Sampling from Discrete Distributions and Computing Fréchet Distances

Karl Bringmann

Dissertation for Obtaining the Title of Doctor of Engineering
of the Faculties of Natural Sciences and Technology
of Saarland University

Saarbrücken, Germany, 2014

Colloquium Details

Colloquium Date:	17.12.2014
Dean:	Prof. Dr. Markus Bläser *Universität des Saarlandes*
Chair:	Prof. Dr. Raimund Seidel *Universität des Saarlandes*
Examiners:	Prof. Dr. Dr. h.c. mult. Kurt Mehlhorn *Max-Planck-Institut für Informatik*
	Prof. Dr. Angelika Steger *ETH Zürich*
	Prof. Pankaj K. Agarwal, Ph.D. *Duke University*
Scientific Assistant:	Dr. Erik Jan van Leeuwen *Max-Planck-Institut für Informatik*

Abstract

In the first part of this dissertation, we study the fundamental problem of sampling from a discrete probability distribution. Specifically, given non-negative numbers $p_1, \ldots, p_n$ the task is to draw i with probability proportional to p_i. We extend the classic solution to this problem, Walker's alias method, in various directions:

1. We improve upon its space requirements by presenting optimal succinct sampling data structures.
2. We present improved trade-offs between preprocessing and query time for sorted inputs, and generalize this from proportional sampling to sampling subsets.
3. For Bernoulli, geometric, and binomial random variates we present optimal sampling algorithms on a bounded precision machine.
4. As an application, we speed up sampling of *internal diffusion limited aggregation*.

The second part of this dissertation belongs to the area of computational geometry and deals with algorithms for the Fréchet distance, which is a popular measure of similarity of two curves and can be computed in quadratic time (ignoring logarithmic factors). We provide the first conditional lower bound for this problem: No polynomial factor improvement over the quadratic running time is possible unless the Strong Exponential Time Hypothesis fails. Our various extensions of this main result include conditional lower bounds under realistic input assumptions, which do not match the known algorithms. We close this gap by presenting an improved approximation algorithm for the Fréchet distance.

Kurzfassung

Im ersten Teil dieser Dissertation untersuchen wir das fundamentale Problem des Ziehens einer Zufallsvariablen anhand ihrer Wahrscheinlichkeitsverteilung. Gegeben nicht-negative Zahlen $p_1, \ldots, p_n$ gilt es i mit Wahrscheinlichkeit proportional zu p_i zu ziehen. Wir erweitern die klassische Lösung dieses Problems in verschiedene Richtungen:

1. Wir entwerfen Datenstrukturen, die weniger Speicher benötigen.
2. Wir erhalten einen optimalen Kompromiss zwischen Vorberechnungs- und Anfragezeit für sortierte Eingaben sowie eine Verallgemeinerung auf das Ziehen von Teilmengen.
3. Für Bernoulli-, geometrische und Binomialverteilungen entwerfen wir optimale Algorithmen für ein Maschinenmodell mit beschränkter Präzision.
4. Als Anwendung verbessern wir die Simulation des *internen diffusionsbegrenzten Wachstums*.

Der zweite Teil dieser Dissertation gehört zum Gebiet der Geometrie und behandelt Algorithmen für die Fréchetdistanz, ein beliebtes Ähnlichkeitsmaß für Kurven, welches in quadratischer Zeit berechnet werden kann. Wir zeigen die erste bedingte untere Schranke für dieses Problem: Keine Verbesserung der quadratischen Laufzeit um einen polynomiellen Faktor ist möglich unter der starken Exponentialzeithypothese. Unsere verschiedenen Erweiterungen dieses Resultats beinhalten bedingte untere Schranken unter realistischen Eingabeannahmen, die nicht mit den bekannten Algorithmen übereinstimmen. Wir schließen diese Lücke mit einem verbesserten Approximationsalgorithmus für die Fréchetdistanz.

Acknowledgments

First and foremost, I want to thank my girlfriend Annika Haß for her absolute support and encouragement, for believing in me and for lifting my spirit in times of need. I also want to thank her for being interested in my scientific passions, and for sharing her own passions with me, showing me interesting sides of the humanities. I am very grateful for the last 7 years with her.

Second, I wish to thank my advisor Kurt Mehlhorn for giving me the freedom to explore my diverse scientific interests and for sharing his wisdom whenever I needed it. He has created an unequaled working atmosphere at the Max Planck Institute for Informatics that brought me in contact with many great researchers from various areas within theoretical computer science.

I also want to thank Tobias Friedrich, who was like an advisor for me from my undergraduate studies to the beginning of my doctoral studies. Since the day I got to know him, he always encouraged me to do research and offered numerous possibilities to collaborate. Without his guidance and him sharing his knowledge about the academic world I would not be where I am today.

My gratitude also goes to all my coauthors, from each of whom I learned a different technique, trick, or mental attitude: Victor Alvarez, S. Anand, Radu Curticapean, Benjamin Doerr, Christian Engels, Tobias Friedrich, Naveen Garg, Danny Hermelin, Christian Igel, Patrick Klitzke, Anton Krohmer, Fabian Kuhn, Amit Kumar, Marvin Künnemann, Kasper Green Larsen, Erik Jan van Leeuwen, Bodo Manthey, Kurt Mehlhorn, Matthias Mnich, Adrian Neumann, Frank Neumann, Konstantinos Panagiotou, Ueli Peter, B. V. Raghavendra Rao, Saurabh Ray, Thomas Sauerwald, Raimund Seidel, Jakub Sliacan, Alexandre Stauffer, He Sun, Henning Thomas, Thomas Voß, and Markus Wagner.

I gratefully acknowledge a *Google Europe Fellowship in Randomized Algorithms* for providing financial support for my studies.

Last but no least, I wish to thank my fellow students, who became very good friends over the years, for many interesting scientific, social, and political discussions, for many jokes, and for all the coffee we had together. In particular, I want to thank Marvin Künnemann, Johanna Goos, and Radu Curticapean for all of our shared good times.

Karl Bringmann
Saarbrücken, August 18, 2014

Preface

The first part of this dissertation focuses on algorithms and data structures for sampling from discrete distributions, and the second part deals with algorithms for the Fréchet distance. During my studies I explored many other research areas within theoretical computer science, with a strong focus on algorithms. This section therefore aims to give a brief overview of my work.

At the time of writing, I have published 26 papers in conferences proceedings, 7 papers in journals, and another 4 manuscripts on ArXiv, which are currently under submission. These papers are listed in the following. Please note that some of these publications predate the start of my doctoral studies in April 2011.

[1] K. Bringmann and M. Künnemann. "Improved approximation for Fréchet distance on c-packed curves matching conditional lower bounds." Submitted. 2014. arXiv: `1408.1340 [cs.CG]`.

[2] K. Bringmann. "Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails." In: *Proc. 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS'14)*. To appear. 2014.

[3] K. Bringmann, T. Friedrich, and A. Krohmer. "De-anonymization of Heterogeneous Random Graphs in Quasilinear Time." In: *Proc. 22th Annual European Symposium on Algorithms (ESA'14)*. To appear. 2014.

[4] K. Bringmann, T. Friedrich, and P. Klitzke. "Generic Postprocessing via Subset Selection for Hypervolume and Epsilon-Indicator." In: *Proc. 13th International Conference on Parallel Problem Solving from Nature (PPSN XIII)*. To appear. 2014.

[5] K. Bringmann, F. Kuhn, K. Panagiotou, U. Peter, and H. Thomas. "Internal DLA: Efficient Simulation of a Physical Growth Model." In: *Proc. 41th International Colloquium on Automata, Languages, and Programming (ICALP'14)*. Vol. 8572. LNCS. 2014, 247–258.

[6] K. Bringmann, T. Friedrich, and P. Klitzke. "Two-dimensional Subset Selection for Hypervolume and Epsilon-Indicator." In: *Proc. 16th Genetic and Evolutionary Computation Conference (GECCO'14)*. 2014, 589–596.

[7] K. Bringmann, T. Sauerwald, A. Stauffer, and H. Sun. "Balls into bins via local search: cover time and maximum load." In: *Proc. 31th Symposium on*

Theoretical Aspects of Computer Science (STACS'14). Vol. 25. LIPIcs. 2014, 187–198.

[8] K. Bringmann, C. Engels, B. Manthey, and B. Raghavendra Rao. "Random Shortest Paths: Non-Euclidean Instances for Metric Optimization Problems." In: *Algorithmica* (2014), 1–21.

[9] K. Bringmann, D. Hermelin, M. Mnich, and E. J. van Leeuwen. "Parameterized Complexity Dichotomy for Steiner Multicut." Submitted. 2013. arXiv: `1404.7006 [cs.DS]`.

[10] V. Alvarez, K. Bringmann, and S. Ray. "A Simple Sweep Line Algorithm for Counting Triangulations and Pseudo-triangulations." Submitted. 2013. arXiv: `1312.3188 [cs.CG]`.

[11] V. Alvarez, K. Bringmann, S. Ray, and R. Seidel. "Counting Triangulations Approximately." In: *Proc. 25th Canadian Conference on Computational Geometry (CCCG'13)*. 2013.

[12] K. Bringmann. "Bringing Order to Special Cases of Klee's Measure Problem." In: *Proc. 38th International Symposium on Mathematical Foundations of Computer Science (MFCS'13)*. Vol. 8087. LNCS. 2013, 207–218.

[13] K. Bringmann, C. Engels, B. Manthey, and B. Raghavendra Rao. "Random Shortest Paths: Non-Euclidean Instances for Metric Optimization Problems." In: *Proc. 38th International Symposium on Mathematical Foundations of Computer Science (MFCS'13)*. Vol. 8087. LNCS. 2013, 219–230.

[14] K. Bringmann and T. Friedrich. "Exact and efficient generation of geometric random variates and random graphs." In: *Proc. 40th International Colloquium on Automata, Languages, and Programming (ICALP'13)*. Vol. 7965. LNCS. 2013, 267–278.

[15] K. Bringmann, B. Doerr, A. Neumann, and J. Sliacan. "Online Checkpointing with Improved Worst-Case Guarantees." In: *Proc. 40th International Colloquium on Automata, Languages, and Programming (ICALP'13)*. Vol. 7965. LNCS. 2013, 255–266.

[16] S. Anand, K. Bringmann, T. Friedrich, N. Garg, and A. Kumar. "Minimizing Maximum (Weighted) Flow-time on Related and Unrelated Machines." In: *Proc. 40th International Colloquium on Automata, Languages, and Programming (ICALP'13)*. Vol. 7965. LNCS. 2013, 13–24.

[17] K. Bringmann and T. Friedrich. "Parameterized Average-Case Complexity of the Hypervolume Indicator." In: *15th Genetic and Evolutionary Computation Conference (GECCO'13)*. 2013, 575–582.

[18] K. Bringmann and K. G. Larsen. "Succinct Sampling from Discrete Distributions." In: *Proc. 45th Annual ACM Symposium on Symposium on Theory of Computing (STOC'13)*. 2013, 775–782.

[19] K. Bringmann, T. Friedrich, C. Igel, and T. Voß. "Speeding Up Many-Objective Optimization by Monte Carlo Approximations." In: *Artificial Intelligence* 204 (2013), 22–29.

[20] K. Bringmann and T. Friedrich. “Approximation quality of the hypervolume indicator.” In: *Artificial Intelligence* 195 (2013), 265–290.

[21] K. Bringmann and K. Panagiotou. “Efficient Sampling Methods for Discrete Distributions.” In: *Proc. 39th International Colloquium on Automata, Languages, and Programming (ICALP'12)*. Vol. 7391. LNCS. 2012, 133–144.

[22] K. Bringmann and T. Friedrich. “Convergence of Hypervolume-Based Archiving Algorithms II: Competitiveness.” In: *14th Genetic and Evolutionary Computation Conference (GECCO'12)*. 2012, 457–464.

