram is the set of point-holding leaves of the terminal state of SEBTreeMC, and every selection procedure that reads the merged histogram and the round counts returns the sequential estimate), distCount_eq_seqCount (the distributed count oracle equals the one-machine one for every round and pair, so it covers every adaptive history of the gap search), thm_exact_regroup, thm_exact_one_partition — built at `bdeef562e`; on the merge half, Pipeline thm_exact_pipeline and Exact thm_exact_distributed, and the count half, DSSC dssc_partition_invariant and dssc_deviation_invariant with Scan scanCount_eq_yCount, built at `a5a2cee59`. With the counts the records carry, (i) as stated, the stored leaves and their counts: ThmExact thm_exact_records (each record carries its partition's local count of its

emitted subtree; the driver walk over the records' own counts outputs the point-holding leaves of the terminal state, its records are those leaves with their counts, and every search reading them agrees with the sequential run), on walk_congr and driver_record_count (under a safe δ a record carries its atom's global count, Lemma 11.4(i), MergeWalk mass_localCounts) — built at `feee9838b`; and at a coarser depth, ThmExact thm_exact_depth (Lemma 11.18(iii)), with every cell a node of s^* as a hypothesis (the premise Lemma 11.9 gives; only that every cell has depth at most D is used); for any leaf cap $\overline{m} > D\lfloor n'/(\overline{\#}+1)\rfloor$, $\overline{\#} = 0$ included, ThmExact thm_exact_mbar, thm_exact_records_mbar, thm_exact_depth_mbar; at range partitioning's $\delta = 1 + \max_i |c_i|$, with $\delta \le D$ and safety proved, not assumed, ThmExact thm_exact_range (on RangeSafe cutDepth_le_of_atDepth, safe_cutDepth); and (iv) written out for Theorem 10.4, ThmExact thm_exact_certified (on CertifiedSearch thm_certified: the distributed estimate's error is at most $3(1+\xi)\,\mathrm{err}_t + 3\epsilon_0 + 4\overline{\Delta})$ — built at `fcb47e5df`. To write: the case without (G), on MergeNoG lem_mergeNoG. (X1)–(X7) are hypotheses about Spark and the code, and are not modelled; floating point is not modelled.] [SIM: the merge and the path, (i)–(ii), on the refined code at $n = 10^7$ (E1a, E1b; `ias.spark` run `e1a-merge`; witnesses in `designs/research/l1mde-simulation-witnesses.md`): a two-dimensional Gaussian mixture, $\overline{\#} = 3163$, $D^{\mathrm{G}} = 17$; range partitioning at $\delta = 1 + \max_i |c_i|$ and subtree partitioning at $\delta =$ `maxSubtreeDepth`, with 1, 2, 7 and 64 partitions, at D^{G} and at $D^{\mathrm{G}} + 5$: all 16 runs return s^* (4656 leaves, all 10^7 points) bit for bit, equal to the one-list walk and to s^* read off the counts; the driver walk merged 4808 collected outputs to 4656 at 64 partitions (Remark 11.5); the backtrack path from s^* is the same at both depths under both rules (Lemma 11.18(iii)). At $n = 10^8$ of the same mixture (`ias.spark` run `e1a-1e8`; 99 999 999 points in the box), $\overline{\#} = 10^4$ and $D^{\mathrm{G}} = 19$: snapped range partitioning ($\delta = 0, 3, 6, 10$) and subtree partitioning ($\delta = 2, 3, 6, 10$) at 1, 2, 7 and 64 partitions, at D^{G} and at 22: all 16 runs return one s^* (14,667 leaves holding every point in the box) bit for bit, equal to the one-list walk and to s^* read off the counts, and the backtrack path (14,666 merges) is the same at both depths under both rules. In that run an executor was killed by the cluster (out of memory, during a local walk at depth 19); Spark dropped it, started another, recomputed the lost blocks and retried, and every run, those after the kill included, stayed bit for bit: an unplanned first instance of a killed executor under (X3)–(X5). The whole pipeline at $n = 10^7$ (E1c; `ias.spark` run `e1c-scores`; each point a function of the seed and its id, split by a hash at $\varphi = 0.3$ into $n' = 6\,999\,749$ training and $\varphi n = 3\,000\,251$ validation points, $\overline{\#} = 2646$, $D^{\mathrm{G}} = 16$): the merge under snapped range and subtree partitioning at 1, 2, 7 and 64 partitions returns one s^* (3913 leaves) in all 8 runs; κ^* (Theorem 11.12(iv)) is identical under 1, 2, 7 and 64 validation partitions; and under both backtrack rules the gap search's scored states, their exact Δ, its rounds and its $\hat{\theta}$ are identical under every validation partitioning (Theorem 11.12's and Theorem 10.4's [SIM] marks). So (i)–(ii) hold end to end on the shipping code at 10^7. To run: E1c at 10^8, and deliberate task failures (a killed executor, speculative execution, a replayed partition), every count and every score compared bit for bit with a one-partition run, L1MDE-PROOF-SIMULATION.]

Corollary 11.17 (The guarantees transfer)**.** *Under the hypotheses of Theorem 11.16, let $\tilde{\Delta}_\theta$ be the computed scores and Δ_θ the real-valued scores of the same counts, and $\varepsilon_{\mathrm{fp}} = 2\max_\theta |\tilde{\Delta}_\theta - \Delta_\theta|$ over the scored states. Then: - Theorem 9.1 for each round, and Proposition 10.1, hold for the distributed estimate with $2\varepsilon_{\mathrm{fp}}$ added; - Theorem 10.4 holds with $(2+3\xi)\varepsilon_{\mathrm{fp}}$ added, the settling tests being made with the computed scores and exact arc lengths.*

Proof. - Thm-Oracle. The computed winner $\hat{\theta}$ satisfies $\Delta_{\hat{\theta}} \le \tilde{\Delta}_{\hat{\theta}} + \varepsilon_{\mathrm{fp}}/2 \le \tilde{\Delta}_\theta + \varepsilon_{\mathrm{fp}}/2 \le \Delta_\theta + \varepsilon_{\mathrm{fp}}$ for every θ: the computed winner minimises Δ_θ up to $\varepsilon_{\mathrm{fp}}$, which is the hypothesis of the conditions on the run of §10.4, so the two definitions of $\varepsilon_{\mathrm{fp}}$ agree. In step 2 of the proof of Theorem 9.1 (§9.3), $2\Delta_{\hat{\theta}} + 2\Delta_\theta \le 4\Delta_\theta + 2\varepsilon_{\mathrm{fp}}$, and the other steps are unchanged. - *Thm-Certified.* The computed lower bound of a gap exceeds the true one by at most $\varepsilon_{\mathrm{fp}}$, so on $\mathcal{E}$ it is at most

$\mathrm{err}(t^*) + \varepsilon_{\mathrm{fp}}$. Step 3 of that proof then gives $\min_U \mathrm{err} \le (1+\xi)\mathrm{err}(t^*) + \xi\varepsilon_{\mathrm{fp}} + \epsilon_0$, and step 1 adds $2\varepsilon_{\mathrm{fp}}$ for the final selection. □

The scores can be exact. On a cell c inside the leaf λ of τ, $\int_{\mathbf{x}_c} f_\tau = \#(\lambda)\, 2^{|\lambda|-|c|}/n'$, since every cell of depth h has volume $2^{-h}\mathrm{vol}(\mathbf{x}_\rho)$. So every integral is an integer over $n'2^D$, every empirical measure is an integer over φn, and every score, every comparison of scores and every settling test can be computed exactly in integer arithmetic on the driver, with numerators of about $D + 2\log_2 n$ bits. Then $\varepsilon_{\mathrm{fp}} = 0$, at a cost that is negligible beside the scans. The arc lengths are integers over n' in any case.

Marks. Proved here. [LEAN: to write — on OracleInequality oracle_finite, with an ε slack.] [SIM: the computed against the exact rational scores on the simulation runs above, recording $\varepsilon_{\mathrm{fp}}$ and whether the winner changes; to run.]

The code at `c0a1837`, against the conditions. - *(X1)* holds where it matters. The partitioner's sample seeds depend on RDD ids (`SubtreePartitioner.scala` l.167–171), which is harmless once Lemma 11.4 holds, since the result does not depend on the partitioner. The data are not persisted and are regenerated by each pass; a seeded generator reproduces them. - *(X2)* fails in the first round's counts (Theorem 11.12's code paragraph). The flag accumulator of Phase 3 is used only for whether it is zero, which repetition does not change. - *(X3)* fails (items 1 and 2 of the code paragraph of §11.1). - *(X4)* fails when the guess of the finer operand is wrong (`LeafMap.scala` l.195–197). In a transliteration of the code run on simulated data (20 runs per data set; read from the code, not run on the cluster), the swap never fired on dense data, changed the collation in 17 runs on line-concentrated data, and in 5 on sparse data (2026-11-04 preprint). - *(X5)* holds for the zoom under (X4) (Remark 11.13(c)). - *(X6)* fails for the first record of each partition. - *(X7)* holds: the integrals and scores are summed on the driver in a fixed order (l.448–465, l.509–528).

So at `c0a1837` the estimate can depend on the partitioner, through (X3) and (X4), and on the execution history, through (X2). Whether any reported estimate differs from the sequential specification is not known. Rerunning only the selection stage with (X4) fixed, on the merged histograms of the earlier runs, settles the collation. The merge defects need the merge rerun.

11.8 The other refinements

Lemma 11.18 (The depth chosen by (G) does not change the estimate)**.** *Let $S_{\overline{\#}} = \{u : \#(u) > \overline{\#}\}$ be as in Definition 8.2, with no cut in depth, and let D^{G} be the (G)-depth. - (i)–(ii) By Lemma 8.3(iii), (G) holds at D exactly when $D \ge D^{\mathrm{G}}$, where $D^{\mathrm{G}} = 1 + \max\{|u| : \#(u) > \overline{\#}\}$ (0 when no node holds more than $\overline{\#}$ points) is finite whenever $\overline{\#} \ge 1$ and the points are distinct; and $s^*_D = s^*$, whose deepest leaves have depth D^{G}, for every such D. - (iii) The sequential specification (merged histogram, path, every count, every score, the estimate) is the same at every $D \ge D^{\mathrm{G}}$. So by Theorem 11.16, the distributed run at D^{G} returns the estimate of a run at any deeper D, for example $D = 60$ whenever (G) holds there. - (iv) If (G) holds at D_0, then $D^{\mathrm{G}} = 1 + \max(\{|u| : |u| < D_0,\ \#(u) > \overline{\#}\} \cup \{-1\})$, computed from the depth-D_0 count map by coarsening (Proposition 6.18(iii)) or one fold (Proposition 6.19), with no new pass over the points.*

Proof. - *(i)–(ii).* Lemma 8.3(iii). - *(iii).* The merged histogram is s^* at every such D (Lemma 11.4(i)). The path uses only the counts of nodes of s^*. A validation point's leaf of s^* is the same at every $D \ge D^{\mathrm{G}}$, since its depth-D label lies below that leaf. So the leaf counts κ^*, and through Theorem 11.12(iv) every count, are the same. The integrals depend on the states alone, and the gap search's choices on these. - *(iv).* Under (G) at D_0, $S_{\overline{\#}} = S_{\overline{\#},D_0}$ (Lemma 8.3(ii)), so the maximum runs over nodes of depth less than D_0, and it is over the

empty set, giving $D^{\mathrm{G}} = 0$, exactly when $\#(\rho) \leq \overline{\#}$. The counts of the nodes of depth at most D_0 are coarsenings of $\mathrm{countOf}_{D_0}$. □

Computed from the training counts, D^{G} keeps P training-determined, as §10.4 requires.

Marks. (i)–(ii) are Lemma 8.3(iii), cited; (iii) and (iv) proved here, new. [LEAN: Depth, from the count map at a finest depth M with (G) there and D^{G} as `gDepth` (iv): condG_iff_gDepth_le ((i): (G) at $D \leq M$ exactly when $D \geq D^{\mathrm{G}}$), condG_gDepth, gDepth_eq_zero_iff, internalSet_coarsen and leafSet_coarsen ((ii): $s^*_D = s^*$ for $D^{\mathrm{G}} \leq D \leq M$), leaf_length_le_gDepth (the leaves have depth at most D^{G}) and exists_leaf_length_eq_gDepth (the deepest has depth exactly D^{G}, built at `fcb47e5df`), condG_mono ((G) at D' gives (G) at every $D \geq D'$, built at `fcb47e5df`), ancOrSelf_truncate_iff and leaf_mass_coarsen ((iii): a point's leaf and every leaf's count, training or validation, are the same at every such D), mass_coarsen and condG_coarsen_iff — built at `bdeef562e`; (iii)'s main claim, Depth seb_terminal_coarsen and seqHist_coarsen (the terminal state and the point-holding leaves with their counts are the same at D as at M), inSet_truncate, yCount_truncate and roundCounts_truncate (the round counts over validation records truncated to D are the same, every cell a node of s^* of depth at most D as a hypothesis, from Lemma 11.9), and depth_invariant (every search reading the histogram with its counts and the round counts returns the same at every $D^{\mathrm{G}} \leq D \leq M$), with ThmExact thm_exact_depth for the distributed run at D — built at `feee9838b`; the path, the scores and the estimate enter as such a search ((X7) of Theorem 11.16).] [SIM: 300 random samples with and without coincident points: (G) at D exactly when $D \geq D^{\mathrm{G}}$, s^*_D constant for $D \geq D^{\mathrm{G}}$, and (iv)'s coarse computation against a direct search, no failure for $\overline{\#} \geq 1$ (the review's `rev10_depth.py`); rebuilt on the Haskell reference in exact arithmetic (run `hs-ref-depth`, 300 trials at finest depth 16, half with coincident points, 163 with (G) there): all of (i)–(iv) held, and where (G) failed at the finest depth it failed at every depth; (ii) and (iii) at $n = 10^7$ on the refined code (E1a, E1b, Theorem 11.16's [SIM] mark): s^* bit for bit and the same backtrack path under both rules at $D^{\mathrm{G}} = 17$ and at $D^{\mathrm{G}} + 5 = 22$; the counts and scores at both depths, to run, L1MDE-PROOF-SIMULATION.]

Lemma 11.19 (Fixed-width labels)**.** *Let $D \leq 62$, or $D \leq 126$ with two words. The heap code of a node of depth $h \leq D$ lies in $[2^h, 2^{h+1})$, so it fits in one signed 64-bit word, or in a 128-bit pair. Consider the label operations the pipeline uses: - the depth (the bit length minus one); - the parent and truncation (right shifts); - the children of a node of depth less than D ($2n$ and $2n+1$); - the sibling (the last bit flipped); - the deepest common ancestor (truncation to the common depth, then a shift by the bit length of the exclusive or); - the left-right order (Lemma 4.28(a)), equality, hashing, and the integer order of the heap codes used by (T).*

On the fixed-width word each computes the same integer as on an arbitrary-precision integer. So every algorithm of Chapters 6 to 11 run on fixed-width labels returns the same labels in the same order, and the pipeline returns the same estimate.

Proof. Two's-complement word arithmetic agrees with integer arithmetic while no intermediate value leaves $[-2^{63}, 2^{63})$, or the 128-bit range. Every operand and intermediate value of the operations listed is a nonnegative integer below $2^{D+1} \leq 2^{63}$, or 2^{127}: - right shifts decrease; - $2n + 1 < 2^{D+1}$ for n of depth less than D; - an exclusive or of two codes is below 2^{D+1}; - bit lengths are at most $D + 1$.

The pipeline forms children only below depth D: point location stops at depth D, and the fills and gaps of the sparse union (Proposition 6.7(iii)) lie between two nodes of depth at most D, so their nodes have depth at most D (Lemma 6.2, Definition 6.6). Hashing is a function of the integer, and a partitioner that hashes differently changes no estimate (Lemma 11.4(ii)). □

Marks. Proved here. [LEAN: Labels enc_injective, enc_bounds, depth_eq_log, enc_parent,

enc_left, enc_right, enc_ancestor, and ScalaJoin xor_size_spec, scalaJoin_eq_join — built at `a5a2cee59`, on unbounded naturals; FixedWidth, on a w-bit word holding its value modulo 2^w read in two's complement, with $D + 2 \leq w$: enc_fits, enc_lt_two_pow_63, enc_lt_two_-pow_127 (the bound), wrap_of_fits, toSigned_wrap, signed_lt_iff, wShr_enc and wShr_enc_-truncate (parent, ancestor, truncation), wChild_enc (children below depth D), enc_sibling_xor and wSibling_enc (the sibling), size_enc (the bit length), xor_fits and wJoin_eq (the deepest common ancestor as the code computes it), wStrictLeft_iff (the left-right order), wEq_iff (equality), lem_fixedWidth_64, and wrap_enc, shift_lt_width, wJoin_shift_lt_width (no intermediate value of the join's shift reaches the width, the proof's bound) — built at `fcb47e5df`. The 128-bit case is modelled as one word with $w = 128$; the code's pair of `Long`s is not modelled, and its agreement is the [SIM] below. Hashing is not modelled (a hash is a function of the integer, the proof's last paragraph), and neither is the conclusion: the Lean proves that each listed operation agrees, and that every algorithm of Chapters 6 to 11 and the pipeline then return the same labels and the same estimate is the argument above.] [SIM: 20 000 random pairs of labels to depth 62: every listed operation in two's-complement 64-bit arithmetic equals the unbounded value, and no intermediate value reaches 2^{63} (the review's `rev10_depth.py`); on the refined SparkDensityTree, `Label64` against `NodeLabel` on 20 000 seeded pairs to depth 62, depth 63 refused: 6 tests, 0 failed (U1, `ias.spark` run `sdt-refined-u1`, 2026-11-11); the 128-bit pair on `Long` pairs against `BigInt`, to run.]

The code at `c0a1837` stores `NodeLabel(lab: BigInt)` (`NodeLabel.scala` l.34), and its join uses the bit length of an exclusive or for labels of one depth (`ScalaJoin`).

The remaining choices of §10.5. These need no lemma of their own. - *The number of partitions, sized to bytes.* Lemma 11.4(ii) and Theorem 11.12 hold for every number of partitions. - *The map-side combine.* Proposition 6.18(iv). - *The persisted data and the partition-invariant split.* (X1). - *The sorted columnar storage.* A sort is a sort; the order read back is the left-right order. - *Exact range partitioning.* A sort and an index, deterministic under (X1), with δ from Corollary 11.6(b).

Chapter 12

Resolution and resources

The mathematics of Chapters 8 to 11 assumes unlimited resources: SEBTreeMC may split any cell that holds data (Definition 8.1 with $D = \infty$), the depths D_n of Corollary 8.11 grow without bound, and every label fits in memory. A run has finite resources: a store that must hold the points, a label of fixed width, a driver of fixed memory, and data recorded at a finite precision. This section says how a run starts (the depth), when it stops, what it estimates, and that every estimate carries its resolution. The last point is the lesson of Chapter 2. In a machine interval experiment the estimator "may not converge in probability to a thin box even if the true measure θ^* is a singleton and all observations are measurable exactly, since the machine precision $p < \infty$", and its diameter "reflects the limit of numerical resolution" [Sai05, §1.3.3]. The L_1 estimator obeys the rule of the sampler of §3.4 in its own form: it returns an estimate at the resolution it reached and says which, rather than an estimate of something it did not resolve.

Throughout, n' is the number of training points, φn the number of validation points, $\overline{\#}$ the count limit, and $\#(u)$ the number of training points in $\mathbf{x}_u$, as in Chapter 11. The heap codes, the depth $|u|$, the ancestor $u|_t$ and the left-right order are those of §4.6.

12.1 Condition (G) and the (G)-depth

Definition 12.1 (Condition (G), the resolution condition)**.** *Fix a root box $\mathbf{x}_\rho$ containing the n' training points, a count limit $\overline{\#} \geq 0$ and a depth D. The training sample satisfies* condition (G) at depth D *when*

$$\max_{|v|=D} \#(v) \leq \overline{\#},$$

that is, when every depth-D cell of $\mathbf{x}_\rho$ holds at most $\overline{\#}$ training points. The (G)-depth D^{G} *is the least D at which (G) holds, and $D^{\mathrm{G}} = \infty$ when there is none. This is the condition inside Definition 8.2, applied to the training sample alone; the validation points, which Phase 2 of §10.2 labels at depth D as well, are not counted.*

(G) is a statement about resolution: the depth-D cells separate the training points into groups of at most $\overline{\#}$. Under it, the merge of Chapter 11 returns the threshold paving s^* (Lemma 11.4), the backtrack from s^* runs SEBTreeMC backwards (Lemma 11.9), and the guarantees of §10.4 and §8.5 apply as stated. The depth at which a run starts is immaterial once (G) holds there, by two results already proved.

By Lemma 8.3(iii), applied to the n' training points in the root box, fixed or computed from them: if (G) holds at D' it holds at every $D \geq D'$, and $s^*_D = s^*_{D'} = s^*$, the paving whose internal nodes are $S_{\overline{\#}} = \{u : \#(u) > \overline{\#}\}$; $D^{\mathrm{G}} = 0$ when $\#(\rho) \leq \overline{\#}$, and otherwise $D^{\mathrm{G}} = 1 + \max\{|u| : \#(u) > \overline{\#}\}$, the depth of s^*; and whenever D^{G} is finite and $\overline{\#} \geq 1$,

$D^{\mathrm{G}} \geq \log_2(n'/\overline{\#})$. By Lemma 11.18(iii) with Theorem 11.16, at every $D \geq D^{\mathrm{G}}$ the final algorithm computes the same merged histogram, the same path P, the same validation counts, the same scores and the same estimate. Two facts are added here.

Remark 12.2 (Ties, and the records at depth D)**.** *(a) For $\overline{\#} \geq 1$, D^{G} is finite exactly when no value occurs more than $\overline{\#}$ times among the training points. Distinct values are separated at some finite depth, since the cells shrink (Lemma 4.23(iii)), after which every cell holds the points of at most one value; and a value occurring more than $\overline{\#}$ times lies in one cell at every depth. For $\overline{\#} = 0$, $D^{\mathrm{G}} = 0$ when $n' = 0$ and $D^{\mathrm{G}} = \infty$ otherwise. (b) The number of occupied depth-D cells, which is the number of training records after Phase 2, is non-decreasing in D and at most $\min(n', 2^D)$: every occupied cell has an occupied child, and holds at least one of the n' points. So a deeper start, once (G) holds, changes the cost, not the estimate (Lemma 11.18(iii)).*

Marks. (a) and (b) proved above. [LEAN: (b) Depth card_support_coarsen_mono, card_support_le_total, card_support_le_pow, card_occupied_le — built at `bdeef562e`; (a) needs the widths of the cells (Lemma 4.23, proposed module `Aspect`), to write; the monotonicity of Lemma 8.3(iii), for depths up to a finest depth at which (G) holds, is Depth condG_iff_gDepth_le.] [SIM: (a), (b) and the two-pass count below, checked exactly on 3000 seeded samples with forced ties (the review's `rev11_gdepth.py`); the merge and the path at D^{G} and at $D^{\mathrm{G}} + 5$ bit for bit at $n = 10^7$ (Lemma 11.18's mark); the scores, to run, L1MDE-PROOF-SIMULATION.]

Finding D^{G}: the two-pass count. Phase 2 of §10.2 computes D^{G} from the training points alone, in at most two passes, so that it is determined by the training points as the conditions of §10.4 require; a third pass then labels every point at D^{G} when $D^{\mathrm{G}} > D_0$. 1. The first pass computes the depth-D_0 count map of the training points at a coarse D_0, combined on the map side (Proposition 6.18). From it, by coarsening, come the counts of every node of depth at most D_0 (Lemma 11.18(iv)), hence the largest count at each depth $d \leq D_0$ and the over-full depth-D_0 cells. 2. If (G) holds at D_0, then $D^{\mathrm{G}} = 1 + \max\{|u| : |u| < D_0,\ \#(u) > \overline{\#}\}$, or 0, by Lemma 8.3(iii), as Lemma 11.18(iv) states; no new pass is needed. 3. Otherwise a second pass labels, at the deepest representable depth $D_{\max}$ of §12.2, only the training points of the over-full depth-D_0 cells: by the monotonicity of counts, a cell over-full at a depth beyond D_0 lies inside one of them. The labels are sorted, one per point with repeats kept, so that the length of a run is a count. The labels below a depth-d node form a contiguous run of the sorted list (Proposition 4.27(vi)), so the depth-d count of a node is the length of the run with that ancestor, and one sweep of the sorted list with a stack of open ancestors returns the largest count at every depth $d \in (D_0, D_{\max}]$ at once: the ancestor counts of a sorted label list are the coarsening of Proposition 6.18(iii) at every depth, as the sweep of Lemma 11.11 computes them. By Lemma 8.3(iii) the predicate "(G) holds at d" is monotone in d, so the sweep gives D^{G}, the outcome that (G) fails at $D_{\max}$ included. One scan does it; a binary search over the depths, at most $\lceil \log_2(D_{\max} - D_0 + 1) \rceil$ scans of one depth each, is the alternative without the stack. 4. If (G) fails at $D_{\max}$, no representable depth satisfies it; the run uses $D_{\max}$ and is the fallback of §12.3.

The training points are then labelled at D^{G}, or at $D_{\max}$: a third pass over all points whenever $D^{\mathrm{G}} > D_0$, since the second touched only the points of the over-full depth-D_0 cells. Labelling every point at $D_{\max}$ in the second pass and truncating (Proposition 6.18) would save that pass at the price of sorting all n' labels; the final algorithm takes the third pass, a map over the persisted points, and Chapter 13 costs it so. The pipeline then continues. Corollary 8.15 says when D^{G} grows slowly enough for consistency: for a bounded density with bounded support and $\overline{\#} = n'^{\alpha}$, it is at most $\lceil b \log_2 n' \rceil$ for all large n', with probability one, on a fixed root box or on the training hull (with n' in place of n, as Remark 8.16 allows for a training sample).

12.2 Resources

Storage. Every training and validation point is held in the distributed store, and Phase 2 maps every point to its depth-D label. A sample that cannot be stored cannot be estimated from. A label is a fixed-width integer, one 64-bit word for $D \le 62$ and two for $63 \le D \le 126$ (Lemma 11.19); the *representable depth* $D_{\max}$ is 62 or 126, according to the label width of the run. The bytes per record are those of Phase 2 of §10.2, and Chapter 13 costs them.

The records. The training records after Phase 2 are the occupied depth-D cells with their counts, at most $\min(n', 2^D)$ of them (Remark 12.2(b)); the validation records are not combined, so there are φn of them, minus those outside the root box.

The driver. It holds the merged histogram, with at most $1+D\lfloor n'/(\overline{\#}+1)\rfloor$ leaves (Lemma 8.3(i) under (G), Lemma 11.7(i) without it), and in each round of the search the common refinement of the round's states with their densities on it. Every cell of that refinement is a union of leaves of the merged paving (Lemma 11.10(i)), so there are at most as many cells as leaves; after the first round the leaf counts κ^* of Theorem 11.12(iv) have at most as many entries. The code at `c0a1837` computes the integrals and scores on the driver (§10.4). So the driver's memory bounds the number of leaves of the merged histogram, and a count limit for which $1 + D\lfloor n'/(\overline{\#}+1)\rfloor$ fits the driver's memory suffices (Lemma 8.3(i)): a sample whose threshold paving does not fit needs a larger count limit, which coarsens the merged histogram, or a final estimate spread over several machines, which neither the code at `c0a1837` nor the final algorithm of Chapter 10 has.

The data's precision. Every side of a depth-D cell is at most $W_\rho 2^{-\lfloor D/d\rfloor}$, with W_ρ the widest side of the root box (Lemma 4.23(iv)); so about D/d bits per coordinate below the scale of the root box decide the depth-D cell of a point. Two points whose records agree lie in one cell at every depth: recorded ties are ties, and by Remark 12.2(a) (G) holds at some depth only if no value occurs more than $\overline{\#}$ times among the records. For a density the points are distinct with probability one, so at a deep enough D every point has a cell of its own and (G) holds for every $\overline{\#} \ge 1$; D^{G} is then finite, and for a bounded density with bounded support and $\overline{\#} = n'^{\alpha}$ it lies between $(1-\alpha)\log_2 n'$ and $\lceil b \log_2 n'\rceil$ for all large n', with probability one (Lemma 8.3(iii); Corollary 8.15(iii) with n' in place of n, Remark 8.16). The machine, however, sees records, not points. A sample of a density recorded at b bits per coordinate is a sample of a discrete law, whose ties are real, and no depth beyond about bd separates them.

12.3 When (G) fails at every representable depth

If $D^{\mathrm{G}} > D_{\max}$, more than $\overline{\#}$ training points share a depth-$D_{\max}$ cell. Either the records are tied, or the representable depth is too shallow for the sample and the count limit. The second case is real in high dimension: at $D_{\max} = 62$ and $d = 10$ a depth-$D_{\max}$ cell has sides at most $W_\rho 2^{-6}$, so a depth-$D_{\max}$ cell in which a density puts a fraction q of its mass holds about qn' training points, and (G) fails for every $\overline{\#}$ below that count, however large the store. Two words give $\lfloor 126/10\rfloor = 12$ bits per coordinate at $d = 10$. And by Corollary 8.15(iv), (G) fails at every representable depth as soon as $n'/\overline{\#} > 2^{D_{\max}}$, so no consistency statement with $\overline{\#}_n/n \to 0$ survives a fixed cap.

The final algorithm of §10.2 then: - uses $D = D_{\max}$; - keeps each over-full depth-D cell as an atom of the merge walk (Definition 11.1), which never extends an atom past the count limit and never splits one; - returns as merged paving π^*_D of Lemma 11.7(iii), the paving whose internal nodes are the nodes u with $|u| < D$ and $\#(u) > \overline{\#}$ whose points do not all lie in one depth-D cell; it is determined by the counts alone, and under (G) it is s^*; - backtracks from π^*_D and selects as before; the statistical results of §10.4 use only the path, not Lemma 11.9, which needs (G); - warns, with the estimate, that it is "a quantised sparse Catalan binary paving with the

given root box and is a quantized integral representation of that partition of the density".

The warning's terms are these.

Definition 12.3 (Integral representation on a paving, and its quantisation)**.** *Let f be a density on $\mathbb{R}^d$, $\mathbf{x}_\rho$ a root box and π an RP of $\mathbf{x}_\rho$. The* integral representation of f on π *is the density f_π that is constant on each leaf cell $\mathbf{x}_l$ of π, with value $\int_{\mathbf{x}_l} f/\mathrm{vol}(\mathbf{x}_l)$, and equals f outside $\mathbf{x}_\rho$. The* density at resolution D *is $f_D = f_{s_D^c}$, the integral representation on the complete tree of depth D. For n' points in $\mathbf{x}_\rho$, the* quantisation of f_π by the counts *is the histogram $f_{n',\pi}$ of Chapter 8: its value $\#(l)/(n'\,\mathrm{vol}(\mathbf{x}_l))$ on a leaf is the sample's estimate of $\int_{\mathbf{x}_l} f/\mathrm{vol}(\mathbf{x}_l)$, an integer multiple of $1/(n'\,\mathrm{vol}(\mathbf{x}_l))$.*

So the warning says: the paving is a Catalan binary tree (Chapter 4) realised on the given root box, stored sparsely by its occupied leaves, and quantised in depth at $D_{\max}$; the estimate $f_{n',\theta}$, for the selected state θ on the path from π_D^*, is the quantisation by the counts of f_θ, the integral representation of f on that partition. It is an estimate of the integrals of f over the cells of θ. What it is not is an estimate of f whose error shrinks with more data at this depth: §12.4 says what shrinks and what does not.

Exactness. The distributed run returns π_D^* for every safe partitioning and every number of partitions (Lemma 11.7(iv)), and the rest of the computation is exact without (G) as with it (Theorem 11.16(i), Theorem 11.12, last clause).

Consistency as the resolution grows. If $D_n \to \infty$ with $D_n/\overline{\#}_n \to 0$ and $\overline{\#}_n/n \to 0$ on a fixed root box, with n' in place of n (Remark 8.16, for a training sample not nested in n), the histogram of $\pi_{D_n}^*$ converges to f in L_1 with probability one, without (G) (Corollary 11.8(i); (ii) of that corollary gives the training hull, for a density with bounded support, under $D_n \log n/\overline{\#}_n \to 0$). This is a statement about a sequence of runs whose representable depth grows, by more words per label, not about one machine.

Under a fixed cap. No consistency for f is claimed, by Corollary 8.15(iv). What converges, for a root box fixed in advance, is the estimate of the cell integrals (Lemma 12.4(iv) below); with the training hull the depth-D cells move with the sample, and the corresponding statement is not made here. Parts (i), (ii) and (v) of Lemma 12.4 are deterministic statements for a given Θ and root box; conditionally on the training points they hold for every root box determined by them, the training hull included, since Δ is formed from the validation points. Only (iv), the $n' \to \infty$ limit, needs the fixed box. Its resolution bias does not shrink with n in either case, as §8.6 already says for data that arrive on a grid.

What the code at `c0a1837` did. It has no walk that keeps an over-full atom. A record over the count limit that starts a later subtree stops the run with "A leaf had exceeded the countLimit from the start, consider splitting more" (`MergeEstimator.scala` l.278–279); the first record of a partition is never compared with the limit (Chapter 11, the code paragraph after Remark 11.5), so whether a run without (G) stops or returns an over-full leaf depends on the partitioner. The library also lets the user raise the count limit first: `getCountLimit` (l.82–85) returns the larger of the requested limit and the largest count among the labelled records it is given, so (G) holds by construction and the count limit depends on the data; the merged histogram is then coarser than the one asked for. Both were read in the archived source at `c0a1837`. The final algorithm does neither: it keeps the atom, returns the estimate and warns, since the estimate is exact, is consistent along a sequence of runs with $D_n \to \infty$ (Corollary 11.8), and says what it is.

12.4 What a run estimates at resolution D

Lemma 12.4 (Estimation at resolution D)**.** *Fix a root box $\mathbf{x}_\rho$, a depth D and a finite set Θ of candidate densities, each constant on the depth-D cells of $\mathbf{x}_\rho$, zero outside it and integrating to one. Let θ^* be the minimum distance estimate over Θ with the validation points, and $\Delta = \Delta(\mathcal{A}_\Theta)$ the deviation of §9.3. Let f be any density, and f_D its density at resolution D (Definition 12.3). - (i) Every Yatracos set of Θ is a union of depth-D cells, and $\int_A f = \int_A f_D$ for each. So Δ is the same for f and for f_D, and Theorem 9.1 holds with f_D in place of f: $\int |f_{\theta^*} - f_D| \le 3 \min_{\theta\in\Theta} \int |f_\theta - f_D| + 4\Delta$. - (ii) For every density g constant on the depth-D cells and zero outside $\mathbf{x}_\rho$, $\int |g - f_D| \le \int |g - f| \le \int |g - f_D| + \int |f_D - f|$. Hence*

$$\int |f_{\theta^*} - f| \le 3 \min_{\theta\in\Theta} \int |f_\theta - f_D| + 4\Delta + \int |f_D - f|.$$

- (iii) The resolution bias $\int |f_D - f| = \int_{\mathbf{x}_\rho} |f_D - f|$ does not depend on the sample, and tends to 0 as $D \to \infty$ for every density f. - (iv) Fix D, and let the training points be i.i.d. with a density f on a root box $\mathbf{x}_\rho$ fixed before they are seen (so $f = 0$ outside it). As $n' \to \infty$, with probability one: the histogram on the depth-D cells satisfies $\int |f_{n',s_D^c} - f_D| \to 0$; the cut-off threshold paving satisfies $\int |f_{n',s_D^*} - f_D| \to 0$; and, with π_D^∞ the paving whose internal nodes are the nodes u with $|u| < D$ that contain at least two depth-D cells of positive μ-mass, $\pi_D^* = \pi_D^\infty$ for all large n' and $\int |f_{n',\pi_D^*} - f_{\pi_D^\infty}| \to 0$. Here $\pi_D^\infty = s_D^c$ exactly when every depth-D cell has positive mass; in general a leaf of π_D^∞ holds at most one depth-D cell of positive mass. - (v) Proposition 10.1 and Theorem 10.4 hold with f_D in place of f, with the same Δ_P, the same event $\mathcal{E}$ and the same constants.

Proof. - (i). $A_{\theta,\vartheta} = \{f_\theta > f_\vartheta\}$ lies in $\mathbf{x}_\rho$, since both are zero outside, and is a union of depth-D cells, since both are constant on them. On a cell v, $\int_v f = \mathrm{vol}(\mathbf{x}_v) f_D(v) = \int_v f_D$; sum over the cells of A. The proof of Theorem 9.1 uses of the target only that it integrates to one and its integrals over the sets of the class, so it holds for f_D. - *(ii).* On a depth-D cell C where $g = c$, $\int_C |c - f| \ge |\int_C (c - f)| = |c\,\mathrm{vol}(C) - \int_C f| = \int_C |c - f_D|$. Outside $\mathbf{x}_\rho$, $g = 0$ and $f_D = f$. Sum over the cells for the first inequality; the second is the triangle inequality. The display is (i) and the triangle inequality $\int |f_{\theta^*} - f| \le \int |f_{\theta^*} - f_D| + \int |f_D - f|$. - *(iii).* The map $h \mapsto h_D$ is linear and $\int_{\mathbf{x}_\rho} |h_D| \le \int_{\mathbf{x}_\rho} |h|$ for integrable h, since the absolute value of an average is at most the average of the absolute value. Given $\epsilon > 0$, take g continuous on the closed root box with $\int_{\mathbf{x}_\rho} |f - g| < \epsilon$. The cells of depth D have diameter at most $\mathrm{diam}_D \to 0$ (Lemma 4.23(iii)), and g is uniformly continuous, so $\sup |g_D - g| \le \omega_g(\mathrm{diam}_D) \to 0$. Then $\int |f_D - f| \le \int |(f - g)_D| + \int |g_D - g| + \int |g - f| \le 2\epsilon + \mathrm{vol}(\mathbf{x}_\rho)\,\omega_g(\mathrm{diam}_D)$. - *(iv), the depth-$D$ histogram.* The depth-D cells are finitely many and fixed, and carry all the mass of f. By the strong law, $\#(v)/n' \to \mu(\mathbf{x}_v)$ for each, so $\int |f_{n',s_D^c} - f_D| = \sum_v |\#(v)/n' - \mu(\mathbf{x}_v)| \to 0$. - *(iv), the pavings.* With probability one, for all large n', every node u of depth at most D with $\mu(\mathbf{x}_u) > 0$ has $\#(u) > \overline{\#}$, since its count grows linearly, and every node with $\mu(\mathbf{x}_u) = 0$ has $\#(u) = 0$ for every n'. Then, for $|u| < D$: $u \in S_{\overline{\#},D}$ exactly when $\mu(\mathbf{x}_u) > 0$; and $u \in I^*$ (Lemma 11.7(iii)) exactly when u contains two depth-D cells of positive mass, since then both hold points, while if u contains one such cell all its points lie in it, and if none, $\#(u) = 0$. A paving is determined by its internal nodes (Lemma 4.2(i)), so s_D^* and π_D^* are eventually the fixed pavings s_D^∞, with internal nodes $\{u : |u| < D,\ \mu(\mathbf{x}_u) > 0\}$, and π_D^∞. On a fixed paving the histogram converges to the integral representation, as for the depth-D cells. A leaf of s_D^∞ is a depth-D cell of positive mass, where $f_{s_D^\infty} = f_D$, or a cell of mass zero, where both are 0; so $f_{s_D^\infty} = f_D$. Every node of depth less than D contains two cells of positive mass exactly when every depth-D cell has positive mass; a leaf of π_D^∞ is not internal, so it holds at most one. - *(v).* The proofs of Proposition 10.1 and Theorem 10.4 use of f only its integrals over the Yatracos sets of states of P, through Theorem 9.1 and Lemma 10.3(iv), and that it integrates to one.