[23] V. Alvarez, K. Bringmann, R. Curticapean, and S. Ray. “Counting Crossing-free Structures.” In: *Proc. 28th Annual Symposium on Computational Geometry (SoCG'12)*. 2012, 61–68.

[24] K. Bringmann. “An improved algorithm for Klee's measure problem on fat boxes.” In: *Computational Geometry: Theory and Applications* 45.5-6 (2012), 225–233.

[25] K. Bringmann and T. Friedrich. “Approximating the least hypervolume contributor: NP-hard in general, but fast in practice.” In: *Theoretical Computer Science* 425 (2012), 104–116.

[26] K. Bringmann, K. Mehlhorn, and A. Neumann. “Remarks on Category-Based Routing in Social Networks.” 2012. arXiv: `1202.2293 [cs.SI]`.

[27] K. Bringmann, T. Friedrich, F. Neumann, and M. Wagner. “Approximation-Guided Evolutionary Multi-Objective Optimization.” In: *Proc. 21st International Joint Conference on Artificial Intelligence (IJCAI'11)*. 2011, 1198–1203.

[28] K. Bringmann and T. Friedrich. “Convergence of Hypervolume-Based Archiving Algorithms I: Effectiveness.” In: *Proc. 13th Genetic and Evolutionary Computation Conference (GECCO'11)*. 2011, 745–752.

[29] T. Friedrich, K. Bringmann, T. Voß, and C. Igel. “The Logarithmic Hypervolume Indicator.” In: *Proc. 11th International Workshop on Foundations of Genetic Algorithms (FOGA'11)*. 2011, 81–92.

[30] K. Bringmann. “Klee's measure problem on fat boxes in time $O(n^{(d+2)/3})$.” In: *Proc. 26th Annual Symposium on Computational Geometry (SoCG'10)*. 2010, 222–229.

[31] K. Bringmann and T. Friedrich. “Tight Bounds for the Approximation Ratio of the Hypervolume Indicator.” In: *Proc. 11th International Conference on Parallel Problem Solving from Nature (PPSN XI)*. Vol. 6238. LNCS. 2010, 607–616.

[32] K. Bringmann and T. Friedrich. “The Maximum Hypervolume Set Yields Near-optimal Approximation.” In: *Proc. 12th Genetic and Evolutionary Computation Conference (GECCO'10)*. 2010, 511–518.

[33] K. Bringmann and T. Friedrich. “An Efficient Algorithm for Computing Hypervolume Contributions.” In: *Evolutionary Computation* 18.3 (2010), 383–402.

[34] K. Bringmann and T. Friedrich. “Approximating the volume of unions and intersections of high-dimensional geometric objects.” In: *Computational Geometry: Theory and Applications* 43.6-7 (2010), 601–610.

[35] K. Bringmann and T. Friedrich. “Approximating the least hypervolume contributor: NP-hard in general, but fast in practice.” In: *Proc. 5th International Conference on Evolutionary Multi-Criterion Optimization (EMO'09)*. 2009, 6–20.

[36] K. Bringmann and T. Friedrich. “Don't be greedy when calculating hypervolume contributions.” In: *Proc. 10th International Workshop on Foundations of Genetic Algorithms (FOGA'09)*. 2009, 103–112.

[37] K. Bringmann and T. Friedrich. “Approximating the volume of unions and intersections of high-dimensional geometric objects.” In: *Proc. 19th International Symposium on Algorithms and Computation (ISAAC'08)*. Vol. 5369. LNCS. 2008, 436–447.

These papers can roughly be categorized as follows. The papers [6, 12, 17, 24, 25, 30, 33–37] study variants of Klee's measure problem from the area of computational geometry, where the task is to compute the volume of the union of a set of n given axis-aligned boxes in d-dimensional Euclidean space. Most prominently, the papers [24, 30] present an improved algorithm for the special case of all boxes being cubes. The papers [34, 37] contain an approximation algorithm for a generalization of this problem. A large part of these papers [6, 12, 17, 25, 33, 35, 36] deals with the special case where all boxes have a common vertex at the origin, which is also called the hypervolume indicator problem. For this special case, the worst-case and average-case complexity is studied in [12, 17]. We also studied the complexity of computing the least contributor [25, 33, 35, 36], i.e., the box whose deletion reduces the volume of the union the least. More generally, the paper [6] investigates the problem of choosing the subset of k boxes maximizing the volume of their union, which is an optimization variant of Klee's measure problem.

The hypervolume indicator problem is motivated by the field of evolutionary multi-objective optimization, where it is used as an indicator for the quality of a population. Beyond questions of computational geometry, we studied properties of the hypervolume indicator that are related to evolutionary multi-objective optimization [4, 19, 20, 22, 27–29, 31, 32]. In particular, the papers [20, 29, 31, 32] relate the hypervolume indicator with classic notions of approximation and the papers [22, 28] study different strategies of selecting offspring populations to maximize the final hypervolume indicator from the perspective of online algorithms.

The papers [10, 11, 23] study another problem of computational geometry: Given a set of points in the plane we want to count the number of different straight-line triangulations of this point set. For this problem we designed improved exact algorithms as well as approximation algorithms, in particular for restricted classes of point sets.

The papers [3, 7–9, 13, 15, 16, 26] study various other problems related to efficient algorithms: the maximum load of a ball-into-bins process augmented by local search [7], the average-case approximation ratio of heuristics for NP-hard problems [8, 13], fixed-parameter tractability of the graph problem Steiner Multicut [9],

online checkpointing [15], scheduling with the objective of minimizing maximum flow-time [16], de-anonymizing social networks, modeled using inhomogeneous random graphs [3], and routing in social networks [26].

The first part of this dissertation is based on the papers [5, 14, 18, 21], which focus on problems related to sampling from a discrete distribution. Chapter 2 studies the space requirements of sampling data structures and is based on [18]. Chapter 3 consists of a significantly improved version of [21], dealing with sorted distributions and a generalization to sampling subsets. In Chapter 4 we consider special distributions that can be sampled efficiently and exactly on a bounded precision machine, this is based on [14] and partly on [5]. Finally, in Chapter 5 we present an application to sampling a model from physics, based on [5].

The second part of this dissertation is based on the more recent papers [1, 2], which consider the Fréchet distance, a popular measure of similarity of two curves. In Chapter 7 we prove a near-quadratic conditional lower bound for computing the Fréchet distance, based on [2]. Chapter 8 is based on [1] and deals with approximation algorithms for realistic input curves.

Contents

PART I

Sampling from Discrete Distributions

1 Introduction

Sampling from a probability distribution is a fundamental problem that lies at the heart of randomized computation, and has never been as important as today, as most sciences perform computer simulations of models involving randomness. We approach this problem area from an algorithm theory perspective. The central problem in the first part of this dissertation is *proportional sampling*, defined as follows. We are given non-negative numbers $p_1, \ldots, p_n$ that define a probability distribution on $\{1, \ldots, n\}$ by picking i with probability proportional to p_i, i.e., the probability of sampling i is $\frac{p_i}{\sum_j p_j}$. The task is to build a data structure that supports sampling from this distribution as a query. The classic solution to this problem is the alias method by Walker from '74 [38], which uses optimal $\mathcal{O}(1)$ query time and optimal $\mathcal{O}(n)$ preprocessing time, i.e., the time for building the data structure is $\mathcal{O}(n)$. We extend this classic result in different directions as follows.

Sampling algorithms such as the alias method are typically analyzed on the Real RAM, where every memory cell can store an arbitrary real number and space usage cannot be meaningfully analyzed. As we will see later, the alias method can also be implemented on a bounded precision machine, specifically the Word RAM. This allows one to analyze space usage and consider sampling as a problem in succinct data structures. We show that the alias method's space usage is sub-optimal, and present novel optimal succinct data structures for proportional sampling, see Section 1.5 and Chapter 2.

The preprocessing time of $\mathcal{O}(n)$ is only optimal for worst-case, unstructured input. As a well-motivated example of structured input, we consider the case of sorted input $p_1 \geqslant \ldots \geqslant p_n$, and present a novel data structure for proportional sampling with optimal trade-off between preprocessing and query time. In particular, we can achieve a preprocessing and query time of $\mathcal{O}(\log n / \log \log n)$, see Section 1.6 and Chapter 3.

Moreover, we consider a different sampling problem, which can be seen as a generalization of proportional sampling: In *subset sampling* we are given $p_1, \ldots, p_n$ and consider n independent events, where event i occurs with probability p_i. The task is to sample the set of occurring events. As for proportional sampling, we consider sorted and unsorted input sequences and in both cases present data structures with optimal preprocessing-query time trade-offs, see Section 1.6 and Chapter 3.

Faster sampling methods are known for special distributions such as Bernoulli, geometric, or binomial random variates. Again, these algorithms are analyzed on the Real RAM and are not exact on bounded precision machines. We present optimal

algorithms for sampling these special distributions on the Word RAM, see Section 1.7 and Chapter 4.

As an application, we use our insights for these fundamental sampling problems to speed up a model from physics. The *internal diffusion limited aggregation* model can trivially be sampled in time $\mathcal{O}(n^2)$. We achieve a reduced sampling time of $\mathcal{O}(n \log^2 n)$, see Section 1.8 and Chapter 5.

In the remainder of this chapter, we make this short discussion precise. First, in Section 1.1 we introduce the two machine models considered in this dissertation, the Real RAM and the Word RAM. We make the reader familiar with Walker's alias method and show how to implement it on a Word RAM in Section 1.2. In Section 1.3 we discuss related literature on sampling algorithms. We fix notation in Section 1.4. Our results are discussed in Sections 1.5 to 1.8 and proven in Chapters 2 to 5.

1.1. Machine Models

We discuss two variants of random access machines (RAMs). Both are abstract computational machine models with an arbitrary number of registers that can be indirectly addressed. The classic RAM has been introduced by Cook and Reckhow [39]. There, each register contains an arbitrarily large integer $\mathbb{N}_0$ and all basic mathematical functions can be performed in constant time. The two models relevant for this dissertation are the Real RAM, which allows computation even with real numbers, and the Word RAM, which only allows computation with bounded precision. The difference is what can be stored in the registers. In addition to the standard definitions, we assume that a uniform random number can be sampled in constant time.

1.1.1. Real RAM Model of Computation

The Real RAM is the main model of computability in computational geometry [40, 41] and is also used in numerical analysis. Here, each register can contain a real number in the mathematical sense. All basic mathematical functions can be computed in constant time. In particular, we will assume that the following operations take constant time:

- Accessing the content of any memory cell.
- Performing any basic arithmetic operation involving real numbers like addition, multiplication, division, comparison, truncation, and evaluating any fundamental function like exp and log.
- Generating a uniformly distributed real number in the interval $[0, 1]$.

The disadvantage of the model is that real numbers are infinite objects and all physical computers can only handle finite portions of these objects. Hence, algorithms designed for the Real RAM are typically not exact when run on real-life computers using floating point approximations. On the positive side, it is easy to design algorithms in this model; this holds in particular for sampling problems.

1.1.2. Word RAM Model of Computation

The Word RAM is a more realistic model of computation [42]. Here, each memory cell contains a *word*, i.e., an integer in the range $\{0, \dots, 2^w - 1\}$, where w is a parameter of the model. We make the usual assumption that $w = \Omega(\log n)$, to be able to store pointers to all input elements (using a constant number of words). The execution of basic instructions on words takes constant time. This includes bit level operations, such as AND, OR, and NOT, and arithmetic operations, like addition, multiplication, and integer division. For sampling we need an additional operation Rand that produces a random word in constant time,[1] i.e., we assume that we can draw w random bits in constant time. Thus, we can only draw a random number from a range $\{1, \dots, 2^\ell\}$, for a positive integer $\ell = \mathcal{O}(w)$, in constant worst-case time, i.e., from a range with size a power of 2. For all other ranges $\{1, \dots, k\}$, $k \in \mathbb{N}$, we can still uniformly sample from it in $\mathcal{O}(1)$ *expected* time when $k \leqslant 2^{\mathcal{O}(w)}$, e.g., by sampling in the range $\{1, \dots, 2^\ell\}$, where $2^\ell \geqslant k$ is the next power of 2, and rejecting as long as the sampled number does not lie in the desired range $\{1, \dots, k\}$.

We can compute with longer integers than w bits by representing them as lists of words. This allows one, e.g., to add two L-bit integers in time $\mathcal{O}(1 + L/w)$. In general, usual logical or arithmetic operations on two L-bit integers can be performed in time $\mathcal{O}(1 + (L/w)^{\mathcal{O}(1)})$. Moreover, we can use floating-point approximations of reals by storing both mantissa and exponent as long integers. This allows one to perform typical operations on two floating-point numbers with L-bit mantissas and E-bit exponents in time $\mathcal{O}(1 + ((L + E)/w)^{\mathcal{O}(1)})$.

Note that the Word RAM offers an intrinsic parallelism where, in constant time, an operation on w bits can be performed in parallel.

1.2. Walker's Alias Method

We first focus on the Real RAM. For proportional sampling, the input consists of non-negative reals $p_1, \dots, p_n$, and we want to build a data structure that supports the operation ProportionalSampling, which returns $i \in \{1, \dots, n\}$ with probability $\frac{p_i}{\sum_j p_j}$. We assume that $\sum_j p_j > 0$. Multiple ProportionalSampling queries shall be independent. This problem has a classic solution by Walker [38], with preprocessing time improved by Kronmal and Peterson [43]; see also [44] for an excellent explanation. The improved version of Walker's alias method needs $\mathcal{O}(n)$ preprocessing time, after which a ProportionalSampling query can be answered in $\mathcal{O}(1)$ worst-case time. While the query time bound is clearly optimal, we will see later that the preprocessing time is optimal as well.

The alias method works as follows. Consider n vases, where vase i contains an amount of $\frac{p_i}{\sum_j p_j}$ of some liquid. We say that vase i contains $\frac{p_i}{\sum_j p_j}$ probability mass

[1] If we instead can only generate a random bit in constant time, then we can clearly simulate Rand in time $\mathcal{O}(w)$. However, even for uniform sampling we need $\Omega(\log n)$ random bits, so that we cannot hope for better query times than $\Theta(\log n)$ in this case.

of outcome i. For preprocessing, we want to repeatedly pour liquid from some vase to some other vase until we end up with (1) all vases containing exactly $1/n$ probability mass and (2) each vase i containing probability mass of at most two different outcomes i and v_i; let q_i denote the probability mass of outcome i so that $1/n - q_i$ is the probability mass of outcome v_i.

This structure allows to quickly sample as follows. First sample a uniformly random $i \in \{1, \dots, n\}$. Then sample a Bernoulli random variate with success probability $n\,q_i$. If this random variate was successful, then return i, else return v_i. Note that we can sample this Bernoulli random variable by sampling a uniformly random number $r \in [0, 1]$ and checking whether $r \leqslant n\,q_i$.

Clearly, this procedure chooses a vase i and outcome j contained in this vase with probability equal to the probability mass of outcome j in vase i. Since we poured probability mass from some vases to others, but did not lose or gain any probability mass, the total probability mass of outcome i over all vases is $\frac{p_i}{\sum_j p_j}$. Hence, we sample i with probability proportional to p_i and Walker's alias method is an exact sampling algorithm.

It is left to show how to pour probability mass. To this end, we call a vase *high* if it contains more than $1/n$ probability mass, *low* if it contains less than $1/n$, and *good* if it contains exactly $1/n$ probability mass. The preprocessing has the property that any non-good vase i contains probability mass only of outcome i, and once a vase becomes good we do not touch it again. As long as not all vases are good, there exist a high and a low vase, since the sum of all probability masses is 1. Pick any high vase i and low vase j, say these vases contain $m_i > 1/n > m_j$ probability mass of i and j, respectively. Now we pour $1/n - m_j$ probability mass of vase i to vase j. This adds probability mass of a second outcome to vase j and makes vase j good. Vase i, on the other hand, may stay high or become good or low. After at most n such pouring steps, all vases are good and we achieved properties (1) and (2). To implement this preprocessing in time $\mathcal{O}(n)$, observe that we only need to maintain a list of high and a list of low vases. Then in every step we can determine a high and a low vase in constant time, and after each pouring step these lists can be updated in constant time.

This method is typically analyzed on the Real RAM, where analyzing space usage is not meaningful, since every memory cell may store an arbitrary real number so that an arbitrary amount of information can be stored in each memory cell. For this reason, we want to analyze the alias method on the Word RAM.

Our sampling problem is the following on the Word RAM: The input consists of non-negative integers $p_1, \dots, p_n$, each of w bits. Build a data structure that supports an operation ProportionalSampling, which returns $i \in \{1, \dots, n\}$ with probability p_i/S, where $S := \sum_j p_j$ is assumed to be positive. All invocations of ProportionalSampling shall be independent.

Is Walker's alias method efficient on the Word RAM? The biggest potential obstacle to this would be the computation with numbers of excessive bit length. However, closely looking at the Real RAM version of the data structure, one can see that all produced numbers are either integers less than n or rationals with denominator nS (more precisely, $\operatorname{lcm}(n, S)$). To see this, note that initially all probability

masses are $\frac{p_i}{S}$, which has denominator dividing nS. Moreover, adding or subtracting two rationals with denominator dividing nS results in a rational with denominator dividing nS. We can store such a number $\frac{q}{nS}$ by storing only its numerator $q \in \{0, \dots, nS\}$ using $\lceil \log(nS+1) \rceil$ bits. Since $S \leqslant n2^w$, the number of bits is bounded by $\lceil \log(n^2 2^w + 1) \rceil \leqslant w + 2\log n + \mathcal{O}(1) \leqslant \mathcal{O}(w)$. Thus, all considered numbers fit in $\mathcal{O}(1)$ words and can be processed in constant time. A second point that needs to be translated to the Word RAM is the sampling of a Bernoulli random variable. However, a Bernoulli random variable with success probability $\frac{q}{nS}$ can be produced by sampling a uniformly random number $r \in \{1, \dots, nS\}$ and testing whether $r \leqslant q$. Hence, the data structure can easily be adapted and has preprocessing time $\mathcal{O}(n)$.

There is one drawback: We need to draw uniformly in $\{1, \dots, n\}$ and in $\{1, \dots, nS\}$, which can only be done in $\mathcal{O}(1)$ *expected* time. Thus, Walker's alias method degenerates to $\mathcal{O}(1)$ *expected* query time, which is theoretically unappealing, but makes not much difference for practice. In any case, no guarantee on worst-case query time is possible for our Word RAM model, as any probability we can generate in a bounded number of steps has as denominator a power of 2, but n and S are not required to be powers of 2.

Considering space usage, for each vase i we need to store v_i, the second outcome that was added to this vase, and q_i, the probability mass of outcome i that is left in vase i. Since $v_i \in \{1, \dots, n\}$ we need $\log n + \mathcal{O}(1)$ bits to store v_i. The number q_i has denominator dividing nS and is at most $1/n$, so we can store $nSq_i \in \{0, \dots, S\}$. Since $S \leqslant n2^w$, we need $w + \log n + \mathcal{O}(1)$ bits to store nSq_i. In total, Walker's alias method needs $n(w + 2\log n + \mathcal{O}(1))$ bits of space. Thus the data structure has a space overhead of more than $2n \log n$ bits compared to just storing the input numbers. Using the terminology from the world of succinct data structures, we say that the data structure has *redundancy* $2n \log n + \mathcal{O}(n)$ bits.

In summary, Walker's alias method can be implemented on the Word RAM and needs $\Theta(n)$ preprocessing time, $\Theta(1)$ expected query time, and $n(w + 2\log n + \mathcal{O}(1))$ bits of space.

1.3. Other Related Work

The fundamental problem of the generation of random values from discrete and continuous distributions has been studied extensively in the literature, see, e.g., Devroye's monograph [45]. Large parts of the literature deal with special distributions such as geometric, binomial, or Bernoulli distributions. These algorithms typically assume the Real RAM model of computation and are, thus, in general not exact on bounded precision machines and cannot be meaningfully analyzed with respect to space usage. We will discuss this branch of the literature in more detail in Section 1.7.

In a seminal work, Knuth and Yao [46] initiated the study of the sampling power of various restricted devices, like finite-state machines. They devise algorithms trying to minimize the use of random bits. However, they do not guarantee efficient

precomputation on general sequences of probabilities, so that their results are incomparable to ours. These ideas have been further developed in [47, 48]. See [49] for a modern approach on sampling random variates such as $\mathrm{Ber}(\mathrm{e}^{-1})$ using simple programs without multi-precision arithmetic.

A dynamic version of proportional sampling, where the input $p_1, \dots, p_n$ may change over time, has been investigated in [50, 51]. Here, an update is of the form "set p_i to v" for some $i \in \{1, \dots, n\}$ and $v \in \mathbb{N}$. These papers achieve an update time of $\mathcal{O}(1)$, while still having optimal preprocessing time $\mathcal{O}(n)$ and expected query time $\mathcal{O}(1)$. Their query time guarantee is slightly weaker than Walker's, since they cannot achieve constant worst-case time, not even on the Real RAM.

In the remainder of this section, we discuss the folklore technique of *rejection sampling* (see, e.g., [52]) that will find numerous applications throughout the first part of this dissertation. Assume that we want to solve proportional sampling on $p_1, \dots, p_n$, and we already have an algorithm for sampling proportional to $\overline{p}_1, \dots, \overline{p}_n$, where $\overline{p}_i \geqslant p_i$ for all i. Then we obtain a sampling algorithm for $p_1, \dots, p_n$ as follows. First, we draw a sample i proportional to $\overline{p}_1, \dots, \overline{p}_n$. Then with probability $\frac{p_i}{\overline{p}_i}$ we return i and are done. Otherwise, we *reject* i, i.e., we throw away i and repeat the process from the beginning.

Note that this algorithm samples proportional to $p_1, \dots, p_n$: In every iteration of the implicit loop, we first sample i with probability $\overline{p}_i / \sum_j \overline{p}_j$, and then we return i with probability $p_i / \overline{p}_i$. Thus, in any iteration we sample i with probability $p_i / \sum_j \overline{p}_j$, i.e., with probability proportional to p_i. In total over all iterations, we still sample proportional to p_i, and since we eventually return a number in $\{1, \dots, n\}$ with probability 1, we sample i with probability $p_i / \sum_j p_j$.

The running time of this algorithm is governed by its expected number of rejections, i.e., number of iterations. Recall that in any iteration the probability of returning i is $p_i / \sum_j \overline{p}_j$. Summing up, in any iteration the probability of not rejecting is $\sum_j p_j / \sum_j \overline{p}_j$. Hence, the expected number of iterations is $\mathcal{O}(\sum_j \overline{p}_j / \sum_j p_j)$. Thus, to obtain a fast sampling algorithm for $p_1, \dots, p_n$ it suffices that $\sum_j \overline{p}_j$ is not much larger than $\sum_j p_j$.

Variants of this technique are present in large parts of the literature on sampling algorithms. Its use for proportional sampling is immediate from the above exposition, as we can reduce the problem to a restricted class of inputs $\overline{p}_1, \dots, \overline{p}_n$, e.g., we can enforce that every $\overline{p}_i$ is an (inverse) power of 2.

1.4. Notation

We say that a statement holds *with high probability* (w.h.p.) if it holds with probability $1 - \mathcal{O}(n^{-c})$ for a constant $c > 0$ that can be chosen arbitrarily large. We abbreviate $[n] = \{1, \dots, n\}$. We write $\ln x$ for the natural logarithm of x, $\log_t x = \ln x / \ln t$, and $\log x = \log_2 x$.

Specific notation, which is only used within a chapter, will be introduced in the respective chapters.