Every state of P is a coarsening of the merged paving (Lemma 11.10(iii)), whose leaves are unions of depth-D cells, so by (i) those integrals are the same for f_D; the event $\mathcal{E}$ concerns μ and $\mu_{\varphi n}$ on the same sets. □

Marks. Proved here; (iv) and (v) new. The chain (i)–(ii) is that of the 2026-11-04 preprint, restated for the final algorithm. [LEAN: to write (proposed module `ResolutionD`): (i)–(ii) and (v) on `OracleInequality oracle_finite` with the target replaced by its quantisation; (iii) by density of continuous functions; (iv) on `VC.EmpiricalSLLN` and `MergeNoG lem_mergeNoG`.] [SIM: (i)–(iii) checked exactly on 2000 seeded instances, a density constant on 16 cells with candidates at depths 0–4: the Yatracos sets unions of cells with $\int_A f = \int_A f_D$, both inequalities of (ii), the contraction of (iii), and the bias of $f(x) = 2x$ equal to 2^{-D-1} (the review's `rev11_resolution.py`); (iv) on a seeded run along $n' = 2^6, \dots, 2^{15}$ at $D = 3$, $\overline{\#} = 5$, for a density with two zero-mass depth-3 cells: $s^*_D = s^\infty_D$ and $\pi^*_D = \pi^\infty_D$ from $n' = 64$ on, the distances to f_D and to $f_{\pi^\infty_D}$ falling from 0.25 to 0.008 while $\int |f_{\pi^\infty_D} - f_D| = 29/119$ stays, and the two structural claims about π^∞_D on all 256 patterns of zero-mass cells (the review's `rev11_fixedD.py`); for the densities with exact cell integrals of §9.6, the resolution bias against D and the gap search's error against the bound of (ii) at several D, to run, L1MDE-PROOF-SIMULATION.]

The same chain holds with f_π in place of f_D for any paving π of which every candidate is a coarsening, in particular the merged paving itself (Lemma 11.10(iii)): that is the representation the warning of §12.3 names, "of that partition". Part (iv) says what a run at a fixed depth converges to. The histogram on the depth-D cells, and the cut-off threshold paving s^*_D, converge to f_D. The merged paving π^*_D converges to a coarser representation, since the walk raises an over-full atom over empty siblings; the two agree when no depth-D cell of the root box has mass zero, as for a density positive on the root box.

Neither bound implies the other. Write $m = \min_\theta \int |f_\theta - f|$, $m_D = \min_\theta \int |f_\theta - f_D|$ and $b = \int |f_D - f|$. By (ii)'s first inequality $m \geq m_D$, and by the contraction in the proof of (iii) $m \geq b/2$; but $3m \geq 3m_D + b$ fails in general. When f_D is itself a candidate, $m_D = 0$ and $m \leq b$, and (ii) is the sharper by at least $b/2$; when every candidate lies on one side of f on each cell, $m = m_D$ and the plain bound is the sharper by b. What (ii) separates is what the candidates can approach, f_D, from what no candidate at resolution D can, f. Why the selection still pays at a fixed D: by (iv) the depth-D histogram alone converges to f_D, so for n' large enough it could be returned as it is. For moderate n' it is noisy, with up to $\min(n', 2^D)$ cells of few points each, and f_D is constant on groups of cells that are not known. The backtrack states are the groupings the counts suggest, and by (v) the selection among them keeps its guarantee, now against the best grouping of f_D on the path, at every n'.

12.5 The selection checks its own resolution

The safety net. The first round of the gap search contains both ends of P: the merged histogram, the finest state, and the root (§10.2, Phase 4; §10.4). If the final winner lies strictly inside P, the selection compared it with finer and with coarser states and preferred it. If the winner is the finest state θ_0, a finer start may do better: lower $\overline{\#}$, which enlarges $S_{\overline{\#}}$ and refines s^*; if (G) then fails at the current depth, raise D to the new D^{G} (§12.1); run again while the resources of §12.2 allow. This is a procedure, not a guarantee: Theorem 10.4 compares the estimate with the states of one path, for one count limit, and says nothing across count limits. In the other direction, for a density uniform on the root box the root state is the density itself, the finer states add noise, and the selection is expected to stay at or near the root whatever D is; this is to be checked by simulation. If no finer start is possible, the estimate is reported as in §12.3.

The report. Besides the estimate, the report of §10.2 carries ξ, ϵ_0 and $\overline{\Delta}$, the length $|P|$ and

the failure bound $2|P|^2 e^{-2\varphi n\overline{\Delta}^2}$, the depth D and whether (G) held. This section adds: the representable depth $D_{\max}$ and, when (G) held, D^{G}; when it failed, the number of over-full depth-D cells and the largest count, with the warning of §12.3 verbatim, and the number of leaves of π^*_D raised over empty siblings, the over-full leaves of depth less than D, which is where π^*_D departs from s^*_D; and whether the winner is the finest state. With these a reader can apply Theorem 10.4, Lemma 12.4 and the safety net without rerunning anything.

At `c0a1837` the check exists as a console message. In `getMDE`, after the first round, "WARNING: best histogram is the largest, consider larger starting histogram!" is printed when the winner has as many stored leaves as its finer neighbour (`MinimumDistanceEstimation.scala` l.595–596), and only with verbose output on; `mdeStep` prints a copy with a doubled word under the same condition (l.303–304). Both were read in the archived source. The final algorithm returns the flag with the estimate.

The open stopping rule. A rule that refines while the finest candidate wins, with a guarantee on the result, is open. Lepski-type selection along an ordered family [GL11] is the natural tool, and §16.9 lists the tool among those for the open questions of the search. The choice of $\overline{\#}$ is the smoothing parameter one level above the path; the guarantees of this monograph are within a path.

12.6 The duality with the auto-validating sampler

The two halves of this monograph's title, exact simulation and L_1 estimation, are the two directions of one construction on one data structure.

The sampler. The Moore rejection sampler of §3.2 partitions its domain into boxes $\Theta^{(1)}, \dots, \Theta^{(W)}$, encloses the range of the target shape p^* over each by its interval extension, $p^*(\Theta^{(i)}) \subseteq [\underline{P}^*(\Theta^{(i)}), \overline{P}^*(\Theta^{(i)})]$, and proposes from the upper histogram $\sum_i \overline{P}^*(\Theta^{(i)})\mathbf{1}_{\Theta^{(i)}}$ [Sai05, eqs. 2.1–2.3]. Its samples are exact for every partition [Sai05, Theorem 7], [SY13, Theorem 4]. Its acceptance probability is at least $\sum_i \mathrm{vol}(\Theta^{(i)})\underline{P}^*(\Theta^{(i)}) / \sum_i \mathrm{vol}(\Theta^{(i)})\overline{P}^*(\Theta^{(i)})$ (the dissertation writes $d(\Theta^{(i)})$ for the length) and tends to one at least linearly with the mesh for a Lipschitz target [Sai05, Theorem 8 and p.72], [SY13, Theorem 4]; in the dissertation's examples a uniform partition into 8192 intervals, or an adaptive one into 2940, 523 or 717, reaches 0.97–0.99 [Sai05, Table 2.4]. The 2005 rule bisects a box along its side of maximal width when the width of its enclosure exceeds a tolerance [Sai05, p.72]; the 2013 rule takes the box of largest $\mathrm{vol} \cdot \mathrm{wid}$ from a priority queue (§3.2). With the first-widest convention these partitions are regular pavings (§4.5), and the sampler's envelope is an interval-valued mapped tree in the sense of §5.5.

Two integral representations on one paving. The dissertation's bound on the acceptance rests on one display: the normalising constant $N_p = \sum_i \int_{\Theta^{(i)}} p^*$ lies in $\sum_i \mathrm{vol}(\Theta^{(i)})\, P^*(\Theta^{(i)})$ [Sai05, p.70]. That is, on each cell the enclosure times the volume encloses the integral of the target over the cell; the upper endpoints give the envelope, the lower the acceptance bound. In the terms of §5.5, an $\mathbb{IR}$-MRP (s, F) that encloses p^* (Definition 5.16, Theorem 5.15) carries on each leaf l an interval $\mathrm{vol}(\mathbf{x}_l) \cdot F(l)$, the interval scaled by the volume, containing $\int_{\mathbf{x}_l} p^*$, and its width, $\mathrm{vol}(\mathbf{x}_l)\,\mathrm{wid}\, F(l)$, is the priority of §6.14; HullPropagate keeps the enclosure at every coarser node (Proposition 6.50(ii)). The estimator carries on each leaf of its paving a count, whose quantisation $\#(l)/(n'\mathrm{vol}(\mathbf{x}_l))$ estimates $\int_{\mathbf{x}_l} f/\mathrm{vol}(\mathbf{x}_l)$ (Definition 12.3). Both are integral representations of a density on a Catalan binary tree realised on a box: the sampler's is an enclosure, guaranteed and two-sided, computed from the expression of the density; the estimator's is an estimate, statistical, computed from points of the density. This much is proved in the sections cited; the pairing is an observation.

The direction of the map, and where the resolution sits. Simulation goes from the density to points, at the resolution of the envelope. The samples are exact whatever the partition; the partition sets the acceptance rate. So resolution costs trials, never correctness, and when the enclosure is too loose the sampler returns no sample within its budget rather than a wrong one (§3.4). Estimation goes from points to f_D, at the resolution of the paving. The computation is exact whatever D (Theorem 11.16); D sets the resolution bias (Lemma 12.4). So resolution costs storage and bias, never the exactness of the arithmetic, and when the data cannot be resolved the run returns its estimate with its resolution rather than an estimate of f. Chapter 1 put it as the inverse problem; here it is the same tree read in the two directions.

The same tree arithmetic runs both. Both refine by a priority queue on the trees of Chapter 4, and both queues are threshold rules. The sampler's 2013 queue, and RPQEnclose$^{\triangledown}$ of §6.14, split a leaf of largest vol·wid, a strict monotone priority (Proposition 6.51(v)); SEBTreeMC splits a leaf of largest count, the monotone priority ψ_D of §8.2; so the states of both are threshold pavings up to ties (Proposition 6.51(i)). Both coarsen by pruning a cherry of smallest priority: RPQEnclose$^{\triangle}$ (Proposition 6.51(ii)), and the merge queue and the backtrack (Lemma 8.5, Lemma 11.9, Proposition 8.6(iii)). The sampling side had, in 2005, a finest resolution, a coarser probing resolution, which §3.5 reads as a truncation of labels, and counts on the cells. An observation, not a theorem: the two priorities are the two sides' measures of how uncertain a cell's integral is, the width of its enclosure on one side, and on the other the count, which fixes the sampling error of the empirical cell integral. A direction (Chapter 16): a loop between the two, in which the sampler draws from a density given by an expression, the estimator estimates it at a resolution D, and the estimate is checked against f_D computed from the enclosure, as §9.6 checks the bounds on mapped pavings with exact cell integrals; and the sampler's final paving as the estimator's starting paving.

Back to Chapter 2. An estimate at a finite resolution says its resolution: in the machine interval experiment, the diameter of the estimated box carries the limit of numerical resolution, and the experiment "can account for the physical limits on empirical as well as numerical resolutions" [Sai05, §1.3.3]. A sample from the sampler is exact, and its resolution is paid in trials. An estimate from the estimator is exact as a computation, and its resolution is paid in bias; the run reports D, whether (G) held, and, when it did not, the warning. Neither returns a number that pretends to a resolution it does not have.

Chapter 13

Cost on real machines

Chapter 6 counts the work of each operation in word operations on labels (Definition 6.1), and Chapter 11 proves that the distributed pipeline of Chapter 10 returns the sequential estimate. Neither says how long a run takes, how much memory it needs, or whether it fits a given cluster. This section is the model that does: it sizes a run before it is paid for, and it reads the timings of a run as a test of itself. It is built in five layers, from the bytes of one record up: the record (§13.1), the machine (§13.2), the operators (§13.3), the pipeline (§13.4) and the calibration (§13.5). The first four were drafted layer by layer in the research record of this monograph, layers 1 and 3 with an adversarial review each; the fifth is the protocol under which the runs of Chapter 14 are made.

What it is not. It is not a benchmark. No number in it is a result of this monograph's own runs, since none has been made (§14.1). The timings it mentions are those of the earlier terabyte runs (§14.2), which recorded neither the executor heap nor the bytes moved (§14.3); the model reads them as hypotheses to test, not as findings.

The discipline. Every number below is one of three things, and the text says which: - **[A]** a *model assumption*: a constant read from the code at `c0a1837` [SGWS23a], from the Spark documentation, or from a standard JVM layout, which a measurement can overturn; - **[M]** a *measured value*, named with the run that measured it; - **[D]** a *derived prediction*, arithmetic from [A] and [M] values by a stated law.

A number that is none of these does not appear. Where the layers below say "about", the value is an [A] or a [D] from one.

Notation. As in Chapter 10 and Chapter 11: n' training points, φn validation points, d dimensions, the finest depth D, the count limit $\overline{\#}$, and $\lambda = \lceil (D+2)/\omega \rceil$ words per label (Definition 6.1). N_D is the number of occupied depth-D training cells, the records after Phase 2; N_V the number of validation records; L' the records collected after the local merges; M_r the rows of the common refinement of round r, and h_r the states that round scores (§10.4); $\Phi = 1 + D\lfloor n'/(\overline{\#}+1) \rfloor$ is the leaf bound of Lemma 8.3(i). The symbols L, H and W are not used for counts here, since §10.4 uses L and H for a path and its index and §10.3 uses W for the walk and the sampled weight; r is the round index throughout. A stage has p partitions, one task each; the cluster has n_{w} workers of c task slots, $m = n_{\mathrm{w}}c$ slots in all; $\nu = N/p$ is the records per task of a stage that reads N records, and $\nu_V = N_V/p_V$ for the validation stage. H_{e} is the executor heap and H_d the driver's. Bytes per record are s with a subscript for the layout.

13.1 Layer 1: the record

A datum enters the pipeline as a point and leaves Phase 2 as a label with a count. The bytes of each form fix every later layer.

Definition 13.1 (Record layouts)**.** *A* record *is a pair (label, count) with the label a node of depth at most D in the heap code of §4.6 and the count a 64-bit integer. Three layouts are distinguished. The* primitive *layout holds the label as one 64-bit word for $D \le 62$, or two for $63 \le D \le 126$ (Lemma 11.19), and the count as one word, with no per-record object. The* row *layout is Spark's internal row of two 64-bit fields. The* heap *layout is a JVM object graph: a tuple of a label object and a boxed count, the label an arbitrary-precision integer, as the code at* `c0a1837` *holds it. The* two's-complement length *of a label of depth D is $\ell_D = \lceil (D+2)/8 \rceil$ bytes, one sign bit included [San23]; it is λ at $\omega = 8$.*

Proposition 13.2 (Bytes per record)**.** *Let $D \le 62$ unless said otherwise. - (i) A point in $\mathbb{R}^d$ of double coordinates takes $8d$ bytes raw [D], and about $8d + 32$ bytes as a Spark MLlib dense vector on the heap (two object headers of 16 bytes and the array) [A]. - (ii) A record takes $s_{\text{raw}} = \lceil (D+1)/8 \rceil + 8$ bytes as raw bits, $\ell_D + 8$ bytes in two's complement, $s_{\text{prim}} = 16$ bytes in the primitive layout, $s_{\text{row}} = 24$ bytes in the row layout (one null-bit word and two fields) [A], and $s_{\text{heap}} \approx 144$ bytes in the heap layout at $D = 60$ [A], namely a tuple (24) with a boxed count (24), the label wrapper (16), the arbitrary-precision wrapper (16), its integer (about 40) and its magnitude array ($16 + 4\lceil (D+1)/32 \rceil$, rounded up to a multiple of 8). At $D = 60$ the raw bits and the two's complement both take 16 bytes, as the primitive layout does by construction [D]. For $63 \le D \le 126$ the second label word (Lemma 11.19, Chapter 12) adds 8 bytes to the primitive and row layouts, 24 and 32 [A]; it costs one more word per record in every shuffle, sort, collect and broadcast, and nothing else in any stage. The heap layout changes one depth later: a depth-63 code has 64 bits, two ints of the magnitude array, so the heap record is about 144 bytes for every $D \le 63$, and the array's third int makes it about 152 for $64 \le D \le 126$ [A]. - (iii) A validation record is one per validation point inside the root box, uncombined (§10.2, Phase 2): $N_V \le \varphi n$ records of the same sizes [D]. - (iv) The merged histogram has between $\lceil n'/\overline{\#} \rceil$ and Φ stored leaves under (G) [D], each a record. - (v) The common refinement of a round has $M_r \le \Phi$ rows [D]. A row carries the label and h_r densities: $8 + 8h_r$ bytes in the primitive layout, which is what the final algorithm broadcasts (a fixed-width label and h_r doubles, Lemma 11.19), and about $96 + 24 + 16 + 8h_r = 136 + 8h_r$ bytes on the heap at $D = 60$ in the form of the code at* `c0a1837` *[A], where 96 is the heap label, 24 the tuple and 16 the header of the array of densities. For the code's zoom, $h_r \le 2k - 2$ (Lemma 11.14(a)); at $k = 10$ and $h_r = 18$ that is 152 bytes in the primitive layout and 280 on the heap [D]. - (vi) A count matrix is $h_r(h_r - 1)$ 64-bit integers per partition: 306 integers, 2448 bytes, at $h_r = 18$ [D].*

Proof. (i), (ii): the widths of the primitive forms follow from Lemma 11.19, which places a depth-$h \le D$ code in $[2^h, 2^{h+1})$, so that $D+1$ bits suffice and one 64-bit word holds $D \le 62$; the heap figures are a standard object layout with compressed references, each component's width as listed. (iii) Phase 2 labels the validation points and does not combine them. (iv) Under (G) every stored leaf holds between 1 and $\overline{\#}$ points (Lemma 11.4(i)), and the stored leaves hold every training point, so there are at least $n'/\overline{\#}$ of them; there are at most $|\mathcal{L}(s^*)| \le \Phi$ (Lemma 8.3(i)). (v) The rows of a round's refinement are leaves of a binary tree whose internal nodes lie in $S_{\overline{\#}}$ (Theorem 11.12(iv), Lemma 11.10), so $M_r \le \Phi$; the bytes are those of a label and h_r doubles. (vi) One entry per ordered pair of distinct states (Lemma 11.11). □

Marks. (i)–(vi) are arithmetic from the layouts stated as hypotheses. The heap figures of (ii) and (v) are [A]: the arbitrary-precision integer's fields were verified with `javap` on OpenJDK 21 and scala-library 2.12.14, a patch or major version from the runs' JDK and Scala

2.12.15 (the research record, review R2 of layer 1); they are measurement 1 of §13.5. [LEAN: Sizing (the proposed `RecordBytes` is part of it) prop_recordBytes_ii, prop_recordBytes_ii_60, prop_recordBytes_ii_two, heapBytes_le63, heapBytes_two, heapBytes_63, enc_fits_labelWords, rawBits_eq ((ii)), prop_recordBytes_iv ((iv), under (G), on `ThresholdPaving`), prop_recordBytes_v, prop_recordBytes_v_18, prop_recordBytes_v_rows ((v); $M_r \le \Phi$ with Lemma 11.10's predicate restated, not composed with `PathRefine`), prop_recordBytes_vi ((vi)) — built at `b6ea25813`, machine-checked-structural for the module (see Lemma 13.9 and Proposition B.9); the layout table is a hypothesis here, so the heap figures stay [A]; (i) and (iii) not built. Finding: (ii)'s 152 heap bytes hold from $D = 64$; at $D = 63$ the code has 64 bits, two ints, and 144 (`heapBytes_63`).] [SIM: measurements 1–4 of §13.5.]

The serialised forms [A], each to be measured (measurements 2 and 3). Java serialisation, the default of the earlier runs [A, from the notebooks' empty Spark configuration], writes a class descriptor once per stream and, per record, the object headers, the integer's fields and its magnitude: several times s_{raw}, the factor unknown until measured. Kryo with registered classes writes the integer as a length and its bytes, about $1 + \ell_D$, and the count as a variable-length integer of at most 9 bytes: about 18 bytes at $D = 60$. Parquet stores a binary label column as a 4-byte length plus ℓ_D, about 12 bytes before encoding and compression, with the counts bit-packed; a sorted fixed-width integer column with delta encoding would take a few bytes per label. Spark's encoder for an arbitrary-precision integer is a 38-digit decimal, about 126 bits, which caps the depth of a label stored that way.

The depth cap. Fixed-width labels change no label (Lemma 11.19) and cap D at 62 or 126. The other cap is numerical (§2.1): the labelling computes each midpoint as the mean of two doubles, and a cell's midpoint stays strictly inside it only while the cell's width exceeds about one unit in the last place of the coordinates. Since the labelling halves the widest axis (`SpatialTree.scala` l.81–85 and l.116–134 at `c0a1837`), axis i is halved about $D/d + \log_2(w_i/\overline{w})$ times, $\overline{w}$ the geometric mean of the root widths, so a label of depth D is meaningful for about $D \lesssim 52d$ on a near-isotropic root box [D]; tighter along the widest axes; safe at $D = 60$ in two and in ten dimensions. The depth at which a midpoint first meets an endpoint, for a given root box, is measurement 5. Chapter 12 treats what the estimate means at the cap.

The terabyte runs through layer 1 (the two-dimensional uniform run of §14.2; every figure [D] from the thesis's parameters [A]). With $n' = 4.375 \cdot 10^{10}$ training points at $D = 60$ and the run's count limit $\overline{\#} = 10^5$ [A, the thesis's Table 4.2 and its notebooks as the research record transcribes them: at most 2 points in any depth-60 cell, about 800 colliding pairs], almost every depth-60 cell holds one point, so $2.2 \cdot 10^{10} \le N_D \le n'$ and $N_D \approx n'$: about $7 \cdot 10^{11}$ bytes raw and about $6.3 \cdot 10^{12}$ bytes on the heap, nine times the raw data. At the shallowest depth satisfying (G) with count limit 10^5, about $D = 19$, there are about $5.2 \cdot 10^5$ records; the label fits $\ell_{19} = 3$ bytes, so about 5.8 MB in two's complement or 8.4 MB with one-word labels: five orders of magnitude less, and within a driver's memory. The record layer alone locates the time and the crashes of the earlier runs; §13.2 sizes them.

13.2 Layer 2: the machine

Definition 13.3 (The machine)**.** *A cluster has n_{w} workers, each running one executor of c task slots, so $m = n_{\text{w}}c$ tasks run at once; an executor heap H_{e} and an off-heap overhead O, whose container holds $H_{\text{e}} + O$; a driver of heap H_d; local disk for spills and shuffle files; and a network. Spark divides H_{e} as follows [A]: 300 MiB reserved; of the rest a fraction* 0.6 *is the* unified region *$M_u = 0.6(H_{\text{e}} - 300\ \text{MiB})$, shared by execution (shuffles, sorts, aggregations) and storage (cached data), of which storage keeps an eviction-proof half; the remaining* $0.4(H_{\text{e}} - 300\ \text{MiB})$ *is* user memory*, for objects the user's code builds and for Spark's own metadata. Execution memory spills*

to disk when full; user memory does not. The default overhead is $O = \max(0.1H_{\mathrm{e}}, 384\ \mathrm{MiB})$*, documented for the YARN and Kubernetes cluster managers.*

The partition as the unit of memory. A stage of p partitions runs in $\lceil p/m \rceil$ waves [D], c tasks sharing one executor at a time. What a task holds inside Spark's accounting (a shuffle's sort buffer, a combine table) can spill; what it builds itself (a materialised partition, the walk's buffer) cannot. So two budgets per task:

$$b_{\mathrm{exec}} \approx \frac{M_u}{c}, \qquad b_{\mathrm{user}} \approx \frac{0.4\,(H_{\mathrm{e}} - 300\ \mathrm{MiB})}{c}.$$

The division by c is an [A]: the JVM does not partition user memory by task, and objects outside Spark's accounting can occupy whatever the unified region is not using at that moment. Measurement 8 is what settles it.

Proposition 13.4 (Sizing the partitions)**.** *Let a stage read N records of s bytes, with a working-space factor $w \geq 1$ (the bytes a task holds per input byte), and let the partitioner be β-balanced: no partition holds more than $\beta N/p$ records. If*

$$p \;\geq\; \left\lceil \frac{\beta\, N\, s\, w}{b} \right\rceil,$$

then every task's working set is at most b bytes. With $b = b_{\mathrm{user}}$ this bounds what a task materialises; with $b = b_{\mathrm{exec}}$ it bounds what it sorts without spilling. Exact range partitioning puts at most $N/p + 1$ records in a partition, one more than $\beta = 1$ allows: with $\beta = 1$ in the condition, every working set is at most $b + s\,w$, one record over b. Subtree partitioning into p partitions with an ε-accurate sample has $\beta = (1+\varepsilon)^2(2 - \frac{1}{p}) + \varepsilon$ (Proposition B.9, §10.3), which rests on Conjecture B.10.

Proof. A task's working set is at most $(\beta N/p)\, s\, w \leq b$; under exact ranks it is at most $(N/p + 1)\, s\, w \leq b + s\, w$. Five records of 16 bytes with $w = 1$, $b = 40$ and $p = 2 = \lceil 80/40 \rceil$ show the extra record is reached: exact ranks put three records, 48 bytes, in one partition. The balance factors are those of §10.2, Phase 3, and of §10.3, step 3. □

Marks. Arithmetic; this is the sizing rule of §10.2, Phase 3, with its symbols named. [LEAN: the form in records is built, `IasCert.L1MDE.MPCSchedule par_le_of_points` (a task of at most Pts records costs at most $a\,\mathrm{Pts} + b$, for any schedule; Chapter 15); the inequality in bytes, Sizing prop_sizing and sizing_le (with β-balance a hypothesis, and with a slack of e records, $b + esw$), prop_sizing_range (range partitioning: exact ranks, at most $N/p + 1$ records a partition, give $b + sw$, since `RangeSafe` states no β), prop_sizing_subtree (the subtree β through Proposition B.9, so conditional on Conjecture B.10) — built at `b6ea25813`, machine-checked-structural for the module. Finding: "$\beta = 1$ plus one record" for exact range partitioning does not give "at most b": range_one_record_over (5 records of 16 bytes, $w = 1$, $b = 40$, $p = 2$; one partition gets 3 records, 48 bytes); the bound is $b + sw$, or $\beta = 1 + p/N$.] [SIM: the largest partition against $\beta N/p$ under both partitioners, on the runs of §13.5.]

Spark's advice is two to three tasks per slot [A], so that the slow tasks of one wave overlap the fast ones of the next; and, for a task whose working set does not fit, to raise p so that each task's input is smaller. The earlier runs did exactly the second, by powers of two, without a byte model; Proposition 13.4 replaces the trial.

The two tiny clusters (§13.5 runs there first). Every run uses one fixed envelope [A], so that runs differ only in data, code and network: a driver of 1 GiB heap (1.5 GiB requested, 4 GiB of local disk) and two executors per profile, each of 1 GiB heap (about 1.4 GiB requested), one task slot and 8 GiB of local disk for spills and shuffle files; the nodes are 4 GB and 8 GB

virtual machines in Helsinki and Nuremberg, and a third profile puts the driver in one region and the executors in the other, to measure the network itself. A namespace quota of 4 CPUs, 8 GiB and 12 pods holds one such run at a time. Two Spark lines exist: the reproduction line, Spark 3.1.3 with Scala 2.12 and JDK 11, the library's own line at `c0a1837` one patch up, and the release line, Spark 4.3.0 with Scala 2.13 and Java 17. In this envelope, by Definition 13.3, $M_u = 0.6 \times 724$ MiB ≈ 434 MiB and user memory is about 290 MiB per executor, with $c = 1$ [D]. The provisional data windows of the cluster's own note, one per dimension, where the data exceed one executor's share but fit the profile (10^6 to $5 \cdot 10^6$ points at $d = 2$, $3 \cdot 10^5$ to $3 \cdot 10^6$ at $d = 10$), are [A] until the calibration run per dimension fixes them.

Remark 13.5 (The 16384-partition bracket)**.** The merge of both two-dimensional terabyte runs used 16384 partitions, "the first power of 2 that didn't crash at this stage", the crashes being "most likely due to out of memory errors in the workers" [San23, §4.4]. The code at `c0a1837` materialises each partition before the walk (`leavesIter.toVector`, `MergeEstimator.scala` l.236) and the first round of the search materialises each validation partition twice over (`toArray` then `sortBy`, `MinimumDistanceEstimation.scala` l.590), both in user memory. Under six assumptions, the model turns these facts into a bracket on the executor heap of those runs, which the runs did not record: - *(a)* the crash was on the uniform data at $D = 60$, with $N_D \approx n' = 4.375 \cdot 10^{10}$ records; the cross run was at $D = 44$ and used 16384 partitions in every training stage and 4096 in the validation stage; - *(b)* a record costs about 148 bytes materialised once (the heap record and one reference) and about 152 bytes in the round-1 double materialisation [A]; - *(c)* the non-spillable budget is user memory, $0.4(H_\mathrm{e} - 300$ MiB), shared by $c = 16$ tasks peaking together; - *(d)* the round-1 materialisation at $p_V = 4096$, which ran, faced the same budget as the merge; the merge additionally held the shuffle-sort's buffer while draining it; - *(e)* the overhead default $\max(0.1H_\mathrm{e}, 384$ MiB) applied on the runs' Databricks cluster manager, which is not recorded, and the operating system took about 1 GiB of the 30 GiB machine; - *(f)* "crash" means an executor out of memory, and 8192 partitions was tried and crashed, which the thesis does not say: it is an inference from "first power of 2".

Then [D]: the merge held $4.375 \cdot 10^{10}/16384 \approx 2.67 \cdot 10^6$ records per task, about 0.39 GB, about 6.3 GB per executor, and ran; at 8192 it would have held about 0.79 GB per task, 12.6 GB per executor, and crashed; the round-1 materialisation held $1.875 \cdot 10^{10}/4096 \approx 4.58 \cdot 10^6$ records per task, about 0.70 GB, 11.1 GB per executor, and ran. So the effective non-spillable budget per executor lay between about 11.1 and 12.6 GB, and under (c) $26.2 \lesssim H_\mathrm{e} < 29.7$ GiB; under (e), $1.1H_\mathrm{e} \lesssim 29$ GiB gives $H_\mathrm{e} \lesssim 26.4$ GiB. The bracket is H_e between about 26.2 and 26.4 GiB, at the very top of what the machine allows. Dropping (f), with only 4096 known to have crashed, the upper bound on the budget rises to about 25.3 GB and the crash puts no upper limit on H_e within the machine; counting the round-1 records at 148 bytes moves the lower end to about 25.5 GiB. Either the runs used a heap close to the largest the machine allows, or assumption (c), the 40% budget, is wrong, which the model itself reads as the likelier. It is a bracket to be measured, not a fact: the executor heap of the runs would settle it at once, and measurement 8 settles it on any cluster.

Marks. The arithmetic is checked in the research record (review R3 of layer 3, items 1–3 and 9). None of (a)–(f) is established. [LEAN: none; the bracket is an inference under unverified assumptions.] [SIM: measurement 8.]

The driver. It holds the partitioner's sample, about $3 \cdot 10^6$ labels (§10.3); the collected L' records, at least $n'/\overline{\#}$ and at most $\min(2^\delta, N_D) + \Phi$ of them (Lemma 13.9; about $4.4 \cdot 10^5$ at the lower bound for the uniform terabyte run, about 63 MB on the heap [D]); each round's refinement, $M_r \le \Phi$ rows; and, under Theorem 11.12(iv), the leaf counts κ^*, at most Φ entries. The count limit is therefore also a driver parameter: $L'\, s_\mathrm{heap} \le H_d$ needs $\overline{\#} \ge n'\, s_\mathrm{heap}/H_d$ [D],

and the measured L' (§13.5) says whether the choice suffices. At these sizes a driver of tens of GiB is not the constraint; the task's user memory is.

13.3 Layer 3: the operators

The operators costed here are the rows of Table 6.1: a map row costs a pass over its partition, a reduction row a shuffle, the `collect` row the driver's memory and a network transfer, and the sort row a shuffle with a range partitioner. A row's law is what makes the result partition-invariant; this layer adds what it costs. Three facts about Spark enter every stage [A] [The]: a transformed dataset is recomputed by every action that reads it unless it is persisted; the map-side and reduce-side tables of a key-wise reduction spill when they do not fit; and a shuffle's files are kept until the datasets that use them are garbage-collected. Serialisation is part of every shuffle, collect and broadcast; the earlier runs used Java serialisation for the pipeline's classes, and Spark's built-in Kryo covers only simple types in shuffles [A].

Proposition 13.6 (Cost of the operators)**.** *In the model of Definition 6.1, with s_{ser} the serialised bytes of a record (measurement 2) and ν the records per task: - (i) Labelling (a map). $\Theta(D)$ halvings per point after copying the $2d$ bounds of the root box, no shuffle, streaming, no materialisation. With the arbitrary-precision label, one integer allocation per point [A]; with fixed-width labels, none. - (ii) Counting (a key-wise reduction). Each map task combines its points per label in a table that spills; the shuffle carries at most one record per map task and occupied cell, so at most $\min(n', p_{\mathrm{in}} N_D)$ records of s_{ser} bytes, and the reduce side adds. The output has as many partitions as the largest parent unless a number is given [A, [The]]. The reduction is a transformation, forced by the first action that reads it. - (iii) Partitioning and sorting (one shuffle). Every record is sent and sorted within its partition: $N_D\, s_{\mathrm{ser}}$ bytes shuffled; about $\nu \log_2 \nu$ comparisons per task, each $O(\lambda)$; the sort's buffer is execution memory and spills [A, measurement 9]. The lookup costs $O(\lambda \log_2 p)$ per record for exact range partitioning, and at most $\lfloor \log_2 S \rfloor + 1$ probes for S subtrees (Theorem B.8). - (iv) The local merge, streaming. On a partition of ν atoms with counts at least 1 (Phase 2 emits only occupied cells), the walk of Lemma 11.4(iii) makes at most $\nu \min(D, \overline{\#} + 1)$ additions and costs $O(\lambda\nu \min(D, \overline{\#} + 1))$ word operations (each record a look-ahead reads costs one $O(\lambda)$ containment test, and a look-ahead reads at most one record more than it adds), and holds at most $\overline{\#} + 1$ records in its buffer, so $O(s_{\mathrm{heap}}(\overline{\#} + 1))$ bytes of user memory, whatever ν is. A record of count 0 voids the buffer bound, not the output. - (v) The driver walk. The driver collects L' records, $L'\, s_{\mathrm{ser}}$ bytes, sorts them in $O(\lambda L' \log L')$, and always runs $W(0)$ (Lemma 11.4(ii)), in $O(\lambda L' + L' \min(D, \overline{\#} + 1))$. - (vi) The counts of a round. Each partition returns its count matrix of $h_r(h_r - 1)$ integers, combined by `treeAggregate` in a tree of chosen depth (Spark's default is 2), each level combining matrices of $h_r(h_r - 1)$ words: $p\, h_r(h_r - 1)$ words sent, no accumulator. With accumulators as at `c0a1837`, each task sends $h_r(h_r - 1)$ named updates, more bytes than the matrix (measurement 6). - (vii) The scan of a round. The refinement, M_r rows, is broadcast once to every executor; partition j with V_j records costs $O(\lambda(M_r + V_j) + V_j h_r^2)$ (Lemma 11.11): at most three truncations per record besides one per pointer move, and $h_r(h_r - 1)/2$ passes of the comparison loop (each a three-way comparison of f_θ and f_ϑ) for a record inside a row, none for a record outside every row. - (viii) The driver's work per round. The backtrack and the collation, $O(\lambda \sum_\theta |\mathrm{supp} f_\theta| + h_r(\lambda + h_r)M_r)$ (§6.10); the Scheffé integrals, at most $h_r^2(h_r - 1)/2$ multiply-adds per row, 2754 at $h_r = 18$ [D], plus $O(d)$ per row for a volume; the maximum over h_r^3 triples. - (ix) Scoring without a scan. Under Theorem 11.12(iv), after one distributed pass that returns the leaf counts κ^* (at most Φ entries, one key-wise reduction), every later round's counts $N_{\theta,\vartheta}$ are sums on the driver over the leaves of s^*, $O(\Phi h_r^2)$, with no broadcast, no scan and no count matrices. The round's rows may be the leaves of s^* themselves, on which every state of P is constant (Lemma 11.10(iii)): then the collation is one pointer pass per state, $O(\lambda h_r \Phi)$, and the integrals and the maximum cost as*

in (viii) with $M_r = |\mathcal{L}(s^*)| \le \Phi$, $O(\Phi h_r^3)$ as coded. The driver's work per round is therefore not reduced; the distributed pass, the broadcast and the matrices are removed.