1.5. Succinct Sampling

The results in this section are proven in Chapter 2. We focus on the question of whether Walker's alias method has optimal space usage, or whether there are data structures for sampling from a discrete distribution that use less space. This question cannot be studied on the Real RAM, as any cell can store an infinite number of bits. However, on the Word RAM this question is well-defined. Specifically, we consider the following question: *On the Word RAM, is there a data structure for sampling from a discrete distribution with redundancy less than* $2n \log n + \mathcal{O}(n)$ *bits?* Of course, such a data structure should use the optimal $\mathcal{O}(n)$ preprocessing time and $\mathcal{O}(1)$ expected query time, if possible.

We answer this question in two ways, in both cases improving upon Walker's data structure. First, we consider the *systematic case*, in which the input is read-only and also available at query time. This is a reasonable model if the input numbers may not be overwritten by the data structure, or if the input numbers are available only implicitly, i.e., we can afford to recompute each p_i when needed, but we cannot afford to store each p_i explicitly. In this case we present a data structure that uses $\mathcal{O}(n + w)$ *redundant bits*. In fact, Walker's classic data structure is *not* systematic, so that all $n(w + 2 \log n + \mathcal{O}(1))$ bits stored in his solution are redundant, i.e., we improve by a factor of $\Theta(w) = \Omega(\log n)$ over the alias method. We then generalize this result to further reduce the redundancy, at the cost of increasing the query time, yielding the following trade-off.

Theorem 1.1. *For any* $1 \leqslant r \leqslant n$ *we can build a systematic data structure for* ProportionalSampling *with* $r + \mathcal{O}(w)$ *bits of redundancy,* $\mathcal{O}(n/r)$ *expected query time, and* $\mathcal{O}(n)$ *preprocessing time.*

The question arises of whether one could save more than a factor of $\Theta(w)$ while still having $\mathcal{O}(1)$ expected query time, or, more generally, whether one can reduce the product rt further, where r is the redundancy and t the expected query time. With the following theorem we prove that this is impossible, showing optimality of the trade-off between redundancy and query time in Theorem 1.1.

Theorem 1.2. *Consider any systematic data structure for sampling from a discrete distribution, having redundancy* r *and supporting* ProportionalSampling *in expected query time* t*. Then* $r \cdot t = \Omega(n)$*.*

This shows that the data structure of Theorem 1.1 is asymptotically optimal with respect to all three aspects: space usage, query time, and preprocessing time (for the latter see Theorem 1.7 in Section 1.6).

So far we considered the systematic case, in which the input is read-only and always available. In the *non-systematic case*, on the other hand, the preprocessing is given access to the input, but the query algorithm is not. Thus, the preprocessing has to encode the input in some way in the data structure it outputs (possibly just storing the input without modifications). It is not immediately clear that such a data structure even needs to use nw bits of space, since two different inputs $p_1, \ldots, p_n$

and $\hat{p}_1, \ldots, \hat{p}_n$ represent the same distribution if there exists some $\alpha > 0$ such that $p_i = \alpha\hat{p}_i$ for all i. E.g., answering *range minimum queries* in an array of n ordered elements (given two indices i and j, return the index of the minimum element in the sub-array from index i through j) can be done using only $\mathcal{O}(n)$ bits, although one might first expect that $\Omega(n \log n)$ bits are necessary, see, e.g., [53]. However, we prove that such savings are not possible for sampling. More specifically, we prove the following result:

Theorem 1.3. *Any non-systematic data structure for* ProportionalSampling *must use at least nw bits of space for any $1 \leqslant w = o(n)$ and sufficiently large n.*

Thus, in contrast to range minimum queries, it is not possible to save even a single bit. As mentioned, Walker's data structure is non-systematic and has space usage $n(w + 2\log n + \mathcal{O}(1))$ bits, i.e., redundancy $2n \log n + \mathcal{O}(n)$ when analyzed as a non-systematic data structure. Very surprisingly, we show that it is possible to vastly improve over this bound in the non-systematic case. More precisely, we show that we need only 1 redundant bit to achieve optimal query time and preprocessing time!

Theorem 1.4. *In the non-systematic case we can build a data structure for* ProportionalSampling *that needs $nw+1$ bits of space, $\mathcal{O}(1)$ expected query time, and $\mathcal{O}(n)$ preprocessing time.*

This is an astonishing result since it is the strongest obtained separation between the systematic and non-systematic case for *any* data structure problem: For redundancy r and expected query time t the optimal bound is $r \cdot t = \Theta(n)$ in the systematic case, while we have $r \cdot t = \mathcal{O}(1)$ in the non-systematic case. The largest previous separation was obtained for Rank and Select queries, where any systematic data structure must satisfy $r = \Omega((n/t)\log t)$ [54], while there exist non-systematic data structures achieving $r = \Theta(n/(\log n/t)^t) + \tilde{\mathcal{O}}(n^{3/4})$ [55].

Finally, we believe that our data structures are not only interesting from a theoretical point of view, but may also be of practical use. In fact, our systematic solution with $\mathcal{O}(1)$ query time is simpler than Walker's alias method, while our non-systematic solution with just 1 redundant bit is only slightly more involved. Furthermore, all constants hidden in the $\mathcal{O}$-notations are small.

Related Work in Succinct Data Structures In the field of succinct data structures, the focus is on designing data structures that have space requirements as close to the information theoretic minimum as possible, while still answering queries efficiently. Here, the space usage of a data structure is measured in the additive number of redundant bits used compared to the information theoretic minimum. As mentioned, previous work has focused on two types of data structures called systematic and non-systematic. Some of the most basic problems in the field include range minimum queries, Rank and Select. The systematic case is well understood for all three problems, with tight bounds for Rank and Select dating back to Raman et al. [56] and Golynski [54]. For constant query time, the redundancy needed for these two problems is $\Theta(n \log\log n/\log n)$. For range minimum, Brodal et al. [57] proved that any systematic data structure with redundancy r must have worst-case

query time $t = \Omega(n/r)$. They complemented this lower bound with a data structure matching the entire trade-off curve.

The strongest separation between the systematic and non-systematic case had been limited, until somewhat recently, to a mere $\log n$ factor in the redundancy for constant query time data structures, see, e.g., [58]. In fact, it had been generally believed that a stronger separation would not be possible for problems such as Rank and Select. This belief was disproved in the seminal paper of Pătraşcu [55]. Here, Pătraşcu demonstrated an exponential separation between the two cases by obtaining non-systematic Rank and Select data structures with redundancy $r = \Theta(n/(\log n/t)^t) + \tilde{\mathcal{O}}(n^{3/4})$. Observe that the redundancy goes down exponentially fast with t and that redundancy $\mathcal{O}(n/\log^c n)$ is possible in constant query time for any constant $c > 0$. Reducing the redundancy all the way to a constant while maintaining constant query time, as we do for our problem, was, however, proved impossible by Pătraşcu and Viola [59]. More specifically, they proved a redundancy lower bound of $r \geqslant n/(\log n)^{\mathcal{O}(t)}$, thus almost matching the upper bound of Pătraşcu, except when $t \geqslant \log^{1-o(1)} n$.

Finally, we mention an interesting problem for which extremely low redundancy and constant query time has been achieved before this work: The input to this problem consists of an array of n trits, i.e., numbers in $\{0, 1, 2\}$, and the goal is to represent the array in as close to $\lceil n \log 3 \rceil$ bits as possible, such that each entry can be retrieved efficiently. For this problem, Dodis et al. [60] showed that constant query time can be achieved with a constant number of input-dependent redundant bits plus $\mathcal{O}(\log^2 n)$ precomputed bits depending only on n and the word size. From a separation point of view, this problem is, however, not interesting, as the problem is not meaningful in the systematic setting.

1.6. Sorted Input and Subset Sampling

The results in this section are proven in Chapter 3. So far we studied proportional sampling on general, worst-case input sequences $p_1, \ldots, p_n$. Now we want to study *structured input*. In particular, many natural distributions have sequences $p_1, \ldots, p_n$ that change monotonicity only few times, meaning that they can be split into a small number of monotone sequences. We focus on these monotone sub-problems, i.e., we consider *sorted* $p_1 \geqslant \ldots \geqslant p_n$.

Moreover, we consider a second fundamental sampling problem, which we call *subset sampling*. Here, we have independent events $1, \ldots, n$ and event i occurs with probability p_i. The task is to sample the set of occurred events. Our results below show that this can be seen as a *generalization* of proportional sampling. We study this problem both with respect to general and sorted input sequences. As we will see, there is a rich interplay in designing efficient algorithms that solve these different variants.

In all cases we obtain efficient preprocessing-query time trade-offs which we prove to be asymptotically optimal.

1.6.1. Proportional Sampling

Recall that in proportional sampling we are given $\mathbf{p} = (p_1, \dots, p_n) \in \mathbb{R}^n_{\geqslant 0}$ and we want to sample a random variable $Y = Y_{\mathbf{p}}$ such that $\Pr[Y = i] = p_i/\mu$, where $\mu = \sum_{i=1}^n p_i$ is assumed to be positive. Depending on whether we consider this problem on sorted or general (i.e., not necessarily sorted) sequences we call the problem SortedProportionalSampling or UnsortedProportionalSampling in this section. Note that in the preceding sections we studied UnsortedProportionalSampling, in fact, all results for this problem variant that are listed in this section follow from Walker's alias method.

A *single-sample algorithm* for SortedProportionalSampling or UnsortedProportionalSampling is given $\mathbf{p}$ as input and outputs a number $s \in [n]$ that has the same distribution as Y. In order to allow for sublinear running times, when we speak of "input $\mathbf{p}$" we mean that the algorithm gets to know n and can access every p_i in constant time. This can be achieved by storing all p_i's in an array, but also, e.g., by a constant depth arithmetic circuit computing p_i. In particular, the algorithm does not know the number of i's with $p_i = 0$, and the input format is not sparse. For this problem we prove the following result.

Theorem 1.5. *There is a single-sample algorithm for* SortedProportionalSampling *with expected time* $\mathcal{O}\big(\frac{\log n}{\log\log n}\big)$ *and for* UnsortedProportionalSampling *with expected time* $\mathcal{O}(n)$*. Both bounds are asymptotically optimal.*

We remark that all our lower bounds only hold for algorithms that work for all n and *all* (sorted) sequences $p_1, \dots, p_n$. They are worst-case bounds over the input sequence $\mathbf{p}$ and asymptotic in n. For particular instances $\mathbf{p}$ there can be faster algorithms. To avoid any confusion, note that we mean worst-case bounds whenever we speak of *(running) time* and expected bounds whenever we speak of *expected (running) time*.

To obtain faster sampling times than in Theorem 1.5, we consider *sampling data structures* that support ProportionalSampling as a *query*. We view building the data structure as *preprocessing* of the input. More precisely, in this preprocessing-query variant we consider the interplay of two algorithms. First, the *preprocessing algorithm* P is given $\mathbf{p}$ as input and computes some auxiliary data $D = D(\mathbf{p})$. Second, the *query algorithm* Q receives input $\mathbf{p}$ and D, and samples Y, i.e., for any $s \in [n]$ we have $\Pr[Q(\mathbf{p}, D) = s] = \Pr[Y = s]$. Here, the probability is taken only over the random choices of Q, so that, after running the preprocessing once, running the query algorithm multiple times generates multiple independent samples. In this setting we prove the following tight result.

Theorem 1.6. *Let* $\beta \in \{2, \dots, n\}$*.* SortedProportionalSampling *can be solved in preprocessing time* $\mathcal{O}(\log_\beta n)$ *and expected query time*

$$t_q^\beta(n, \mu) = \mathcal{O}\big(\min\big\{\beta, \tfrac{\log n}{\log\log n}\big\}\big).$$

This is optimal, as for some constant $\varepsilon > 0$*,* SortedProportionalSampling *has no data structure with preprocessing time* $\varepsilon \log_\beta(n)$ *and expected query time* $\varepsilon\, t_q^\beta(n, \mu)$ *for any* $\mu = \mu(n)$*.*

Note that if we can afford a preprocessing time of $\mathcal{O}(\log n)$ then the query time is already $\mathcal{O}(1)$, which is optimal. Thus, larger preprocessing times cannot yield better query times. This is why we assume that the preprocessing time is of the form $\mathcal{O}(\log_\beta n)$. With varying $\beta \in \{2, \dots, n\}$ this takes all values in the interval $[\Theta(1), \Theta(\log n)]$ of reasonable values. Varying β yields a trade-off between preprocessing and query time; if one wants to have a large number of samples, one should set $\beta = 2$ to minimize query time, while large β yields superior running times if one wants only a small number of samples. Note that we prove a matching lower bound for this trade-off for all β.

For general input sequences, ProportionalSampling can be solved by Walker's alias method. This result will be used in the proofs of Theorem 1.9 and Theorem 1.10 below, so we include it here for completeness. Moreover, we prove a matching lower bound.

Theorem 1.7. UnsortedProportionalSampling *can be solved in preprocessing time $\mathcal{O}(n)$ and query time $\mathcal{O}(1)$. This is optimal, as for some constant $\varepsilon > 0$,* UnsortedProportionalSampling *has no data structure with preprocessing time εn and expected query time εn for any $\mu = \mu(n)$.*

Note that any data structure with preprocessing time t_p and expected query time t_q can be transformed into a single-sample algorithm with expected time $t_p + t_q$, so the single-sample variant of the problem is also solved by the preprocessing-query variant. Moreover, a single-sample algorithm with expected time t yields a data structure with no preprocessing time and expected query time t. These arguments show that Theorem 1.5 follows from Theorems 1.6 and 1.7.

1.6.2. Subset Sampling

In the previous section we considered the problem of sampling from a distribution. Here, we consider n independent events with indicator random variables $X_1, \dots, X_n$, and $\Pr[X_i = 1] = p_i$. As a shorthand, we write $\mathbf{p} = (p_1, \dots, p_n)$ and $\mu = \mu_{\mathbf{p}} = \sum_{i=1}^n p_i = \mathbb{E}\big[\sum_{i=1}^n X_i\big]$. Consider the random variable $X = X_{\mathbf{p}} = \{i \in [n] \mid X_i = 1\}$, i.e., the set of all events that occurred; in particular, for any $S \subseteq [n]$ we have

$$\Pr[X = S] = \Big(\prod_{i \in S} p_i\Big) \cdot \Big(\prod_{i \in [n] \setminus S} (1 - p_i)\Big).$$

We call the problem of sampling X SortedSubsetSampling or UnsortedSubsetSampling, if we consider it on sorted or general input sequences, respectively. As previously, we consider two variations of the problem. In the *single-sample* variant we are given $\mathbf{p}$ and we want to compute an output that has the same distribution as X. Moreover, in the *preprocessing-query* variant we have a precomputation algorithm that is given $\mathbf{p}$ and computes some auxiliary data D, and a query algorithm that is given $\mathbf{p}$ and D and has an output with the same distribution as X. The results of multiple calls to the query algorithm are independent.

Any query algorithm cannot run faster than $\Theta(1 + \mu)$, as its expected output size is μ and any algorithm requires a running time of $\Omega(1)$. Whether this query time

is achievable depends on μ and the alloted preprocessing time, as our results below make precise. Note that the single-sample variant of UnsortedSubsetSampling can be solved trivially in time $\mathcal{O}(n)$; we just toss a biased coin for every p_i. This algorithm is optimal, as shown by the following tight result.

Theorem 1.8. *There is a single-sample algorithm for* SortedSubsetSampling *with expected time*

$$t(n,\mu) = \begin{cases} \mathcal{O}(\mu), & \text{if } \mu \geqslant \frac{1}{2}\log n, \\ \mathcal{O}\big(1 + \frac{\log n}{\log(\frac{\log n}{\mu})}\big), & \text{otherwise,} \end{cases}$$

and for UnsortedProportionalSampling *with expected time $\mathcal{O}(n)$. Both bounds are asymptotically optimal for any $\mu = \mu(n)$.*

Again, this theorem follows from our results on the preprocessing-query variant presented in the following two theorems.

Theorem 1.9. *Let $\beta \in \{2, \dots, n\}$.* SortedSubsetSampling *can be solved in preprocessing time $\mathcal{O}(\log_\beta n)$ and expected query time*

$$t_q^\beta(n,\mu) = \begin{cases} \mathcal{O}(\mu), & \text{if } \mu \geqslant \frac{1}{2}\log n, \\ \mathcal{O}(1 + \beta\mu), & \text{if } \mu < \frac{1}{\beta}\log_\beta n, \\ \mathcal{O}\big(\frac{\log n}{\log(\frac{\log n}{\mu})}\big), & \text{otherwise.} \end{cases}$$

In particular, the query time is always bounded by $\mathcal{O}(1 + \beta\mu)$. This is optimal, as for some constant $\varepsilon > 0$, SortedSubsetSampling *has no data structure with preprocessing time $\varepsilon \log_\beta n$ and expected query time $\varepsilon\, t_q^\beta(n,\mu)$ for any $\mu = \mu(n)$.*

Observe that setting $\beta = 2$ in the above result yields a preprocessing time of $\mathcal{O}(\log n)$ and an (optimal) expected query time of $\mathcal{O}(1 + \mu)$.

The next result addresses the case of general, i.e., not necessarily sorted, probabilities.

Theorem 1.10. UnsortedSubsetSampling *can be solved in preprocessing time $\mathcal{O}(n)$ and expected query time $\mathcal{O}(1+\mu)$. This is optimal, as for some constant $\varepsilon > 0$,* UnsortedSubsetSampling *has no data structure with preprocessing time εn and expected query time εn for any $\mu = \mu(n)$.*

Both positive results in the previous theorems highly depend on each other. In particular, we prove them by repeatedly reducing the instance size n and switching from the one problem variant to the other.

We also present a relation between ProportionalSampling and SubsetSampling that suggests that the classic problem ProportionalSampling is the easier of the two problems (or can be seen as a special case of SubsetSampling). Specifically, we present a reduction that allows one to infer the upper bounds for ProportionalSampling (Theorems 1.6 and 1.7) from the upper bounds for SubsetSampling (Theorems 1.9 and 1.10), see Section 3.3 for details.

Related work for Subset Sampling A classic algorithm solves SubsetSampling for $p_1 = \ldots = p_n = p$ in the optimal expected time $\mathcal{O}(1 + \mu)$, see, e.g., the monographs [45] and [52], where also many other cases are discussed. Indeed, observe that the index i_1 of the first sampled element is geometrically distributed, i.e., $\Pr[i_1 = i] = (1-p)^{i-1}p$. Such a random value can be generated by setting $i_1 = \lfloor \frac{\log r}{\log(1-p)} \rfloor$, where r is chosen uniformly at random in $(0, 1)$. Moreover, after having sampled the index of the first element, we repeat the process starting at $i_1 + 1$ to sample the second element, and so on, until we arrive for the first time at an index $i_k > n$. In [61] the "orthogonal" problem is considered, where we want to uniformly sample a fixed number of elements from a stream of objects. The problem of UnsortedSubsetSampling was considered also in [62], where algorithms with linear preprocessing time and sub-optimal query time $\mathcal{O}(\log n + \mu)$ were designed. Our results improve upon this running time, and provide matching lower bounds.

1.6.3. Real RAM vs Word RAM

We present our results of Chapter 3 on the Real RAM model of computation. However, in Section 3.4 we argue that our algorithms can also be adapted to work on the Word RAM model of computation, using that geometric random variates can be sampled exactly and efficiently on the Word RAM, see Section 1.7 and Chapter 4. The lower bounds hold for both models since we bound the number of probed inputs p_i, and in both models in unit time we can only read a single input p_i.

1.7. Sampling from Special Distributions

The results in this section are proven in Chapter 4. We consider special distributions like the geometric, binomial, or Bernoulli distribution. Folklore or classic results show that such random variates can be efficiently sampled; we discuss these results in detail below. These algorithms are, however, typically analyzed on the Real RAM, which is highly problematic as real numbers are infinite objects and all physical computers can only handle finite portions of these objects. Typical implementations with, e.g., double floating point precision are efficient but *not exact*, i.e., some outcomes might not be reachable and others might become more likely than they should.

Our aim is the design of exact and efficient algorithms for sampling from special distributions on a bounded precision machine, specifically on the Word RAM. We show that this is possible in many cases and present fast algorithms in particular for Bernoulli, geometric, and binomial random variates. As an application, this yields exact and efficient sampling algorithms for Erdős-Rényi and Chung-Lu random graphs. It also allows the exact and efficient generation of very large non-uniform random variates (e.g., for cryptographic applications [63]).

For exponential and normal distributions, there already exist implementations of exact and efficient random number generators [64], which work similarly to our algorithms for Bernoulli, geometric, and binomial random variates.

1.7.1. Bernoulli Random Variates

A Bernoulli random variate $\mathrm{Ber}(p)$, $p \in [0,1]$, has outcome 1 with probability p and outcome 0 otherwise. To sample this on a Real RAM, simply draw a uniformly random real $r \in [0,1]$ and return whether $r \leqslant p$. This is an exact sampling algorithm, since $r \leqslant p$ happens with probability p. Moreover, as we can draw r in constant time by assumption, this simple algorithm takes $\mathcal{O}(1)$ time.

If p is a rational $\frac{a}{b}$ with small denominator b, then on the Word RAM we can sample $\mathrm{Ber}(p)$ by drawing a uniformly random number $s \in \{1, \ldots, b\}$ and returning whether $s \leqslant a$. However, if b is large or we can only approximate p then this method is infeasible.

We describe a simple method for generating Bernoulli random variates on a Word RAM, which is closely related to [48] and cannot be found in standard text books. We simulate the Real RAM algorithm, which draws $r \in [0,1]$ uniformly at random and compares it with p, by computing *approximations* of r and p. For r, an i-bit approximation consists of i random bits. For p, we assume that we can compute an i-bit approximation in time $i^{\mathcal{O}(1)}$, although it would suffice if such an approximation could be computed in time $\mathcal{O}((2-\varepsilon)^i)$ for some $\varepsilon > 0$. If r and p are sufficiently far apart, then we can decide whether $r \leqslant p$ just by comparing their i-bit approximations. Otherwise we increase the precision parameter i and repeat. Since r is uniformly random in $[0,1]$, it is unlikely to lie too close to p, and one can show that this yields an algorithm with expected constant running time. See Section 4.2 for details.

Theorem 1.11. *Let $p \in [0,1]$ and assume that for any $i \in \mathbb{N}$ we can compute a number $p_i \in [p - 2^{-i}, p + 2^{-i}]$ in time $i^{\mathcal{O}(1)}$. Then the Bernoulli random variate $\mathrm{Ber}(p)$ can be sampled in expected running time $\mathcal{O}(1)$ on a Word RAM.*

1.7.2. Geometric Random Variates

A geometric random variate $\mathrm{Geo}(p)$, $p \in (0,1]$, counts the number of times we can draw $\mathrm{Ber}(p)$ obtaining only zeroes before we see the first 1. Clearly, we can follow this definition and sample $\mathrm{Geo}(p)$ by repeatedly sampling $\mathrm{Ber}(p)$. This method has expected running time $\mathcal{O}(1/p)$, which is *not efficient* for p close to 0, i.e., the asymptotic running time in terms of the parameter p is far from being optimal.

On the Real RAM, a folklore algorithm for sampling geometric random variates in constant time works as follows. We draw $r \in (0,1)$ uniformly at random and return

$$\left\lfloor \frac{\log(r)}{\log(1-p)} \right\rfloor.$$

This takes value $k \in \mathbb{N}$ if and only if $k \leqslant \log(r)/\log(1-p) < k+1$, which happens if and only if $(1-p)^k \geqslant r > (1-p)^{k+1}$, which happens with probability $(1-p)^k - (1-p)^{k+1} = p(1-p)^k$, and this is equal to the probability of $\mathrm{Geo}(p) = k$. Thus, this method exactly samples the geometric random variate $\mathrm{Geo}(p)$.