Proof. (i)–(iii) and (v)–(viii) restate, in the model's units, the costs read from the code at `c0a1837` in the research record (layer 3), with (ii)'s record bound from §10.2, Phase 2, and the lookup bounds from Theorem B.8. (ix): the counts are Theorem 11.12(iv), $N_{\theta,\vartheta} = \sum\{\kappa^*(\ell) : f_\theta(\ell) > f_\vartheta(\ell)\}$ over the leaves of s^*, $h_r(h_r - 1)$ sums of at most Φ terms; the score of Theorem 11.12(ii) needs the integrals $\int_{A_{\theta,\vartheta}} f_\tau$ as well, which (iv) does not give, so they are computed as in (viii) on the rows of the round, and the leaves of s^* serve as rows by Lemma 11.10(iii).

(iv) is new. In the streaming walk, each atom costs at most two containment tests (the drop test and the overlap test) and, if it is not dropped, one join, each $O(\lambda)$; each record a look-ahead reads costs one more containment test; a look-ahead reads at most one record past the ones it adds, and the extension that starts it adds the atom's own count, so the tests number at most the additions plus 2ν. The additions are made only when a decision reads records below $A = t \wedge a_i$; by the proof of Lemma 11.4(iii) such a decision reads at most $\overline{\#} + 1$ records, and they lie below A on the side of a_i, that is, in $I(A\mathsf{R})$ where $A = a_{i-1} \wedge a_i$ is a branching node of the atoms. Distinct decisions have distinct branching nodes, since in a list in left-right order exactly one adjacent pair crosses from $A\mathsf{L}$ to $A\mathsf{R}$ (the argument of Theorem B.3, §10.3). An index j is read by the decision at A only if $a_j \sqsubseteq A\mathsf{R}$, so at most once per strict ancestor of a_j among the branching nodes, at most $\min(D, \nu - 1)$ times, and at most $\overline{\#} + 1$ records are read per decision. Hence at most $\nu \min(D, \overline{\#} + 1)$ additions. The buffer bound is Lemma 11.4(iii), whose proof uses that every count is at least 1 (the partial sums increase strictly); with counts of 0 allowed, a run of zero-count cells below one decision is read whole. □

Marks. (iv) derived here; the rest from the code and Chapter 11. [LEAN: (iv) Stream prop_opCost_iv (at most $(\nu-1)D$ additions for atoms in left-right order of depth at most D, any counts, by a potential argument in place of the double count over branching nodes; at most $(\nu-1)(\overline{\#}+1)$ with counts ≥ 1; so at most $\nu \min(D, \overline{\#}+1)$), streamAdds_le_depth, streamAdds_le_cap, lem_merge_iii_buffer — built at `8afd4d5f2`; the containment tests, StreamCounts streamTests_le_adds (the drop test, the overlap test and the look-ahead's reads, at most the additions plus 2ν) and prop_opCost_iv_tests (at most $\nu(\min(D, \overline{\#} + 1) + 2)$ tests), with lookAhead_ends (a look-ahead whose run ends first makes one addition per record of the run) — built at `15a5577ef`; the $O(\lambda)$ word cost of a join or a containment test and the bytes are not modelled; (vii) PathRefine prop_opCost_vii, moves_le, truncTotal_le, loop_mul_two (the pointer moves at most M times, at most $M + 3V$ truncations, and $h(h-1)/2$ comparisons per record inside a row and none outside, a comparison being one pass of the double loop, which evaluates $>$ and, when it fails, $<$) — built at `133fe3210`, on `Scan`; the $O(\lambda)$ word cost of a truncation and the broadcast are not modelled; (vi) and (ix) are counts of words from the matrix dimensions.] [SIM: measurements 6, 8, 9 and 10; the buffer's high-water mark against $\overline{\#} + 1$.]

What the code at `c0a1837` runs, and what the final algorithm removes. As the notebooks of the earlier runs ran it, the training data were generated twice (for the hull and for the labelling), written to and read from a checkpoint, read again for the sketch and again for the partitioning shuffle, then merged: seven passes over the training data (two generations, a write, four reads, the merge's among them), two of them full shuffles of about N_D records; the validation data were generated twice and labelled once, with one shuffle of the round-1 records. Each pass that reads an unpersisted generator runs it again [A]. The final algorithm (Chapter 10; the first table of §10.5) generates once and persists; labels at the shallowest depth satisfying (G), so that the key-wise reduction of (ii) reduces, which shrinks the second shuffle, the sketch, the merge and the collected records by the factor n'/N_D, about 10^5 for the two-dimensional uniform run [D]; streams the merge, (iv), so that the sort before it can spill; always runs the driver walk,

(v); returns count matrices, (vi); and may score every round after the first on the driver, (ix), which removes the per-round broadcast, pass and matrices and leaves the driver's work as it is. Whether (ix) is adopted is a question for §10.2 (the end matter).

13.4 Layer 4: the pipeline

Definition 13.7 (The time of a stage)**.** *A stage of p tasks on m slots, reading N records with $\nu = N/p$ per task, takes*

$$T_{\text{stage}} \approx \left\lceil \frac{p}{m} \right\rceil \cdot \max_{\text{tasks}} \left(\nu\, c_{\text{op}} + \nu\, s \left(\frac{1}{\beta_{\text{disk}}} + \frac{1}{\beta_{\text{net}}} \right) + \text{spill}(\nu) + \text{gc}(\nu) \right) + T_{\text{driver}},$$

where c_{op} is the stage's processor time per record, s the bytes per record it writes and reads (zero without a shuffle or a write), β_{disk} and β_{net} the per-task bandwidths, $\text{spill}(\nu)$ *is zero until $\nu\, s_{\text{heap}}$ exceeds b_{exec} and then about $2\nu\, s_{\text{ser}}/\beta_{\text{disk}}$ (the 2 for the write and the reread; the trigger is in heap bytes, the disk traffic in serialised bytes, as measurement 9 predicts them),* $\text{gc}(\nu)$ *is the garbage collector's time, which grows faster than ν once the live objects of c tasks approach the heap, and T_{driver} is the serial work at the driver. The maximum over tasks carries the skew, β of Proposition 13.4. Every constant is to be fitted (§13.5).*

The earlier cost accounting of this work assumed at most one partition per machine in every stage; no run of the earlier series met that hypothesis, at 4096 to 16384 partitions on 16 machines. Definition 13.7 replaces it by the two numbers a run actually has, p and m, through the waves.

Proposition 13.8 (The pipeline, composed)**.** *For the final algorithm of Chapter 10, the stages have the following inputs, per-record costs, bytes written, non-spillable peaks and driver work, in the symbols of Definition 13.7 and Proposition 13.6:*

Stage	N	c_{op}	s	Peak per task, non-spillable	T_{driver}
hull	n' points	$c_{\text{read}} + O(d)$	0	none	$p \cdot 2d$ doubles
labels and counts	n' points, then N_D records	$c_{\text{read}} + c_{\text{lab}} D + c_{\text{comb}}$	s_{ser} (shuffle) + storage	none (the table spills)	the depth, from coarse counts (Lemma 11.18)
partition and merge	N_D records	$c_{\text{read}} + c_{\text{part}} \log p + c_{\text{cmp}} \log \nu + c_{\text{walk}}$	s_{ser} (shuffle)	$s_{\text{heap}}(\overline{\#} + 1)$, the buffer	collect, sort and walk L' records
validation labels and round 1	φn points, N_V records	$c_{\text{read}} + c_{\text{lab}} D + c_{\text{cmp}} \log \nu_V + c_{\text{trunc}} + c_{\text{sch}} h_1^2/2$	small (matrices, or κ^*)	none (the sorts spill)	the path, the collation, integrals $O(M_1 h_1^3)$, broadcast $M_1(8 + 8h_1)$ bytes to every executor
rounds $r \geq 2$, scanned	N_V persisted records	$c_{\text{trunc}} + c_{\text{sch}} h_r^2/2$	0	none	as round 1, with M_r, h_r
rounds $r \geq 2$, from κ^*	none	—	0	none	counts $O(\Phi h_r^2)$, collation $O(\lambda h_r \Phi)$, integrals $O(\Phi h_r^3)$

The bytes communicated are: the two shuffles, $\min(n', p_{\text{in}} N_D)$ *and N_D records of s_{ser} bytes; the collect, $L' \leq \min(2^\delta, N_D) + \Phi$ records (Lemma 13.9); per scanned round, one broadcast of $M_r(8 + 8h_r)$ bytes (the primitive layout, Proposition 13.2(v)) to each of n_{w} executors and $p\, h_r(h_r - 1)$ words of matrices; and, from κ^*, nothing. The memory of a run is, with the partitions sized by Proposition 13.4 under its β (so the merge's sort input is at most $\beta N_D/p$ records per task): per task, non-spillable, $s_{\text{heap}}(\overline{\#} + 1)$ plus constants, for records of count at least 1; per task, spillable, the combine table and the sort buffers, within b_{exec}; on the driver, the sample, the L' records, the refinement and κ^*.*

Proof. Each row is the sum of the operators of Proposition 13.6 that the stage runs, in the order of §10.2. The bound on L' is Lemma 13.9. □

The depth choice and the fallback (what Chapter 12 asks of this section). The "labels and counts" stage finds the depth D^{G} in at most two passes (Lemma 11.18, Chapter 12): pass 1 labels and counts at D_0; pass 2 labels the N_{over} points of the over-full depth-D_0 cells at $D_{\max}$, sorts them, and one sweep of the sorted labels with a stack of open ancestors gives the largest count at every depth in $(D_0, D_{\max}]$ at once, $O(\lambda N_{\mathrm{over}})$. Pass 2 is one more row of the table with $N = N_{\mathrm{over}}$, $c_{\mathrm{op}} = c_{\mathrm{lab}} D_{\max} + c_{\mathrm{cmp}} \log \nu$ and $s = s_{\mathrm{ser}}$, and nothing on the driver but the depth. When $D^{\mathrm{G}} > D_0$ a third pass labels every point at D^{G}: a map at $c_{\mathrm{lab}} D^{\mathrm{G}}$ per point over the persisted points, no shuffle, before the key-wise reduction of Proposition 13.6(ii). The alternative, labelling every point at $D_{\max}$ in pass 2 and truncating, saves that map at the price of sorting all n' labels, a shuffle of n' records; the model takes the third pass. A deeper start D_0 grows N_D and with it the records of every later stage, by Proposition 13.6(ii)–(v), and not the estimate (Chapter 12). Without (G) the pipeline runs at $D_{\max}$ with the over-full cells as atoms (Lemma 11.7): no stage's form changes, and N_D is then the number of occupied depth-$D_{\max}$ cells, so the fallback costs exactly what those records cost. The two-word label of $63 \le D \le 126$ costs one word per record in every shuffle, sort, collect and broadcast (Proposition 13.2(ii)) and nothing else.

Lemma 13.9 (Sizes)**.** *Assume (G) and a depth limit δ safe for the partitioner (Corollary 11.6). Then the records collected after the local merges satisfy $L' \le \min(2^\delta, N_D) + \Phi$, and every round's refinement has $M_r \le \Phi$ rows.*

Proof. The local walk $W(\delta)$ never raises an output above depth δ (Definition 11.1, Lemma 11.4(ii)), so a local output w either has depth exactly δ, and distinct outputs are disjoint (Lemma 11.2(c) within a partition, the antichain of Lemma 11.4(ii) across partitions), so there are at most $\min(2^\delta, N_D)$ such; or $|w| > \delta$, and then its parent contains $e \wedge e'$ for two emitted subtrees of its partition with $|e \wedge e'| \ge \delta$ (Lemma 11.2(c)), which holds more than $\overline{\#}$ points locally (Lemma 11.2(b)), hence globally under (Safe-1). As $\#(w) \le \overline{\#}$ under (G), w is a leaf of s^* (Lemma 8.3(ii)), and there are at most Φ of those. $M_r \le \Phi$ is Proposition 13.2(v). □

Marks. Kept from the 2026-11-04 preprint, restated on Chapter 11's walk. [LEAN: Sizing lem_sizes, lem_sizes_count, heavy_parent_of_raised, mass_localCounts_le — built at `b6ea25813`, on `MergeWalk` and `ThresholdPaving`; machine-checked-structural: Lemma 11.2(b) for $W(\delta)$ is a hypothesis (`MergeWalk` proves the heaviness for $W(0)$ only, `walk_spec`).] [SIM: L' against the bound under range and subtree partitioning at several δ.]

Remark 13.10 (The terabyte stage table as four hypotheses)**.** The stage times of the earlier two-dimensional runs [San23, SGWS23b], read through Proposition 13.8 with the code's choices of §10.5, give four hypotheses. Each is a testable claim with the measurement that decides it; none is a finding. 1. *Generation.* The hull stage only generates the points and reduces by the hull. It took 0:06:47 and 0:07:13 for $6.25 \cdot 10^{10}$ points on 256 slots: about 1.4–$1.5 \cdot 10^8$ points per second over the cluster, $6 \cdot 10^5$ per slot [D]. The training points, $n' = 4.375 \cdot 10^{10}$ of them, are generated again inside the labelling stage; at that rate this takes at most $n'/n = 0.7$ of the hull stage (at most, since the hull stage also reduces), 0:04:44 and 0:05:03, which is 2% of the labelling stage's 4:08:00 (uniform) and 5% of its 1:35:00 (cross); with the whole hull time instead, 0:06:47 and 0:07:13, it is 2.7% and 7.6%. *Claim:* generation does not explain the labelling stage. *Decided by:* measurement 10, the generator alone. 2. *Spill and garbage collection.* The labelling stage took 4:08:00 for the uniform run ($p = 4096$, $D = 60$) and 1:35:00 for the cross run ($p = 16384$, $D = 44$), on the same n'. Labelling is linear in D, a factor of about 1.4, not 2.6. The records per task differ by 4: $1.07 \cdot 10^7$ against $2.67 \cdot 10^6$, about 1.5 GB against 0.4 GB on the heap [D], where $b_{\mathrm{exec}} \approx 0.6(H_{\mathrm{e}} - 300\ \mathrm{MiB})/16$ is about 1 GB for a heap near 26 GiB [D]. *Claim:* the uniform run's combine table spilled and its collector worked harder; costs superlinear in ν explain the gap and linear ones do not. *Decided by:* measurement 9 at the two partition counts

on one data set, with the collector's time from the event log. 3. *The second shuffle.* The merge stage took 2:53:00 and 2:22:00 at 16384 partitions, of the same order as the labelling stage at the larger partition count; the count-limit scan between them took 0:03:25 and 0:03:01. *Claim:* both are dominated by one shuffle of about N_D records of s_{ser} bytes. *Decided by:* measurement 2, which gives s_{ser}, and the stage's shuffle bytes from the event log, which give β_{net}. 4. *The driver's search.* The validation and search stages took 0:16:05 (uniform) and 0:34:30 (cross). The uniform estimate has 21 leaves and the cross estimate 596 603, so the collations differ as much. *Claim:* the difference is the driver's serial work per round, $O(M_r h_r^3)$ for the integrals plus the backtrack and collation. *Decided by:* the driver's time per round logged against M_r.

The times are the thesis's Table breakdown_1TB as the research record transcribes it, in h:m:s, with its five stages (the hull; the training labels; the count-limit scan; the merge; the validation labels with the search). Its five uniform-2D entries sum to 7:27:17 against the 7 h 11 min of its Table 4.2 (§14.2), a discrepancy of exactly the last stage that is inside the source and unresolved.

For the final algorithm at the shallowest depth, the model predicts [D] that the second shuffle, the sketch, the merge and the collect shrink by about n'/N_D, that the combine table no longer spills, and that the run is dominated by reading and labelling the points, about two passes at the rate of hypothesis 1, plus the driver's search. That prediction is what §14.1 tests first.

Marks. The times, in h:m:s, and the leaf counts are [A], read from the thesis's tables and the notebooks as the research record transcribes them; the rates are [D]. [LEAN: none.] [SIM: measurements 2, 9 and 10, and the driver's per-round log.]

Proposition 13.11 (The cost of the search)**.** *The gap search of §10.4 scores at most $|U| \le |P| \le \Phi$ states (Lemma 11.14(b)(i)), and no bound on $|U|$ in terms of the sample alone is smaller in general: through ξ alone a gap settles only when its arc length is at most $4\xi/(1+\xi)$, and first-round gaps can have arc length near $D/(k-1)$. In practice two things bound it: $\epsilon_0 > \sigma$, where σ is the largest step of P, gives at most $r^* + 1$ rounds after the first and $|U| < k + (D + (k-1)\sigma)/(\epsilon_0 - \sigma)$ (Lemma 11.14(b)(iii)); and the error profile along P, for which a bound is open (§16.9). The driver's work is $\sum_r O(M_r h_r^3 + \lambda h_r M_r + d M_r)$ over the rounds, the last term the volumes, and the final selection over all scored states costs $O(|U|^3 M)$ as the integrals are coded at* `c0a1837`*, M the rows of the final collation; each scanned round adds one pass over N_V records.*

Proof. Lemma 11.14(b) and the integral loop of Proposition 13.6(viii). □

Marks. Stated as §10.4 states it (the review of that section, M9); nothing added. [LEAN: `LemRounds` covers (a) of Lemma 11.14; GapRounds covers (b): `Run.card_le` ($|U| \le |P|$), `Run.lem_rounds_b_iii` (at most $r^* + 1$ rounds after the first, the count of $|U|$ non-strictly), `Run.lem_rounds_b_iii_strict` (strictly when $\sigma > 0$), `Run.lem_rounds_b_iii_strict_D` (the strict D-form as stated here, without $\sigma > 0$, built at `050df0219`), `xi_settle` (with scores at most 1, a gap settles through ξ alone only at arc length at most $4\xi/(1+\xi)$) — built at `166f3514c`; that first-round gaps can have arc length near $D/(k-1)$, the driver's work and the $O(|U|^3 M)$ of the final selection are not checked.] [SIM: the gap search's $|U|$ and rounds on the calibration runs, against the bounds.]

Proposition 13.12 (Memory of the merge)**.** *Let the bytes per record be as measured (s_{heap}, s_{ser}: measurements 1 and 2), let the sort before the walk spill (measurement 9), and let the walk stream over records of count at least 1 (Lemma 11.4(iii); Phase 2 emits only occupied cells). Then no task of the merge materialises more than $s_{\text{heap}}(\overline{\#} + 1)$ bytes plus constants, whatever p; and with p as in Proposition 13.4 and $b = b_{\text{exec}}$, no task's sort exceeds its execution budget: exactly, for range partitioning with exact ranks, up to one record ($b_{\text{exec}} + s\,w$, Proposition 13.4); under Conjecture B.10, for subtree partitioning (β of Proposition B.9). This is the "no out-of-memory*

failure in the merge" of §10.5, and it is all that is claimed: the driver's memory (§13.2) and the collector's headroom are outside it.

Proof. Lemma 11.4(iii) bounds the buffer; Proposition 13.4 bounds the sort's input under β-balance; the two balance factors are those of §10.2 and §10.3. □

Marks. The review of Chapter 10 (M10) asked that this be said exactly so. [LEAN: the buffer bound and the sizing inequality, as above; nothing about Spark's spilling, which is a hypothesis.] [SIM: measurement 8 at two partition counts, with the buffer's high-water mark.]

13.5 Layer 5: calibration

The layers above are a model: [A] constants and [D] laws. This layer is the protocol that turns the [A]s into [M]s and tests the laws, in the order the author ruled on 2026-11-10: everything is checked at sizes the two tiny clusters handle, with the model's certified arithmetic, before any run at scale; the terabyte problems are then the last test, on Scaleway, within a budget of about 500 EUR, stopping when it is spent, and nothing at scale starts without the author's go (§14.1).

Definition 13.13 (The calibration protocol)**.** *A* calibration step *is a quadruple: a predicted quantity with the law that predicts it, one of the propositions above; a run, named, that measures it; a tolerance, fixed before the run; and the layer the prediction belongs to. A step* passes *when the measured value lies within the tolerance of the prediction, and* fails *otherwise. A failure falsifies its layer, which is corrected, and the correction re-derives every prediction that depends on it, before the next run. For a law, the tolerance is zero: a count of records, bytes in a fixed-width layout, or a buffer's high-water mark must match exactly. For an [A] constant, the tolerance is a factor, stated with the step. The steps run in order: the small runs on one executor, then the multi-worker runs in the data windows of §13.2, then the terabyte runs. No quantity is reported as measured until its step has run, and then with its run.*

The ten measurements. Each turns an [A] of a layer into an [M], or tests a law. The runs are through the cluster's door, on the reproduction line at `c0a1837` with the law driver added as an overlay to the pinned sources, unless the step says otherwise; a step whose law needs several workers runs with two executors. The tolerances are choices of the protocol, not measured values.

#	Quantity	Predicted by	Run	Tolerance	Layer
1	heap bytes per record, arbitrary-precision and fixed-width label, at $D = 19$, 44, 60 and 63 (the heap layout's independence of D up to 63) and at $D = 64$ and 70 (the magnitude array's third int; the second label word from 63)	Proposition 13.2(ii): about 144 for $D \le 63$, and 16 (24 as a row) for $D \le 62$; about 152 for $D \ge 64$, and 24 (32 as a row) for $D \ge 63$	one executor, a heap histogram of a materialised partition	10% for the heap layout; exact for the primitive and row widths	1
2	shuffle bytes per record, Java and Kryo serialisation	§13.1: Java several times 16, unknown factor; Kryo about 18	one executor, the shuffle write of the counting stage from the event log, divided by its records	Kryo within a factor 2; Java is fitted, not predicted	1
3	storage bytes per record, binary and fixed-width label column, unsorted and sorted	§13.1: about 12 plus the count, before compression	one executor, file sizes	a factor 2	1
4	heap bytes per point as a dense vector; per refinement row, as the final algorithm broadcasts it and in the heap form of `c0a1837`	Proposition 13.2(i), (v): $8d + 32$; $8 + 8h_r$ as broadcast, $136 + 8h_r$ on the heap	one executor, a heap histogram; the broadcast's size from the event log	10% for the heap figures; exact for the primitive row	1

#	Quantity	Predicted by	Run	Tolerance	Layer
5	the depth at which a midpoint first meets an endpoint for a given root box; a candidate cause of the earlier runs' "negative volume"	§13.1: about $52d$ on a near-isotropic box	one executor, the labelling alone at increasing depth	exact (a depth)	1
6	bytes per task of the counts sent to the driver, accumulators against matrices	Proposition 13.6(vi): $8h_r(h_r-1)$ for the matrix, 2448 at $h_r = 18$; accumulators more	two executors, the task results' sizes from the event log	exact for the matrix; the accumulator overhead is fitted	3
7	the serialised size of the subtree partitioner	S labels of ℓ_D bytes plus an S-entry map, Java-serialised	two executors, the shuffle dependency's size	a factor 2	1
8	peak heap per task in the merge and in round 1, at two partition counts, with execution and storage memory alongside	Proposition 13.12: the buffer $s_{\text{heap}}(\overline{\#}+1)$ with the streaming walk; $r\,s_{\text{heap}}$ with the code's `toVector`; the budgets of Definition 13.3	two executors, in the data window, a heap histogram per task	the buffer: exact in records; the budget split (user against execution): 20%	2, 3
9	bytes spilled, memory and disk, by the counting reduction and by the partitioning sort, at $D = 60$ and at the shallowest depth satisfying (G)	Definition 13.7: zero while $\nu\, s_{\text{heap}} \le b_{\text{exec}}$; above it, memory bytes spilled about $\nu\, s_{\text{heap}} - b_{\text{exec}}$ [D] and disk bytes about the same count of records at s_{ser} [A]	two executors, in the data window, the spill metrics of the event log (the in-memory size spilled and the bytes written)	zero against nonzero is exact; each volume within a factor 2	2, 4
10	the wall time of each operator of Proposition 13.6 on one executor, with the generator alone as the baseline	Definition 13.7 with $\lceil p/m \rceil = p$: the fitted c_{gen}, c_{lab}, c_{comb}, c_{cmp}, c_{trunc}, c_{sch}	one executor, the data window's midpoint, each stage timed separately	the constants are fitted here; their linearity in ν is tested at two sizes, within 20%	3, 4

Measurements 1–5 and 7 are small runs on one executor; 6, 8 and 9 need the multi-worker profiles; 10 fits the constants of Definition 13.7. After them, every constant of Proposition 13.8 is an [M] with its run.

The multi-worker runs in the window. One run per dimension at the window's midpoint, on each of the three profiles, so that the network's share is separated from the arithmetic (the third profile differs only by the network). Each predicts, before it starts: the records and bytes of each shuffle (Proposition 13.6(ii)–(iii)), L' (Lemma 13.9), the peak per task (Proposition 13.12), the number of rounds and $|U|$ (Proposition 13.11), and each stage's time (Definition 13.7 with the constants of measurement 10). The runs also carry the exactness laws of Chapter 11 (the merge and the counts across partitionings, bit for bit), which the law driver checks in the same run; a run that fails an exactness law is a defect of the code, not of the model, and stops the protocol until fixed.

The terabyte runs, last. With every constant an [M], Proposition 13.8 predicts each stage of each terabyte problem of §14.1: its partitions from Proposition 13.4 with the measured s_{heap} and the cluster's b_{exec} and b_{user}; its bytes shuffled; its time, hence its cost in the cluster's price. The predicted cost of each stage is set against the remaining budget before the stage starts, and the run stops at the budget. The terabyte runs confirm or falsify a calibrated model; they do not explore one. Each is also run at the earlier depths and count limits, so that the speed-up of each refinement of §10.5 is read off separately, with the generation timed apart.

The record of a run. The cluster's door refuses an overlay or script whose working copy differs from its commit, so every run is reproducible from a commit. For each run the protocol keeps, as the check of the release repository keeps it: the pinned source tree hash, the sha256 of each overlay, and the JDK and Spark line; and, as §14.1 lists: the cluster (machines, processor model, cores and whether virtual, executor heap and overhead, driver memory, Spark version, serialiser); the profile, the number of executors, n, d, the density and the seed; the time to generate the data, apart from the estimator's stages; each stage's input and output records and partitions; its

wall time; the bytes it shuffles, spills and collects; and its peak heap per task. The event log is the source of the bytes and the spills.

What has been measured so far. The library at `c0a1837` builds on the cluster's reproduction line and its own tests pass there: 122 tests in 5 suites, 0 failed, in 8 min 6 s on JDK 11 (run `sdt-test11-c0a1837`, 2026-11-08) [M]. The law driver's first runs, at correctness sizes, passed the merge exactness law under (G) with 4 partitions against 1 (141 leaves) and the counts law with 1, 4 and 8 input partitions [M; reported in the cluster's note of 2026-11-08, run names not recorded]. None of the ten measurements has run. No constant of this section is an [M].

Remark 13.14 (What a certificate can check)**.** The model has two kinds of statement. The first kind is arithmetic from stated hypotheses, and a machine-checked certificate (Chapter 15) can hold it: - the bytes per record in each layout as a function of D and h_r, given the layout table (Proposition 13.2(ii), (v), (vi)) [LEAN: Sizing prop_recordBytes_ii, prop_recordBytes_v, prop_recordBytes_vi, with the table a hypothesis — built at `b6ea25813`]; - the bounds on records: $N_V \le \varphi n$, $\lceil n'/\overline{\#} \rceil \le |\mathcal{L}(s^*)| \le \Phi$, $L' \le \min(2^\delta, N_D) + \Phi$, $M_r \le \Phi$ (Proposition 13.2(iv), Lemma 13.9) [LEAN: Sizing prop_recordBytes_iv, lem_sizes (Lemma 11.2(b) for $W(\delta)$ a hypothesis), prop_recordBytes_v_rows — built at `b6ea25813`, on `ThresholdPaving` and `MergeWalk`; $N_V \le \varphi n$ (Proposition 13.2(iii)) not built]; - the sizing inequality (Proposition 13.4) and the waves $\lceil p/m \rceil$ [LEAN: the waves and the balance-on-the-data form of the sizing rule are built, `IasCert.L1MDE.MPCSchedule` (`waves_unit_lower`, `waves_unit_upper`, `waves_bounded`, `par_le_of_points`; Chapter 15); the inequality in bytes, Sizing prop_sizing, prop_sizing_range, and the waves as $\lceil p/m \rceil$, waves_eq — built at `b6ea25813`]; - the buffer bound $\overline{\#} + 1$ of the streaming walk (Lemma 11.4(iii)) and its addition count (Proposition 13.6(iv)) [LEAN: Stream lem_merge_iii_buffer, prop_opCost_iv — built at `8afd4d5f2`; the look-ahead's reads against its additions, `lookReads_le_lookAdds`, for one look-ahead only; the walk's containment tests, StreamCounts streamTests_le_adds and prop_opCost_iv_tests (at most $\nu(\min(D, \overline{\#}+1)+2)$ — built at `15a5577ef`; the word operations of a join or a test are not counted]; - the scan's cost, pointer moves at most M_r and $h_r(h_r-1)/2$ comparisons per matched record (Lemma 11.11, Proposition 13.6(vii)) [LEAN: PathRefine prop_opCost_vii — built at `133fe3210`, on `Scan`]; - the candidates per round, $h_r \le 2k-2$ for the zoom (Lemma 11.14(a)), and $|U| \le \Phi$ for the gap search (Lemma 11.14(b)) [LEAN: (a) as Chapter 11 marks it; (b) GapRounds `Run.card_le`, built at `166f3514c`].

Of these cost counts only those marked built above are built (the waves and the balance-on-the-data sizing, the streaming walk's buffer bound and addition count, the gap search's $|U| \le \Phi$); the correctness halves of Chapter 11 are built where its marks say so. The second kind is a property of a JVM, of Spark or of a machine: a heap layout, a serialiser's bytes, whether a sort spills, a bandwidth, the collector's time, the 40% budget. No certificate reaches these. They are the [A]s, and the protocol above is the only check they have. The section's claim is therefore of the form: *if* the measured constants are these, *then*, by the certified arithmetic, the run has these sizes and this cost.

Marks. [LEAN: as listed: the waves and the balance-on-the-data sizing, the streaming walk's buffer bound and addition count, the gap search's $|U| \le \Phi$, the bytes per record, the bounds on records but N_V, the sizing inequality in bytes (Sizing, with the layout table, β-balance and Lemma 11.2(b) for $W(\delta)$ as hypotheses) and the scan's cost (PathRefine) are built; the rest to write.] [SIM: the protocol itself.]

Chapter 14

Scale: our own runs

This section reports the final algorithm of Chapter 10 run at the terabyte scale. Each stage's configuration, time, bytes and memory are recorded, and the measurements calibrate the cost model of Chapter 13. Until the runs exist, the section states their protocol and reports no numbers of its own.

14.1 The runs of the final algorithm

What is run. The final distributed algorithm of Chapter 10. Each of its refinements runs only with its exactness lemma of Chapter 11; a refinement without one is not run. The refinements are: - data generated once and persisted, with a partition-invariant split; - the root box from the training points; - the shallowest depth satisfying (G), or else the deepest representable depth, said so (Chapter 12); every run reports the fields of §10.2's report, so that Lemma 12.4 applies to its estimate; - fixed-width labels: one word ($D_{\max} = 62$) unless a ten-dimensional problem fails (G) at that depth on the small cluster, which is checked before any terabyte run; the two-word label ($D_{\max} = 126$) is then the option, costed in Chapter 13; - partitions sized to bytes per record, memory per task and balance; - range partitioning in snapped runs (Corollary 11.6(c)), or subtree partitioning; exact-rank cuts alone give a depth limit near D and merge almost nothing locally (the remark after Corollary 11.6's sharp example); - a merge that can spill, and a driver walk that always runs; - backtrack ties broken by heap code, under both backtrack rules of §10.2, Phase 4 (least count; least $|a - b|$), each run recording its rule and the selected leaf count m_n (§16.14); and the gap search, with validation counts combined once per partition.

The code version is recorded with every run.

The two backtrack rules can select different estimates from the same s^*, which is why every run records its rule. The first run of the whole pipeline on the refined code (E1c, $n = 10^7$ points of a two-dimensional Gaussian mixture; Theorem 10.4's [SIM] mark) already shows it. From one s^* of 3913 leaves, the gap search under least count chose a state with 3461 leaves, and under least $|a - b|$ one with 2693: on this smooth density the arc-length rule chose the coarser estimate. Which of the two is closer to f in L_1 is not yet measured; the final runs report both errors.

The problems. The same four as the earlier runs: the uniform density and the cross-shaped Beta line mixture of Gaussians, each in 2 and 10 dimensions, at about 10^{12} bytes. The larger dimensions follow once these have run.

The earlier timings include generating the data, on different machines, at fixed depths and count limits. So, to make the two comparable, each problem is also run at the earlier depths and count limits, alongside the runs at the shallowest depth satisfying (G). The speed-up of each

refinement can then be read off separately.

What each stage records: - the cluster: machines, CPU model, cores (and whether these are virtual), executor heap and overhead, driver memory, Spark version, serialiser; - the time to generate the data, separately from the estimator's stages; - the stage's input and output records and partitions; - its wall time; - the bytes it shuffles, spills and collects; - its peak heap per task.

These are the measurements of Chapter 13's layer 5. Each estimate of Chapter 13 is replaced by a measured constant, and the model's prediction for each stage is compared with its measured time.

The protocol. The steps run in this order, each depending on the one before. 1. Small simulations on one machine: the per-record measurements of Chapter 13, layer 5, and the simulation witnesses of the theorems of Chapters 8 to 11. 2. Runs at 10^9–10^{10} bytes, checking the model's predictions stage by stage. 3. The terabyte runs.

At each step the predicted cost is set against the budget before anything is started.

The order and the budget. The terabyte-sized problems above are the only reruns at scale. Before them, every stage is checked at sample sizes that the small Spark cluster on Hetzner handles, against the cost model of Chapter 13 with its machine-checked parts, so that the terabyte runs confirm a model rather than explore one. The terabyte runs are on Scaleway, with a budget of about 500 EUR, and stop when it is spent. Nothing at scale starts without the author's go.

14.2 Earlier runs

Graner (2022). The first runs of the bottom-up merge and the minimum distance search on Spark, with a merge built on Spark datasets. The densities were uniform densities on the unit cube in 1, 10, 100 and 1000 dimensions, and standard Gaussians in 1, 2 and 5. Each used 10^7 and 10^8 training points and half as many validation points. The clusters were four workers of 4 cores and nine workers of 2 cores, with 8 GB each. The reported timings include generating the data [Gra22, ch. Scalable histograms, Results].

Sandstedt (2023). Runs with subtree partitioning: - on up to 10^8 points in up to 1000 dimensions [San23, App. A]; - on a terabyte of simulated data, split 70/30 into training and validation [San23, §4.4, Table 4.2]: - $6.25 \cdot 10^{10}$ points in two dimensions, in 7 h 11 min (uniform) and 4 h 42 min (cross); - $1.25 \cdot 10^{10}$ points in ten dimensions, in 45 min (uniform) and 46 min (cross).

The terabyte runs used 16 workers of 16 cores each (AWS c4.4xlarge, 30 GB) [San23, §4.4]; the notebooks hold the rest of the environment [SGWS23b].

The merge of both two-dimensional terabyte runs used 16384 partitions, "the first power of 2 that didn't crash at this stage", and the crashes were "most likely due to out of memory errors in the workers" [San23, §4.4]. The ten-dimensional runs merged at 8192. As the thesis itself explains, the merge cannot spill [San23, §4.5]; Chapter 13 sizes it.

The public record of these runs is the thesis's Tables 4.2–4.5 and A.1–A.4 and the notebooks [SGWS23b] at `2cf1b6d`. The unpublished research record of this monograph also holds the as-run details: which code ran, the configuration, the stage tables and the partition counts.

14.3 What the earlier runs do not establish

The earlier runs did not record the executor heap or the bytes moved. Their exported notebooks are the code after the runs. The exactness conditions of Chapter 11 were found later, from the 2026-11-04 reading of the code at `c0a1837`, and no run was checked against them. These are the reasons for the runs of §14.1; the details are in the research record.

Chapter 15

Machine-checked certificates

A mark "[LEAN: built]" in this monograph names a Lean 4 module and theorems in it, and means the following. The module is in the `provedog` tree of the source repository, checked with Lean 4 v4.32.0 and Mathlib at its v4.32.0 tag, as the toolchain file and the Lake manifest pin them. It was *crunched*: a sealed job on the Kubernetes cluster (§13.2 describes it) cloned the named commit of the repository, built the module with `lake`, found no `sorry` in it, and ran `#print axioms` on every named theorem, which listed only `propext`, `Classical.choice` and `Quot.sound`. The job's name, commit and verdict are written into the certificate register, `provedog/decision-universe.json`, as the row's `verified` field, and the row's status flips from `partial` to `machine-checked` only then; a row whose statement keeps a hypothesis the Lean does not discharge is `machine-checked-structural` and says which. Every row below was crunched at commit `a5a2cee59` on 2026-11-10, except `MergeNoG`, at `279d904f6` on 2026-11-08; the commit at which each module first built green is in the aggregate root `IasCert.lean`, whose import list is what the build target checks. The register, the root and the modules are the record; the tables are a reading of them.

Three things a certificate here is not. It does not check the Scala: the modules state the *specification* of Chapter 10, and where a module models a routine of the code at `c0a1837` (the raising, `findSubtree`, the scan) it models the lines named, as the text argues they run. It does not model Spark: the execution hypotheses (X1)–(X7) of Theorem 11.16 and every cost of Chapter 13 that is a property of a machine are hypotheses, as Remark 13.14 says. And it does not model floating point: the scores of Theorem 11.12(ii) are certified in exact arithmetic, which Corollary 11.17 makes the arithmetic of the final algorithm.

The namespace. The modules of the decision-theory chain are `ProveDog.DecisionTheory.*`; the modules of the pipeline's specification and of the cost model are `IasCert.L1MDE.*`, and they move under `ProveDog.*` with the public release of the certificate library (Chapter 16); the marks follow the move once the moved modules build, never before.