We consider the problem of sampling geometric random variates on the Word RAM. We first observe the following lower bound for any exact sampling algorithm,

which follows from the expected output size being $\Omega(\log(1/p))$ bits. Recall that w is the word size of our Word RAM, i.e., the number of bits in any cell.

Theorem 1.12. *On a Word RAM, any algorithm sampling a geometric random variate* $\mathrm{Geo}(p)$ *with parameter* $p \in (0,1]$ *has expected running time* $\Omega(1+\log(1/p)/w)$.

The Real RAM algorithm can be translated to the Word RAM by computing $\log(r)$ and $\log(1-p)$ using multi-precision arithmetic with a precision that is high enough to determine $\lceil \log(r)/\log(1-p) \rceil$. This simple approach yields the following result, which we prove in Section 4.3.

Theorem 1.13. *On a Word RAM with word size* $w = \Omega(\log\log(1/p))$*, a geometric random variate* $\mathrm{Geo}(p)$ *with parameter* $p \in (0,1]$ *can be sampled in expected running time* $\mathcal{O}(1 + \log(1/p)\operatorname{poly}\log\log(1/p)/w)$.

To the best of our knowledge, this observation is not discussed in the literature so far. It not only applies to geometric distributions, but to all distributions where the inverse of the cumulative distribution is efficiently computable on a Word RAM. The assumption on w is needed to handle pointers to an array as large as the expected output size in constant time. This result is independent of the rest of this section and demonstrates that the classical Real RAM algorithm implemented on a Word RAM *does not give an optimal running time* matching Theorem 1.12, since this algorithm, as well as many other approaches, *does not avoid taking logarithms.* Note that it is a long-standing open problem in analytic number theory and computational complexity whether the logarithm can be computed in linear time.

Our aim is a Word RAM algorithm which samples geometric random variates exactly and in the optimal running time. In Section 4.4 we present an algorithm that proves the following theorem. Here, we assume that a relative 2^{-i}-approximation of p can be computed in time $i^{\mathcal{O}(1)}$, see Section 4.1 for details.

Theorem 1.14. *On a Word RAM with word size* $w = \Omega(\log\log(1/p))$*, a geometric random variate* $\mathrm{Geo}(p)$ *with parameter* $p \in (0,1]$ *can be sampled in expected running time* $\mathcal{O}(1 + \log(1/p)/w)$*, which is optimal.*

Observe that, as a sample of a geometric random variate can be arbitrarily large, the aforementioned sampling algorithm cannot work in bounded worst-case time or space. We remark that on a parallel machine with P Word RAM processors, the running time decreases to $\mathcal{O}(1 + \log(1/p)/(wP))$.

1.7.3. Bounded Geometric Random Variates

For our applications in sampling random graphs, we need to sample a *bounded geometric random variate* $\mathrm{Geo}(p,n) := \min\{n, \mathrm{Geo}(p)\}$ with $p \in (0,1]$ and $n \in \mathbb{N}$. Similarly to Theorem 1.12 for (unbounded) geometric random variates, we observe the following lower bound.

Theorem 1.15. *On a Word RAM, any algorithm sampling a bounded geometric random variate* $\mathrm{Geo}(p,n)$ *with parameters* $p \in (0,1]$ *and* $n \in \mathbb{N}$ *has expected running time* $\Omega(1 + \log(\min\{1/p, n\})/w)$.

We present an algorithm which achieves this optimal running time bound and prove the following theorem in Section 4.5. Again, we assume that a relative 2^{-i}-approximation of p can be computed in time $i^{\mathcal{O}(1)}$, see Section 4.1 for details.

Theorem 1.16. *On a Word RAM with word size $w = \Omega(\log\log(1/p))$, a bounded geometric random variate* $\mathrm{Geo}(p, n)$ *with parameters $p \in (0, 1]$ and $n \in \mathbb{N}$ can be sampled in expected running time $\mathcal{O}(1 + \log(\min\{1/p, n\})/w)$, which is optimal.*

If p is a rational number with numerator and denominator fitting in $\mathcal{O}(1)$ words, then our algorithm needs $\mathcal{O}(n)$ space in the worst case.

If p is an arbitrary real, then we cannot bound the worst-case space usage of a sampling algorithm for $\mathrm{Geo}(n, p)$ in general. However, if p is a rational with numerator and denominator fitting into a constant number of words of the Word RAM, then Theorem 1.16 shows that this is indeed possible.

1.7.4. Random Graphs

As an application of our sampling algorithms for bounded and unbounded geometric random variates, we present optimal sampling algorithms for Erdős-Rényi and Chung-Lu random graphs on the Word RAM.

This is motivated by the fact that a large fraction of empirical research on graph algorithms is performed on random graphs. Random graph generation is also commonly used for simulating networking protocols on the Internet topology and the spread of epidemics (or rumors) on social networks (see, e.g., [65]). It is also an important tool in real world applications such as detecting motifs in biological networks (see, e.g., [66]). There is a large body of work on generating random regular graphs (see, e.g., [67]), graphs with a prescribed degree distribution (see, e.g., [68]), and graphs with a prescribed joint degree distribution (see, e.g., [69]). All these algorithms converge to the desired distribution for $n \to \infty$. Note that this typically implies for finite n that only an approximation of the true distribution is achieved.

The most studied random graph model is certainly the Erdős-Rényi [70] random graph $\mathcal{G}(n, p)$, where each edge of a graph of n vertices is present independently with probability $p \in [0, 1]$. Many experimental papers use algorithms with running time $\Theta(n^2)$ to draw from $\mathcal{G}(n, p)$. The reason for this is probably that most graph algorithm software libraries such as JUNG, LEDA, BGL, and JDSL also do not contain efficient random graph generators. However, there are several algorithms which can sample from $\mathcal{G}(n, p)$ in expected time $\mathcal{O}(m + n)$ on a Real RAM, where $m = \Theta(pn^2)$ is the expected number of edges [71, 72]. This is done by using the fact that in an ordered list of all $\Theta(n^2)$ pairs of vertices the distance between two consecutive edges is geometrically distributed. Using the Real RAM algorithm for sampling geometric random variates, the resulting distribution is *not exact* if the algorithm is run on a physical computer, which can only handle bounded precision. The available implementation in the library NetworkX [73] therefore also does not return the desired distribution exactly. It is not obvious how to get an exact implementation even by using algebraic real numbers [74] and/or some high accuracy floating-point representation. The problem of sampling $\mathcal{G}(n, p)$ on a bounded precision model has been studied by Blanca and Mihail [75]. They showed how to achieve an approximation

of the desired distribution efficiently. Our aim is an *exact and efficient* generation on a bounded precision model instead.

We also consider the Chung-Lu random graph $\mathcal{G}(n, W)$ [76]. In this model, we are given $n \in N$ and $W = (W_1, \ldots, W_n) \in \mathbb{R}^n_{>0}$ and build a graph on $[n]$ by connecting $i, j \in [n]$, $i \neq j$, independently with probability $\min\{W_i W_j / \sum_k W_k, 1\}$. Usually the weights W follow a power-law distribution. In this case, the expected number of edges m is $\Theta(\sum_{i=1}^n W_i)$ (in general, the expected number of edges m is only bounded from above by $\mathcal{O}(\sum_{i=1}^n W_i)$). Moreover, in this case Miller and Hagberg [72] gave an algorithm to sample Chung-Lu random graphs in expected time $\mathcal{O}(n+m)$ on the Real RAM.

Our Results We focus on homogeneous and inhomogeneous random graphs and consider Erdős-Rényi [70] and Chung-Lu graphs [76]. The key ingredient for generating such graphs with n vertices faster than the obvious $\Theta(n^2)$ algorithm is an efficient algorithm for sampling geometric random variates. In fact, our results below follow from plugging our algorithm for sampling geometric random variables on the Word RAM into the best algorithm known for sampling the respective graph class.

For generating graphs with n vertices it is natural to assume $w = \Omega(\log n)$.

Theorem 1.17. *On a Word RAM with word size $w = \Omega(\log n)$, the Erdős-Rényi random graph $\mathcal{G}(n, p)$ can be sampled in expected time $\mathcal{O}(n+m)$, where $m = \Theta(pn^2)$ is the expected number of edges. This is optimal if $w = \mathcal{O}(\log n)$. If p is a rational number with numerator and denominator fitting into $\mathcal{O}(1)$ words, then the worst-case space complexity of our algorithm is asymptotically equal to the size of the output graph, which is optimal.*

For Chung-Lu random graphs, we assume that the expected number of edges is $\Theta(\sum_{i=1}^n W_i)$. In particular, this holds if W follows a power-law distribution. In this case, we obtain the following result.

Theorem 1.18. *Let $W = (W_1, \ldots, W_n)$ be rationals with common denominator, where all numerators and the common denominator fit into $\mathcal{O}(1)$ words. Then on a Word RAM with word size $w = \Omega(\log n)$, the Chung-Lu random graph $\mathcal{G}(n, W)$ can be sampled in expected time $\Theta(n+m)$, where m is the expected number of edges. This is optimal if $w = \mathcal{O}(\log n)$. The worst-case space complexity of the algorithm is asymptotically equal to the size of the output graph, which is optimal.*

1.7.5. Binomial Random Variates

A binomial random variate $\mathrm{Bin}(n, \frac{1}{2})$ counts the number of times we see heads when flipping n unbiased coins. Since flipping a coin is equivalent to sampling $\mathrm{Ber}(\frac{1}{2})$, we can turn this definition into an algorithm to sample $\mathrm{Bin}(n, \frac{1}{2})$ in time $\Theta(n)$.

The literature on sampling contains a large amount of algorithms that sample binomial random variates much faster, specifically in constant time (see, e.g., [45]), and implementations of these algorithms are readily available. However, all of these algorithms are exact only in the Real RAM model of computation.

On a Word RAM we assume $w = \Omega(\log n)$, so that the output fits in a constant number of cells. Until recently there was no algorithm known that samples $\mathrm{Bin}(n, \frac{1}{2})$ faster than the trivial coin flipping on the Word RAM. The first algorithm with sublinear preprocessing and polylogarithmic sampling time was proposed in [77]. We improve upon this algorithm here and show that constant expected sampling time is possible, even without any preprocessing.

Theorem 1.19. *On the Word RAM with $w = \Omega(\log n)$, a binomial random variable $\mathrm{Bin}(n, \frac{1}{2})$ can be sampled in expected time $\mathcal{O}(1)$. Moreover, our algorithm has a running time that is larger than $t > 0$ with probability $\exp(-t^{\Omega(1)})$.*

1.8. Application: Sampling a Model from Physics

The results in this section are proven in Chapter 5. As an application of our insights for the fundamental sampling problems discussed above, we consider a model that was introduced in the physics community, but has also drawn attention of mathematicians as well as computer scientists. Our general goal is to design more efficient algorithms for generating such models in order to allow larger computer experiments. Moreover, we want these algorithms to sample from the exact distribution, maximizing reliability of their results. Indeed, for the simple model that we consider we present an exact sampling algorithm with significantly reduced running time.

Internal diffusion limited aggregation (IDLA) is a random process that places n particles on the two-dimensional integer grid $\mathbb{Z}^2$. Let $A(i) \subset \mathbb{Z}^2$ denote the set of occupied grid points after placing i particles. The first particle is placed on the origin, i.e., $A(1) = \{(0, 0)\}$. From there on, $A(i + 1)$ is constructed from $A(i)$ by adding the first grid point in $\mathbb{Z}^2 \setminus A(i)$ that is reached by a random walk on $\mathbb{Z}^2$ starting at the origin.

Particle diffusion processes are of considerable significance in various branches of science. In fact, the IDLA process was introduced by Meakin and Deutch [78], who used it as a model to describe the dynamics of certain chemical and physical processes like corrosion or the melting of a solid around a source of heat. Since then, the study of the typical properties of $A(n)$, and most prominently its "shape," has been the topic of many works. In particular, numerical simulations in [78] indicated that the surface of $A(n)$ is typically extremely smooth such that the fluctuations from a perfect circle are only of logarithmic order. Proving this rigorously turned out to be a difficult and challenging mathematical problem, which was resolved only recently, after many attempts by several different authors (see, e.g., [79–82]), by Jerison, Levine and Sheffield [83].

We try to understand IDLA from a computational perspective by giving an efficient algorithm for determining the set $A(n)$. This line of research is driven by the pursuit to get efficient algorithmic tools for coping with random walks and by the wish to speed up models from physics, so that one may perform larger experiments. Moreover, understanding such models from a computational perspective might add to their understanding in general.

Using the aforementioned results it is easy to see that a direct simulation of every individual step for determining $A(n)$ is likely to require a total time of $\Omega(n^2)$, i.e., time $\Omega(n)$ per particle. Indeed, since $A(n)$ typically resembles a perfect circle, it has a radius of order $n^{1/2}$. Moreover, the random walk of a particle can be viewed as a combination of two independent one-dimensional random walks, one along the horizontal and one along the vertical axis. Thus, if a particle is placed initially at the origin, one of these two random walks has to travel a distance of order $n^{1/2}$ in some direction in order to escape $A(n)$. A quadratic running time then follows immediately from the well-known fact that a one-dimensional random walk of length ℓ in expectation only deviates $\Theta(\ell^{1/2})$ hops from its initial position.

The computational complexity of determining $A(n)$ was studied by Moore and Machta [84]. Among other results they showed that the simulation of IDLA (given a string of random bits) is complete for the class $\mathcal{CC}$ (even in the case of one particle), which is the subset of $\mathcal{P}$ characterized by circuits that are composed of comparator gates only. Moreover in [85], Friedrich and Levine give an algorithm that samples $A(n)$. They do not provide an analysis of the complexity (and it seems a quite difficult task to do so), but their experiments indicate that it scales like $\mathcal{O}(n^{3/2})$, while they inherently use space $\Omega(n)$.

Our Results We develop time and space-efficient algorithms for determining the set $A(n)$. We present the first algorithm that provably improves upon the "naive" step-by-step simulation of the particles. Our best algorithm yields the following result. It works on the Real RAM as well as the Word RAM.

Theorem 1.20. *IDLA can be simulated in $\mathcal{O}(n \log^2 n)$ time and $\mathcal{O}(n^{1/2} \log n)$ space, both in expectation and with high probability.*

Our algorithm simulates all particles consecutively. It crucially uses that the shape of $A(n)$ is almost a perfect circle, as discussed above. Let the *in-circle* be the largest circle centered at the origin that contains only occupied grid points. As long as the current particle $n+1$ is within the in-circle of $A(n)$, the random walk will typically stay in $A(n)$ for many steps. More precisely, if the current distance of the particle to the in-circle of $A(n)$ is d, then typically in the next $\Theta(d^2)$ random walk steps the particle will stay in $A(n)$. We want to utilize this fact by combining many steps to a single *jump* of the particle, without simulating all of these steps explicitly. Building on this, we use drift analysis to show that typically $\mathcal{O}(\log n)$ such jumps are sufficient to simulate one particle. Intuitively, such a combination of steps to a jump simply amounts to sampling the position of the particle after $T \approx d^2$ steps, which can be done by sampling two binomial random variables $\mathrm{Bin}(T, \frac{1}{2})$. However, there is an obstacle to this simple intuition: Within $\Theta(d^2)$ steps we leave $A(n)$ with positive probability, so simply jumping to the outcome of $\Theta(d^2)$ steps necessarily introduces an error. As we want to design an *exact* sampling algorithm, we have to overcome this hurdle.

We discuss different *jump procedures* to tackle this problem. The most elementary jump procedure J_{time} simply jumps to the outcome of d random walk steps (instead of d^2). As the distance to the in-circle is d, we cannot leave $A(n)$ in d steps and we thus get an exact sampling algorithm. However, the (expected) jumping distance of

J_{time} is small, so that we need many jumps to finally leave $A(n)$. For the second approach J_{shape} we take a ball S of radius d around the current particle position and directly sample the position where the particle first leaves S. This yields a large jumping distance, but it is computationally expensive, as the leaving distribution of a ball is not sufficiently simple. Finally, we balance these two approaches to get long jumps that can be efficiently computed. For this we either do $d^2/\log n$ steps of the random walk or go to the position where the particle first leaves the ball S, *whichever happens first*. With high probability this jump will end somewhere inside S after $d^2/\log n$ steps. Thus, we may sample the outcome of $d^2/\log n$ steps, conditioned on falling into S, and return this as the outcome of the jump. Of course, this does not yet sample from the correct distribution, but we can patch this algorithm by running, with a small probability p_{fail}, a slow algorithm that corrects the probability of unlikely events. Overall, this works in constant time in expectation and with high probability and has a large jumping distance.

2 Succinct Sampling

This chapter is based on [18]. Approximately, it can be split into two parts: Kasper Green Larsen contributed the lower bounds and I contributed the novel data structures.

[18] K. Bringmann and K. G. Larsen. "Succinct Sampling from Discrete Distributions." In: *Proc. 45th Annual ACM Symposium on Symposium on Theory of Computing (STOC'13)*. 2013, 775–782.

In this chapter, we study ProportionalSampling from the perspective of succinct data structures. We show that the space usage of Walker's alias method is sub-optimal. In fact, we present optimal systematic and non-systematic succinct data structures. We first restate the results of Section 1.5 that we will prove in this chapter.

Theorem 1.1. *For any $1 \leqslant r \leqslant n$ we can build a systematic data structure for* ProportionalSampling *with $r + \mathcal{O}(w)$ bits of redundancy, $\mathcal{O}(n/r)$ expected query time, and $\mathcal{O}(n)$ preprocessing time.*

Theorem 1.2. *Consider any systematic data structure for sampling from a discrete distribution, having redundancy r and supporting* ProportionalSampling *in expected query time t. Then $r \cdot t = \Omega(n)$.*

Theorem 1.3. *Any non-systematic data structure for* ProportionalSampling *must use at least nw bits of space for any $1 \leqslant w = o(n)$ and sufficiently large n.*

Theorem 1.4. *In the non-systematic case we can build a data structure for* ProportionalSampling *that needs $nw + 1$ bits of space, $\mathcal{O}(1)$ expected query time, and $\mathcal{O}(n)$ preprocessing time.*

This chapter is structured as follows. In Section 2.1, we present our systematic data structures. In Section 2.2, we then demonstrate that it is possible to do much better in the non-systematic case. In Section 2.3, we complement our systematic data structures with a matching lower bound. Finally, in Section 2.4, we prove a lower bound on the amount of bits needed to represent an input distribution.

2.1. Sampling with Read-Only Input

In this section, we present a data structure that supports sampling from a discrete distribution if the input is read-only, i.e., in the systematic case. We achieve any desired redundancy of $r + \mathcal{O}(w)$ bits with expected query time $\mathcal{O}(n/r)$ and optimal preprocessing time $\mathcal{O}(n)$, proving Theorem 1.1.

First, in the next section, we present a novel and practical data structure that uses $\mathcal{O}(n \log n + w)$ redundant bits, $\mathcal{O}(1)$ expected query time, and $\mathcal{O}(n)$ preprocessing time. Then, in Section 2.1.2, we modify it to use only $\mathcal{O}(n + w)$ redundant bits. In Section 2.1.3, we show how to get any smaller redundancy while increasing query time.

2.1.1. Redundancy of $\mathcal{O}(n \log n + w)$

Preprocessing We compute $S = \sum_i p_i$ and store it using $\mathcal{O}(w)$ bits. Additionally, we store a sorted array A that contains numbers in $[n] = \{1, \ldots, n\}$, specifically, A contains, for each index i, the number i exactly $\lfloor np_i/S \rfloor + 1$ times. This finishes space usage.

Observe that A has size at most $2n$, since we have

$$|A| = \sum_i \left(\left\lfloor \frac{n\,p_i}{S} \right\rfloor + 1 \right) \leqslant \sum_i \left(\frac{n\,p_i}{S} + 1 \right) = \frac{nS}{S} + n = 2n.$$

Moreover, A has entries in $[n]$, so that we can store it using $\mathcal{O}(n \log n)$ bits. Note that A can easily be constructed in time $\mathcal{O}(n)$.

Sampling Intuitively, if we return the value $A[k]$ for a uniform random $k \in \{1, \ldots, |A|\}$, then this is close to sampling from the input distribution. We can make this into an exact sampling method with a slight modification, using the rejection method as follows.

1. Pick a uniformly random $k \in \{1, \ldots, |A|\}$.

2. Rejection: If $k = 1$ or $A[k-1] \neq A[k]$ then
   with probability $1 - \mathrm{frac}(n\,p_{A[k]}/S)$ goto step 1.

3. Return $A[k]$.

Here, $\mathrm{frac}(x) = x - \lfloor x \rfloor$ is the fractional part of x. Note that in step 2 we check whether k is the first occurrence of $A[k]$ in A. If so, with some probability we throw away k and go to step 1 again, i.e., there is an implicit loop.

Let $i \in [n]$. What is the probability q_i of returning i in the first iteration of the implicit loop? There are $\lfloor n\,p_i/S \rfloor + 1$ occurrences of i in A. If we randomly pick k to be the first occurrence of i in A, then we return i only with probability $\mathrm{frac}(n\,p_i/S)$

and reject it otherwise. If we pick k to be any other occurrence of i, then we return i right away. Thus, we have

$$q_i = \frac{\lfloor n\,p_i/S \rfloor + \operatorname{frac}(n\,p_i/S)}{|A|} = \frac{n\,p_i}{S|A|}.$$

Let $Q = \sum_j q_j$ denote the probability of returning anything in the first iteration of the implicit loop, i.e., the probability of leaving the loop in the first iteration. The total probability of sampling i with the above method is

$$\sum_{t \geqslant 0} q_i (1-Q)^t = \frac{n\,p_i}{S|A|} \sum_{t \geqslant 0} (1-Q)^t.$$

On the right hand side, the only term dependent on i is p_i. Hence, the probability of sampling i is proportional to p_i. Since we only sample numbers from $[n]$, this implies that the probability of sampling i is p_i/S, proving that the above method is indeed an exact sampling algorithm.

To show that it is also fast, note that the probability of leaving the loop in the first iteration is

$$Q = \sum_j q_j = \sum_j \frac{n\,p_j}{S|A|} = \frac{n}{|A|} \geqslant \frac{1}{2}.$$

Hence, the expected number of iterations is constant. In every iteration we sample uniform numbers in $[n]$ and $[S]$, which can be done in $\mathcal{O}(1)$ expected time, and, in particular, in time independent of the sampled number. Thus, the above sampling method needs in total $\mathcal{O}(1)$ expected time.

2.1.2. Redundancy of $\mathcal{O}(n + w)$

A simple encoding of A as in the last section is very wasteful. We show how to reduce the redundancy to $\mathcal{O}(n+w)$ bits. For this, we construct a bit array B of length $|A|$. The entry $B[k]$ is 1 if k is the first occurrence of $A[k]$ in A, and 0 otherwise. We store B in a data structure supporting Rank queries, where $\textsc{Rank}_B(k) := \sum_{j=1}^{k} B[k]$ (with summation over the integers). Using, e.g., [86], Rank queries can be answered in constant time using a data structure of size $(1 + o(1))|B| = \mathcal{O}(n)$ bits.

Observe that we have $\textsc{Rank}_B(k) = A[k]$. Hence, using the Rank data structure for B, we can simulate the query algorithm from the last section and whenever it reads an array entry $A[k]$ we instead query $\textsc{Rank}_B(k)$. Since we only need to store S and the Rank data structure for B, this reduces the redundancy to $\mathcal{O}(n + w)$ bits.

2.1.3. Arbitrary Redundancy

We show that we can further reduce the redundancy to $\mathcal{O}(n/k + w)$ bits at the cost of increasing the query time to $\mathcal{O}(k)$ for any integer $k \geqslant 1$. Choosing $k = cn/r$ for a sufficiently large constant $c > 0$ implies Theorem 1.1.