15.1 The decision-theory chain

The oracle inequality and the consistency theorem rest on this chain, which is independent of the pipeline. The Vapnik–Chervonenkis inequality for partition families is not in Mathlib and is proved here.

Module (`ProveDog.DecisionTheory.`)	Certifies	Named theorems	Status
`ScheffeFinite`	the Scheffé identity on a finite set, the L_1 view of §9.3	`scheffe_argmax`, `scheffe_eq_half_L1`, `tv_eq_zero_iff_eq`	machine-checked
`MinDistance`	Theorem 9.1(i), the algebraic core: the factor-3 bound given its proof steps as hypotheses; the Scheffé step on a pair	`min_distance_oracle`, `yatracos_pair_dist`	machine-checked
`L1MDERate`	the rate of Theorem 9.1(ii) from a per-set tail bound B: the union bound over the Yatracos class turns B into $\|\mathcal{A}\|B$; the tail is a hypothesis (`hB`), which `VC.EmpiricalHoeffding` discharges	`error_exceed_subset_union`, `l1mde_rate`	machine-checked-structural
`VC.SignCube`	Hoeffding on the sign cube, step 4 of the VC inequality in the form of Devroye, Györfi and Lugosi (1996), Theorem 12.5	`sum_exp_signSum`, `prod_cosh_le`, `card_signSum_ge_le`, `card_exists_signSum_ge_le`	machine-checked
`VC.Trace`	traces, the shatter coefficient, Sauer–Shelah for traces	`card_traces_le_shatterCoeff`, `shatterCoeff_le_two_pow`, `vcDim_traces_le`, `card_traces_le_sum_choose`, `shatterCoeff_le_sum_choose`	machine-checked
`VC.GhostSwap`	the exchangeability of a sample and its ghost sample, step 2	`measurePreserving_swapAt`, `prod_preimage_fst`, `prod_preimage_snd`	machine-checked
`VC.EmpiricalHoeffding`	Hoeffding for the empirical count of one measurable set under an i.i.d. sample, step 1	`integral_comp_eval`, `hoeffding_pi`, `measure_abs_cnt_sub_ge_le`	machine-checked
`VC.Inequality`	the VC inequality: $\mathbb{P}\{\sup_A \lvert\mu_n(A)-\mu(A)\rvert > \varepsilon\} \le 8S\,e^{-n\varepsilon^2/32}$ for a countable family with shatter coefficient S, and with VC dimension	`cnt_swapAt_sub`, `vc_inequality`, `vc_inequality_shatterCoeff`, `vc_inequality_vcDim`	machine-checked
`VC.Boxes`	axis-parallel boxes: VC dimension at most $2d$, and the inequality for boxes with rational corners (cited by Lemma 8.12(ii)'s marks)	`vcDimLE_boxes`, `measurableSet_box`, `vc_inequality_ratBoxes`	machine-checked
`VC.LugosiNobel`	[LN96, Lemma 1]: the VC inequality for families of partitions	`traceOf_unionCell`, `card_traces_unionCell_le`, `cnt_biUnion`, `partDev_le`, `lugosi_nobel_lemma1`	machine-checked
`VC.GlivenkoCantelli`	[LN96, Corollary 1]: almost-sure uniform convergence over partition families, by Borel–Cantelli	`map_firstN`, `ae_eventually_firstN_notMem`, `measurable_partDev`, `lugosi_nobel_corollary1`	machine-checked
`VC.Summability`	Corollary 1 under the paper's (a) and (b): they make Lemma 1's bound summable (the derivation §8.4 gives)	`summable_of_sublinear`, `lugosi_nobel_corollary1_ab`	machine-checked
`VC.HistogramDev`	Theorem 1, the estimation term: the L_1 distance from the empirical histogram to the histogram of the true cell masses is the partition's deviation; it vanishes almost surely under (a), (b)	`histDev_eq_partDev`, `histDev_le_partDev`, `histDev_tendsto_zero_ab`, `integral_abs_histo_sub_le`	machine-checked
`VC.HistogramOp`	Theorem 1, step (A): the histogram operator is an L_1 contraction	`sum_indicator_le`, `histOp_contraction`	machine-checked

Module (`ProveDog.DecisionTheory.`)	Certifies	Named theorems	Status
`VC.HistogramApprox`	Theorem 1, eqs. (11)–(15): the approximation error of the histogram operator, split into cells of small oscillation, the other cells and the uncovered region	`histOp_apply_of_mem`, `abs_sub_cellAvg_le`, `setIntegral_abs_sub_cellAvg_le_two`, `integral_abs_sub_histOp_le`	machine-checked
`VC.HistogramDensity`	Theorem 1, decomposition (10) on one partition, the deterministic half	`integrable_histo`, `histOp_eq_histo_of_law`, `sum_setIntegral_le`, `integral_abs_sub_histo_le`	machine-checked
`VC.HistogramConsistency`	Theorem 8.8: strong L_1 consistency of histograms on data-driven partitions under (a), (b), (c), for a countable family of partitions, each a finite family of measurable, pairwise disjoint cells of finite reference measure, the partition any function of the outcome	`integral_abs_sub_histo_le_g`, `measurableSet_coarse`, `lugosi_nobel_theorem1`	machine-checked
`VC.EmpiricalSLLN`	the strong law for empirical cell frequencies, for one set and for a countable family at once (Theorem 8.10(c)'s shallow cells; Lemma 12.4(iv))	`cnt_div_tendsto`, `cnt_div_tendsto_all`	machine-checked
`VC.Coarsen`	Theorem 8.10(c) in abstract form: (M2) and a cell hierarchy whose deep cells are small give condition (c), hence consistency under (a), (b)	`coarse_tendsto_zero`, `consistency_of_coarsening`	machine-checked
`VC.RegularPaving`	the regular-paving cells of $[-1,1)^d$: exact split counts under first-widest bisection, the widths, small deep cells (Lemma 4.23(iii) for the cube)	`nsplit_eq`, `width_eq`, `small_of_deep`, `measurableSet_cell`, `countable_finset_paths`	machine-checked
`VC.CorCoarsen`	Corollary 8.11: the threshold paving cut off at depth D_n is strongly L_1-consistent, proved from `lugosi_nobel_theorem1`, on the half-open cube, any regular reference measure, $D_n/(\overline{\#}_n+1)$ for $D_n/\overline{\#}_n$, $f \geq 0$ everywhere	`card_goodFam_le`, `card_cellTraces_le`, `mass_cmap`, `shallow_cnt_le`, `plab_thr_iff`, `cor_coarsen`	machine-checked

Not in this table: Hoeffding's inequality and Sauer–Shelah, which are Mathlib's (`Probability.Moments.SubGaussian`, `Combinatorics.SetFamily.Shatter`) and are rows of the register with status `cite-mathlib`; the skeleton estimate and the minimax lower bounds, which the register lists as stated and absent. No statement of this monograph cites those rows: Proposition 16.6(iii) proves the skeleton bound it needs from Theorem 9.1 and Hoeffding's inequality, and its [LEAN] mark names the module to write.

15.2 The specification of the pipeline

These modules state the objects of Chapters 4, 6, 8, 10 and 11 and prove the statements their marks name. What each leaves outside its model is said in the second column.

Module (`IasCert.L1MDE.`)	Certifies	Named theorems	Status
`Labels`	the heap code of §4.6 and Proposition 4.27 except (iv), Lemma 4.28(a), (b), the code's labels: a bijection with the positive integers; the left-right order a strict partial order on all labels, total on an antichain; the subtrees contiguous; on unbounded naturals	`enc_injective`, `dec_enc`, `enc_dec`, `enc_left`, `enc_right`, `enc_parent`, `enc_ancestor`, `enc_bounds`, `depth_eq_log`, `scala_isAncestorOf_iff`, `strictLeft_iff_block`, `leftRight_trans`, `leftRight_total_on_antichain`, `leftRight_asymm`, `leftRight_irrefl`, `leftRight_not_antisymmetric`, `strictLeft_same_depth`, `subtree_contiguous`, `sibling_chain_antichain`, `suffix_join`	machine-checked
`PropBlock`	Proposition 4.27(iv): for disjoint nodes u, v, $u \prec v$ exactly when u lies below the left child of $u \wedge v$	`strictLeft_iff_left_join`, `ancOrSelf_iff_suffix`, `strictLeft_of_ancOrSelf`, `left_strictLeft_right`, `exists_turn_below_join`, `turns_differ`, `turn_unique`	machine-checked
`ScalaJoin`	the code's `join` through the bit length of an exclusive or is the deepest common ancestor (§4.6; Lemma 11.19's marks)	`xor_size_spec`, `scalaJoin_eq_join`	machine-checked
`SparseUnion`	Definition 6.6 and Proposition 6.7(i): the sparse closure as a predicate, its cells an antichain, each inside an element of N, each in or disjoint from every element, covering exactly what N covers; (ii), (iii) argued in the text	`cl_antichain`, `cl_sub`, `cl_in_or_disj`, `cl_cover`, `gaps_disj`, `disj_of_sub`, `disj_or_nested`, `ancOrSelf_parent_of_below`	machine-checked
`CountMap`	Proposition 6.18(i)–(v) on count maps (finitely supported maps to $\mathbb{N}$); Phase 2 is partition-invariant	`merge_comm`, `merge_assoc`, `merge_zero`, `relabel_add`, `total_relabel`, `relabel_comp`, `relabel_id`, `coarsen_add`, `coarsen_coarsen`, `total_coarsen`, `coarsen_countOf`, `countOf_append`, `countOf_perm`, `countOf_flatten`, `mass_add`, `mass_children`	machine-checked

Module (`IasCert.L1MDE.`)	Certifies	Named theorems	Status
`MRPLift`	Definition 6.17 and Proposition 6.18(i), (ii), (iii first part), (v) for values in any commutative monoid; summing out the ids of a labelled MRP (Definition 5.20, Proposition 6.33): it commutes with coarsening; the case $\mathbb{N}$ is `CountMap`	`relabel_zero`, `relabel_single`, `relabel_add`, `total_relabel`, `relabel_comp`, `relabel_id`, `coarsen_add`, `coarsen_coarsen`, `coarsen_of_shallow`, `total_coarsen`, `mass_add`, `mass_eq_sum`, `mass_children`, `atDepth_add`, `atDepth_coarsen`, `mergeSubtree_add`, `total_mergeSubtree`, `sumOutIds_add`, `total_sumOutIds`, `coarsen_sumOutIds`, `countMap_mass`	machine-checked
`ThresholdPaving`	Lemma 8.3(i), (ii) under (G), and the paving cut off at depth D without it, with the bounds behind (M1), (M2) of Corollary 8.11 (the shallow leaves' mass at most $2^k\overline{\#}$); (iii) is `GDepthBound`	`internal_parent`, `internal_depth_lt`, `leaf_mass_le`, `isLeaf_root_iff`, `isLeaf_cons_iff`, `leaf_cover`, `card_internalSet_le`, `card_leavesOf`, `leafD_cover`, `card_leafSet`, `card_leafSet_le`, `shallow_leaves_mass_le`	machine-checked
`PavingCount`	Proposition 4.33 and Theorem 8.10(b): at most $C_k \le 4^k$ pavings with k internal nodes (tree shapes, so partitions for a fixed root box)	`toTree_injective`, `numNodes_toTree`, `card_family_le_catalan`, `catalan_le_four_pow`, `card_family_le_four_-pow`	machine-checked
`Approx`	Lemma 4.25(iv) and (iii) on the finest cells: the average over the leaves of the complete paving of depth k keeps each leaf integral, is an L_1 contraction, and returns a function constant on the leaves of s_{k_0} for every $k \ge k_0$; (i), (ii) and the limit for a general density argued in the text	`avg_eq_of_const`, `sum_avg_block`, `sum_abs_avg_block_le`, `sum_abs_avg_le`, `lem_approx_iv`	machine-checked
`OracleInequality`	Theorem 9.1(i) on a common refinement: `min_distance_oracle` instantiated for candidate histograms on a finite leaf set, the candidate-pair step from the finite Scheffé lemma; the two truth-side steps hypotheses (`hdev`, `htri`)	`gap_le_dev`, `pair_step`, `exists_min_dev`, `oracle_finite`	machine-checked
`OneDelta`	Proposition 10.1, the deterministic part and the tail: under (T) every round's candidates lie on one path; its Yatracos class has at most $\lvert P\rvert^2$ sets; every round has the oracle inequality with $4\Delta_P$; $\mathbb{P}\{\Delta_P > \epsilon\} \le \lvert P\rvert^2 B$ for a per-set tail B; the bound on $\mathbb{E}\,\Delta_P$ argued in the text	`onPath_iterate`, `candidates_onPath`, `card_yatClass_le`, `devOn_mono`, `round_dev_le`, `round_oracle`, `tail_le`	machine-checked

Module (`IasCert.L1MDE.`)	Certifies	Named theorems	Status
`CertifiedSearch`	Lemma 10.3(i)–(iv), the gap lower bound, Theorem 10.4 (with Theorem 9.1(i) over U a hypothesis) and the tail of Corollary 10.5; the Borel–Cantelli step and the limit argued in the text	`cherry_step`, `arc_add`, `err_lipschitz`, `total_arc_le`, `err_between`, `nearer_end`, `score_lower`, `score_lower_sup`, `two_abs_mass_sub_le_L1`, `score_lower_finite`, `gap_lower`, `certified_min`, `thm_certified`, `tail_eq`, `cor_certified_tail`	machine-checked
`MergeWalk`	Definition 11.1 and Lemma 11.2(a)–(c) (the walk $W(0)$ and $W(\delta)$, raising over atoms in left-right order); Lemma 11.4(i) and (ii) (the local walks under a safe δ, the driver's atoms an antichain that covers); Corollary 11.6's first condition when every stored label lies below a subtree root. It models the specification (the driver walk always runs; no root relabelling); the streaming form Lemma 11.4(iii) to write (`Stream`)	`walk_spec`, `walkD_spec`, `output_eq_leaves`, `raised_eq_leaves`, `lem_merge`, `local_walk_props`, `mass_localCounts`, `local_outputs_not_nested`, `driver_atom_props`, `driver_atoms_cover`, `driver_atoms_antichain`, `safeDepth_of_roots`	machine-checked
`Raise`	Lemma B.2 and Lemma 11.2(d): `maximalSubtreesDecreaseDepth` as the code runs it (l.236–287) equals the raising of `MergeWalk` for $S \geq 2$ and any depth limit, and the statement of Lemma B.2 at depth limit 0; the single-subtree case with a positive limit outside the model	`left_right_join`, `raiseD_eq_of_max`, `raiseD_zero`, `decreaseDepth_eq`, `raiseAt_eq_raiseD`, `decreaseDepth_eq_raiseD`, `lem_raise`, `lem_raise_props`	machine-checked
`MergeNoG`	Lemma 11.7(i), (ii) for one list: the walk without (G) emits an over-full atom on its own, every internal node is shallower than D and heavier than the count limit, the leaf bound; the empty leaves, (iii), (iv) and Corollary 11.8 argued in the text	`walk_spec_noG`, `mass_raise`, `raise_parent_heavy`, `raise_not_nested`, `mem_outFinset`, `mem_internalOf`, `ancClosed_internalOf`, `lem_mergeNoG`	machine-checked
`Exact`	Lemma 8.4 (SEBTreeMC as a relation, ties open, under (G) and $\overline{m} > D\lfloor n/(\overline{\#}+1)\rfloor$); the merge part of Theorem 11.16; Lemma 11.9(i) per state under any tie-breaking (the single path under (T) not modelled); the scoring step outside the model	`reach_inv`, `seb_step_exists`, `seb_no_step_at_threshold`, `lem_seb`, `thm_exact`, `thm_exact_merge`, `thm_exact_single`, `thm_exact_distributed`, `seb_split_noninc`, `seb_last_split`, `seb_reverse_unique`, `back_inv`, `back_step_is_seb`, `reach_of_back`	machine-checked
`Pipeline`	the merge half of Theorem 11.16 end to end, with $\overline{m} = 1 + \lceil Dn'/\overline{\#}\rceil$: each partition's emitted subtrees are its local walk $W(\delta)$ over its stored labels, and the driver's walk over the raised outputs returns the nonempty leaves of every terminal run of SEBTreeMC	`mbar_bound`, `localEmitted_props`, `thm_exact_pipeline`	machine-checked

Module (`IasCert.L1MDE.`)	Certifies	Named theorems	Status
`ThmExact` (`ProveDog.L1MDE`)	Theorem 11.16 under (G), the two halves composed: the merged histogram is the set of point-holding leaves of the terminal state of SEBTreeMC, the distributed count oracle (each partition's scan, added on the driver) is the one-machine count for every round and pair, so every selection procedure reading the merged histogram and the counts returns the sequential estimate, for every partitioning and with one partition per round; the driver walk over the counts the records carry (the stored leaves and their counts), and the distributed run at a coarser depth $D^{\mathrm{G}} \le D \le M$ (every cell a node of s^* as a hypothesis); any leaf cap; the instance at range partitioning; (iv) for Theorem 10.4; the case without (G) to write; (X1)–(X7) and floating point outside the model	`distCount_eq_seqCount`, `thm_exact`, `thm_exact_regroup`, `thm_exact_one_partition`, `walk_congr`, `mergeOutput_congr`, `mass_single`, `driverCounts_cons`, `mass_driverCounts`, `list_sum_map_eq_zero`, `list_sum_eq_single`, `list_sum_finsupp_sum`, `mass_eq_sum_atoms`, `mass_driverCounts_eq`, `output_below_atom`, `driver_record_count`, `records_leaves`, `thm_exact_records`, `thm_exact_depth`, `thm_exact_mbar`, `thm_exact_records_mbar`, `thm_exact_depth_mbar`, `thm_exact_range`, `thm_exact_certified`	machine-checked
`Depth` (`ProveDog.L1MDE`)	Lemma 11.18(i)–(iv) from the count map at a finest depth with (G) there: (G) at D exactly when $D \ge D^{\mathrm{G}}$, $s^*_D = s^*$ and every leaf count the same for $D^{\mathrm{G}} \le D \le M$, the leaves of depth at most D^{G}; (iii)'s main claim, every search reading the histogram with its counts and the round counts the same at every such D (every cell a node of s^* as a hypothesis); Remark 12.2(b); Remark 12.2(a) outside the model	`mass_coarsen`, `condG_coarsen_iff`, `gDepth_le`, `condG_iff_gDepth_le`, `condG_gDepth`, `gDepth_eq_zero_iff`, `internalSet_coarsen`, `leafSet_coarsen`, `leaf_length_le_gDepth`, `ancOrSelf_truncate_iff`, `leaf_mass_coarsen`, `seb_terminal_coarsen`, `seqHist_coarsen`, `inSet_truncate`, `yCount_truncate`, `roundCounts_truncate`, `depth_invariant`, `card_support_coarsen_mono`, `card_support_le_total`, `card_support_le_pow`, `card_occupied_le`, `le_gDepth`, `condG_mono`, `exists_leaf_length_eq_gDepth`	machine-checked
`FixedWidth` (`ProveDog.L1MDE`)	Lemma 11.19 on a w-bit two's-complement word, $D + 2 \le w$: the bound, shifts, children, sibling, bit length, the code's join, the left-right order and equality agree with the unbounded label operations; hashing, the pair of `Long`s and the conclusion for the algorithms and the pipeline outside the model	`wrap_of_fits`, `toSigned_wrap`, `signed_lt_iff`, `enc_fits`, `enc_lt_two_pow_63`, `enc_lt_two_pow_127`, `size_enc`, `wShr_enc`, `wShr_enc_truncate`, `wChild_enc`, `enc_sibling_xor`, `wSibling_enc`, `xor_fits`, `wJoin_eq`, `wStrictLeft_iff`, `wEq_iff`, `lem_fixedWidth_64`, `wrap_enc`, `shift_lt_width`, `wJoin_shift_lt_width`	machine-checked

Module (`IasCert.L1MDE.`)	Certifies	Named theorems	Status
`RangeSafe` (`ProveDog.L1MDE`)	Corollary 11.6(a) in general form, (b) for any partitioner read along the sorted list (δ safe exactly when $\delta > \max_i \lvert c_i \rvert$), and (c)	`le_cutDepth`, `anc_between`, `part_eq_of_below`, `safe_of_cutDepth_le`, `cutDepth_le_of_safe`, `safe_iff`, `safe_cutDepth`, `not_safe_of_lt`, `safe_of_no_boundary`, `cutDepth_le_of_snapped`, `safe_of_factor`, `safe_of_snapped`, `join_length_lt_left`, `cutDepth_le_D`, `cutDepth_le_of_atDepth`	machine-checked
`ArcPath` (`ProveDog.L1MDE`)	Theorem 16.7(i), the order, on the event as a hypothesis: every run of the arc-length backtrack, under every tie-break, passes through π_f, and the merges inside the leaves of π_f come first; (ii) only as Theorem 10.4 at the index of π_f, the error of a state abstract; the refinement at the algorithm's own s^* (positive-mass leaves and the strong-law event as hypotheses), a run's existence, the event's satisfiability; the event's probability, the identity for the error of π_f, its expectation and (T) outside the model	`notMem_iff_below_leaf`, `child_mem_leavesOf_iff`, `arcStep_subset`, `arcStep_internal`, `thm_arcPath`, `thm_arcPath_order`, `thm_arcPath_certified`, `subset_internalSet`, `arcEvent_internalSet`, `thm_arcPath_-internalSet`, `arcRun_exists_aux`, `arcRun_exists`, `thm_arcPath_exists`, `mass_single_eq`, `arcEvent_satisfiable`	machine-checked
`GapRounds` (`ProveDog.L1MDE`)	Lemma 11.14(b) and the bounds of Proposition 13.11, the gap search as a structure whose fields are the statement's hypotheses: a round adds a state exactly when a gap is unsettled, the search stops, $\lvert U \rvert \le \lvert P \rvert$; the first-round and bisection gap bounds; for $\epsilon_0 > \sigma$ every gap settled after r^* rounds and the count of $\lvert U \rvert$, the A-form strictly when $\sigma > 0$ and the D-form strictly always; the stopping round and the set the search stops with; r^*'s existence and a witness run; θ_0, θ_H in every round; settling through ξ alone; $\lvert P \rvert$'s bound (Lemma 8.3(i)) and $A \le D$ (Lemma 10.3(ii)) as cited, the cost statements outside the model	`arc_succ`, `arc_nonneg`, `arc_mono`, `bracket`, `unsettled_lt`, `settled_mono`, `settled_of_small`, `xi_settle`, `iterate_halves`, `consec_right`, `lend_eq`, `Run.mono`, `Run.le_H`, `Run.card_le`, `Run.adds_iff`, `Run.grow`, `Run.stops`, `Run.pick_within`, `Run.first_gap`, `Run.bisect_halves`, `Run.gap_inv`, `Run.settled_after`, `Run.subset_of_le`, `Run.card_new_le`, `Run.card_le_pow`, `Run.card_real`, `Run.lem_rounds_b_iii`, `Run.lem_rounds_b_iii_-strict`, `Run.exists_stop`, `Run.stopRound`, `Run.stopRound_le`, `Run.stopRound_spec`, `Run.ends_mem`, `Run.lem_-rounds_b_iii_strict_D`, `Run.lem_rounds_b_iii_stop`, `exists_rstar`, `settled_range_two`, `trivialRun`, `run_nonempty`	machine-checked

Module (`IasCert.L1MDE.`)	Certifies	Named theorems	Status
`Union` (`ProveDog.L1MDE`)	Proposition 6.5: the union of internal-node sets is the coarsest common refinement, its leaves the nonempty intersections, a commutative idempotent monoid with the root paving as identity, refinement a partial order, the leaf count; Proposition 6.7(ii), and (iii) at the level of sets ($\mathrm{cl}(A \cup B)$ through the runs and gaps; the list merge is `UnionMerge`); Lemma 6.4 and the local law for MRPOperate; refinement stated on labels; the merge's cost outside the model	`prop_union`, `refines_iff`, `union_monoid`, `refines_order`, `card_leaves_union_le`, `sparseUnion_ii`, `sparseUnion_ii_noGap`, `run_antichain`, `run_max`, `gaps_eq_leaves`, `cl_merge`, `lem_localRefine`, `cl_below_unique`, `cl_local`, `union_local`, `operate_local`, `upVal_local`, `upVal_congr`, `below_of_common`, `restrictTo_union`, `mem_restrictTo`, `exists_leaf_below`, `mem_leavesOf_union`, `not_mem_of_below_leaf`, `mem_of_below_leaf`, `ancClosed_union`, `ancClosed_empty`, `below_mem_other`, `ne_nil_of_below`, `below_parent`, `exists_below_deep`, `exists_below_not_mem`	machine-checked
`UnionMerge` (`ProveDog.L1MDE`)	Lemma 6.2 (Fill's output: the leaves between a and b, in left-right order, $\lvert b\rvert - \lvert a\rvert + 1$ labels) and Proposition 6.7(iii) as lists: the run's contiguity, `Gaps`, and the merge of Algorithm 2 branch for branch, its output in left-right order with no duplicates and members exactly $\mathrm{cl}(A \cup B)$; its cost outside the model	`lem_fill`, `fill_length`, `fillLeft_trans`, `fill_spec_self`, `run_contiguous`, `right_of_join`, `gapsList_spec`, `leafOf_refine`, `pairwise_refine`, `mem_refine`, `children_leaf_iff`, `inCl_sorted`, `inCl_union_disj`, `inCl_run`, `sparseMerge_nil_left`, `sparseMerge_nil_right`, `sparseMerge_cons_cons`, `single_step`, `run_step`, `sparseMerge_spec`, `sparseUnion_iii`	machine-checked

Module (`IasCert.L1MDE.`)	Certifies	Named theorems	Status
`Canonical` (`ProveDog.L1MDE`)	Theorem 5.5 (i)–(iii) with Lemma 5.4, the post-order pass in pointer form; Proposition 6.11's correctness, sparse and dense; Proposition 5.22(v); the scheme's cells nonempty as in §5.1; the cost sentences, Proposition 5.22(i)–(iv) and floating point outside the model	`Scheme.mem_of_ancOrSelf`, `Scheme.nested_of_mem`, `below_cases`, `nset_ancClosed`, `nset_subset`, `terminal_eq_canonical`, `thm_canonical_i`, `thm_canonical_ii`, `thm_canonical_ii_min`, `mergeAt_represents`, `mergeAt_noCherry`, `minimiseAt_frame`, `minimiseAt_vals`, `minimiseAt_represents`, `minimiseAt_noCherry`, `thm_canonical_iii`, `canonical_unique`, `transform_represents`, `prop_transform_v`, `prop_transform_v_inj`, `sib_adjacent`, `reduce_spec`, `pass_spec`, `stack_leaf`, `stack_leaf_eq`, `stack_canonical`, `prop_minimiseStack_sparse`, `prop_minimiseStack_dense`	machine-checked
`CanonicalScheme` (`ProveDog.L1MDE`)	Theorem 5.5's scheme hypothesis met: the dyadic scheme on the half-open $[0,1)$, its cells nonempty, and (iii) on it; Proposition 5.22(v) for every MRP (every MRP with an ancestor-closed set of internal nodes represents its own leaf function), and its first law with no scheme; a box $[a,b)$, $d>1$, the closed root box and Proposition 5.22(i)–(iv) outside it	`dyCell_split`, `dyCell_disj`, `dyCell_nonempty`, `dyadic`, `dyadic_cell`, `dyadic_root`, `thm_canonical_iii_dyadic`, `leaf_eq_of_mem`, `represents_leafFn`, `exists_represents`, `leafFn_comp`, `prop_transform_v_all`, `prop_transform_v_inj_all`, `prop_transform_v_mrp`	machine-checked
`Stream` (`ProveDog.L1MDE`)	Lemma 11.4(iii): the streaming walk emits exactly $W(\delta)$ and raises as the materialised walk does, with no count or depth hypothesis; its buffer of at most $\overline{\#}+1$ records (counts ≥ 1); Proposition 13.6(iv)'s addition count, at most $\nu \min(D, \overline{\#}+1)$; the partition case; the word cost of joins and tests and the bytes outside the model	`lem_merge_iii`, `stream_eq_walkD`, `lem_merge_iii_buffer`, `prop_opCost_iv`, `lem_merge_iii_partition`, `streamAdds_le_depth`, `streamAdds_le_cap`, `streamAdds_le_slack`, `streamBuf_le`, `lookEnds_iff`, `lookReads_le`, `lookAdds_le`, `lookReads_le_lookAdds`, `lookReads_zero_counts`, `runSum_eq`, `le_runSum`, `runSum_eq_mass`, `mass_eq_sum_cover`, `raiseNb_eq_raiseD`, `raiseD_alone`, `delayRaise_eq`, `raiseStream_eq`, `slack_refuse`, `slack_extend`, `sum_slack_le`, `pairwise_emit`, `listSum_eq_zero`, `listSum_eq_single`, `listSum_finsupp_sum`	machine-checked

Module (`IasCert.L1MDE.`)	Certifies	Named theorems	Status
`StreamCounts` (`ProveDog.L1MDE`)	Lemma 11.4(iii)'s emitted counts: the counting walk emits the streaming walk's labels, each with the mass of its subtree, so $W(\delta)$'s subtrees with their counts; the look-ahead's running sum is the run's; Proposition 13.6(iv)'s containment tests (the drop test, the overlap test and the look-ahead's reads), at most the additions plus 2ν, so at most $\nu(\min(D, \overline{\#}+1)+2)$; the raised output's counts and the word cost of a test outside it	`lookSum_eq_runSum`, `lookAdds_eq_length`, `lookAhead_ends`, `streamWalkC_fst`, `streamInv_start`, `streamInv_drop`, `streamInv_refuse`, `streamInv_extend`, `streamWalkC_eq`, `streamWalkC_count`, `lem_merge_iii_counts`, `streamTests_le_adds`, `prop_opCost_iv_tests`	machine-checked
`Swap` (`ProveDog.L1MDE`)	Theorem 6.38 (i)–(v): every one-step statistic, its sums over intervals, on any coarser partition, and the count maps of a regular paving, preserved by any sanitization; Corollary 6.41 (i)–(iv) for the product state; Remark 6.39 and that Corollary 6.41(iv)'s formula is the chain binomial's likelihood outside it	`oneStep_swap`, `oneStep_sanitized`, `thm_swap_i`, `thm_swap_ii`, `thm_swap_iii`, `thm_swap_iii_stats`, `thm_swap_countMap`, `thm_swap_iv`, `thm_swap_v_step`, `thm_swap_v_block`, `glue_spec`, `isSwap_glue`, `glue_fibrePerm`, `cor_swapSIR_i`, `cor_swapSIR_ii`, `cor_swapSIR_iii`, `cor_swapSIR_iv`, `likelihood_congr`	partial-deliberate
`Operate` (`ProveDog.L1MDE`)	Proposition 6.14(i): MRPOperate represents $x \mapsto f_1(x) \star f_2(x)$ for sparse MRPs with antichain supports and $e \star e = e$; commutative, associative and with identity e as $\star$ is, associativity compared at points also deeper than the two intermediate supports, which is equality as represented functions since the supports are finite; Proposition 5.3(i)–(iii): the constants, closure under pointwise operations and the laws, the subring and subalgebra, the zero divisors and the units; Proposition 6.14(ii) and Proposition 5.3(iv) outside it	`prop_operate_i`, `operate_comm`, `operate_assoc`, `operate_id`, `isOperate_antichain`, `upVal_cl`, `upVal_point`, `algebra_closure`, `algebra_law`, `inF_const`, `algebraSubring`, `algebraSubalgebra`, `algebra_iii`, `algebra_units`, `inF_indicator_child`	machine-checked
`Fold` (`ProveDog.L1MDE`)	Proposition 6.19(i), (ii), (iv) for count maps: the fold in any monoid, coarsening and insertion on a folded MRP, the count at every node; Proposition 6.50(i)–(iv) over an abstract commutative idempotent monoid with the interval model's facts as hypotheses (the points map and the width monotone in the hull order, vol ≥ 0, cells covered by their children's, G isotone), which makes the row structural; Proposition 6.18(vi); Proposition 6.33(iv); Proposition 6.19(iii), the costs and Mathlib's intervals outside it	`prop_fold_i`, `foldRoot_leaf`, `prop_fold_ii_coarsen`, `prop_fold_ii_insert`, `prop_fold_iv_count`, `prop_fold_iv_countMap`, `folded_mass`, `prop_hull_i`, `prop_hull_ii`, `prop_hull_iii`, `prop_hull_iv`, `prop_relabel_vi`, `prop_relabel_vi_adjacent`, `prop_sumOut_iv`, `prop_sumOut_iv_add`, `prop_sumOut_iv_remove`	machine-checked-structural

Module (`IasCert.L1MDE.`)	Certifies	Named theorems	Status
`PathRefine` (`ProveDog.L1MDE`)	Lemma 11.10(i)–(iii): leaves of the states and cells of cl(N) are unions of leaves of s, cl(N) = $\mathcal{L}(s)$ for a nonempty sample with s and the root among the states, the states constant on the leaves of s, kept by dropped cells and MinimiseLeaves; Theorem 11.12(i)–(ii) with the skipping and MinimiseLeaves (exact arithmetic, histogram states), (iii) condition (R) and the first round, (iv) $N_{\theta,\vartheta}$ as a sum of κ^* over the leaves of s and the point location; Proposition 13.6(vii): the pointer's moves, the truncations and $h(h-1)/2$ comparisons per record inside a row; (C) over points, (S) after a sort, (E), (N), Spark, floating point, Proposition 6.9's binary search and word costs outside it	`lem_pathRefine_i`, `lem_pathRefine_ii`, `lem_pathRefine_iii`, `lem_pathRefine_iii_keep`, `onLeaves_cl`, `skip_flat`, `yatracos_cl`, `inRows_minimise`, `inSet_record`, `thm_dssc_i`, `thm_dssc_ii`, `thm_dssc_iii_R`, `thm_dssc_iii_first`, `kappa_perm`, `thm_dssc_iv`, `mem_iff_below_stored`, `point_location`, `moves_le`, `truncTotal_le`, `prop_opCost_vii`	machine-checked
`Sizing` (`ProveDog.L1MDE`)	Proposition 13.2(ii), (iv), (v), (vi): the bytes per record in each layout as functions of D and h with the layout table a hypothesis, the stored leaves between $\lceil n'/\overline{\#} \rceil$ and Φ under (G), the rows at most Φ; Lemma 13.9 with Lemma 11.2(b) for $W(\delta)$ a hypothesis; Proposition 13.4 with β-balance a hypothesis, and exact ranks for range partitioning ($b + sw$; finding: "$\beta = 1$ plus one record" does not give b); the waves $\lceil p/m \rceil$; Proposition B.9 with ε-accuracy and Corollary B.7's loads as hypotheses, which with Lemma 11.2(b) makes the row structural; (i), (iii), the heap figures themselves, Spark's memory regions, the JVM, Cor-LPTCap and Conjecture B.10 outside it	`prop_recordBytes_ii`, `prop_recordBytes_ii_two`, `heapBytes_63`, `enc_fits_labelWords`, `prop_recordBytes_iv`, `prop_recordBytes_v`, `prop_recordBytes_v_rows`, `prop_recordBytes_vi`, `lem_sizes`, `heavy_parent_of_raised`, `sizing_le`, `prop_sizing`, `prop_sizing_range`, `range_one_record_over`, `prop_sizing_subtree`, `waves_eq`, `prop_balance`, `prop_balance_lpt`, `prop_balance_par`	machine-checked-structural
`Coverage` (`ProveDog.L1MDE`)	Proposition 6.47(i), (iii), (iv) and the remark after Definition 6.46 on a finite leaf set: c_α exists with $c_\alpha \geq h_+ > 0$, R_α overshoots $(1-\alpha)M(0)$ by less than the mass of the leaves of height c_α, the regions are nested, and a point lies in R_α exactly when cov $> \alpha$, $\alpha = 1$ included; (v) on the leaf masses from the finite Scheffé step, and its stated form with the step from leaf masses to $\int \lvert p - \bar{f} \rvert$ a named hypothesis; (ii), that step, the anomaly score and Algorithm 16's cost outside it	`prop_coverage_i`, `prop_coverage_iii`, `prop_coverage_iv`, `prop_coverage_v`, `prop_coverage_v_of_le`, `exists_hplus`, `calpha_pos`, `calpha_ge_hplus`, `calpha_mono`, `levelMassGt_calpha_lt`, `calpha_le_iff`, `lt_cov_iff`, `cov_le_one`	machine-checked

Module (`IasCert.L1MDE.`)	Certifies	Named theorems	Status
`PQThreshold` (`ProveDog.L1MDE`)	Proposition 6.51(i), (ii), (iv) for pavings as ancestor-closed internal-node sets and a priority in any linear order, under every tie-break: the split priorities do not increase and $\{\psi > t\} \subseteq I(s_k) \subseteq \{\psi \geq t\}$ with t the k-th largest value; the pruned priorities do not decrease and $\{\psi < t\} \subseteq P \subseteq \{\psi \leq t\}$ with t the k-th smallest value on $I(s)$, P closed downwards; the pruned sets of the runs exactly those sets; (iii) and Lemma 8.17 without their uniform laws: the nodes of value t pairwise disjoint, the tree or the pruned set the strict side with a subset of $\{\psi = t\}$ of the stated size, every such subset reached by splitting; those uniform laws and (v) outside it	`prop_pqThreshold_i`, `prop_pqThreshold_ii`, `prop_pqThreshold_iv`, `prop_pqThreshold_iii`, `lem_pruneTies`, `splitRun_order`, `pruneRun_order`, `prune_reach`, `split_reach`, `isKthLargest_unique`, `isKthSmallestOn_unique`	machine-checked
`Aspect` (`ProveDog.ProveDog.RegularPaving`)	Lemma 4.23(i)–(iv) for a general root box under the first-widest rule: every box of aspect ratio at most a; all boxes of one depth with one width vector and one split axis; the closed box of depth k, and the half-open cell, of volume $2^{-k}\mathrm{vol}(\mathbf{x}_\rho)$; the diameter bounds as bounds on pairs of points of the closure in the Euclidean distance, $\mathrm{diam}_k \to 0$ and shrinking cells; every width at most $W_\rho 2^{-\lfloor k/d \rfloor}$; the $O(dD)$ count, the cells of the closed convention as sets and the bridge to the cube of `VC.RegularPaving` outside it	`lem_aspect_i`, `lem_aspect_ii`, `lem_aspect_ii_same`, `lem_aspect_iii_volume`, `lem_aspect_iii_volume_cell`, `lem_aspect_iii_diam`, `lem_aspect_iii_diam_closure`, `diamK_small`, `diamK_nonneg`, `lem_aspect_iii_tendsto`, `lem_aspect_iii_shrink`, `lem_aspect_iii_shrink_cell`, `lem_aspect_iv_width`, `lem_aspect_iv_diam`, `lem_aspect_iv_diam_closure`, `pair_le`, `widths_pos`, `narrowest_widths_pos`, `widths_succ`, `firstWidest_le`, `box_cons`	machine-checked
`GDepthBound` (`ProveDog.ProveDog.DecisionTheory.L1MDE`)	Lemma 8.3(iii) from the count map at a finest depth M with (G) at M, which stands for "D^{G} is finite" when M is the deepest resolved depth: $D^{\mathrm{G}} = 1 + \max\{\lvert u\rvert : \#(u) > \overline{\#}\}$, $n \leq 2^{D^{\mathrm{G}}}\overline{\#}$, $D^{\mathrm{G}} \geq \log_2(n/\overline{\#})$, the leaf count, and (G) at M when every depth-M cell holds at most one point; the first sentence from `Depth`; the finiteness of D^{G} for distinct points of $\mathbb{R}^d$ outside it	`lem_threshold_iii`, `gDepth_eq_one_add_max`, `total_le_two_pow_mul`, `total_le_two_pow_gDepth_mul`, `div_cap_le_two_pow_gDepth`, `logb_div_cap_le_gDepth`, `card_leafSet_le_gDepth`, `condG_of_cells_le_one`	machine-checked

Module (`IasCert.L1MDE.`)	Certifies	Named theorems	Status
`SEB` (`ProveDog.ProveDog.DecisionTheory.L1MDE`)	Lemma 8.4 with and without (G), for a finite depth cap: the chain as the splitting queue of ψ_D on `PQThreshold`'s runs, under any tie-breaking, stops by its count criterion exactly when $\overline{m} > \lvert S_{\overline{\#},D}\rvert$, and then at s^*_D; under (G), `Exact`'s chain is that chain step by step; without (G), `Exact`'s chain is not SEBTreeMC; $D = \infty$, the law of the ties and the construction of a run outside it	`lem_seb_noG`, `lem_seb_noG_bound`, `lem_seb_noG_threshold`, `lem_seb_noG_-threshold_bound`, `lem_seb_iff`, `sebStep_iff_psiD`, `reach_countCrit`, `seb_stop_iff`, `seb_stopped`, `countCrit_iff`, `isMaxLeaf_psiD_iff`, `maxLeaf_mem_-internalSet`, `sebRun_inv`, `sebRun_card_le`, `reach_lt_mbar`, `exists_-of_not_countCrit`	machine-checked
`DSSCConditions` (`ProveDog.ProveDog.DecisionTheory.L1MDE`)	Theorem 11.12(iii)'s (C) and (S), in `ThmExact`'s shape: (C) over the points of the root box for the depth-D labels, kept by any map to an ancestor-or-self, so for their pushforward to the leaves of any paving; under (C) and (R), the count over the records is the number of points of the root box in DSSC's set; (S) after one merge sort of each partition whose distinct labels form an antichain; with (R) from `PathRefine`, every round's sorted partitions meet `ThmExact`'s hypotheses; the aggregated κ^* records, the code's TimSort, (E), (N), Spark and floating point outside it	`thm_dssc_iii`, `thm_dssc_iii_S`, `sortRecs_sorted`, `count_points`, `condC_depth`, `condC_map`, `condC_leaves`, `depth_records`, `leaf_records`, `hR_of_records`, `sorted_of_key`, `sortRecs_perm`, `flatten_sort_perm`, `flatten_singletons`, `sum_ind`, `yCount_witness`	machine-checked
`FindSubtree`	Theorem B.8(iv) and Corollary 11.6(a): `findSubtree` as the code runs it (l.401–439) depends on the leaf only through its depth-δ ancestor, for any list of subtrees of depth at most δ; `getPartition` is constant below any node of depth at least `maxSubtreeDepth`; Theorem B.8(i)–(iii) outside the model	`join_truncate`, `findSubtree_truncate`, `cor_safe`, `le_maxSubtreeDepth`, `cor_safe_-maxSubtreeDepth`, `pairwise_strict_of_-sorted`	machine-checked
`DSSC`	Theorem 11.12(i), and (ii) in exact arithmetic: the count of an ordered pair over a list of records adds over any split into partitions and does not depend on their order; every deviation, hence the round's winner, is that of one partition	`yCount_append`, `yCount_flatten`, `yCount_perm`, `dssc_-partition_invariant`, `dssc_deviation_-invariant`	machine-checked
`Scan`	Lemma 11.11 and Proposition 6.9(iii): the scan of `scheffeSetsValidationCount` (l.319–390) under (S) and (R): the pointer after each record, the leaf it finds, and the matrix entry it builds is `yCount`; the cost bound and the returned records of Theorem 11.12(iii) outside the model	`scanTest_iff`, `advance_eq`, `lem_scan_ptrs`, `lem_scan_ptrs_zero`, `before_le_length`, `scan_finds_leaf`, `inLeafAt_iff`, `codeAdds_iff_inSet`, `scanCount_eq_yCount`	machine-checked

15.3 The cost model

These four modules certify arithmetic of Chapter 13's kind: a cost model whose constants are hypotheses. What they do not certify is what Remark 13.14 says no certificate reaches.

Module (`IasCert.L1MDE.`)	Certifies	Named theorems	Status
`MPCCost`	a cost model of Spark-style runs: a stage is P tasks with their work and output words, a kind (narrow, shuffle, collect, accumulator, broadcast) and the driver's work; a schedule maps tasks to m slots; the parallel time is the largest slot load, the driver charged in full; words and the h-relation counted apart; the laws: conservation of work over slots, total$/m \le$ par $\le$ total, monotonicity, hrel $\le$ words $\le m \cdot$ hrel, additivity over a plan, and that no schedule is slower than one slot	`Stage.load_sum`, `Stage.par_le_total`, `Stage.total_le_mul_par`, `Stage.par_eq_total_of_one`, `Stage.par_mono_work`, `Stage.words_broadcast`, `Stage.hrel_le_words`, `Stage.words_le_mul_hrel`, `Plan.par_append`, `Plan.words_append`, `Plan.rounds_append`, `Plan.syncs_append`, `Plan.compTime_le_T1`, `Plan.T1_le_mul_par_add`	machine-checked
`MPCSchedule`	schedules onto m task slots: greedy schedules (each task to a least-loaded slot; list scheduling is one, LPT another) exist for every order; Graham's bound $m\cdot\text{par} \le \text{total} + (m-1)\,\max\text{work}$ for them (the second bound of Theorem B.6); the waves: with unit tasks par $= \lceil P/m \rceil$, with tasks of work at most c a greedy par $\le \lceil P/m \rceil c$ (Definition 13.3's waves); balance on the data: a task of at most Pts records with work at most $a\,\text{Pts} + b$ gives par $\le a\,\text{Pts} + b\,k$ for k tasks per slot, for any schedule (Proposition 13.4 in records); contiguous schedules	`Stage.maxWork_le_par`, `Stage.graham`, `Stage.waves_unit_lower`, `Stage.waves_unit_upper`, `Stage.waves_bounded`, `Stage.par_le_of_points`, `Stage.exists_greedy`, `Stage.isContiguous_const`	machine-checked
`LemRounds`	Lemma 11.14(a) as arithmetic on the search's recursion at `c0a1837` (span, step, candidates, size difference as hypotheses, the code's facts): a running round has a positive step and at most $2k-2$ candidates, so does the final round; the shrinking $(k-1)^r\,\text{sd}_r \le 2^r s_0$; the round bound in division-free form; termination for $k \ge 4$; a $k = 3$ run in which every round runs; a $k = 4$ witness; that the Scala realises the recursion argued in the text; (b) is `GapRounds`	`Run.steps_pos`, `Run.cands_le`, `Run.final_cands_le`, `Run.final_after_loop`, `Run.shrink`, `Run.rounds`, `Run.stops`, `pow_bound`, `k3_runs_forever`, `witness_k4`	machine-checked

Module (`IasCert.L1MDE.`)	Certifies	Named theorems	Status
`MPCPipeline`	the cost table of the notebook flow of the terabyte runs, not of the final algorithm (§14.2; the library at `c0a1837` as the notebooks ran it, stage by stage, each row a named hypothesis in the run's sizes): $T_m \le \sum_{\text{rows}} (a\,\text{Pts} + b\,t) + \sum \text{driver}$, the words, three shuffle rounds, the driver exchanges $7 + [\text{resampled}] + 3(R+2)$ and the barriers $9 + [\text{resampled}] + (R+1)$, derived from the rows' kinds; one finding: the refinement scan of round 1 is paid once per *task*, 16 times per slot at 4096 tasks on 256 slots, where the earlier accounting charged it once per machine (Proposition 13.6(vii) counts it per partition)	`sort_clause`, `Row.par_le`, `Row.words_le`, `Row.driver_le`, `rowsPlan_par`, `rowsPlan_words`, `rowsPlan_rounds`, `rowsPlan_syncs`, `NotebookRun.thm_cost`, `NotebookRun.thm_cost_expanded`, `NotebookRun.words_le`, `NotebookRun.driver_le`, `NotebookRun.rounds_eq`, `NotebookRun.exchanges_eq`, `NotebookRun.barriers_eq`	machine-checked

15.4 What is not checked

Theorem 8.8 is machine-checked, with the Vapnik–Chervonenkis inequality for partition families it needs, and Corollary 8.11 is proved from it rather than assumed. The specification of the labels, the count maps, the threshold paving, the merge walk with and without (G), exactness end to end, the scan and the count of the selection, the safety of subtree partitioning, the one-path property and the certified search's bound are checked. The following are not, and the sections' marks say so where they arise: Theorem 8.9 (its three conditions for SEBTreeMC's partitions); Theorem 9.1(ii) beyond its structural form; Lemma 4.28(c); the finiteness of D^{G} for distinct points of $\mathbb{R}^d$ in Lemma 8.3(iii); Lemma 8.4 at $D = \infty$ and the law of its ties; the $O(dD)$ count of Lemma 4.23(ii); the aggregated κ^* records of Theorem 11.12(iii); Proposition 6.7(iii)'s cost; Lemma 11.2(b) for $W(\delta)$, a hypothesis of Lemma 13.9; Theorem B.3, Theorem B.8(i)–(iii), the $\frac{4}{3} - \frac{1}{3m}$ bound of Theorem B.6, and Corollary B.7, whose loads Proposition B.9 takes as hypotheses; every cost count of Remark 13.14's first list except the waves, the balance on the data, the bytes per record, the bounds on records but N_V, the sizing inequality in bytes and the scan's cost; the cost statements of Chapter 6; Proposition 6.47(ii), and (v)'s step from the leaf masses to $\int |p - \bar{f}|$, a hypothesis of `prop_coverage_v_of_le`; Proposition 6.51(v), and the uniform laws over ties in Proposition 6.51(iii) and Lemma 8.17; and the new results of §6.11 other than Theorem 6.38 and Corollary 6.41 (Proposition 6.44, Lemma 6.45), and that Corollary 6.41(iv)'s formula is the chain binomial's likelihood. Conjecture B.10 is open mathematics, not a formalisation task. A statement whose mark says "[LEAN: to write]" has a proposed module name; a statement whose mark says "[LEAN: none]" is a hypothesis about a machine, a program or a measurement, and the protocol of §13.5 and the runs of Chapter 14 are its only check.

As of the commit the tables name, the register holds 58 rows: 53 machine-checked (33 of the pipeline's specification and the cost model, among them wave 2's `Stream` at `8afd4d5f2`, among them wave 1's `RangeSafe` at `bdeef562e` and `FixedWidth` re-crunched at `fcb47e5df` with the red-team's fold, its `ThmExact` and `Depth` re-crunched at `feee9838b` with the red-team's fold, and wave 2's `ArcPath`, `GapRounds` and `Union` at `050df0219` and `Canonical` at `dbf40e07f`; 20 of the decision-theory chain), 1 machine-checked-structural (`L1MDERate`), 1 stated (`Skeleton`), 1 absent (the minimax lower bounds) and 2 cited from Mathlib. The statements whose marks say "to write" fall under the modules those marks propose; the crunch after each module lands is

what changes these numbers.

Marks. A reading of the register; no new claim. [LEAN: the tables.] [SIM: none; the register's own check is the crunch.]

Chapter 16

Directions

Everything in this section is open, except three observations in §16.5, which are proved there. Each item says what exists, what is missing, and where the earlier sections already reach.

16.1 Trans-dimensional estimation

The Moore rejection sampler of §3.3 samples exactly from a density on finitely many labelled domains, each label's density taken with respect to its own Lebesgue measure. That is, it samples from a density with respect to the sum of these measures [SY13, Theorems 1 and 4]. Its Example 1 mixes the diagonal of $[0,1]^2$, which carries a one-dimensional density, with the square itself. The diagonal's domain is to be read through its parametrisation by $[0,1] \subset \mathbb{R}^1$, although the source writes every domain as a subset of the $\mathbb{R}^{d_k}$ of its label.

Estimating such a density from data, with L_1 guarantees, is not addressed in [SY13] or elsewhere in this line of work. Every consistency result of Chapters 8 to 9 is for one density on one $\mathbb{R}^d$.

The natural estimator is a labelled mapped paving: - one regular paving per root box ${}^k\mathbf{x}_\rho$ in the parameter space of label k; - heights relative to that label's Lebesgue measure; - counts normalised over all labels.

Scheffé's identity holds for densities with respect to any dominating measure, so the selection of Chapter 9 applies in form. Whether the coarsening theorem of Chapter 8 extends is open, with each label's probability interacting with its own rate. This would close the circle opened by the trans-dimensional sampler of §3.3: simulation from trans-dimensional random variables there, and estimation of them here.

16.2 Many samples, many treatments

Avelin and Sainudiin pose exact Bayesian A/B testing as a trans-dimensional model choice, run on Spark (§3.6) [AS18]. For many groups, the binomial partition model of [SY13, §5.7] takes the set partitions of the m samples as labels, and samples exactly from the posterior over all of them. For $m = 4$ samples of pine seedlings there are 15 set partitions. From 10^7 exact samples, the posterior probability of $\{\{1\},\{2,3,4\}\}$ is estimated as 0.5548, with a standard error of about 1.6×10^{-4}.

Lifting this from binomial parameters to densities, with mapped-paving histograms in place of parameters, is open. A natural statistic is the largest difference between the empirical measures of two of the K samples over the Yatracos class of their collated histograms (§9.4). It costs one K-way collation and one reduction. Its use as a test is not studied here.

16.3 A revived C++ library

The C++ library MRS, built on C-XSC, implements the algorithms of Chapter 3 and Chapters 6 to 9 [SYH+18]: - the Moore rejection sampler and its enclosures; - the hull propagation and pruning of Chapter 6; - the randomised priority queues; - the Markov chains on pavings (§7.5); - the minimum distance selection with `CheckBounds` (Chapter 9).

Its arithmetic is the older recursive descent on pointer trees, which, as the author notes, never scaled to massive data or dimensions. Two directions follow, both open: - updating MRS for current compilers and for a current C-XSC; - lifting its arithmetic to the sparse labels of Chapters 4 to 6.

Set inversion and global optimisation [JKDW01] would then run on the same keys as the estimator.

16.4 Moving objects: air traffic and images

§§ 7.2 to 7.4 showed trajectories added, subtracted and pruned on one machine. §7.6 gave, as a design, the same operations as maps over keys $(i, \ell_{\delta^*}(x))$: a flight's id together with its cell, a root-labelled paving (Chapter 6, `Dfn-Forest`). Three directions follow. The first and third are the author's; the second is the co-trajectory paper's own closing question [SMT+20, §4.3, §6]. - **Air traffic at scale.** The design has not been built. Mock data could be made from the trajectories kept in the PostScript figures of the 2012 paper [TKS12]. - **Mobility and privacy.** The paper asks for a framework "integrated across multiple spatial and temporal resolutions in a distributed computational setting" [SMT+20, §6]. The root-labelled sparse paving of Chapter 6, with the coarsening of §7.8, could supply its multi-resolution and distributed data structure. Sufficiency at every coarser resolution, for counts and transition counts, is the lemma of §7.8. Still open are: - privacy guarantees; - sequences of origin–destination matrices across resolutions; - building the framework for a metropolitan network, as in [Tor23], or larger. - **Images.** Microscopy images of biological cells, in two, three or four dimensions (one, two or three of space, with time), are the third direction. Given segmented and tracked cells, a cell's identity plays the role of a flight's, and the arithmetic of occupancies applies.

Contacts read off the same root-labelled pavings, and the transmissions along them, are §16.11.

- **Air traffic at planetary scale.** A radar's range is a cone in three-dimensional space, over time. A distributed arithmetic of all aircraft co-trajectories, with the radar ranges of every airport and the position of every flight over the planet, needs one root box for them all: the annulus of the stratosphere over the earth, crossed with time. Defining that root box, with the cells of §16.5 on the surface and a coordinate for altitude, is the first problem. The records of each flight then carry position, velocity and acceleration in latitude, longitude and altitude, and the arithmetic serves collision avoidance and flight-management operations, the motivation of the 2012 paper [TKS12] at the scale of the planet. Simulated data for this setting can be reconstructed from the trajectories in the PostScript figures of that paper, for an implementation in SparkDensityTree.

16.5 The surface of the sphere

Aircraft fly over a surface that is nearly a sphere. Navigation uses an ellipsoid, and equal area on the sphere is not equal area on the ellipsoid. The author names three ways to pave it: - (a) over a radar's range, map the region to a rectangle; - (b) project from a pole onto a large enough box, leaving out a cap around the other pole; - (c) use regular-paving trees that partition the compact surface itself.

Only (c) covers the whole sphere. The costs of (b) are a disc-shaped domain inside a box, and a chart that does not preserve area (the stereographic projection preserves angles, not areas).

Three constructions of kind (c) are already at hand. - **A cylindrical paving.** The cylindrical projection preserves area [GHB+05, p.7]. So a single regular paving of the root box $[-1,1]\times[0,2\pi)$ in the coordinates $(z,\phi)=(\cos\theta,\phi)$ has cells whose spherical area is exactly $4\pi/2^\delta$ at depth δ. Its costs: cells near the poles are long and thin, though their diameters still shrink to zero, and the seam at $\phi=0$ needs the half-open convention. - **HEALPix** [GHB+05]. The sphere is split into twelve base pixels of equal area, three rings of four. Each is subdivided as a nested quadrilateral tree, and all pixels at resolution N_{side} have the same area $\pi/(3N_{\text{side}}^2)$ (their Chapter 6). The equality "by integration" is in their Chapter 6.3. In the nested scheme, $N_{\text{side}}=2^j$ (Table 1). Within a base pixel, the nested index p'_n interleaves the bits of two pixel coordinates: x along the north-east direction, y along the north-west (eqs. (13)–(14)). A pixel's "daughters inherit the pixel index of their parent and acquire two new bits" (Fig. 1). - **S2** [S2 26]. - The six faces of a cube are projected onto the sphere, and each is subdivided as a quadtree to level 30. - A cell's identifier is 64 unsigned bits: a 3-bit face, then two bits per level, then a single 1 that marks the level, then zeros. - Cells are square in the face's (s,t) coordinates and subdivided at their centres there. In the cube-space coordinates (u,v), a nonlinear transform of (s,t), they are rectangles "not necessarily subdivided around their center point" (the guide, Coordinate Systems). - S2 is not equal-area. The transform makes cells "roughly the same size on the sphere", and the guide's average-area formula is accurate only to within a factor of about 1.7.

Three observations, each with its proof. The interleaving is Górski et al.'s; reading it as a regular paving, and the halving, are this monograph's. 1. **A quadtree level is two paving depths.** On a square, two levels of bisection make one quadtree split: one cut of the square, then one cut of each half. So a level-j cell of a quadtree on a square is a depth-$2j$ cell of a regular paving of that square. - For HEALPix the square is the base pixel's (x,y) index square. - For S2 it is the face's (s,t) square, and only that square. In (u,v) the same trees are realised with off-midpoint cuts (§16.6). 2. **HEALPix's index already is a paving label.** Read most significant bit first, p'_n alternates a y bit and an x bit, coarsest level first. So it is the depth-$2j$ label of the regular paving that cuts y before x, with the lower index as the left child, and its numeric order is that paving's left-right order. The full index $p_n=fN_{\text{side}}^2+p'_n$ prefixes the label with the base pixel f: a labelled root box (§3.3, Chapter 6).

S2's identifier is also a fixed-width label with the face as a 3-bit prefix, but it differs from the labels of Chapter 4 in two ways. - *Across levels:* its trailing marker places a parent at the centre of its descendants' range, where a leading-1 encoding orders by depth first. - *Along a level:* it orders cells along a Hilbert curve, not left to right. Faces 4 and 5 set the top bit, so S2 ids must be compared unsigned.

Both orders keep every subtree a contiguous range of identifiers (S2's `range_min` and `range_max`), so the contiguous range partitioning of Chapter 10 applies to both. 3. **On HEALPix, every depth halves the area exactly.** - *Proof.* A depth-$2j$ cell is a nested pixel at $N_{\text{side}}=2^j$, of area $\pi/(3\cdot 4^j)$. A midpoint cut of its (x,y) index square along either axis separates its four level-$(j+1)$ pixels by one index bit. So each half is the union of two pixels of area $\pi/(3\cdot 4^{j+1})$, half the parent's area. The cut follows pixel boundaries, which are well defined on the sphere (their Chapter 6.3). - *Consequence.* At every depth δ, a cell of a base pixel has area $(\pi/3)/2^\delta$, so the volume rule of §7.1 holds with spherical area. The heights of §7.1 and the selection of Chapter 9 carry over unchanged. - *Scope.* The proof needs only equal pixel areas, not an equal-area chart. It fails for midpoint cuts in (z,ϕ) inside a base pixel, and for S2: there the rule holds for the (s,t) measure, and a density with respect to spherical area needs each cell's exact area, or a change of dominating measure (§16.1, §16.7).

Boundaries. With one *closed* root box per face, a point on a face edge would lie in two root

boxes, a cube vertex in three, and a HEALPix pole in four. That is double counting on a null set that real data do hit: equator and meridian values, poles, rounded coordinates. So the root boxes must be half-open, with a fixed rule for shared edges and vertices, or points must be assigned by the scheme's own map from points to cells. S2 states that its cells are closed sets, and gives a conversion to make each point lie in exactly one cell.

What remains open: - a root box for altitude alongside the surface; - distribution over finitely many labelled root boxes. Chapter 11 is stated for one root box. The extension is expected by prefixing the label, but it is still to be proved and certified. - for options (a) and (b), whose root boxes are not square, whether the first widest side should be measured in the chart or on the sphere.

The author expects this to interest people in dynamics and low-dimensional topology. One link is already in place: pairs of binary trees with the same number of leaves represent the elements of Thompson's group F (Chapter 1, Chapter 4) [CFGS15], which refers to Cannon, Floyd and Parry's introduction for the group.

16.6 The same trees, other cuts

A regular paving cuts at midpoints. The trees of Chapter 4, the Catalan binary trees, do not depend on where the cuts are. Another cut rule gives the same trees, the same labels and the same arithmetic on labels; only the volumes change, and with them the heights. - S2's (u, v) cuts are a ready example. - Cuts at equal mass are another.

The volume rule $\mathrm{vol}/2^{\delta}$ is then replaced by each cell's own volume, which must be carried with the label or computed from it.

16.7 Data near lower-dimensional sets

Storage already adapts: the sparse table holds only the occupied finest cells, and their number depends on how many cells of that size the support of the data meets, not on the size of the grid. Dyadic trees adapt to the intrinsic dimension of data in classification [SN06]. In the L_1 view, however, a law carried by a lower-dimensional set has no Lebesgue density on $\mathbb{R}^d$, and the dominating measure must change. That is the question of §16.1 again.

16.8 When the coarsening conditions are necessary

Lugosi and Nobel state their conditions as sufficient [LN96]. For fixed partitions, Abou-Jaoude gives necessary and sufficient conditions for convergence in L_1 in probability [AJ76a].

For the merges of Chapter 8, which coarsen the finest dyadic partition of a fixed box, Lugosi and Nobel's condition on the growth of the partition class follows from the leaf count (Chapter 8). So the question narrows to the other two conditions.

Conjecture (departitioning rules). For each n, let π_n be the paving returned by a merge rule of Chapter 8 at depth D_n, and $\pi_n[x]$ the cell containing x. Then the merged histograms converge in L_1 almost surely, for every density on the box, if and only if both conditions below hold almost surely: - (SC) $\mu\{x : \mathrm{diam}(\pi_n[x]) > \gamma\} \to 0$ for every $\gamma > 0$; - (V) $\mu\{x : \#(\pi_n[x]) \le K\} \to 0$ for every K.

What is known about each direction: - *Necessity of (V)* is suggested by the relative error $1/\sqrt{K}$ of a cell holding K points, as for k-spacing estimates, which need $k_n \to \infty$. - *Sufficiency* needs control of the empty leaves.

No claim is made on either. The results of [AJ76b] on random partitions, and later work on data-dependent partitions, bear on both, and they have not been checked against (V).

16.9 Tighter bounds for the search

The concentration term of the search of §9.5 can be taken over one Yatracos class fixed by the training points. Under a tie-breaking condition, every candidate of every round is a state of one backtrack path, and one deviation over its Yatracos class, of order $\sqrt{\log|P|/(\varphi n)}$, serves all rounds. Without that condition, the set Q of states the search can score, also fixed by the training points, takes the place of the path (§9.3; §10.4, `Rmk-Ties`).

The approximation term is settled for the gap search of §10.4, against every state of the path. The search as coded has only Theorem 2 against its last round's candidates, since a round need not contain the previous winner (§10.4). Open: - whether the factor 3 can be lowered for the nested states of one path, and whether the improper factor 2 survives a validation-dependent set of scored states; - a bound on how many states the gap search scores, in terms of the shape of $\theta \mapsto \int |f_{n',\theta} - f|$ along the path; - a stopping rule across count limits, refining while the finest state wins, with a guarantee on the result (§12.5).

The tools are: - the optimal factors of selection [BKM19]; - linear-time selection [MŠ07]; - selection with noisy comparisons [AFJ+18]; - the trade-off of time and accuracy [AACS25]; - Lepski-type selection along ordered families [GL11].

16.10 Conditional densities for sequence models

The slices and the marginals of Chapter 6 together give exact conditional distributions of a joint mapped paving given a context: the slice, divided by the marginal. Whether this arithmetic, on joint densities of discrete or embedded tokens, is of use in large language models is an exploratory question only.

16.11 Contacts and transmission trees

A root-labelled paving gives a contact network that moves with the ids (§6.11, Definition 6.42). At each time t_k, a neighbourhood rule $\mathcal{N}$ around each id's occupied leaf gives a contact graph $G_{\mathcal{N}}(t_k)$. Over k these graphs make one temporal graph, whose edges are read off positions.

Here every id keeps its identity. What passes between ids is a token, a message or an infection, and who passes it to whom is the event of interest. The swaps of §6.11 read the same paving another way, for another decision problem: they hide which future is whose. The two readings are kept apart below.

The neighbourhoods need not come from a partition of space. - A ball of radius r around an id's leaf, or the m ids whose leaves are nearest to it, finds its contacts directly. - The cell rule at depth j is one case of $\mathcal{N}$. It misses close pairs whose leaves have no common ancestor at depth j, that is, pairs separated by the cut of an ancestor shallower than j; across the root's first cut it misses them at every depth $j \geq 1$ (Definition 6.42). - The nearest-neighbour rule gives a directed graph: i' may be among the m nearest to i without i being among the m nearest to i'.

Finding neighbourhoods and the arithmetic of the trees are separate computations (Remark 6.43). They are joined only where a decision problem needs both, as for the incidence of an infection per cell given who met whom.

Transmission processes on a moving network. [SW16] grows ranked transmission trees by an SI process on a fixed contact network. [God21] extends the process to SI, SIR, SIS and SIRS on directed networks, with a parameter that blocks a node's edges once it is infected. It leaves time-varying networks open. The graphs $G_{\mathcal{N}}(t_k)$ are such a network, and a directed one under the nearest-neighbour rule. On them, with the steps of the process at the times t_k, the processes keep their form: - a token passes at t_k only along the edges live at t_k; - an id holds the token between its contacts (store-carry-forward, §6.11).

Two open questions, those of §6.11: - *The forward question.* Fix a rule $\mathcal{N}$ and a law of trajectories. Which law on ranked transmission trees do they induce, and how does it change with r, m or the depth j of the cell rule? For the complete graph, [God21] reports SIR runs in which every ranked planar shape is equally likely. That is one baseline to compare against. - *The inverse question.* What does an observed set of transmission trees say about the paving's contacts? This is the inverse problem of [SW16], posed on a network that moves. Its unknown is a sequence of graphs, read off positions through $\mathcal{N}$, in place of one fixed graph.

Privacy. A transmission tree records who passed a token to whom, across intervals. A swap does not preserve which ids one id meets at different times (Theorem 6.38(v)), so transmission trees cannot be read from swapped data. What swaps with a status do preserve are counts: the compartment counts per cell, the new infections and removals per cell, and with them the likelihood of a cell-by-cell chain binomial, at the swap cells and at every coarser partition (Corollary 6.41). So Corollary 6.41 bears on releasing epidemic counts with the identities hidden. It is not a method for transmission trees, which need the identities kept.

16.12 Halo-aware subtree sampling

Subtree partitioning (§10.3) samples the roots of the subtrees that become partitions, and Conjecture B.10 is about what the sample says of the cells a partition receives. The search of Proposition 6.44 adds a second cost to a cut: the *halo* of a partition, its occupied cells of depth j whose grid coordinate is extreme along some axis of the subtree's box, whose neighbours can lie in another partition. A ball or nearest-neighbour query at a halo cell needs the records of two partitions, so the mass in the halos is the part of the search that is not local to a partition. The direction: estimate the halo mass together with the weights, and choose subtree roots whose cuts carry little mass near the boundary, so that both the merge (Chapter 11) and the search stay inside a partition. For a node u and a depth j the halo is a set of labels (Proposition 6.44(ii) says which), so its mass is a sum over labels read from the summed field (Definition 6.32), with no geometry. Whether a sample of the labels estimates the halo mass as well as it estimates the weights is the question this adds to Conjecture B.10.

16.13 Adaptive approximation, and values beyond constants

Theorem 5.10 and Corollary 5.11 describe approximation on the complete pavings s_k: 2^k cells of one size, a rate $2^{-\alpha k/d}$ for Hölder-α functions, and a ceiling at $\alpha = 1$ for constant values. Two directions go past both limits, and both are already latent in the arithmetic of Chapter 6.

Adaptive pavings. With m leaves placed where the function varies, rather than $m = 2^k$ cells of one size, the functions approximable at a given rate form much larger classes. For piecewise constants on adaptive dyadic partitions the theory is that of nonlinear approximation: the functions approximable in L_p at rate $m^{-s/d}$ by m-leaf dyadic pavings are, up to the usual endpoint subtleties, the Besov classes $B^s_\tau(L_\tau)$ with $1/\tau = s/d + 1/p$, which for L_1 densities is $1/\tau = s/d + 1$; the greedy tree that refines where the local error is largest attains the rate. The references are DeVore's survey of nonlinear approximation and, for exactly these objects, the

piecewise-constant learning algorithm on adaptive dyadic partitions of Binev, Cohen, Dahmen, DeVore and Temlyakov (BIB TO ADD below; the theorem numbers are to be taken from the archived papers before either claim is cited in a statement). These classes contain functions whose singularities lie on lower-dimensional sets (§16.7), with a large s, which belong to no Hölder class; on them the complete paving is stuck at $m^{-1/d}$. The link to this monograph is the threshold paving s^* of §8.2: it refines where the empirical mass is largest, the greedy tree applied to the empirical measure, and the oracle inequality Theorem 9.1 says the selection of Chapter 9 pays a factor 3 over the best state of its path. The theorem to aim at is that the estimator attains the adaptive rate of the Besov class the density lies in, without being told the class, balanced against the estimation term of §10.4; what is open is whether the path of §10.4 contains a near-best m-leaf paving, since the threshold paving refines by count and the greedy tree by local error. For a start, Lemma 5.12 gives the rate $1/m$ on an interval for every density of bounded variation with no adaptivity at all.

Polynomial values. The saturation of Corollary 5.11(iii) is a limit of constant values. Let the value set be the polynomials $\mathbb{R}[x_1, \dots, x_d]$ in the coordinates of the root box, or those of total degree at most r with the product allowed to raise the degree. Proposition 5.3(i) applies verbatim: the pointwise sum, difference and product of two polynomial-valued MRPs on their common refinement are polynomial-valued MRPs (division leaves the ring, as it must), so Proposition 6.14 and its Spark form hold for this value ring unchanged. Copying a value down to a finer cell is literal when the polynomial is kept in the coordinates of the root box, and an affine change of variables when it is kept in the cell's own coordinates. Integrals over a cell are exact, monomial by monomial, so marginals (integrate out the dropped coordinates, a polynomial in the kept ones), slices (substitute the fixed coordinates), collation and MinimiseLeaves (equality of coefficient vectors) all survive. What does not survive as a union of cells is a level set: $\{p \geq c\}$ on a cell is a semi-algebraic set, so the highest density regions of §6.12 need either root-finding or an enclosure, and here the interval arithmetic of §5.5 returns, since the Bernstein coefficients of p on a box enclose its range and tighten as the box shrinks. For estimation, a piecewise-polynomial histogram is a candidate density like any other: the selection of Chapter 9 needs only whether a validation point lies in a Yatracos set $\{f_i > f_j\}$, a sign test of a polynomial at the point, so Theorem 9.1 carries over; and the uniform rate for C^{r+1} functions becomes $2^{-(r+1)k/d}$. An implementation is a new value type for the leaf map of Chapter 6: a sparse map from exponent tuples to coefficients, $\binom{d+r}{r}$ of them for total degree r, in cell-local coordinates scaled to $[0,1)^d$ for conditioning, with exact rational coefficients where exactness is wanted, and three operations the code at `c0a1837` lacks, re-centering on refinement, exact integration over a box, and evaluation at a point. Nothing else in the pipeline changes.

16.14 The selected leaf count as a complexity measure

Two numbers attach to a density through the dyadic Catalan trees alone, one of f and one of the run.

Definition 16.1 (Dyadic approximation numbers; the selected leaf count)**.** *For a density f on $\mathbf{x}_\rho$ and $m \geq 1$, let $\sigma_m(f) = \min\{\int |f_\pi - f| : \pi$ an RP of $\mathbf{x}_\rho$ with at most m leaves$\}$, with f_π the integral representation of Definition 12.3. For the final algorithm of Chapter 10 run on n points, with its parameters ($\overline{\#}_n$, φ, the backtrack rule) fixed in advance as functions of n, let $m_n(f) = |\mathcal{L}(\hat{\theta})|$ be the number of leaves of the selected state, a random variable, and $\kappa(f) = \limsup_n \log \mathbb{E}\, m_n(f) / \log n$ its growth exponent.*

$\sigma_m(f) \to 0$ for every density, by Proposition 5.8(i) on the complete pavings, and the speed is a smoothness statement and nothing else: Corollary 5.11 and Lemma 5.12 for the Hölder and bounded-variation classes, §16.13 for the Besov classes of adaptive pavings, and no rate at all

for the class of all densities [DG85]. $\sigma_m(f) = 0$ for some finite m exactly when f is itself an $\mathbb{R}$-MRP; then the nodes on which f is almost everywhere constant form a hereditary family whose maximal elements are disjoint and cover the box, so there is a unique coarsest paving π_f with f constant on its leaves, and $\sigma_m(f) = 0$ exactly for $m \geq |\mathcal{L}(\pi_f)|$. The uniform density has π_f the root paving, in every dimension. This is the sense in which the measure is stated without d: it is a count of leaves, the same count for the uniform density in every dimension; d enters through the regular scheme's cyclic cuts and the geometry of f, as the exponent κ below shows.

Lemma 16.2 (The integral representation has the largest entropy)**.** *Let f be a density on $\mathbf{x}_\rho$ with finite entropy $H(f) = -\int f \log f$ and π an RP. Then $H(f) = H(f_\pi) - D(f \,\|\, f_\pi)$, with D the relative entropy. So among the densities with the cell masses of f on the leaves of π, f_π has the largest entropy, and $f = f_\pi$ almost everywhere exactly when $H(f) = H(f_\pi)$.*

Proof. f_π is constant on each leaf l, with value $\mu(\mathbf{x}_l)/\mathrm{vol}(\mathbf{x}_l)$, so $\int f \log f_\pi = \sum_l \mu(\mathbf{x}_l) \log\big(\mu(\mathbf{x}_l)/\mathrm{vol}(\mathbf{x}_l)\big) = \int f_\pi \log f_\pi$. Hence $D(f\|f_\pi) = \int f \log f - \int f \log f_\pi = -H(f) + H(f_\pi)$, and $D \geq 0$ with equality exactly when $f = f_\pi$ almost everywhere. □

Proposition 16.3 (Braiding below a cell on which the density is constant)**.** *Let $X_1, X_2, \ldots$ be i.i.d. with density f, and let u be a node with $\mu(\mathbf{x}_u) > 0$ on which f is constant. - (i) Conditionally on $\#(u)$, the points of $X_1, \ldots, X_n$ in $\mathbf{x}_u$ are i.i.d. uniform on $\mathbf{x}_u$. - (ii) The splits that SEBTreeMC (Definition 8.1, uniform ties, no depth cap) makes below u, taken in order, are the run of SEBTreeMC on those points with $\mathbf{x}_u$ as root box, for as long as the global run makes them. - (iii) Fix M and $K = 2^M - 1$, and let the count limit $\overline{\#}_n$ and the leaf cap $\overline{m}_n$ of the run satisfy $\overline{\#}_n/n \to 0$ and $\overline{m}_n \to \infty$. As $n \to \infty$, the law of the split-queue state with $k \leq K$ splits below u converges in total variation to q_k; the merges of the backtrack by least count (§10.2, Phase 4) below u, taken in order from the first state with K splits below u, are the merge queue of Theorem 8.18(ii) on $\mathbf{x}_u$ with probability tending to one; and the two agree with probability tending to one: Corollary 8.22(i)–(ii) hold for the subtree below u.*

Proof. (i) Conditionally on which points fall in $\mathbf{x}_u$, each of them has the law of f restricted to $\mathbf{x}_u$ and normalised, which is uniform there; they are independent. (ii) The regular scheme is local to the box: the cuts of the descendants of u depend on the widths of $\mathbf{x}_u$ alone (Lemma 4.23(ii)), so they are the cuts of the regular scheme with $\mathbf{x}_u$ as root box. SEBTreeMC splits a leaf of largest count. Whenever the leaf it splits lies below u, that leaf has the largest count among the leaves below u, and between two such splits the leaves below u do not change; with uniform ties, a uniform choice among the global maxima restricted to the maxima below u is uniform among them. So the sequence of splits below u is the run on $\mathbf{x}_u$ with its own points and the same rule. (iii) *The split queue.* $\#(u)/n \to \mu(\mathbf{x}_u) > 0$ almost surely, so $\#(u) \to \infty$, and $\overline{\#}_n/n \to 0$ with $\overline{m}_n \to \infty$ let the global run make K splits below u for all large n, with probability tending to one (the event $\mathcal{E}_n$ of Corollary 8.22's proof, with a cap $o(n)$ in place of a fixed one). Conditionally on $\#(u) = N$, (i) and (ii) make the states below u those of SEBTreeMC on N uniform points of $\mathbf{x}_u$, whose laws converge to q_k as $N \to \infty$ by Corollary 8.22(i); integrating over $\#(u)$, which tends to infinity, gives the limit. *The merge queue.* The backtrack merges the cherry of least count; whenever that cherry lies below u it has the least count among the cherries below u, and between two such merges the cherries below u do not change, with uniform ties restricted as in (ii); so the merges below u, in order, are the merge queue on the subtree. The first state of the backtrack with K splits below u is the complete tree of depth M on $\mathbf{x}_u$ with probability tending to one: by the strong law every node of depth $|u| + M - 1$ below u holds about $\mu(\mathbf{x}_u)2^{1-M}n$ points and every node of depth at least $|u| + M$ at most about half as many, so the least-count queue merges every cherry whose children have depth greater than $|u| + M$ before any cherry whose children have depth $|u| + M$, and at the moment K internal nodes remain below u they

are exactly the nodes of depth less than $|u| + M$. From there the merges below u are the merge queue of Theorem 8.18(ii) from s^{c}_M, and Corollary 8.22(i)–(ii) apply to the subtree, conditionally on $\#(u)$ and then integrated. □

Marks. Proved here, from Corollary 8.22. [LEAN: to write, after the braiding theorems.] [SIM: the visit law of the split queue and of the backtrack below each leaf of π_f against q_k, for the three-leaf f of Example 16.4, $n = 10^4$ to 10^6, both backtrack rules; to run, L1MDE-PROOF-SIMULATION.]

Where f is not constant on $\mathbf{x}_u$, the two children of u have unequal masses unless their integrals happen to agree, the population queue splits the heavier child first, and there is no braiding at u: at a finite n a node looks braided only while the sampling noise, of order $\sqrt{\#(u)}$, masks the difference of its children's counts. So the nodes at which the split and merge paths braid are, in the limit, the leaves of π_f and everything below them, the max-entropy completion of Lemma 16.2; for a density that is no $\mathbb{R}$-MRP the braiding recedes to deeper cells as n grows, without end.

What the selected leaf count measures. For the uniform density the selection is expected to stay at the root (§12.5): $m_n = 1$. For a density smooth on the root box, the oracle balance of $\sigma_m \asymp m^{-1/d}$ (by Corollary 5.11, since $L_1 \le \mathrm{vol} \cdot \sup$ on a box, with its saturation at order one for the sharpness) against the deviation term of order $\sqrt{m/n}$ suggests $m_n \asymp n^{d/(d+2)}$, so $\kappa = d/(d+2)$; for a density smooth on a k-dimensional set, adaptive pavings should give $\kappa = k/(k+2)$, the information dimension of the law rather than the ambient dimension, which the regular scheme cannot see since its cyclic cuts spend splits on the coordinates where f is uniform. None of this is proved: Theorem 10.4 bounds the error of the selected state by that of the best state on the path, and says nothing about the number of leaves it selects. One thing is settled, and it changes what the measure measures.

Example 16.4 (The least-count backtrack inflates the leaf count)**.** Let f on $[0,1]$ take the values 0.08, 0.12 and 1.9 on $[0, \frac14)$, $[\frac14, \frac12)$ and $[\frac12, 1)$, with masses 0.02, 0.03 and 0.95, so π_f has three leaves. In the backtrack by least count (§10.2, Phase 4), a cherry inside $[\frac12, 1)$ whose two children have depth $j \ge 2$ holds about $0.95 \cdot 2^{2-j} n$ points, while the cherry $[0, \frac12)$, whose two children are the first two leaves of π_f, holds about $0.05n$. The first exceeds the second exactly for $j \le 6$ ($0.95/16 = 0.059$ against $0.95/32 = 0.030$). So, with probability tending to one, the path merges $[0, \frac14)$ with $[\frac14, \frac12)$ while $[\frac12, 1)$ is still cut into its 32 cells of depth 6: no state on the path has the leaves of π_f. The states on the path are either refinements of π_f with at least 34 leaves, whose error tends to 0, or later states, each containing the cell $[0, \frac12)$ or a coarsening of it (33 leaves just after the cross merge), whose error is at least $|0.02 - 0.03| = 0.01$ for every n. The selection takes the former once n is large: $m_n(f) \ge 34$ with probability tending to one, the best state on the path having 34 leaves, not 3.

The inflation is the path's, not the trees': the least-count rule merges where the points are few, not where they are uniform. The rule that follows the braiding merges the cherry of least $|a - b|$, the children's counts, which is the arc length of the step (Lemma 10.3(i)): inside a leaf of π_f the two children have counts that differ by $O(\sqrt{\#})$, across two leaves of different densities by $\Theta(n)$, so every internal merge precedes every cross merge for large n, the path passes through π_f, and $m_n(f) \to |\mathcal{L}(\pi_f)|$. This rule gives a chain under a total tie-break, so (T) and Theorem 10.4 hold for it as for the least-count rule; it is not monotone in the sense of Proposition 6.51, so its states are not threshold pavings, and the exactness of its distributed backtrack is to be shown as Lemma 11.9 shows it for the count. That the arc-length backtrack reaches π_f for every $\mathbb{R}$-MRP f with probability tending to one is Theorem 16.7 below. Open: the growth of m_n under each rule for smooth f; and a shape statistic for uniformity on a cell from the law q_k of Theorem 8.18, in the spirit of the tree-shape counts of Chapter 4. [SIM: run for the uniform density and the three-leaf

f above, both rules, $n = 10^3$ to 10^6, as the leaves of the path's L_1-best state (T4, `ias.spark` run `t4-all`; the witnesses in `designs/research/l1mde-simulation-witnesses.md`): the uniform density stays at the root under both rules; for the three-leaf f the least-count path never holds π_f, its best state being coarser at $n \le 10^5$ (2 leaves, error at least 0.01, the example's merged $[0, \frac{1}{2})$) and finer at 10^6 (43 leaves, against the bound of 34), while the arc-length path's best state is π_f from 10^5, ten times more accurate at 10^6. The selected m_n (with the gap search, T5) and the Gaussian mixtures in 2 and 10 dimensions, whose expected log–log slopes are near $\frac{1}{2}$ and $\frac{5}{6}$ under the balance above, are still to run, L1MDE-PROOF-SIMULATION, reported in §14.1's protocol.]

The first term of the oracle inequality. The approximation numbers and the selected leaf count meet in the first term of Theorem 9.1, $3 \min_\theta \int |f_{n',\theta} - f|$, the distance from f to the best *empirical* candidate. At a state with m leaves that distance is the approximation number at m plus the sampling noise on the state's cells, and nothing else.

Proposition 16.5 (The first term of the oracle inequality at a state)**.** *Let θ be a paving of $\mathbf{x}_\rho$ with m leaves, f a density on $\mathbf{x}_\rho$, f_θ its integral representation on θ (Definition 12.3), and $f_{n',\theta}$ the histogram of n' i.i.d. points of f on θ, with $\mu_{n'}$ their empirical measure. - (i) $\frac{1}{2}\int |f_\theta - f| \le \int |f_{n',\theta} - f| \le \int |f_\theta - f| + \sum_{l \in \mathcal{L}(\theta)} |\mu_{n'}(\mathbf{x}_l) - \mu(\mathbf{x}_l)|$, for every sample; and for a paving fixed before the sample is seen, $\mathbb{E}\sum_l |\mu_{n'}(\mathbf{x}_l) - \mu(\mathbf{x}_l)| \le \sqrt{m/n'}$. - (ii) So $\frac{1}{2}\sigma_m(f) \le \int |f_{n',\theta} - f|$ for every state with m leaves, fixed or chosen from the data, and the first term of Theorem 9.1 over any set of states is at least $\frac{3}{2}\sigma_m(f)$ with m the largest number of leaves among them. For the path P of the final algorithm, whose states are data-driven, $\min_{\theta \in P} \int |f_{n',\theta} - f| \le \min_{\theta \in P} \big(\int |f_\theta - f| + 2 \sup_{B \in \mathcal{B}_{|\mathcal{L}(\theta)|}} |\mu_{n'}(B) - \mu(B)| \big)$, with the supremum over unions of boxes bounded by Lemma 8.12(ii). - (iii) Splitting a leaf l of θ into its two children lowers $\int |f_\theta - f|$ by $\int_{\mathbf{x}_l} |f_\theta - f| - \int_{\mathbf{x}_l} |f_{\theta'} - f| \le \operatorname{osc}(f, \mathbf{x}_l) \operatorname{vol}(\mathbf{x}_l)$, and by nothing when f is constant on $\mathbf{x}_l$; it raises the sum in (i) by at most $2\min(|\mu_{n'}(\mathbf{x}_{l_1}) - \mu(\mathbf{x}_{l_1})|, |\mu_{n'}(\mathbf{x}_{l_2}) - \mu(\mathbf{x}_{l_2})|)$, of expectation at most $2\sqrt{\mu(\mathbf{x}_l)/n'}$ for a fixed θ.*

Proof. (i) $f_{n',\theta} - f_\theta$ is constant on each leaf, with integral $\mu_{n'}(\mathbf{x}_l) - \mu(\mathbf{x}_l)$ over $\mathbf{x}_l$, so $\int |f_{n',\theta} - f_\theta|$ is the sum, and the upper bound is the triangle inequality. For the lower bound, the first inequality of Lemma 12.4(ii), whose proof is cell by cell and holds on any paving, gives $\int |g - f_\theta| \le \int |g - f|$ for every g constant on the leaves of θ; with $g = f_{n',\theta}$, $\int |f_\theta - f| \le \int |f_\theta - g| + \int |g - f| \le 2\int |g - f|$. For a fixed paving, $n'\mu_{n'}(\mathbf{x}_l)$ is binomial with mean $n'\mu(\mathbf{x}_l)$, so $\mathbb{E}|\mu_{n'}(\mathbf{x}_l) - \mu(\mathbf{x}_l)| \le \sqrt{\mu(\mathbf{x}_l)(1 - \mu(\mathbf{x}_l))/n'}$, and $\sum_l \sqrt{\mu(\mathbf{x}_l)/n'} \le \sqrt{m \sum_l \mu(\mathbf{x}_l)/n'} \le \sqrt{m/n'}$ by the Cauchy–Schwarz inequality. (ii) $\sigma_m(f) \le \int |f_\theta - f|$ by definition, and the lower bound of (i) is deterministic. For the path, the leaves of a state are at most $|\mathcal{L}(\theta)|$ boxes, so Lemma 8.12(i) bounds the sum in (i) by twice the supremum over $\mathcal{B}_{|\mathcal{L}(\theta)|}$, which does not depend on the state. (iii) Outside $\mathbf{x}_l$ nothing changes. On $\mathbf{x}_l$ the new representation is the cell averages on the two children, and $\int_{\mathbf{x}_l} |f_{\theta'} - f| \ge 0$, while $\int_{\mathbf{x}_l} |f_\theta - f| \le \operatorname{osc}(f, \mathbf{x}_l)\operatorname{vol}(\mathbf{x}_l)$ since the average lies between the infimum and the supremum; when f is constant on $\mathbf{x}_l$ both integrals are 0. In the sum, $|a_1| + |a_2|$ replaces $|a_1 + a_2|$ with $a_i = \mu_{n'}(\mathbf{x}_{l_i}) - \mu(\mathbf{x}_{l_i})$, and $|a_1| + |a_2| - |a_1 + a_2| \le 2\min(|a_1|, |a_2|)$; the expectation bound is that of (i) for one child. □

Marks. Proved here, from Lemma 12.4(ii), Lemma 8.12 and the Cauchy–Schwarz inequality. [LEAN: to write, on `ResolutionD` and `VC.Boxes`.] [SIM: the sandwich (i) on the three-leaf f of Example 16.4 and on $f(x) = 2x$, every state of the path, $n' = 10^3$ to 10^6; to run, L1MDE-PROOF-SIMULATION.]

Read with Theorem 10.4, whose bound is $3(1 + \xi) \min_{\theta \in P} \int |f_{n',\theta} - f|$ plus terms that do not depend on f, the proposition says three things. The error of the final algorithm is, within constants, $\min_{\theta \in P} \big(\int |f_\theta - f| + \text{noise on } \theta \big)$: the bias–variance balance along the path, whose

minimiser is the selected leaf count $m_n(f)$ up to the factor the oracle inequality loses. A split pays only while $\operatorname{osc}(f, \mathbf{x}_l)\operatorname{vol}(\mathbf{x}_l)$ exceeds the noise it adds, of order $\sqrt{\mu(\mathbf{x}_l)/n'}$; on a cell where f is constant it never pays, at any n, which is the braiding of Proposition 16.3 seen from the inequality, while on a cell where f varies the criterion keeps splitting as n grows, at the resolution where the children's counts cease to look like a fair coin. And the lower bound is in σ_m while the upper bound is in the path's own $\min_{\theta\in P}\int|f_\theta - f| \geq \sigma_{|\mathcal{L}(\theta)|}(f)$: the gap between the two is the backtrack rule's defect of Example 16.4, which the arc-length rule closes for the $\mathbb{R}$-MRP densities.

The Catalan skeleton. The second way to use the approximation numbers is the skeleton estimate of [DL01, ch. 7]: a finite ϵ-net of a class, with the minimum distance estimate run over the net. The Catalan trees give the net for free, with its size and its description length.

Proposition 16.6 (The Catalan skeleton)**.** *Let $\mathcal{F}_m$ be the densities on $\mathbf{x}_\rho$ that are $\mathbb{R}$-MRPs on regular pavings with at most m leaves, and $0 < \epsilon \leq 1$. - (i) $\mathcal{F}_m$ has an ϵ-net in L_1 of at most $N_m(\epsilon) = C_{m-1}\,(2m/\epsilon + 1)^{m-1}$ densities of $\mathcal{F}_m$, so $\log N_m(\epsilon) \leq (m-1)\big(\log 4 + \log(2m/\epsilon + 1)\big)$; every density f lies within $\sigma_m(f) + \epsilon$ of the net. - (ii) An element of the net is coded in $L_m(\epsilon) \leq 3m + (m-1)\log_2(2m/\epsilon + 1)$ bits, the tree in $2(m-1)$ and the masses in the rest. The* Catalan complexity *of f at precision ϵ, $K_\epsilon(f) = \min\{m : \sigma_m(f) \leq \epsilon\}$, is finite for every density and every $\epsilon > 0$, and f is described to within 2ϵ in L_1 by a net element of $L_{K_\epsilon(f)}(\epsilon)$ bits. - (iii) Let $\hat{f}$ be the minimum distance estimate, with the φn validation points, over the union of the nets of $\mathcal{F}_1, \ldots, \mathcal{F}_M$, of $N \leq \sum_{m\leq M} N_m(\epsilon)$ candidates. For every density f and $\delta \in (0,1)$, with probability at least $1-\delta$,*

$$\int |\hat{f} - f| \;\leq\; 3 \min_{m\leq M} \sigma_m(f) + 3\epsilon + 4\sqrt{\frac{\log(2N^2/\delta)}{2\varphi n}}.$$

Proof. (i) The regular pavings with m leaves are the Catalan trees with m leaves realised on the box, C_{m-1} of them (Proposition 4.33). For one paving, a density of $\mathcal{F}_m$ is its vector of cell masses $p \in \Delta_m$, and two such densities on the same paving are at L_1 distance $\sum_l |p_l - p'_l|$. Round each p_l down to a multiple of $\eta = \epsilon/(2m)$; the rounded vector is within $m\eta = \epsilon/2$ of p and its deficit from 1 is at most $\epsilon/2$ and a multiple of η; add the deficit to one coordinate. The result is a probability vector on the grid $\eta\mathbb{Z}^m$ within ϵ of p, and such vectors are determined by $m-1$ coordinates in $\{0, \eta, \ldots, 1\}$, at most $(2m/\epsilon + 1)^{m-1}$ of them. $C_{m-1} \leq 4^{m-1}$. The last claim is the triangle inequality with a paving attaining $\sigma_m(f)$, a minimum over the finitely many pavings with at most m leaves, whose integral representation f_π lies in $\mathcal{F}_m$. (ii) A Catalan tree with m leaves is a word of $2(m-1)$ parentheses; the grid vector takes $(m-1)\lceil\log_2(2m/\epsilon + 1)\rceil$ bits. $K_\epsilon(f) < \infty$ by Proposition 5.8(i), and the description is (i) at $m = K_\epsilon(f)$. (iii) Theorem 9.1 over the N candidates, with min over the candidates at most $\min_{m\leq M}\sigma_m(f) + \epsilon$ by (i); the Yatracos class of N candidates has at most N^2 sets, and Hoeffding's inequality with the union bound gives $\Delta \leq \sqrt{\log(2N^2/\delta)/(2\varphi n)}$ with probability at least $1-\delta$, as in Proposition 10.1. □

Marks. Proved here; (iii) is the skeleton estimate of [DL01, ch. 7] on this dictionary. [LEAN: to write, on `OracleInequality.oracle_finite` with the count of (i).] [SIM: $N_m(\epsilon)$ against the actual net; the bound of (iii) on the three-leaf f with $M = 4$, $\epsilon = 0.05$, $n = 10^4$; to run, L1MDE-PROOF-SIMULATION.]

With $\epsilon = 1/n$ and M chosen to balance the two terms, a density with $\sigma_m(f) \leq m^{-s}$ is estimated at the rate $(n/\log n)^{-s/(2s+1)}$ by the skeleton, since $\log N \leq M\big(3 + \log(2Mn + 1)\big)$: the rate of nonlinear approximation, which the class of all densities does not have ([DG85]; $K_\epsilon(f)$ can grow as fast as one likes as $\epsilon \to 0$). The final algorithm does not run over the skeleton: its path has at most $n+1$ states, so its deviation term is $\sqrt{\log n/n}$ with no M in it, and its price is the path's $\min_{\theta\in P}\int|f_\theta - f|$ in place of $\sigma_m(f)$, by Proposition 16.5(ii). The path is the computable

skeleton: the data choose $n+1$ of the $\sum_m N_m(\epsilon)$ net points, and the rule that chooses them, the backtrack, decides how much of σ_m it keeps. The code length $L_m(\epsilon)$ is a two-part description of a density to precision ϵ, the tree and then the masses, in the sense of the minimum description length principle; it is relative to this dictionary, as any computable complexity must be, where the Kolmogorov complexity it stands in for is universal and not computable. The union over m of the ϵ-balls around the net of $\mathcal{F}_m$ is the class $\{f : K_\epsilon(f) \le m\}$, and these classes exhaust the densities as $m \to \infty$ at every ϵ; the skeleton of trees and the continuum of densities over it are the same object seen at two resolutions. Open: a skeleton weighted by $2^{-L_m(\epsilon)}$, whose deviation term for the chosen candidate would be of order $\sqrt{L_m(\epsilon)/n}$, the code length of the answer rather than the logarithm of the size of the net, which is a question here and not a theorem; and, beyond the $\mathbb{R}$-MRP densities, whose paths under the arc-length rule pass through π_f (Theorem 16.7), whether that rule makes the path's minimum attain σ_m at the noise level ϵ_n of the run, which would make $m_n(f)$ the data's estimate of $K_{\epsilon_n}(f)$.

The arc-length backtrack reaches the finite-complexity densities. The gap between the two bounds of Proposition 16.5(ii) closes, for the $\mathbb{R}$-MRP densities, under the backtrack rule that follows the braiding. The statement needs no condition beyond f being an $\mathbb{R}$-MRP, and Example 16.4 shows that the least-count rule does not have the property.

Theorem 16.7 (The arc-length path contains the coarsest constant paving)**.** *Let f be a density on $\mathbf{x}_\rho$ that is an $\mathbb{R}$-MRP, with π_f its coarsest constant paving and $m = |\mathcal{L}(\pi_f)|$. Run the final algorithm of Chapter 10 on n i.i.d. points of f with a count limit $\overline{\#}_n$ such that $\overline{\#}_n/n' \to 0$ and $\overline{\#}_n \ge 1$, with (G) holding at its depth, and with Phase 4's backtrack merging at each step the cherry of least $|a-b|$, the difference of its children's training counts, ties broken by the heap code. Then: - (i) with probability tending to one, the merged histogram s^* refines π_f, every merge of a cherry inside a leaf of π_f precedes every merge of a cherry whose children are distinct leaves of π_f, and the path P passes through π_f; - (ii) on that event, $\min_{\theta\in P}\int|f_{n',\theta}-f| \le \int|f_{n',\pi_f}-f| = \sum_{l\in\mathcal{L}(\pi_f)}|\mu_{n'}(\mathbf{x}_l)-\mu(\mathbf{x}_l)|$, of expectation at most $\sqrt{m/n'}$ (Proposition 16.5(i)), so by Theorem 10.4 the estimate satisfies, on $\mathcal{E}$,*

$$\int |f_{n',\hat\theta} - f| \;\le\; 3(1+\xi)\sum_{l\in\mathcal{L}(\pi_f)} |\mu_{n'}(\mathbf{x}_l) - \mu(\mathbf{x}_l)| + 3\epsilon_0 + 4\overline{\Delta}\,:$$

the finite-complexity densities are estimated at the parametric rate in their number of leaves, up to the constants of the search.

Proof. The rule is a deterministic function of the training counts with a total tie-break, so P is a chain fixed by the training points, (T) holds, and Theorem 10.4 applies with this P; the backtrack runs on the driver over the collected s^* as before, so nothing in Chapter 11 changes.

s^* refines π_f. A leaf of π_f has positive mass, or is a zero-mass cell ($f = 0$ on it); such a leaf holds no point, is never split, and is an empty leaf of s^*, which the refinement below includes. For a leaf l of positive mass, $\#(l)/n' \to \mu(\mathbf{x}_l) > 0$ almost surely while $\overline{\#}_n/n' \to 0$, so $\#(l) > \overline{\#}_n$ for all large n, and every strict ancestor of l, holding at least as many points, is internal in s^* (Lemma 8.3(ii), under (G)), provided the run's depth exceeds the depth of every leaf of π_f: true for all large n when the depth grows (Corollary 8.15), and an assumption at a fixed depth (it holds in the runs of the [SIM] mark, whose π_f have depth at most 3 and $D = 20$). Every internal node of π_f is a strict ancestor of a leaf of positive mass. So, with probability tending to one, every internal node of π_f is internal in s^*: s^* refines π_f, and every leaf of s^* lies inside a leaf of π_f.

Internal cherries are nearly balanced. Call a node u *internal to* π_f when it lies inside a leaf l of π_f, the leaf itself included; then f is constant on $\mathbf{x}_u$, so $\mu(\mathbf{x}_{u\mathsf{L}}) = \mu(\mathbf{x}_{u\mathsf{R}})$. Conditionally on the points

in $\mathbf{x}_u$, they are i.i.d. uniform there (Proposition 16.3(i)), so the count a of $u\mathsf{L}$ is binomial $(\#(u), \frac{1}{2})$ and $a-b = 2a-\#(u)$; by Hoeffding's inequality $\mathbb{P}\{|a-b| > t \mid \#(u)\} \le 2e^{-t^2/(2\#(u))} \le 2e^{-t^2/(2n')}$. Let $\overline{D}_n$ be a deterministic bound on the depth of the run: the representable depth, or the d_n of Corollary 8.15. The cherries that can arise on P are internal nodes of s^* of depth less than $\overline{D}_n$, and one that holds no point has $a = b = 0$. For a fixed node u, the event that u holds a point and $|a-b| > t$ has probability at most $2e^{-t^2/(2n')}\,\mathbb{P}\{\#(u) \ge 1\}$, and at each depth h, $\sum_{|u|=h} \mathbb{P}\{\#(u) \ge 1\} \le \sum_{|u|=h} \mathbb{E}\,\#(u) = n'$. So the expected number of internal nodes of depth less than $\overline{D}_n$ that hold a point and have $|a-b| > t$ is at most $2e^{-t^2/(2n')}\,\overline{D}_n n'$, which is $1/n'$ at $t_n = \sqrt{2n' \log(2\overline{D}_n n'^2)}$, and by Markov's inequality there is none with probability at least $1 - 1/n'$. And $t_n = o(n')$, since $\overline{D}_n$ is fixed or grows like $\log n'$ (Corollary 8.15(iii)).

Cross cherries are unbalanced. A cherry of a state on P whose node u is not internal to π_f and is not a leaf of π_f contains two leaves of π_f; the first such cherry to be merged has both children leaves of the current state that are not internal to π_f, hence (no cross merge having happened yet) unions of leaves of π_f that are themselves leaves of π_f: so u is an internal node of π_f with both children leaves of π_f. For every such u, $\mu(\mathbf{x}_{u\mathsf{L}}) \ne \mu(\mathbf{x}_{u\mathsf{R}})$: the two children are halves of $\mathbf{x}_u$ of equal volume (Definition 4.22), with f constant on each, so equal masses would give equal constants, f would be almost everywhere constant on $\mathbf{x}_u$, and u would lie in the hereditary family whose maximal elements are the leaves of π_f, against u internal (0 against 0 is the case $f = 0$ on $\mathbf{x}_u$). So $\gamma = \min_u |\mu(\mathbf{x}_{u\mathsf{L}}) - \mu(\mathbf{x}_{u\mathsf{R}})| > 0$ over the finitely many such u. By the strong law, $|a-b| \ge \gamma n'/2$ for all of them, for all large n, with probability one.

The order. On the intersection of the three events, which has probability tending to one, every available internal cherry has $|a-b| \le t_n = o(n')$ while every available cross cherry has $|a-b| \ge \gamma n'/2$, so the rule merges internal cherries as long as any exists. A merge inside a leaf l of π_f produces a node inside l or l itself, never a cross cherry, so when the last internal cherry is merged every leaf of π_f has been assembled and nothing else has: the state is π_f. This is (i). (ii) is Proposition 16.5(i) at $\theta = \pi_f$, where $f_{\pi_f} = f$, and Theorem 10.4. □

Marks. Proved here. The first cross merge cannot be tied: that is where the regular scheme's equal-volume siblings enter. A scheme whose siblings carry unequal reference measure, such as the S2 cells of §16.5 (the cylindrical paving and HEALPix are equal-area and keep the argument), needs the condition $\mu(\mathbf{x}_{u\mathsf{L}}) \ne \mu(\mathbf{x}_{u\mathsf{R}})$ for the internal nodes of π_f with two leaf children as a hypothesis. A tie at a later cross merge, between a leaf of π_f and a union of leaves (on $[0,1)$, density c on $[0,\frac{1}{2})$ and $\frac{3}{2}c$, $\frac{1}{2}c$ on the quarters of $[\frac{1}{2},1)$, so the two halves have equal mass), comes after the path has passed π_f and does not touch (i). [LEAN: (i), the order, built at `519e23794` in `ProveDog.L1MDE.ArcPath` (`thm_arcPath`: every run of the arc-length backtrack, under every tie-break, passes through π_f, every state before it refining π_f and π_f refining every state after it; `thm_arcPath_order`: step i merges inside a leaf of π_f exactly when it comes before π_f; with `arcStep_internal`, `arcStep_subset`, and `notMem_iff_below_leaf`, `child_mem_leavesOf_iff` for the event's encoding on the split sets of `Exact`), on the event of the three steps above as a hypothesis (`ArcEvent`, one t; its probability is the text's); the step "s^* refines π_f" at the algorithm's own s^*, `subset_internalSet`, `arcEvent_internalSet`, `thm_arcPath_internalSet` (with every internal node of π_f above a leaf of positive mass, and the strong-law event $\#(l) > \overline{\#}$, as hypotheses), a run exists and passes through π_f, `arcRun_exists`, `thm_arcPath_exists`, and the event is satisfiable, `arcEvent_satisfiable` — built at `050df0219`. (ii) only as Theorem 10.4 evaluated at the index of π_f on the run (`thm_arcPath_certified` on `CertifiedSearch.thm_certified`), the error of a state an abstract function of its index as in `CertifiedSearch`: the identity $\int |f_{n',\pi_f} - f| = \sum_l |\mu_{n'}(\mathbf{x}_l) - \mu(\mathbf{x}_l)|$, its expectation bound, and (T), that the path is a function of the training counts, are argued here and not formalised.] [SIM: T4 on the cluster (`ias.spark` run `t4-all`, seed 20261111, $D = 20$, $\overline{\#} = \lceil\sqrt{n}\rceil$, one sample per cell; the study in `designs/research/l1mde-t4-path-study-2026-11-11.md`,

its 32 witnesses in `designs/research/l1mde-simulation-witnesses.md`, none failed): the three-leaf f of Example 16.4, the uniform density, a five-piece step density and the later-cross tie above, $n = 10^3$ to 10^6, both rules. Under the arc-length rule π_f is on the path in every non-uniform cell from $n = 10^5$, from 10^3 for the five-piece and later-tie densities, and its error is within $\sqrt{m/n'}$ wherever it is (three-leaf at 10^6: 5.7×10^{-4} against 1.7×10^{-3}); the onset is near $n' \approx \mu/\gamma^2$ (γ the least mass difference of two sibling leaves of π_f, μ the mass of the largest competing cell: about 9 500 for the three-leaf f, which the grid passes between 10^4 and 10^5), a heuristic the one-sample grid is consistent with, not a proof. The later tie does not keep the path from π_f. Over 20 replicates per cell (`t4-reps`, the three non-uniform densities at $n = 3 \cdot 10^3$, 10^4, $3 \cdot 10^4$, 10^5; 24 witnesses, 480 runs), the share of runs with π_f on the arc-length path rises through the onset for the three-leaf f, 0.15, 0.50, 0.95, 0.95, its midpoint at $n' \approx 9\,500$, and is 1 for the other two from $3 \cdot 10^3$; on the least-count path it is 0 in all 160 runs of the three-leaf and five-piece densities; the arc-length rule's mean error is the smaller in every cell, by a factor 1.05 to 5.5. The Haskell reference (`spark/l1mde-reference/`, its own SplitMix64 sampler and exact rational errors, `ias.spark` run `hs-ref-t4`, 360 runs at $n = 3 \cdot 10^3$ to $3 \cdot 10^4$) reproduces the law: the three-leaf share 0.15, 0.45, 0.90, the least-count share 0, the other two densities 1 under the arc-length rule, and the uniform density ending at the root with error exactly 0 under both rules. In $d = 2$ (`t4-2d`, a three-leaf and a five-leaf step density of the square under the first-widest scheme, 20 runs per cell at $n = 3 \cdot 10^3$ to 10^5) the arc-length path holds π_f in all 160 runs; the least-count rule's behaviour there is Proposition 16.8's.]

Two things follow and one is open. The theorem is the $\kappa = 0$ case of the growth exponent: under the arc-length rule the path of every $\mathbb{R}$-MRP density passes through π_f, a state with $|\mathcal{L}(\pi_f)|$ leaves whose error is the sampling noise on them alone, and Example 16.4 shows the least-count rule has no such property. The parametric rate holds for the gap search as specified except for its backtrack rule, which the final algorithm can offer as an option at no cost in exactness. Open: whether the rule gives the balance $m_n \asymp n^{d/(d+2)}$ for smooth densities.

The least-count rule is not always wrong; when it reaches π_f is a property of the density's geometry, not of n.

Proposition 16.8 (When the least-count path holds the coarsest constant paving)**.** *Let f be an $\mathbb{R}$-MRP whose π_f is not the root, $m^* = \max\{\mu(\mathbf{x}_l) : l \in \mathcal{L}(\pi_f)\}$ and $m_* = \min\{\mu(\mathbf{x}_u) : u \in I(\pi_f)\}$. Run the final algorithm as in Theorem 16.7, with Phase 4's backtrack merging the cherry of least count. - (i) If $m^* < m_*$, then with probability tending to one the path P passes through π_f. - (ii) If $m^* > m_*$, then with probability tending to one no state of P is π_f. - (iii) If $m^* = m_* = m$, let T be the set of pairs (l, u) of a leaf l and an internal node u of π_f, both of mass m, with l not below u, and let (Z_v) be the centred Gaussian vector over the leaves and internal nodes of π_f with $\mathrm{Cov}(Z_v, Z_w) = \mu(\mathbf{x}_v \cap \mathbf{x}_w) - \mu(\mathbf{x}_v)\mu(\mathbf{x}_w)$. Then $\mathbb{P}\{\pi_f$ is a state of $P\} \to \mathbb{P}\{Z_l < Z_u$ for every $(l, u) \in T\}$: the limit is 1 when T is empty and $\frac{1}{2}$ when T has one pair.*

The minimum m_* is attained on a cherry of π_f (descend from any internal node into an internal child, which has no more mass), so (i) reads: every leaf of π_f is lighter than every cherry of π_f.

Proof. With probability tending to one, s^* refines π_f (the first step of the proof of Theorem 16.7, which does not use the rule), every leaf of π_f of positive mass is internal in s^*, and by the strong law, over the finitely many nodes involved, $\#(l)/n' \to \mu(\mathbf{x}_l)$ for the leaves l of π_f and $\#(u)/n' \to \mu(\mathbf{x}_u)$ for $u \in I(\pi_f)$. The merged nodes have non-decreasing counts (the "Counts" step of the proof of Lemma 11.9(i), which uses the rule alone and none of that lemma's caps), so a node of strictly smaller count is merged before one of strictly larger count. - *(i).* Every internal node of s^* lies inside a leaf of π_f or is in $I(\pi_f)$. A node v inside a leaf l has $\#(v) \le \#(l)$, and for large n, $\#(l) < \#(u)$ for every leaf l and every $u \in I(\pi_f)$, since $m^* < m_*$. So every node inside

a leaf of π_f is merged before every node of $I(\pi_f)$; when the last of them is merged, the internal nodes of the state are exactly $I(\pi_f)$, and the state is π_f. - *(ii).* Take a leaf l with $\mu(\mathbf{x}_l) = m^*$ and $u \in I(\pi_f)$ with $\mu(\mathbf{x}_u) = m_*$; l has positive mass, and l is not below u, since l below u would give $\mu(\mathbf{x}_l) \le \mu(\mathbf{x}_u)$. For large n, $\#(u) < \#(l)$, so u is merged before l. A state equal to π_f has u internal and l a leaf, that is, it comes after the merge of l and before the merge of u: there is none. - *(iii).* The pairs of a leaf l and an internal node u with $\mu(\mathbf{x}_l) < \mu(\mathbf{x}_u)$ separate as in (i): $\#(l) < \#(u)$ for large n. A pair in which l lies below u with equal mass has $\#(u) = \#(l)$ almost surely (the rest of $\mathbf{x}_u$ has mass 0), and never blocks: u is merged after everything below it. The remaining pairs are those of T, whose cells are disjoint. By the multivariate central limit theorem over the finitely many cells, $\big((\#(v) - n'\mu(\mathbf{x}_v))/\sqrt{n'}\big)_v$ converges in law to (Z_v), and for $(l,u) \in T$, $\mathrm{Var}(Z_u - Z_l) = 2m(1-m) + 2m^2 = 2m > 0$, so no event $\{\#(l) = \#(u)\}$ keeps a positive probability. On the event that the strict pairs have separated, the argument of (i) shows that π_f is a state of P when $\#(l) < \#(u)$ for every $(l,u) \in T$, and that of (ii) that it is not when $\#(l) > \#(u)$ for some pair. The set $\{z : z_u > z_l \text{ for every } (l,u) \in T\}$ has a boundary of probability 0 under the law of (Z_v), so by the portmanteau theorem its probability is the limit. With T empty the event is sure; with one pair it is $\{Z_u - Z_l > 0\}$, of probability $\frac{1}{2}$ by symmetry. □

The limit can be 0 at a tie: with mass $\frac{1}{2}$ on each of $[0, \frac{1}{4})$ and $[\frac{1}{2}, \frac{3}{4})$ and 0 elsewhere, T pairs each of the two leaves with the other half of $[0,1)$, whose count equals the other leaf's almost surely, and the two conditions contradict each other.

All three cases are seen in the examples. Example 16.4 is case (ii): $m^* = 0.95 > 0.05 = m_*$, the cell $[0, \frac{1}{2})$. The density 1 on the left half of $[0,1)$ and $\frac{3}{2}$, $\frac{1}{2}$ on the quarters of the right half has $m^* = m_* = \frac{1}{2}$, a tie with one pair in T (the leaf $[0, \frac{1}{2})$ and the internal node $[\frac{1}{2}, 1)$), so case (iii) gives $\frac{1}{2}$. The density 2 on $[0, \frac{1}{2})$ and 0 on $[\frac{1}{2}, 1)$ is a tie with T empty (its leaf of mass 1 lies below the root), so the least-count path holds π_f with probability tending to one. The arc-length rule needs no such condition (Theorem 16.7): it reaches π_f for every $\mathbb{R}$-MRP once n is large, the onset, near $n' \approx \mu/\gamma^2$, depending on f.

Marks. Proved here, new; (i)–(ii) from A1_0-0's reading of the two-dimensional T4 runs, (iii) from its proposal in review. [LEAN: to write, (i) and (ii), with `ArcPath`, on `CertifiedSearch` and the merge order of Lem-Backtrack (i) (`Exact`); (iii) open, since it needs the multivariate central limit theorem, which Mathlib has only in part.] [SIM: the T4 witnesses in `designs/research/l1mde-simulation-witnesses.md`. Case (i): the two-dimensional three-leaf density (the left half 0.3, the right half cut in y into 0.5 and 0.2; $m^* = 0.5 < 0.7 = m_*$), π_f on the least-count path in all 80 runs at $n = 3 \cdot 10^3$ to 10^5. Case (ii): the three-leaf and five-piece densities of $[0,1)$, π_f on the least-count path in none of 160 replicate runs. Case (iii) with one pair in T: the later-cross tie in $d = 1$ and a five-leaf density of the square ($m^* = m_* = 0.3$), π_f on the least-count path in 41 and 35 of 80 runs, against the limit $\frac{1}{2}$, at every n with no trend. The Haskell reference (`ias.spark` run `hs-ref-t4-2d`), which also computes each density's case exactly, agrees with its own sampler: case (ii) in none of 120 runs, case (i) in all 60, the ties in 23 and 34 of 60 (64 and 69 of 140 pooled with the Spark runs).] * * *

Appendix A

Algorithms: index and pseudocode

Every algorithm of Chapters 3 to 14, in one place. Each has up to three forms: - **MRS2**: recursive descent on pointer trees, in C++; - **sparse**: maps over leaf labels (Chapters 4 to 6); - **Spark**: as coded at `c0a1837`, where that code differs from the sparse form.

The table records where each form is stated today. The pseudocode of the algorithms of Chapter 3, Chapter 7, Chapter 9 and Chapter 10 follows it (A.1–A.10). The paving arithmetic of Chapters 4 to 6 is stated in the body beside its results (A.11), and A.12–A.14 give SEBTreeMC, the merge walk $W(\delta)$ and the (G)-depth.

Algorithm	§	MRS2 form	Sparse form	Spark form
Moore rejection sampler (TRS, Alg. 1) and its refinement queue by vol × wid	2	[SY13, Alg. 1]; MRS	—	[AS18] (no code cited)
Moore–Metropolis–Hastings chain at two resolutions	2.5	[Sai05, §2.3.3–2.3.4]	truncation of labels (§3.5)	—
`makeSRP`, `insertData`	6.2	[TKS12, Algs. 1–2]	label at depth δ^*, reduce by key (§7.6)	`quickToLabeled` (Chapter 10)
`AddSRP`, linear combinations	6.3	[TKS12, Alg. 3]	label-wise $\alpha a + \beta b$ after coarsening (§7.6)	—
`ReuniteEmptyNodes`	6.4	[TKS12, Alg. 4]	not needed: absent keys (§7.6)	—
`AddSRPHist`, averaging	6.5	[STHL13, Alg. 1]	label-wise sum of heights on a common refinement	—
stay-split-merge Metropolis–Hastings on pavings	6.5	[STHL13, §3, eq. (6)]	split $\ell \mapsto \ell\mathtt{0}, \ell\mathtt{1}$, merge the reverse (§7.6)	—
`InitSEBPQ` / `SEBTreeMC` (randomised priority queue)	6.5, 7	[STHL13, Alg. 2], [ST19, Alg. 1]	Appendix A.12	the bottom-up merge (Chapter 10)
`SRPCollate`	8.4	[ST19, Alg. 3]	collation as a K-way union (Chapter 6)	`spacedHistToCRP` (Chapter 10)
`GetYatracos` (one-box update)	8.4	[ST19, Alg. 4]	— (to write)	Scheffé integrals at the driver, $O(Lh^3)$ (cost layer 3)
`GetDelta`	8.4	[ST19, Alg. 2]	— (to write)	`scheffeSetsValidationCount` + accumulators (Chapter 10)
zoom-in search	8.5	`NUM_CHECKS`, `NUM_ITERS` [SYH+18]	—	`getMDE` / `mdeStep` (Chapter 10)
`CheckBounds`	8.6	[SYH+18] (`mrs-1.0-YatracosThis`)	—	— (the SIMULATION node)
subtree partitioning: sketch, maximal-weight subtrees, LPT, `findSubtree`	9	—	—	`SubtreePartitioner.scala`
merge to the count limit	9	—	the walk of `Lem-Merge`	`maximalCountSubtreeGeneration`

A.1 Conventions

- The pseudocode states each algorithm in its **sparse** form: a paving is a map from leaf labels to values (Chapters 4 to 6), and a label is a bitstring read left to right from the root. Where the MRS2 or Spark form differs in a way that matters, a note says how.
- `anc(ell, d)` is the ancestor of label ell at depth d: the first d bits of ell (Chapter 4); `meet(u, v)` is $u \wedge v$, the longest common prefix of two labels (§4.6).
- `split(ell)` gives the two children ell0 and ell1; `vol(ell)` is the volume of the cell of ell in the root box.
- "Sorted" means sorted in left-right order of labels (Chapter 4, `Prop-Block`).
- In the code blocks, `Dbar` is $\overline{\Delta}$, `wbar` is $\overline{w}$, `cbar` is the count limit, `mbar` is the leaf cap $\overline{m}$ and `theta_hat` is $\hat{\theta}$.
- Each algorithm names the section that proves what it owes. The pseudocode states nothing that section does not.

A.2 The Moore rejection sampler

```
MooreRejection(p*, root box Θ, box budget W_max, acceptance threshold a_min,
  trial budget T):
  # The envelope, by the 2013 rule (\cref{sec-moore-sampler}).
  Q ← priority queue of boxes keyed by vol(Θ_i) · wid P*(Θ_i);  Q ← {Θ}
  while |Q| < W_max and AcceptanceLowerBound(Q) < a_min:
      Θ_i ← pop the box of largest key
      bisect Θ_i at the midpoint of its FIRST side of maximal width into Θ_i0,
        Θ_i1
      push Θ_i0 and Θ_i1 with their keys
  # The proposal: a box with probability proportional to vol(Θ_i) · upper
    P*(Θ_i), by Walker's alias table.
  alias ← WalkerAlias({vol(Θ_i) · upper P*(Θ_i)})          # O(W) set-up, O(1)
    per draw
  for trial in 1..T:
      i ← alias.draw();  t ← a uniform point in Θ_i
      h ← Uniform[0, upper P*(Θ_i)]
      if h <= p*(t): return t                               # exact:
        \cref{sec-moore-sampler}
  return NONE                                              # no sample rather
    than a wrong one: \cref{sec-nosample}

AcceptanceLowerBound(Q) = Σ vol(Θ_i) · lower P*(Θ_i)  /  Σ vol(Θ_i) · upper
  P*(Θ_i)
```

Note. The trans-dimensional sampler (§3.3) is the same loop over labelled boxes; the label k selects the domain ${}^{k}\mathbb{T}$ and the expression evaluated.

A.3 Statistical regular pavings, sparse form

```
MakeSRP(points x_1..x_n, root box, depth delta*):            # makeSRP +
  insertData, sparse
  counts ← empty map from labels to integers
  for each x_j:  counts[label of the depth-delta* cell containing x_j] += 1
  return counts                                             # an absent label has
    count 0
```

```
Coarsen(counts, d):                                          # pushforward by
  anc(·, d): \cref{sec-arith}, Prop-Relabel
  out ← empty map
  for (ell, c) in counts with |ell| >= d:  out[anc(ell, d)] += c
  return out

AddSRP(a, b, alpha, β):                                      #
  \cref{sec-traj-arith}, label-wise after coarsening:
  \cref{sec-sparse-labels-change}
  require a and b at the same depth delta*
  out ← empty map
  for ell in keys(a)  union  keys(b):  out[ell] ← alpha·a[ell] + β·b[ell]    #
    absent = 0
  return out
```

Note. The invariant of §7.6 (every leaf shallower than δ^* holds 0) is what makes `AddSRP` at a common depth exact; §7.6 states its closure and the operations that break it. `ReuniteEmptyNodes` has no sparse form: an empty cell is an absent key.

A.4 The stay-split-merge chain on pavings

```
SSMStep(paving s, data, rule):                               # one
  Metropolis--Hastings step
  with the stay mass of \cref{sec-posterior-mean} (MRS proposeChangeMCMCState):
    return s
  propose as in \cref{sec-posterior-mean}, either
    split: a splittable leaf ell of s;  s' ← s with ell replaced by ell0, ell1
    merge: a cherry u of s;           s' ← s with u0, u1 replaced by u
  alpha ← min(1, π(s') q(s | s') / (π(s) q(s' | s)))          # π: the
    posterior of \cref{sec-posterior-mean}
  with probability alpha: return s'
  return s
```

Note. Which leaves are splittable, and so the proposal ratio, depends on the rule and the caps of §7.5, which states when the ratio is at least one. Coincident points make a chain dive at every depth unless a cap or $\#_n \geq 2$ stops it (§7.5).

A.5 The Yatracos matrix along one path, and the objective

```
GetYatracosStep(Y, state i−1 → state i by splitting the leaf v*):    #
  GetYatracos, one split
  # Y[j][i] = {f_j > f_i}, as a set of leaves; exact by the three facts of
    \cref{sec-mde-pavings}.
  for each earlier state j < i:
      Y[j][i] ← Y[j][i−1] minus {v*};   Y[i][j] ← Y[i−1][j] minus {v*}
      for c in split(v*):
          if f_j(c) > f_i(c): add c to Y[j][i]
          if f_i(c) > f_j(c): add c to Y[i][j]
  return Y

Delta(theta, Y, validation counts, n_val):                   # GetDelta, with
  PER-SET accumulators
  best ← 0
  for each set A in Y:
```

```
    integral ← 0;  mass ← 0                                  # reset for EVERY
      set: \cref{sec-checkbounds}
    for each leaf v in A:
        integral += f_theta(v) · vol(v);  mass += valcount(v) / n_val
    best ← max(best, |integral − mass|)
  return best
```

Note. `getMappedFunctionDelta` in `mrs-1.0-YatracosThis` sets the two accumulators once, before the loop over the sets (§9.6); the reset above is the repair.

A.6 The zoom-in search

```
ZoomMDE(path P from the root to its cap, k = NUM_CHECKS, I = NUM_ITERS):
  Θ ← the root and k states spaced uniformly along P        # k + 1 states
  for it in 1..I:
      score the states of Θ by Delta on the collation of Θ
      [lo, hi] ← the range spanned by the three lowest scores
      step ← floor((hi − lo) / k);  if step <= 1: break
      Θ ← Θ  union  {lo, lo + step, …, hi}
  return the state of least Δ over the Yatracos class of Θ, the smallest index
    on ties
```

Note. The MRS form rebuilds the path with the same seed at each iteration (§9.3). The Spark form at `c0a1837` (`getMDE`) keeps each round's winner and its next larger neighbour, and selects from the last round only (§9.5).

A.7 `CheckBounds`, as the repair runs it

```
CheckBounds(known density f on [0,1]^d, n, phi, CRITLEAVES, NUM_CHECKS,
  replicates R):
  for r in 1..R:
      x ← n points from f;  validation ← the first floor(phin);  training ←
        the rest
      path ← SEBTreeMC on the training points to CRITLEAVES leaves
      Θ ← the states with 1, c, 2c, …, CRITLEAVES leaves, c = floor(CRITLEAVES
        / NUM_CHECKS)
      theta* ← the MDE over Θ (A.5, per-set Delta)
      IAE*   ← integral|f_theta* − f|                              # paving
        arithmetic: \cref{sec-arith}
      minIAE ← min over theta in Θ of integral|f_theta − f|
      Δtrue  ← max over A in Yatracos(Θ) of |integral_A f − validation mass of
        A|
      record Thm2_holds ← (IAE* <= 3·minIAE + 4·Δtrue)
```

Note. This is the check the simulation node reruns. The published code rounds φn instead of flooring it, and computes `maxDelta` as the running sum (§9.6).

A.8 The final distributed algorithm

```
FinalMDE(data source, phi, count limit, k, xi, eps0, Dbar):
  # Phase 0: the data, once; a partition-invariant split.
  X ← read or generate;  persist
  (train, val) ← split by a hash of a stable point id, or two samples generated
    separately
```

```
  # Phase 1: the root box, from the training points only.
  B ← reduce(train, box hull)                          # val points outside B: no
    label, counted only in phin
  # Phase 2: depth, labels, counts, all from the training points.
  D ← the shallowest depth meeting (G) (\cref{sec-resolution}):
        count at D0;  if (G) holds, coarsen these counts;  else label the
          over-full cells' points
        at the deepest representable depth and binary-search the depths
      (if none is representable: the deepest representable one, over-full cells
        kept as atoms, said so)
  counts ← train.map(x -> (fixed-width label of x at depth D, 1)).reduceByKey(+)
    # exact: Prop-Relabel
  valrecs ← val.map(x -> (label of x at depth D, 1)), not combined;  persist
  # Phase 3: the merge.
  m ← ceil(β · records · bytes per record · working factor / memory per
    concurrent task)
  if the count map can be ranked exactly:
      parts ← counts in m contiguous runs of left-right order, by exact rank
      delta ← 1 + max over boundaries i of |c_i|,  c_i = (last label of run i)
        meet  (first label of run i+1)
          (delta ← 0 for one run)
  else:
      (parts, delta) ← SubtreePartition(counts, m)                    # A.9
  local ← parts.mapPartitions(the merge walk W(delta), streaming:
              look-ahead <= count limit + 1 records, last emitted label,
                one-subtree delay;
              never relabel to the root)                          # Lem-Merge:
                \cref{sec-exact}
  s* ← at the driver: W(0) on sort(local), ALWAYS                  # Lem-Merge,
    Cor-Safe: \cref{sec-exact}
  # Phase 4: the selection.
  return GapSearch(s*, valrecs, phin, k, xi, eps0, Dbar)                # A.10
```

Note. The code at `c0a1837` runs the driver walk only when a depth limit refused a join; §10.2 gives a case where that trigger misses.

A.9 Subtree partitioning

```
SubtreePartition(counts, m):
  S ← a reservoir sample of each input partition, about 3·min(hint·m, 10^6)
    labels in all
  weight(ell) ← records of its partition / its sample size   (1 / fraction for a
    resampled partition)
  W ← Σ weights;  wbar ← W / m
  roots ← the merge walk on S, with weights for counts, limit wbar and no depth
    limit;
          each emitted subtree raised to its highest ancestor that overlaps no
            other emitted one   # Thm-MWS
  sort roots by decreasing weight;  give each to a partition of least current
    load     # LPT: Thm-LPT
  delta ← the largest depth of a root
  partitioner(ell) ← the partition of FindSubtree(roots in left-right order,
    ell)
  return (counts partitioned by partitioner, each partition sorted in left-right
    order, delta)
```

```
FindSubtree(R in left-right order, ell):                          # Thm-Find: at
  most floor(log2 |R|) + 1 probes
  binary-search R for the root whose subtree contains ell
  if there is none: return the neighbour whose common ancestor with ell is
    deepest
```

A.10 The gap search

```
GapSearch(s*, valrecs, phin, k, xi, eps0, Dbar):          # Dbar fixed from the
  training points only
  P ← the backtrack from s* to the root, each step merging the cherry least in
    (count, heap code)   # (T)
  arc(t) ← Σ over i < t of |a_i − b_i| / n'                    # Lem-Arc (i)
  U ← k states of P spaced by arc length, including theta_0 = s* and the root
    theta_H
  Score(U, U)
  repeat:
      new ← {}
      for each gap J between consecutive u, u' of U that holds a state:
          low_J ← Δ_u + Δ_u' − 2·Dbar − arc_J / 2           # scores over
            classes inside A_P
          if arc_J / 2 > max(xi · low_J, eps0):  add the arc-length midpoint of J
            to new
      Score(new, new  union  the end-points of their gaps);  U ← U  union  new
  until new is empty                                          # every gap settled
  theta_hat ← the MDE over U with the Yatracos class of U
  return (theta_hat, xi, eps0, Dbar, |P|, 2|P|² exp(−2 phin Dbar²), D, whether
    (G) held)
                                    # Thm-Certified: 3(1+xi) min over P + 3eps0
                                      + 4·Dbar, on the event E

Score(new, T):                                                # scores the states
  of new over Yatracos(T)
  collate T on its common refinement; skip leaves common to its coarsest and
    finest state; MinimiseLeaves
  per partition of valrecs (the round-1 records, reused unchanged): count the
    records in every set
  combine the partitions' count matrices by treeAggregate     # each partition
    once: Thm-DSSC (E)
  Δ_u ← max over sets A of |integral_A f_u − N_A / phin|  for u in new  # phin
    includes points outside the box
```

Note. The bounds need the conditions on the run of §10.4. With floating-point integrals whose computed winner minimises Δ_θ only up to $\varepsilon_{\mathrm{fp}}$, they carry $2\varepsilon_{\mathrm{fp}}$ (Chapter 11).

A.11 The rows stated in the body

The paving arithmetic of Chapters 4 to 6 has its pseudocode in the body, in both the pointer form of MRS2 and the leaf-list form, each beside the result that proves it. It is not repeated here: - the union and the common refinement, Algorithms 1 and 2; - point location, Algorithms 3 and 4; - MinimiseLeaves, Algorithms 5 and 6; - MRPOperate and MRPTransform, Algorithms 7 to 9; - the fold, coarsening and insertion, Algorithms 10 and 11; - marginals and slices, Algorithms 12 to 15; - coverage regions, UniformApprox and the enclosure queues, Algorithms 16 to 19.

Lemma 11.14 counts the rounds of the zoom (Appendix A.6) and of the gap search (Ap-

pendix A.10); it adds no algorithm. Three rows remain: SEBTreeMC, the merge walk $W(\delta)$, and the (G)-depth.

A.12 SEBTreeMC

```
SEBTreeMC(sample, count limit cbar, leaf cap mbar, depth cap D):       # Dfn-SEB
  s ← the root alone;  path ← [s]
  while some leaf of s has depth < D                                   # a
    splittable leaf exists
        and s has fewer than mbar leaves
        and the largest count c of a splittable leaf exceeds cbar:
      u ← a splittable leaf of count c, drawn uniformly among them      # only
        ties are randomised
      s ← s with u replaced by its children u0, u1;  append s to path
  by_count ← every splittable leaf of s holds at most cbar points       # true
    when none is splittable
  return (path, by_count)                                              # S_0, ...,
    S_T; the terminal state is S_T
```

Note. The chain stops by its count criterion exactly when $\overline{m} > |S_{\overline{\#},D}|$, and then its terminal state is the threshold paving s^*_D whatever the tie-breaking (Lemma 8.4). In MRS the loop is `AdaptiveHistogram::prioritySplit` with `CompCount` [SYH+18]. Under (G), the final algorithm computes the nonempty leaves of the same paving from below, by the merge walk (Appendix A.13, Lemma 11.4).

A.13 The merge walk $W(\delta)$

```
MergeWalk(atoms a_1, ..., a_N in left-right order, counts kappa, count limit
  cbar, depth limit delta):   # Dfn-Walk; delta <= min |a_i|
  count(u) = Σ kappa(a_i) over the atoms a_i below u                    # #_W
  e_last ← none;  t ← a_1;  E ← []                                      # no
    node contains none
  for i in 2..N:
      A ← meet(t, a_i)
      if |A| >= delta and count(A) <= cbar and A does not contain e_last:
          t ← A                                                        # an
            extension
      else:
          append t to E;  e_last ← t;  t ← a_i                          # a
            refusal
  append t to E                                                        # E = e_1,
    ..., e_S: Lem-CountWalk (a)
  for j in 1..S:                                                       # the
    raising: Lem-CountWalk (d)
      k ← max(delta, |meet(e_{j-1}, e_j)| + 1, |meet(e_j, e_{j+1})| + 1)   # a
        term with no neighbour is dropped
      r_j ← anc(e_j, k)
  return r_1, ..., r_S, each with count(r_j)                           #
    count(r_j) = count(e_j): Lem-CountWalk (c)
```

Note. For $S = 1$ the raising gives $e_1|_\delta$, by Definition 11.1. Under (G), with a depth limit δ safe for the partitioner (Definition 11.3), each partition runs $W(\delta)$ on its stored labels, and the driver runs $W(0)$ on the sorted outputs of all partitions, always (Lemma 11.4(ii), Appendix A.8). An atom below the current subtree is absorbed by an extension that leaves t unchanged; the code

at `c0a1837` jumps past the whole run of atoms below A, with the same effect (§11.1).

A.14 The (G)-depth, in two passes

```
GDepth(training points, root box, count limit cbar, coarse depth D0 <= deepest
  representable depth Dmax):   # sec-gdepth
  # Pass 1: the depth-D0 count map, combined on the map side (Prop-Relabel).
  C ← countOf_D0(training points)
  for d in 0..D0:  m_d ← the largest count at depth d, by coarsening C          #
    Lem-Depth (iv)
  if m_D0 <= cbar:                                                              #
    (G) holds at D0
      return 0 if m_0 <= cbar, else 1 + max{ d < D0 : m_d > cbar }              #
        Lem-Threshold (iii)
  # Pass 2: only the points of the over-full depth-D0 cells.
  L ← their depth-Dmax labels, sorted, one per point, repeats kept
  one sweep of L with a stack of open ancestors:  m_d ← the longest run with one
    depth-d ancestor, d in (D0, Dmax]
                                                  # over L only: not the global
                                                    m_d, but exact for the test
                                                    m_d <= cbar
  if m_Dmax > cbar:  return NONE                                                #
    no representable depth: sec-fallback
  return the least d in (D0, Dmax] with m_d <= cbar                             #
    (G) is monotone in d: Lem-Threshold (iii)
```

Note. Only the training points are counted (Definition 12.1). A cell over-full at a depth beyond D_0 lies inside an over-full depth-D_0 cell, so the second pass needs no other points. A binary search over the depths, with at most $\lceil \log_2(D_{\max} - D_0 + 1) \rceil$ scans of one depth each, replaces the stack; Appendix A.8 writes that form. The training points are then labelled at D^{G}, or at $D_{\max}$ when the result is NONE, by a third pass over all of them whenever $D^{\mathrm{G}} > D_0$.

Appendix B

Proofs

This appendix proves the five results that §10.3 states for subtree partitioning and cites: the cost of the weighted walk (Theorem B.3), the longest-processing-time bound (Theorem B.6) and its sampled-load corollary (Corollary B.7), the lookup `findSubtree` (Theorem B.8), and the transfer from sampled loads to received cells (Proposition B.9), which rests on Conjecture B.10, a conjecture stated here with no proof. They make precise the complexity results for the subtree partitioner of [San23]. Every statement is about the code at commit `c0a1837`, which the final algorithm of Chapter 10 keeps for these four steps; every hypothesis is stated with its statement. Costs are counted as in Definition 6.1, and the left-right order, the join $\wedge$, the truncation $x|_t$ and the interval $I(u)$ of a node among sorted labels are those of Proposition 4.27 and Lemma 4.28. The exactness results on the merge and the count (Lemma 11.4, Theorem 11.12, Lemma 11.11) are proved in Chapter 11, and the cost of a round in Proposition 13.6; neither is repeated here. Every bound below was derived by reading the code; none has been confirmed by a run (§14.1).

B.1 The weighted walk

The partitioner's second step runs the merge walk over the sampled labels with their weights in place of counts (§10.3, step 2): `maximalWeightSubtreeGeneration` (`SubtreePartitioner.scala` l.302–337) followed by `maximalSubtreesDecreaseDepth` (l.236–287) with depth limit 0. Lemma 11.4(iii) shows that the code's walk is the walk $W(\delta)$ of Definition 11.1 on counts; here only its cost is at issue, and the weights are reals. Each iteration does a binary search, but it also sums weights over part of the common ancestor's interval, and a terminating step can sum $\Theta(N)$ of them; so the weight additions must be counted. The code adds to the current subtree's weight only the weights from the current index on (l.316), where `SubtreePartition` of Appendix A.9 sums the whole interval of the common ancestor; Lemma B.1 shows that the two values agree wherever the branch taken depends on them.

Setting. There are $N \geq 1$ pairs $(\ell_0, w_0), \ldots, (\ell_{N-1}, w_{N-1})$ with weights $w_i \geq 0$, and a threshold $\overline{w}$, such that (H1) the distinct labels form an antichain of nodes of depth at most D, and (H2) the list is in left-right order, with copies of a label adjacent. In the partitioner the labels are distinct finest-cell labels of depth D, sorted by `leftRightOrd`, so (H1) and (H2) hold. Br is the set of *branching nodes*, the nodes both of whose children contain a label; |Br| is one less than the number of distinct labels. Weights are added in exact arithmetic; the code uses 32-bit floats, and Theorem B.6 and Corollary B.7 are stated in exact arithmetic as well.

Lemma B.1 (The walk)**.** *Assume (H1) and (H2). The main loop of* `maximalWeightSubtreeGeneration` *runs at most $N-1$ times and emits subtrees $t_1, \ldots, t_S$ such that the sets $I(t_1), \ldots, I(t_S)$ are consecutive intervals partitioning $\{0, \ldots, N-1\}$; the t_j are pairwise disjoint*

with $t_1 \prec \cdots \prec t_S$; and the weight carried by t_j is $\sum_{i \in I(t_j)} w_i$, at most $\overline{w}$ unless t_j is one of the labels. The walk of `SubtreePartition` *(Appendix A.9) emits the same subtrees with the same weights.*

Proof. Write s for the index at which the current subtree t was started. Initially $s = 0$, $t = \ell_0$, $i = 1$ and oldMaxIndex $= -1$. At the head of each iteration we keep the invariants (i) oldMaxIndex $= s - 1$; (ii) $[s, i) \subseteq I(t) \subseteq [s, N)$, and t carries the weight $\sum_{j \in [s,i)} w_j$; (iii) the subtrees emitted so far satisfy $I(t_j) = B_j$ for consecutive intervals $B_1, B_2, \ldots$ partitioning $[0, s)$. We also show that every iteration increases i. Initially (i) and (iii) are trivial, and (ii) holds because $I(\ell_0)$ is the block of copies of ℓ_0, which starts at 0 by (H2).

Take an iteration at $i < N$, let $a = t \wedge \ell_i$, and write $I(a) = [\mathrm{lo}, \mathrm{up})$, an interval by Lemma 4.28(c) since by (H1) no label strictly contains $a \sqsupseteq \ell_s$. Since $a \sqsupseteq t$, lo $\leq s$, and $i \in I(a)$ gives up $> i$. By (ii) the code's sum is $W_a = \sum_{j \in [s,\mathrm{up})} w_j$. There are three cases.

- **Extension** ($W_a \leq \overline{w}$ and oldMaxIndex $<$ lo). By (i), lo $\geq s$, so lo $= s$ and $I(a) = [s, \mathrm{up})$. The code sets $t = a$ with weight W_a and $i =$ up, so (ii) holds with equality.
- **Copies** ($\ell_i = a$ and the extension is refused). Here $\ell_i \sqsupseteq t \sqsupseteq \ell_s$, so $\ell_i = \ell_s$ by (H1); hence $t = \ell_i =: x$ and $I(a)$ is the block of copies of x. We claim lo $= s$: otherwise the copy ℓ_{lo} lies in some earlier block $B_j = I(t_j)$, so $x \sqsubseteq t_j$ and $s \in I(t_j) \subseteq [0, s)$, which is absurd. So W_a is the weight of $I(x) = [s, \mathrm{up})$. If up $< N$, the code emits x, a block for (iii), and restarts with $s =$ up, $t = \ell_{\mathrm{up}}$, oldMaxIndex $=$ up $- 1$ and $i =$ up $+ 1$; (ii) holds because no copy of ℓ_{up} lies before up, since it would lie in a block $I(t_j)$, which would then contain up. If up $= N$, the code sets $i = N$, and (ii) holds with $I(x) = [s, N)$.
- **Termination** (every other case). The code emits t and sets $s = i$, $t = \ell_i$, oldMaxIndex $=$ $i - 1$ and $i = i + 1$. The emitted t has $I(t) = [s, i)$: by (ii) it suffices that $\ell_i \not\sqsubseteq t$. If $\ell_i \sqsubseteq t$ and t was never extended, then $t = \ell_s$ and (H1) gives $\ell_i = t = a$, the copies case; if t was last extended at an earlier iteration, that iteration set $i = \mathrm{up}(t)$, so $i \notin I(t)$. Invariant (ii) for the new subtree follows as in the copies case.

At exit $i = N$, and by (ii) the last emission has $I(t) = [s, N)$. The emitted sets are nonempty and pairwise disjoint, so no t_j contains another, and disjoint nodes whose label blocks are consecutive in left-right order are in that order. A subtree emitted after at least one extension carries the weight tested at its last extension, at most $\overline{w}$; every other emitted subtree is a label. The pseudocode tests $\sum_{[\mathrm{lo},\mathrm{up})} w \leq \overline{w}$ together with oldMaxIndex $<$ lo; since lo $\leq s =$ oldMaxIndex $+ 1$, this holds exactly when lo $= s$ and $W_a \leq \overline{w}$, the code's test, and in the copies case lo $= s$, so the stored weights agree as well. □

Marks. Proved here, from the 2026-11-04 preprint. [LEAN: the walk on counts is `IasCert.L1MDE.MergeWalk` (Lemma 11.4); the weighted walk's emission over reals, to write (proposed module `WeightedWalk`, the same invariants with weights in a linearly ordered commutative monoid).] [SIM: the code's walk against the pseudocode on random sorted antichains with random nonnegative weights and random $\overline{w}$: the same subtrees and weights, 0 failures expected; to run, L1MDE-PROOF-SIMULATION.]

Lemma B.2 (Raising)**.** *Let $t_1 \prec \cdots \prec t_S$ be pairwise disjoint nodes. With depth limit 0,* `maximalSubtreesDecreaseDepth` *(*`SubtreePartitioner.scala` *l.236–287) returns, with $O(S)$ label operations, nodes $u_1, \ldots, u_S$ such that $t_j \sqsubseteq u_j$ for every j, the u_j are pairwise disjoint with $u_1 \prec \cdots \prec u_S$, and no u_j contains t_k for $k \neq j$. For $S = 1$ it returns the root.*

Proof. Let $S \geq 2$ and $a_j = t_j \wedge t_{j+1}$, so that t_j lies in the left subtree of a_j and t_{j+1} in its right one. The code returns $u_1 = a_1\mathsf{L}$, $u_S = a_{S-1}\mathsf{R}$, and, for $1 < j < S$, the deeper of $a_{j-1}\mathsf{R}$ and $a_j\mathsf{L}$. Both candidates for u_j contain t_j, so the deeper lies inside the other. Hence $u_j \sqsubseteq a_j\mathsf{L}$ for

$j < S$ and $u_{j+1} \sqsubseteq a_j\mathsf{R}$, so $u_j \prec u_{j+1}$, and $\prec$ is transitive (Proposition 4.27). By construction u_j contains neither t_{j-1} nor t_{j+1}, and a node containing t_j and t_k contains everything between them in left-right order; so u_j contains no t_k with $k \neq j$. Each u_j costs two joins and one depth comparison; the list is built by prepending and reversed once. □

Marks. Proved here. [LEAN: certified, `IasCert.L1MDE.Raise` (`left_right_join`, `raiseD_eq_of_max`, `decreaseDepth_eq`, `raiseAt_eq_raiseD`, `decreaseDepth_eq_raiseD`, `lem_raise`, `lem_raise_props`: the code as it runs, l.236–287, equals the raising of `MergeWalk` for $S \geq 2$ and any depth limit, and the statement above for depth limit 0; Chapter 15). The cost and the single-subtree case with a positive depth limit, where the code returns the root, are outside the model.] [SIM: covered by Lemma 11.2's check (`rev10_walk.py`, 4000 cases against the transliterated `maximalSubtreesDecreaseDepth`).]

Theorem B.3 (Cost of the weighted walk [San23])**.** *Assume (H1) and (H2). `maximalWeightSubtreeGeneration` runs at most $N-1$ iterations, each with one join and at most $2(\lceil \log_2 N \rceil + 1)$ binary-search steps, then performs $O(S)$ label operations, and makes in total at most*

$$(N-1) + \sum_{a \in \mathrm{Br}} \big|I(a\mathsf{R})\big| \;\leq\; (N-1) + N \min(D, N-1)$$

weight additions. Its cost is therefore $O(\lambda N \log N + N \min(D, N))$ word operations. The walk of `SubtreePartition` (Appendix A.9) obeys the same bound, with at most $\sum_{a \in \mathrm{Br}} |I(a)| + N \leq N(\min(D, N-1)+1)$ additions. The bound is attained up to a constant factor: for every $N \geq 2$ and $D \geq N-1$ there is an input with distinct labels of depth D, unit weights and $\overline{w} = 1$ on which the code makes exactly $N(N-1)/2$ additions while its label operations stay $O(N \log N)$. If the sums are read from prefix sums $\mathrm{pre}_j = \sum_{k<j} w_k$, which cost N additions to precompute, every iteration does $O(1)$ arithmetic and the cost is $O(\lambda N \log N)$.

Proof.

- **Iterations and label operations.** Each iteration increases i (Lemma B.1), so there are at most $N-1$. Each does one `join` and one `subtreeAt`, with at most $2(\lceil \log_2 N \rceil + 1)$ predicate evaluations by Lemma 4.28(c), then a loop of $\mathrm{up} - i$ weight additions. The raising pass costs $O(S)$ (Lemma B.2). It remains to count the additions.
- **Extension and copies steps.** There $\mathrm{up} - i$ is at most the amount by which i advances, so these steps add at most $N-1$ weights in total.
- **One termination step** at index i. As in the proof of Lemma B.1, $\ell_i \not\sqsubseteq t$; also $t \not\sqsubseteq \ell_i$, or $\ell_i \sqsupseteq \ell_s$ would give $\ell_i = \ell_s = t$. So $t \prec \ell_i$ are disjoint, and $a = t \wedge \ell_i$ has t in its left subtree and ℓ_i in its right one. The indices in $[i, \mathrm{up})$ lie in $I(a)$ at or after ℓ_i, so in $I(a\mathsf{R})$, and the step adds at most $|I(a\mathsf{R})|$ weights.
- **All termination steps.** In the step above $\ell_{i-1} \sqsubseteq t$, so $a = \ell_{i-1} \wedge \ell_i$, with ℓ_{i-1} below $a\mathsf{L}$ and ℓ_i below $a\mathsf{R}$. In a list in left-right order exactly one adjacent pair crosses from $a\mathsf{L}$ to $a\mathsf{R}$, so distinct terminations have distinct $a \in \mathrm{Br}$, and together they add at most $\sum_{a \in \mathrm{Br}} |I(a\mathsf{R})|$ weights. Index j is counted once for each $a \in \mathrm{Br}$ with $\ell_j \sqsubseteq a\mathsf{R}$; each such a is a strict ancestor of ℓ_j, so there are at most $\min(|\ell_j|, |\mathrm{Br}|) \leq \min(D, N-1)$ of them.
- **The pseudocode.** A step whose ancestor a is not a label sums $|I(a)|$, with $a = \ell_{i-1} \wedge \ell_i \in \mathrm{Br}$ distinct across steps, and the same double count bounds these sums by $N \min(D, N-1)$. A step whose ancestor is a label sums a block of copies, which the walk then consumes; such steps sum at most N in total.
- **The lower bound.** Take $D \geq N-1$, $\ell_k = \rho\,\mathsf{R}^k\mathsf{L}$ for $k < N-1$ and $\ell_{N-1} = \rho\,\mathsf{R}^{N-1}$, each continued to depth D by left turns. These labels are pairwise disjoint, of depth D and in left-right order, and $\ell_{i-1} \wedge \ell_i = \rho\,\mathsf{R}^{i-1}$, whose right subtree holds $\ell_i, \ldots, \ell_{N-1}$. With unit weights and $\overline{w} = 1$ no two labels fit together, so every iteration terminates with $t = \ell_{i-1}$,

and its loop adds the $N - i$ weights of $\ell_i, \ldots, \ell_{N-1}$ (l.316): $\sum_{i=1}^{N-1}(N-i) = N(N-1)/2$ additions.

- **Prefix sums.** Then $W_a = pre_{\text{up}} - pre_s$ in the code, and $pre_{\text{up}} - pre_{\text{lo}}$ in the pseudocode. □

Marks. Proved here. The addition count is the double count over branching nodes that Proposition 13.6(iv) reuses for the streaming merge. The final algorithm keeps the code's loop; whether the prefix sums pay is measurement 10 of §13.5, since the walk runs once on the driver over N_s sampled labels (Remark B.4) and its additions are at most $N_s \min(D, N_s)$. [LEAN: to write — on `WeightedWalk`, the addition count by the double count; the lower-bound family as a witness.] [SIM: the addition count on random inputs against the bound, and the lower-bound family at $N \le 64$ against $N(N-1)/2$; to run, L1MDE-PROOF-SIMULATION.]

Remark B.4 (The walk inside the partitioner)**.** In the subtree partitioner, N is the number N_s of sampled cells. The code takes a target of $\min(\eta\, p_{\text{in}}, 10^6)$ labels for p_{in} input partitions, where η is `samplePointsPerPartitionHint`, and oversamples three-fold, so N_s is at most $3 \min(\eta\, p_{\text{in}}, 10^6)$ plus the number of input partitions, plus a resample of the over-full input partitions whose expected size is at most $\min(\eta\, p_{\text{in}}, 10^6)$ (l.86–115; about $3 \cdot 10^6$ labels in the terabyte runs, §10.3). Building the subtrees costs the driver $O(\lambda N_s \log N_s + N_s \min(D, N_s))$, including the sort at l.119; this is linear in N_s times the depth, not $N_s \log N_s$, as soon as $D > \log_2 N_s$, which is the usual case. The sampled labels are distinct. A label sampled from partition p carries the weight $n_p/|\text{sample}_p|$, the records of its partition over its sample size, and a label from a resampled partition carries $1/\text{fraction}_p$, the inverse of its sampling fraction (l.99–115); every weight is therefore at most the largest ratio $n_p/|\text{sample}_p|$ over the input partitions (or $1/\text{fraction}_p$ for a resampled one), which is about $\max(1, L/\min(\eta\, p_{\text{in}}, 10^6))$ when the sample sizes are proportional to the partitions' records, L being the number of nonempty finest cells; the ratio of one partition can exceed the global one. With no over-full input partition the sampled total is $W = L$, so every sampled weight is at most $\overline{w} = W/m$ when that largest ratio is at most W/m, which holds when $\eta \ge 1$, $m \le \min(L, 10^6)$, no partition is resampled and the samples are proportional: these are the conditions under which Corollary B.7 applies. A sampled label heavier than $\overline{w}$ is kept as a subtree of its own (Lemma 11.7(ii)), as §10.3 says.

B.2 Longest-processing-time assignment

`distributeSubtreesToPartitions` (`SubtreePartitioner.scala` l.363–391) sorts the subtrees by decreasing weight with a stable sort (l.370), keeps the m partition loads in a binary heap ordered by load alone (l.372–380), and gives each subtree in turn to a partition of least current load (l.381–385). This is Graham's longest-processing-time rule, with ties broken by the heap. Its bound $(\frac{4}{3} - \frac{1}{3m})$ always holds and is attained for every m [Gra69]. The proof below reduces to the instance cut after the critical job, and then uses that the rule is optimal when every job exceeds a third of the optimum (Lemma B.5). The loads are 32-bit floats; the statements are for exact arithmetic. Here, as in §10.3, m is the number of partitions requested; Chapter 13 writes it p.

Lemma B.5 (LPT is optimal on large jobs)**.** *Let $p_1 \ge \cdots \ge p_K$ be job weights on m machines with optimal largest load T, and suppose every $p_k > T/3$. Then LPT, with any tie-breaking, has largest load T.*

Proof.

- **Big and medium jobs.** All jobs are positive, and in a schedule with largest load T a machine holds at most two jobs. A *big* job ($p > 2T/3$) shares its machine with no other.

Let b be the number of big jobs, $m' = m - b$, and $q_1 \geq \cdots \geq q_M$ the remaining *medium* jobs. An optimal schedule puts each big job alone and the medium jobs on the other m' machines, at most two each, so $M \leq 2m'$.

- **What LPT does.** LPT places the big jobs first, each on an empty machine (empty machines have load 0, all others positive load). When a medium job is placed, fewer than $M \leq 2m'$ medium jobs have been placed, by induction at most two per non-big machine, so some non-big machine holds at most one medium job and has load at most $2T/3$; every big machine, and every machine with two medium jobs, has load above $2T/3$. So each medium job goes to a non-big machine holding at most one medium job. The first medium jobs go to the empty machines; if $M > m'$, the job $q_{m'+i}$ $(1 \leq i \leq r = M - m')$ goes to the least loaded machine holding one medium job, and by induction on i these machines hold $q_1, \ldots, q_{m'+1-i}$, so $q_{m'+i}$ joins $q_{m'+1-i}$ (machines with equal loads are interchangeable). The final loads are the big jobs, the singles $q_1, \ldots, q_{m'-r}$, and the pairs $q_{m'+1-i} + q_{m'+i}$ for $1 \leq i \leq r$.
- **Each LPT load is at most T.** Every single is at most $p_1 \leq T$. For a pair, fix $i \leq r$. In an optimal schedule the $m' + i$ largest medium jobs lie on m' machines, at most two each, so at least i machines hold two of them. These cannot all take both their jobs from the $2i - 1$ jobs ranked $m' + 2 - i, \ldots, m' + i$. So one machine holds a job of rank at most $m' + 1 - i$ and another of rank at most $m' + i$; its load is at least $q_{m'+1-i} + q_{m'+i}$, which is therefore at most T. □

Theorem B.6 (LPT bound [Gra66, Gra69, San23])**.** *Let $m \geq 1$ and $p_1 \geq p_2 \geq \cdots \geq p_K \geq 0$ be job weights with total p_{tot}. Assign the jobs in this order, each to a machine of least current load, breaking ties arbitrarily. Let C_{LPT} be the largest final load and C^* the least possible largest load over all assignments. Then*

$$C_{\text{LPT}} \leq \left(\frac{4}{3} - \frac{1}{3m}\right) C^* \qquad \text{and} \qquad C_{\text{LPT}} \leq \frac{p_{\text{tot}}}{m} + \left(1 - \frac{1}{m}\right) \max_k p_k.$$

The first bound is Graham's for this order [Gra69] and is attained for every m; the second is the bound of list scheduling in any order [Gra66]. In particular `distributeSubtreesToPartitions` *satisfies both bounds in exact arithmetic, whatever its heap's tie-breaking, and costs $O(K \log K + K \log m + \lambda K)$.*

Proof.

- **The critical job.** Let ℓ be the last job placed on a machine whose final load is C_{LPT}. When ℓ was placed, that machine had the least of the m loads, say s, and the loads summed to $\sum_{k<\ell} p_k \leq p_{\text{tot}} - p_\ell$. So $C_{\text{LPT}} = s + p_\ell \leq (p_{\text{tot}} - p_\ell)/m + p_\ell = p_{\text{tot}}/m + (1 - 1/m)p_\ell$, which gives the second bound; it uses only that each job goes to a least-loaded machine, not the order.
- **Small critical job.** Since $p_{\text{tot}}/m \leq C^*$, the first bound follows when $p_\ell \leq C^*/3$.
- **Large critical job.** Otherwise consider the instance $p_1, \ldots, p_\ell$. LPT on it makes the first ℓ choices of the full run, and its largest load is C_{LPT}, since the critical machine receives nothing after ℓ and loads only grow. Its optimum T_ℓ is at most C^*, and all its jobs are at least $p_\ell > C^*/3 \geq T_\ell/3$. By Lemma B.5, $C_{\text{LPT}} = T_\ell \leq C^*$.
- **Tightness.** Graham's instance [Gra69] has $2m + 1$ jobs of weights $2m - 1, 2m - 1, 2m - 2, 2m - 2, \ldots, m + 1, m + 1, m, m, m$, LPT largest load $4m - 1$ and optimum $3m$.
- **Cost.** The code sorts with $O(K \log K)$ comparisons, makes K heap operations of cost $O(\log m)$, and K hash insertions. □

Nothing in the bound is new: the bound, its tightness and the proof through the critical job and the large-job lemma are Graham's. Two things are added here: the bound does not depend on

tie-breaking, so it holds for the code's heap, and combined with the weights of Lemma B.1 it gives a guarantee relative to the ideal load W/m.

Marks. Cited, with the proof restated for any tie-breaking. [LEAN: the second bound for every greedy schedule, $m\,C \le p_{\mathrm{tot}} + (m-1)\max_k p_k$, is certified in $\mathbb{N}$ (`IasCert.L1MDE.MPCSchedule graham`, with `exists_greedy`; Chapter 15); the first bound, $\frac{4}{3} - \frac{1}{3m}$, is not formalised and is kept for comparison only.] [SIM: both bounds on random weights against a brute-force optimum at $K \le 10$, $m \le 4$, and Graham's instance for $m \le 8$; to run, L1MDE-PROOF-SIMULATION.]

Corollary B.7 (Sampled load of subtree partitioning)**.** *Suppose every sampled cell weight is at most $\overline{w} = W/m$ (Remark B.4 gives the conditions: $\eta \ge 1$, $m \le \min(L, 10^6)$), no over-full input partition and no resample). Then every subtree carries weight at most W/m, there are $S \ge m$ subtrees, and the largest sampled load given to a partition is at most $\min\{(\frac{4}{3} - \frac{1}{3m})C^*,\ (2 - \frac{1}{m})W/m\}$.*

Proof. The sampled labels are distinct, so by Lemma B.1 a subtree either passed an extension test or is a single label, and in both cases its weight is at most W/m; raising does not change weights (Lemma B.2). The weights total W, so there are at least m subtrees. Apply Theorem B.6 with $\max_k p_k \le W/m$. □

The condition $S \ge m$ matters to the code: the partitioner reports $\min(m, S)$ partitions (l.147) while the heap assigns indices among all m (l.380–384), and with $S < m$ an index could reach the reported number or beyond, which Spark rejects. Relative to W/m, the constant 2 cannot be improved as $m \to \infty$: $m+1$ subtrees of weight $W/(m+1)$, each at most W/m, force a load of $2W/(m+1) = \frac{2m}{m+1} \cdot \frac{W}{m}$ on some partition under any assignment.

B.3 The lookup

The search `findSubtree` makes $O(\log S)$ probes for $S \ge 1$ subtrees (for $S = 0$ the code reads `subtrees(-1)`, l.422), and its fallback does $O(1)$ extra work. It is a binary search, but the comparison at l.410 is monotone along the array only when the subtrees form an antichain in left-right order and the query is at least as deep as every subtree: a query shallower than `mid` is returned unchanged by `truncate`, and a query strictly containing `mid` compares as lying to its right (Lemma 4.28(b)). The theorem assumes both. When no subtree contains the query, none lies on its root path, and the fallback returns the neighbour whose common ancestor with the query is deeper, which maximises that depth over all subtrees.

Theorem B.8 (findSubtree [San23])**.** *Let $S \ge 1$, let $u_0 \prec \cdots \prec u_{S-1}$ be an antichain in left-right order, let $\delta = \max_k |u_k|$, and let x be a node with $|x| \ge \delta$. Then `findSubtree` on x and the array of the u_k (`SubtreePartitioner.scala` l.401–439) (i) runs its loop at most $\lfloor \log_2 S \rfloor + 1$ times and costs $O(\lambda(1 + \log S))$ word operations; (ii) returns u_k if $u_k \sqsupseteq x$, and there is at most one such k; (iii) otherwise returns a u_k that maximises $|x \wedge u_k|$ over all k, namely whichever of the two neighbours $u_{\mathrm{high}} \prec x \prec u_{\mathrm{low}}$ (those that exist) has the deeper common ancestor with x, with ties kept at the last probed index; and (iv) depends on x only through $x|_\delta$.*

Proof.

- **The comparison values.** Let κ_k be the value at l.410 of `leftRightOrd.compare(x.truncate(|u_k|), u_k)`. As $|x| \ge |u_k|$, the truncation is $x|_{|u_k|}$, which equals u_k or is disjoint from it. By Lemma 4.28(a), $\kappa_k = 0$ if $u_k \sqsupseteq x$, $\kappa_k = -1$ if $x \prec u_k$, and $\kappa_k = +1$ if $u_k \prec x$.
- **Monotonicity.** The u_k are disjoint and sorted, so $k \mapsto \kappa_k$ is a run of $+1$'s, then at most one 0, then a run of -1's: if $u_k \prec x$ then $u_{k'} \prec x$ for every $k' < k$, similarly on the right, and two subtrees containing x would be nested.

- **(i).** The loop keeps $\kappa_k = +1$ for $k <$ low and $\kappa_k = -1$ for $k >$ high. It probes mid $=$ low $+ \lfloor$(high $-$ low)$/2\rfloor$ (l.406), and each half it can keep has at most $\lfloor$(high $-$ low $+ 1$)$/2\rfloor$ indices. Each probe costs $O(1)$ label primitives, and the fallback at most two joins.
- **(ii).** If some $\kappa_k = 0$, the invariant keeps k in [low, high] until k is probed and returned.
- **(iii).** Otherwise the loop ends with low $=$ high $+ 1$; the last probe is high if it moved low, and low if it moved high. The fallback (l.422–438) compares $|x \wedge u_{\text{mid}}|$ with $|x \wedge u|$ for the other neighbour, when it exists, and keeps the strictly deeper. For optimality, let $k < k' \le$ high: the box of $x \wedge u_k$ is an interval of the left-right order containing $\mathbf{x}_{u_k}$ and $\mathbf{x}_x$, so it also contains $\mathbf{x}_{u_{k'}}$; hence $x \wedge u_k \sqsupseteq x \wedge u_{k'}$, and the join depth increases towards x. The same holds on the right.
- **(iv).** Each κ_k depends only on $x|_{|u_k|} = (x|_\delta)|_{|u_k|}$. If $u_k \not\sqsupseteq x$, then u_k and $x|_\delta$ are disjoint ($u_k \sqsupseteq x|_\delta$ would give $u_k \sqsupseteq x$, and $x|_\delta \sqsupset u_k$ is impossible since $|u_k| \le \delta$), so an ancestor of x containing u_k is a strict ancestor of $x|_\delta$, and $x \wedge u_k = x|_\delta \wedge u_k$. □

Marks. Proved here. Part (iv) is what Corollary 11.6(a) uses, and Chapter 11 cited the certificate in its place until this appendix landed. [LEAN: (iv) certified, `IasCert.L1MDE.FindSubtree` (`join_truncate`, `findSubtree_truncate`: with every subtree of depth at most δ, `findSubtree` of x and of $x|_\delta$ agree, for any list of subtrees, no sortedness assumed; `cor_safe`, `cor_safe_maxSubtreeDepth`, `pairwise_strict_of_sorted`; Chapter 15). (i)–(iii) and the assignment of subtrees to partitions, to write.] [SIM: `findSubtree` against a brute-force maximiser of $|x \wedge u_k|$ on random antichains and random queries of depth at least δ, with the probe count against $\lfloor \log_2 S \rfloor + 1$; to run, L1MDE-PROOF-SIMULATION.]

B.4 From sampled loads to received cells

Corollary B.7 bounds the *sampled* load of a partition. What a partition receives in Phase 3 is the number of finest cells that `findSubtree` sends to its subtrees, and that number is estimated, not known, by the sample. The transfer is a conditional proposition. Here, as in Corollary B.7, m is the number of Phase-3 partitions, LPT's machines (the p of Proposition 13.4), not the $m = n_{\text{w}}c$ task slots of Definition 13.3.

Proposition B.9 (Phase-3 balance from LPT)**.** *Assume the hypotheses of Corollary B.7, and suppose that the sample is* ε-accurate: *the sampled total satisfies* $W \le (1+\varepsilon)L$, *and for every set* U *of subtrees,* `getPartition` *sends at most* $(1+\varepsilon)w(U) + \varepsilon L/m$ *finest cells to* U, *where* $w(U)$ *is the sampled weight of* U *and* L *the number of nonempty finest cells. Then every partition of Phase 3 receives at most* $\big((1+\varepsilon)^2(2-\frac{1}{m}) + \varepsilon\big)L/m$ *cells, and at most* $(1+\varepsilon)(\frac{4}{3} - \frac{1}{3m})C^* + \varepsilon L/m$, *where* C^* *is the optimal largest sampled load. So, after the subtree repartition, every partition of Phase 3 holds at most a fraction* β/m *of the finest cells, with* $\beta = (1+\varepsilon)^2(2-\frac{1}{m}) + \varepsilon$: *the balance factor of Proposition 13.4. The balance of the stages before the repartition depends on how the Phase-2 output is partitioned and is not covered.*

Proof. Let U_j be the subtrees assigned to partition j. By accuracy and Corollary B.7, partition j receives at most $(1+\varepsilon)w(U_j) + \varepsilon L/m$ cells, where $w(U_j)$ is at most $(2-\frac{1}{m})W/m \le (1+\varepsilon)(2-\frac{1}{m})L/m$, and also at most $(\frac{4}{3} - \frac{1}{3m})C^*$. □

Marks. Proved here, conditionally. [LEAN: Sizing prop_balance, prop_balance_lpt, prop_balance_par (the one line on `MPCSchedule par_le_of_points`) — built at `b6ea25813`, machine-checked-structural: ε-accuracy and Corollary B.7's loads are hypotheses (Cor-LPTCap is not built; `graham` is on natural work and the $\frac{4}{3}$ bound is not formalised).] [SIM: the received cells per partition against $\beta L/m$ on the calibration runs of §13.5 (the largest partition is their own [SIM] of Proposition 13.4).]

Conjecture B.10 (Accuracy of the partitioner's sample)**.** *Let the partitioner sample as the code does: a target of* $\min(\eta\, p_{\text{in}}, 10^6)$ *labels over* p_{in} *input partitions by reservoir sampling, oversampled three-fold, with the weights of Remark B.4. There is a constant* C *such that, for every* $\varepsilon \in (0,1)$*, the sample is* ε*-accurate in the sense of Proposition B.9 with probability at least* $1-1/m$ *whenever* $\eta \geq C\varepsilon^{-2}(D+\log m)$ *and the target is not capped, that is,* $\eta\, p_{\text{in}} \leq 10^6$*.*

No proof is claimed. The conjecture is plausible for the following reasons, which also say what a proof must handle. The estimated weight of a fixed node is a sum of stratified sample counts scaled by inverse inclusion probabilities, and for a node holding about L/m cells Bernstein's inequality gives a relative error of order $\sqrt{(D+\log m)/\eta}$. Every subtree is a node of depth at most D, so a union bound over the at most 2^{D+1} such nodes accounts for the subtrees being chosen from the same sample. A proof must also control the cells in the gaps between subtrees, which the sample does not see and `findSubtree` gives to a neighbour (Theorem B.8(iii)), and it must handle the resampled partitions. The cap matters: at 10^6 labels over 16384 partitions the effective η is about 61 before oversampling (§10.3), whatever the hint says. The library's default is $\eta = 20$ and its tests use $\eta = 100$; for a fixed η the conjectured ε does not tend to zero, so Proposition 13.4 with β near 2 is the realistic reading, not perfect balance. Range partitioning with exact ranks (§10.2, Phase 3) replaces the sample by the exact order and removes the conjecture from the balance argument; the halo direction of §16.12 adds a second question to it.

Marks. A conjecture; open. [LEAN: none.] [SIM: the empirical ε of the sample against η, D and m on the calibration runs, as the evidence the conjecture would need; to run, L1MDE-PROOF-SIMULATION.]

Bibliography

[AACS25] Anders Aamand, Maryam Aliakbarpour, Justin Y. Chen, and Sandeep Silwal. How fast can you find a good hypothesis?, 2025.

[AFJ+18] Jayadev Acharya, Moein Falahatgar, Ashkan Jafarpour, Alon Orlitsky, and Ananda Theertha Suresh. Maximum selection and sorting with adversarial comparators and an application to density estimation, 2018.

[AJ76a] Saab Abou-Jaoude. Conditions nécessaires et suffisantes de convergence L_1 en probabilité de l'histogramme pour une densité. *Annales de l'Institut Henri Poincaré, Section B*, 12(3):213–231, 1976.

[AJ76b] Saab Abou-Jaoude. Sur la convergence L_1 et L_∞ de l'estimateur de la partition aléatoire pour une densité. *Annales de l'Institut Henri Poincaré, Section B*, 12(4):299–317, 1976.

[Ald01] David J. Aldous. Stochastic models and descriptive statistics for phylogenetic trees, from Yule to today. *Statistical Science*, 16(1):23–34, February 2001.

[And66] T. W. Anderson. Some nonparametric multivariate procedures based on statistically equivalent blocks. In P. R. Krishnaiah, editor, *Multivariate Analysis*, pages 5–27. Academic Press, 1966. to verify: as SainudiinTengHarlowLee2012's reference list prints it.

[AS18] B. Avelin and R. Sainudiin. Exact Bayesian A/B testing using distributed fault-tolerant Moore rejection sampler, 2018. Preprint, 2 pages.

[BF06] Michael G. B. Blum and Olivier François. Which random processes describe the tree of life? A large-scale study of phylogenetic tree imbalance. *Systematic Biology*, 55(4):685–691, 2006.

[BKM19] Olivier Bousquet, Daniel M. Kane, and Shay Moran. The optimal approximation factor in density estimation. In *Proceedings of the Thirty-Second Conference on Learning Theory*, volume 99 of *Proceedings of Machine Learning Research*, pages 318–341, 2019.

[CC13] Timothy Chu and Sean Cleary. Expected conflicts in pairs of rooted binary trees. *Involve: A Journal of Mathematics*, 6(3):323–332, 2013.

[CERT10] Sean Cleary, Murray Elder, Andrew Rechnitzer, and Jennifer Taback. Random subgroups of Thompson's group F. *Groups, Geometry, and Dynamics*, 4(1):91–126, 2010.

[CFGS15] S. Cleary, M. Fischer, R. C. Griffiths, and R. Sainudiin. Some distributions on finite rooted binary trees. Research Report UCDMS2015/2, UCDMS, 2015.

[CFP96] J. W. Cannon, W. J. Floyd, and W. R. Parry. Introductory notes on Richard Thompson's groups. *L'Enseignement Mathématique (2)*, 42:215–256, 1996.

[CPT15] Sean Cleary, John Passaro, and Yasser Toruno. Average reductions between random tree pairs. *Involve: A Journal of Mathematics*, 8(1):63–69, 2015.

[CRW13] Sean Cleary, Andrew Rechnitzer, and Thomas Wong. Common edges in rooted trees and polygonal triangulations. *The Electronic Journal of Combinatorics*, 20(1), 2013. Paper P39.

[CSW16] Yu-Fang Chen, Lei Song, and Zhilin Wu. The commutativity problem of the MapReduce framework: A transducer-based approach, 2016.

[Dah19] Joel Dahne. Privacy and analysis of trajectories and co-trajectories. Examensarbete i matematik, 30 hp, Department of Mathematics, Uppsala University, July 2019. U.U.D.M. Project Report 2019:53.

[DF95] Robert P. Dobrow and James Allen Fill. On the markov chain for the move-to-root rule for binary search trees. *The Annals of Applied Probability*, 5(1):1–19, 02 1995.

[DG85] Luc Devroye and László Györfi. *Nonparametric Density Estimation: The L_1 View.* Wiley, New York, 1985.

[DL01] Luc Devroye and Gábor Lugosi. *Combinatorial Methods in Density Estimation.* Springer Series in Statistics. Springer, New York, 2001.

[Fil96] James Allen Fill. On the distribution of binary search trees under the random permutation model. *Random Structures & Algorithms*, 8(1):1–25, 1996.

[Fis22] Ronald A. Fisher. On the mathematical foundations of theoretical statistics. *Philosophical Transactions of the Royal Society of London. Series A*, 222:309–368, 1922.

[FS09] Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics.* Cambridge University Press, New York, NY, USA, 1 edition, 2009.

[GCI+17] Shelly Grossman, Sara Cohen, Shachar Itzhaky, Noam Rinetzky, and Mooly Sagiv. Verifying equivalence of Spark programs. In *Computer Aided Verification (CAV 2017)*, 2017.

[Ges70] M. P. Gessaman. A Consistent Nonparametric Multivariate Density Estimator Based on Statistically Equivalent Blocks. *The Annals of Mathematical Statistics*, 41(4):1344–1346, 1970.

[GGSZ14] S. Garcia, M. Grill, J. Stiborek, and A. Zunino. An empirical comparison of botnet detection methods. *Computers and Security Journal*, 45:100–123, 2014. to verify: as SainudiinTeng2019's reference list prints it.

[GHB+05] K. M. Górski, E. Hivon, A. J. Banday, B. D. Wandelt, F. K. Hansen, M. Reinecke, and M. Bartelmann. HEALPix: a framework for high-resolution discretization and fast analysis of data distributed on the sphere. *The Astrophysical Journal*, 622:759–771, 2005. arXiv:astro-ph/0409513.

[GL11] Alexander Goldenshluger and Oleg Lepski. Bandwidth selection in kernel density estimation: oracle inequalities and adaptive minimax optimality. *The Annals of Statistics*, 39(3):1608–1632, 2011.

[GM03] Alexander G. Gray and Andrew W. Moore. Nonparametric density estimation: toward computational tractability. In *Proceedings of the SIAM International*

Conference on Data Mining, 2003. to verify: as SainudiinTengHarlowLee2012's reference list prints it.

[God21] Simon Godskesen. Simulating transmission processes on networks. Bachelor's thesis in applied mathematics and statistics, Uppsala University, 2021.

[GR92] Andrew Gelman and Donald B. Rubin. Inference from iterative simulation using multiple sequences. *Statistical Science*, 7(4):457–472, 1992.

[Gra66] Ronald L. Graham. Bounds for certain multiprocessing anomalies. *The Bell System Technical Journal*, 45(9):1563–1581, 1966.

[Gra69] Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.

[Gra22] Johannes Graner. Scalable algorithms in nonparametric computational statistics. Master's thesis, Uppsala University, Uppsala, Sweden, 2022.

[Har12] Jennifer Harlow. MCMC histograms: random walk on a graph and independent Metropolis–Hastings Markov chains. Unpublished note, July 2012 (lamastex/manuscripts/HistogramNotes/MCMC/IMH/IndependentMH_-MCMC.tex). PENDING THE OP whether it may be cited (section 7, question 1), 2012.

[HHKR95] R. Hammer, M. Hocks, U. Kulisch, and D. Ratz. *C++ Toolbox for Verified Computing: Basic Numerical Problems*. Springer-Verlag, Berlin, Germany, 1995.

[HST12] Jennifer Harlow, Raazesh Sainudiin, and Warwick Tucker. Mapped regular pavings. *Reliable Computing*, 16:252–282, 2012.

[Hyn96] Rob J. Hyndman. Computing and graphing highest density regions. *The American Statistician*, 50(2):120–126, 1996.

[JKDW01] Luc Jaulin, Michel Kieffer, Olivier Didrit, and Éric Walter. *Applied Interval Analysis: With Examples in Parameter and State Estimation, Robust Control and Robotics*. Springer, London, 2001.

[KS93] M. Kirkpatrick and M. Slatkin. Searching for evolutionary patterns in the shape of a phylogenetic tree. *Evolution*, 47:1171–1181, 1993.

[lam] Mob-Spark: Scalable mobility data analysis over Apache Spark. https://github.com/lamastex/mob-spark. Scala and Spark code for analysis of co-trajectories, in particular privacy analysis of SwapMob. GitHub repository lamastex/mob-spark, MIT licence.

[lam26] lamastex. mob: a github repository of mobility research. https://github.com/lamastex/mob, 2026. Retrieved 2026-11-10.

[Lav92] Michael Lavine. Some aspects of Pólya tree distributions for statistical modelling. *Annals of Statistics*, 20:1222–1235, 1992.

[Lav94] Michael Lavine. More aspects of Pólya tree distributions for statistical modelling. *Annals of Statistics*, 22:1161–1176, 1994.

[Lin13] Jimmy Lin. Monoidify! Monoids as a design principle for efficient MapReduce algorithms, 2013.

[LN96] Gábor Lugosi and Andrew Nobel. Consistency of Data-Driven Histogram Methods for Density Estimation and Classification. *The Annals of Statistics*, 24(2):687–706, 1996.

[Mah92] Hosam M. Mahmoud. *Evolution of Random Search Trees*. Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley, New York, 1992.

[MC05] G. McGarvey and B. Cloitre. Sequence A000108. The On-Line Encyclopedia of Integer Sequences, 2005. to verify: as SainudiinTengHarlowLee2012's reference list prints it.

[Mei08] J. Meier. *Groups, Graphs and Trees: An Introduction to the Geometry of Infinite Groups*. Cambridge University Press, Cambridge, United Kingdom, 2008.

[Moo66] Ramon E. Moore. *Interval Analysis*. Prentice-Hall Series in Automatic Computation. Prentice-Hall, Englewood Cliffs, N.J., 1966.

[Moo00] Andrew W. Moore. The anchors hierarchy: using the triangle inequality to survive high dimensional data. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 397–405, 2000. to verify: TOMACS prints the venue as the Twelfth conference; UAI 2000 was the Sixteenth.

[MŠ07] Satyaki Mahalanabis and Daniel Štefankovič. Density estimation in linear time, 2007.

[Neu90] A. Neumaier. *Interval Methods for Systems of Equations*. Cambridge University Press, Cambridge, United Kingdom, 1990.

[OEI] Sequence A056971: Number of (binary) heaps on n elements. The On-Line Encyclopedia of Integer Sequences, https://oeis.org/A056971.

[R85] Jean-Luc Rémy. Un procédé itératif de dénombrement d'arbres binaires et son application à leur génération aléatoire. *RAIRO. Informatique théorique*, 19(2):179–195, 1985.

[Ram19] Kristoffer Ramström. Botnet detection on flow data using the reconstruction error from autoencoders trained on Word2Vec network embeddings. Master's thesis, Uppsala University, 2019. to verify: as SainudiinTeng2019's reference list prints it.

[S2 26] S2 Geometry Library. S2 cells. Developer guide, page s2cell_hierarchy, https://s2geometry.io, 2026. Retrieved 2026-11-10.

[Sai05] Raazesh Sainudiin. *Machine Interval Experiments*. PhD thesis, Cornell University, Ithaca, New York, 2005.

[Sai12] R. Sainudiin. Sequence A185155, The On-line Encyclopedia of Integer Sequences. published electronically, Feb 2012.

[San23] Axel Sandstedt. Scalable nonparametric l1 density estimation via sparse subtree partitioning. Master's thesis, Uppsala University, Statistics, AI and Data Science, 2023.

[Sch47] Henry Scheffé. A useful convergence theorem for probability distributions. *The Annals of Mathematical Statistics*, 18(3):434–438, 1947.

[SGWS23a] Axel Sandstedt, Johannes Graner, Tilo Wiklund, and Raazesh Sainudiin. SparkDensityTree: An Apache Spark library for scalable density estimation, anomaly detection and conditional density regression with universal performance guarantees through distributed sparse binary trees, September 2023.

[SGWS23b] Axel Sandstedt, Johannes Graner, Tilo Wiklund, and Raazesh Sainudiin. SparkDensityTree-examples: User-guide with examples for SparkDensityTree library, September 2023.

[SH10] Raazesh Sainudiin and Jennifer Harlow. A C++ class library for statistical set processing. In R. Bhatia, editor, *Short communications in mathematical software, International Congress of Mathematicians*, page 670, India, August 2010. Hindustan Book Agency. to verify: as TengKuhnSainudiin2012's reference list prints it.

[SHT13] R. Sainudiin, J. Harlow, and W. Tucker. There and back again: Split and prune to tighten. In *2013 IEEE International Conference on Fuzzy Systems*, 2013.

[SMT18] Julián Salas, David Megías, and Vicenç Torra. SwapMob: Swapping trajectories for mobility anonymization. In *Privacy in Statistical Databases: UNESCO Chair in Data Privacy, International Conference, PSD 2018, Valencia, Spain, September 26–28, 2018, Proceedings*, volume 11126 of *Lecture Notes in Computer Science*, pages 331–346, Cham, 2018. Springer International Publishing.

[SMT+20] Julián Salas, David Megías, Vicenç Torra, Marina Toger, Joel Dahne, and Raazesh Sainudiin. Swapping trajectories with a sufficient sanitizer. *Pattern Recognition Letters*, 131:474–480, 2020. Sec. 24 sufficient-sanitiser prior art. CC-BY. Archived sha256 39463d6dd883a8b2e232b80b90758405ede5c076926e454891f780018466fd95.

[SN06] Clayton Scott and Robert D. Nowak. Minimax-optimal classification with dyadic decision trees. *IEEE Transactions on Information Theory*, 52(4):1335–1353, 2006.

[SS03] Charles Semple and Mike Steel. *Phylogenetics*, volume 24 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, 2003.

[ST19] Raazesh Sainudiin and Gloria Teng. Minimum distance histograms with universal performance guarantees. *Japanese Journal of Statistics and Data Science*, 2:507–527, 2019. Published title; an earlier version circulated as "Minimum distance estimation with universal performance guarantees over statistical regular pavings".

[Sta97] Richard P. Stanley. *Enumerative combinatorics. Vol. 1*, volume 49 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, Cambridge, 1997. With a foreword by Gian-Carlo Rota, Corrected reprint of the 1986 original.

[Sta99] Richard P. Stanley. *Enumerative combinatorics. Vol. 2*, volume 62 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, Cambridge, 1999.

[STHL13] R. Sainudiin, G. Teng, J. Harlow, and D. S. Lee. Posterior expectation of regularly paved random histograms. *ACM Transactions on Modeling and Computer Simulation*, 23(26):6:1–6:20, 2013.

[STW20] Raazesh Sainudiin, Warwick Tucker, and Tilo Wiklund. Scalable multivariate histograms, 2020.

[STW+26] Raazesh Sainudiin, Warwick Tucker, Tilo Wiklund, Johannes Graner, Axel Sandstedt, Lauri Viitasaari, Pauliina Ilmonen, and Yannis G. Yatracos. Distributed sparse binary trees for L_1 minimum distance estimation of any unknown density, 2026. Manuscript, about to be submitted.

[SV15] R. Sainudiin and A. Véber. A Beta-splitting model for evolutionary trees. Research Report 2015/3, UCDMS, 2015.

[SW16] Raazesh Sainudiin and David Welch. The transmission process: A combinatorial stochastic process for the evolution of transmission trees over networks. *Journal of Theoretical Biology*, 410:137–170, 2016. CC-BY.

[SY13] Raazesh Sainudiin and Thomas York. An auto-validating, trans-dimensional, universal rejection sampler for locally Lipschitz arithmetical expressions. *Reliable Computing*, 18:15–54, 2013.

[SYH+18] Raazesh Sainudiin, Thomas York, Jennifer Harlow, Gloria Teng, Warwick Tucker, and Dillon George. MRS 2.0, a C++ class library for statistical set processing and computer-aided proofs in statistics. https://github.com/raazesh-sainudiin/mrs2, 2008–2018.

[The] The Apache Software Foundation. RDD Programming Guide: Accumulators. Apache Spark documentation. Accessed 2026-11-04.

[THS14] G. Teng, J. Harlow, and R. Sainudiin. L_1-consistent adaptive multivariate histograms from a randomized queue prioritized for statistically equivalent blocks. Research Report 2014/2, UCDMS, 2014.

[TKS12] Gloria Teng, Kenneth Kuhn, and Raazesh Sainudiin. Statistical Regular Pavings to Analyze Massive Data of Aircraft Trajectories. *Journal of Aerospace Computing, Information, and Communication*, 9(1):14–25, 2012.

[Tor23] Kristoffer Torp. Large scale mobility patterns from co-trajectories of mobile phone network data using L1 and weighted Jaccard distance measures. Master's thesis, Uppsala University, 2023. The thesis is dated December 2023; to verify on DiVA (record diva2:1879932) whether the publication year is 2024.

[Tuk47] John W. Tukey. Non-parametric estimation II. Statistically equivalent blocks and tolerance regions—the continuous case. *The Annals of Mathematical Statistics*, 18(4):529–539, 1947. as the reference list of SainudiinTeng2019, p. 527, prints it.

[vN63] John von Neumann. Various techniques used in connection with random digits. In A. H. Taub, editor, *John von Neumann: Collected Works*, volume V, pages 768–770. Pergamon Press, Oxford, 1963. Verified by $A^1{}_{0\text{-}0}$ through ias.research.search; earlier sources print the publisher as Oxford University Press.

[Wil91] David Williams. *Probability with Martingales*. Cambridge University Press, Cambridge, 1991. Theorem 14.2: Lévy's upward theorem.

[Wol57] J. Wolfowitz. The minimum distance method. *Annals of Mathematical Statistics*, 28:75–88, 1957.

[Yat85] Yannis G. Yatracos. Rates of convergence of minimum distance estimators and Kolmogorov's entropy. *Annals of Statistics*, 13:768–774, 1985.

[Yul25] George Udny Yule. A mathematical theory of evolution, based on the conclusions of Dr. J. C. Willis, F.R.S. *Philosophical Transactions of the Royal Society of London. Series B*, 213(402–410):21–87, 1925.

[YZXS11] Jing Yuan, Yu Zheng, Xing Xie, and Guangzhong Sun. Driving with knowledge from the physical world. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '11)*, pages 316–324, 2011. as salas2020sanitizer prints it, ref. [33].

[YZZ+10] Jing Yuan, Yu Zheng, Chengyang Zhang, Wenlei Xie, Xing Xie, Guangzhong Sun, and Yan Huang. T-drive: driving directions based on taxi trajectories. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS '10)*, pages 99–108, 2010. as salas2020sanitizer prints it, ref. [34].

[ZXW+16] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, et al. Apache Spark: a unified engine for big data processing. *Communications of the ACM*, 59(11):56–65, 2016.

About the author

Raazesh Sainudiin is a mathematical statistician, coder and farmer. After degrees in mathematics and biology (Minnesota State University, Mankato) and in biometrics and statistics (M.S. and Ph.D., Cornell University), Sainudiin was a Research Fellow of the Royal Commission for the Exhibition of 1851 at the University of Oxford, Lecturer and then Senior Lecturer at the University of Canterbury, New Zealand, and Associate Professor of Mathematics with specialization in Data Science at Uppsala University (2019–2025), where Sainudiin became Docent in Applied Mathematics and Statistics. In industry, Sainudiin was Principal Data Scientist at Combient and Director of Technical Strategy and Research at Combient Mix.

The research runs from mathematical and statistical genetics, epidemiology and physics, and the mathematical theology of ideologies in media (from the retweeting of politicians and hate groups in the 2016 US presidential election to the Black Lives Matter movement in social and mass media), through auto-validating simulation, interval arithmetic and computer-aided proofs, including mapped regular pavings, and computational nonparametric statistics, to scalable data science and distributed machine learning. Its applications reach from molecular evolution and the conservation of a critically endangered bird to air traffic, gas turbines, privacy in human mobility and word sense disambiguation, and it extends to mathematical art and the history of Indian combinatorics. Current research adds statistical contract theory and thermoeconomics. It has been funded by the Royal Societies of the United Kingdom, New Zealand and Sweden, the US National Science Foundation and the European Union's Marie Curie Actions, among others, and Sainudiin has lectured on it in many countries. Sainudiin co-founded and leads VakeWorks in Sweden and India, whose Laboratory for Mathematical Statistical Experiments wrote this book with a commons of human and digital agents. Since 2023 Sainudiin has also farmed at Miðgarðr, a field station and gene bank for the Swedish heritage breed of pigs (linderödssvin) in Torpshammar, Sweden.



Raaz resting with Alpha-13, our Matriarch Linderödssvin (Swedish heritage pig breed), Spring 2025.
Miðgarðr Family Farm and Genebank of VakeWorks AB, Gim 238, 84175 Torpshammar, Sweden.
Photo: Elisabeth Bolund.