We will partition $[n]$ into blocks of k elements. First, we show how to sample the block that contains the final sample. Then we show how to sample inside a block.

For ease of readability, assume that k divides n. On input $p_1, \dots, p_n$, consider the auxiliary instance $q_1, \dots, q_m$, where $m = n/k$ and $q_i = \sum_{j=1}^{k} p_{ik+j}$. We first show how to sample with respect to $q_1, \dots, q_m$. For this we make use of the data structure from the last section. Since we are not given input $q_1, \dots, q_m$, but $p_1, \dots, p_n$, we have to simulate this data structure and whenever it reads an input number q_i, we compute q_i on the fly from the input $p_1, \dots, p_n$. This incurs an additional factor of k on both preprocessing and query time, totaling in $\mathcal{O}(mk) = \mathcal{O}(n)$ preprocessing and $\mathcal{O}(k)$ query time. Moreover, we need only $\mathcal{O}(m + w) = \mathcal{O}(n/k + w)$ bits of redundancy.

Next, given i as sampled above, we show how to sample $j \in \{ik+1, \dots, (i+1)k\} =: J$ with probability p_j/q_i. To do so, we use a simple linear time sampling algorithm (see, e.g., [52, p. 120]): First we compute $q_i = \sum_{j \in J} p_j$. Then we sample a uniform random integer $R \in \{1, \dots, q_i\}$. Finally, via linear search we determine the smallest index ℓ such that $\sum_{j=1}^{\ell} p_{ik+j} \geqslant R$ and return $ik + \ell$. This needs $\mathcal{O}(k)$ query time, and no preprocessing or redundancy.

Putting both parts together, for any index j there is a block i with $j \in \{ik + 1, \dots, (i+1)k\}$. We have probability q_i of sampling i in the first part and probability p_j/q_i of sampling j in the second part. In total, this yields a probability of p_j for sampling j, so we indeed described an exact sampling algorithm. We get the desired preprocessing time $\mathcal{O}(n)$, expected query time $\mathcal{O}(k)$, and redundancy $\mathcal{O}(n/k + w)$.

2.2. One Additional Bit

In this section, we show that there is a data structure for sampling from a discrete distribution with redundancy 1 in the non-systematic case where the input is not read-only, proving Theorem 1.4. More precisely, we construct a data structure using $nw + 1$ bits of space in total that supports the operation ProportionalSampling in $\mathcal{O}(1)$ expected time and can be built in $\mathcal{O}(n)$ preprocessing time. Since we prove in Section 2.4 that it takes at least nw bits to describe the input, this corresponds to a redundancy of only 1 bit.

The First Bit Let c be a sufficiently large constant integer to be fixed later. We start the description of the data structure by explaining the usage of the first bit of memory: In this bit we store whether we have

$$\sum_i p_i \geqslant 2^{w-c-1} n. \tag{*}$$

Note that this can be computed in $\mathcal{O}(n)$ time preprocessing. The use of the rest of the bits depends on this first bit; we describe both cases in the next two paragraphs.

Large Sum Assume that (*) holds, i.e., the first bit that we have stored is 1. Then the bound $p_i \leqslant 2^w$ for all i is on average a tight upper bound, which is the standard situation to apply the rejection method.

In this case, in the remaining nw bits we simply store the plain input $p_1, \ldots, p_n$. This completes the description of space usage and preprocessing.

To perform a ProportionalSampling operation we proceed as follows.

1. Pick a uniformly random number $i \in [n]$.
2. Rejection: With probability $1 - p_i/2^w$ goto 1.
3. Return i.

The analysis of this sampling method is similar to the analysis in Section 2.1.1. Note that in step 2 with some probability we go back to step 1, so there is an implicit loop. The probability of returning $i \in [n]$ in the first iteration of this loop is $\frac{p_i}{2^w n}$. Let Q denote the probability of returning anything in the first iteration of the implicit loop, i.e., the probability of leaving the loop in the first iteration. Then the total probability of sampling i with above method is

$$\sum_{t \geqslant 0} \frac{p_i}{2^w n}(1-Q)^t.$$

Note that here the only term dependent on i is p_i. Hence, the probability of sampling i is proportional to p_i, so it has to be p_i/S, and we indeed have an exact sampling method.

To bound the method's expected running time, consider the probability Q in more detail. We have

$$Q = \sum_i \frac{p_i}{2^w n} \overset{(*)}{\geqslant} 2^{-c-1} = \Omega(1).$$

Hence, the expected number of iterations of the implicit loop is bounded by a constant. In every iteration we sample a random number in $[n]$ and in $[2^w]$, which can be done in $\mathcal{O}(1)$ expected time, and, in particular, in time independent of the sampled number. Hence, in total the above method uses $\mathcal{O}(1)$ expected time.

Small Sum Assume that we do not have (*), i.e., $\sum_i p_i < 2^{w-c-1}n$ and the first bit is 0. The intuition of how to proceed is as follows. Conditioned on $\sum_i p_i < 2^{w-c-1}n$, the entropy of the input is much less than nw. This allows one to compress the input to $nw - \Omega(n)$ bits, while still guaranteeing efficient access to each input number p_i. Now we use the algorithm of Section 2.1, which generates $\mathcal{O}(n)$ redundant bits and performs a ProportionalSampling operation in $\mathcal{O}(1)$ expected time, if it is given access to the input numbers. The total space usage of writing down the compressed input and the redundant bits is then $nw - \Omega(n) + \mathcal{O}(n)$ bits, which is at most nw bits after adjusting constants.

In the following, we describe the details of the compression step. Let $I := \{i \in [n] \mid p_i \geqslant 2^{w-c}\}$. We store $p_1, \ldots, p_n$ in order, using w bits if $i \in I$, and only the $w - c$ least significant bits otherwise. This yields a bit string B. In order to

read a value p_i from B we need to know where its encoding begins in B, and how many bits it uses. We achieve this by storing (the characteristic bit vector of) I in a data structure supporting Rank queries in $\mathcal{O}(1)$ query time using $\mathcal{O}(n)$ bits of space (using, e.g., [86]). Using this data, any value p_i can be read in constant time: Given i, we compute $k = \textsc{Rank}_I(i-1)$. Then there are k input numbers encoded with w bits and $i-1-k$ input numbers encoded with $w-c$ bits preceding p_i. Hence, the encoding of p_i starts at position $kw + (i-1-k)(w-c)$. Moreover, the length of its encoding is w, if $i \in I$ (if and only if $\textsc{Rank}_I(i) > \textsc{Rank}_I(i-1)$), and $w-c$, otherwise.

Since this compression of the input allows us to read any input value in constant time, we can simulate the algorithm from Section 2.1 on the compressed input, decoding p_i whenever the algorithm reads it. This produces additional data of $\mathcal{O}(n)$ bits and allows us to sample from the input distribution in $\mathcal{O}(1)$ expected time.

In total the compressed input and the auxiliary data need $nw - (n - |I|)c + \mathcal{O}(n)$ bits. Since we are in the case where $\sum_i p_i < 2^{w-c-1}n$ and every $i \in I$ has $p_i \geqslant 2^{w-c}$, we can bound $|I| \leqslant n/2$. Thus, the total number of bits is at most $nw - \frac{n}{2}c + \mathcal{O}(n)$. For sufficiently large c, this is less than nw.

In both cases we need $\mathcal{O}(n)$ preprocessing, $\mathcal{O}(1)$ query time, and at most $nw+1$ bits of storage, which finishes the proof of Theorem 1.4.

2.3. Lower Bounds for Read-Only Inputs

In this section, we show a tight lower bound on the trade-off between redundancy and expected query time for read-only data structures for sampling from a discrete distribution, proving Theorem 1.2. Throughout the section, we assume the availability of such a data structure using r redundant bits and supporting Proportional-Sampling in expected time t.

Hard Distribution For the proof, we consider a hard distribution over input numbers. Let $B \geqslant 1$ be a parameter to be fixed later and assume B divides n. We draw a random input $X = X_1, \ldots, X_n$ in the following manner: Partition the indices $\{1, \ldots, n\}$ into B consecutive groups of n/B indices each. For each group, select a uniform random index j in the group and let the corresponding input number X_j have the value 1. For all other indices in the group, we let the corresponding input number store the value 0. This constitutes the hard input distribution.

As a technical remark regarding our input distribution, note that the previous work on systematic Rank and Select structures allows access to multiple input elements by assuming the input is packed in machine words. Even though our hard distribution uses only 0's and 1's, we assume that the data structure may only access a single input number in one read operation. We note that this is a completely valid assumption, since we could just replace all 1's with $2^w - 1$ (or some other very large number) to enforce this restriction. Also, assuming that only one input number can be accessed with one read operation is more appropriate for situations in which the input numbers are given implicitly, i.e., have to be computed when requested.

Finally, we note that the tight lower bound for range minimum by Brodal et al. [57] also assumes that only one input element may be read in one operation.

For an intuition on why the above input distribution is hard, think of B as being a sufficiently large constant times r. Running ProportionalSampling on such an input must return an index j for which $X_j = 1$. Furthermore, the index returned is uniformly random among the indices having the value 1. Thus, the ProportionalSampling operation must find the location of a 1 inside a random block. Since there are so few redundant bits, we have less than 1 bit of information about each block on average, thus the only way to locate a 1 inside a block is to perform a linear scan, costing $\Omega(n/B) = \Omega(n/r)$ time. We prove that this intuition is correct in the rest of the section.

Note that $H(X) = B \log(n/B)$ bits, where $H(\cdot)$ denotes binary Shannon entropy. This is easily seen since X contains B 1's, each uniformly distributed inside a range of n/B indices.

An Encoding Proof We prove our lower bound using an encoding argument. More specifically, we show that if a sampling data structures exists that is too efficient in terms of redundancy r and expected query time t, then we can use this data structure to encode (and decode) the random input X in less than $H(X)$ bits in expectation. This is an information theoretic contradiction.

The basic idea in the encoding procedure is to implement the claimed data structure on the input X and then run ProportionalSampling for $k = B/2$ times. We will then write down the input numbers X_j that the data structure reads during these executions along with the redundant bits. This will (essentially) be enough to recover the entire input X and thus we derive a contradiction if these numbers X_j and the redundant bits can be described in much less than $H(X)$ bits. Since the number of read operations depends on the query time, we derive lower bounds on the trade-off between the redundant bits and the query time.

As a last technical detail before we present the encoding and decoding procedures, we assume that on each invocation of ProportionalSampling, the sampling data structure is given access to a finite stream of uniform random bits that it uses to determine the index to return. Thus, if we fix the stream of random bits given to the data structure, the latter becomes completely deterministic and always returns the same index on the same input. The encoding and decoding procedures will share such random streams, thus they both know what "randomness" was used by the data structure when performing the k ProportionalSampling operations. More formally, let $R_1, \ldots, R_k$ be k finite sequences of uniform random bits. Both the encoding and decoding procedure are given access to these sequences. Since X is independent of $R_1, \ldots, R_k$, we have $H(X \mid R_1 \cdots R_k) = H(X)$, i.e., we still derive a contradiction if the encoding uses less than $H(X)$ bits in expectation when the encoder and decoder share $R_1, \ldots, R_k$. We are finally ready to present the encoding procedure.

Encoding Procedure Upon receiving the input numbers $X = X_1, \ldots, X_n$, we first implement the claimed data structure on X. Then we run Proportional-

SAMPLING for k times, using R_i as the source of randomness in the i-th invocation. We now do the following:

1. We write down the r redundant bits stored by the data structure on input X.

2. Now construct an initially empty set of indices C and an initially empty string z. For $i = 1, \ldots, k$ we examine the input numbers X_j read during the i-th PROPORTIONALSAMPLING operation. For each such number X_j, in the order in which they are read, we first check whether $j \in C$. If not, we add j to C and append the value X_j (just a bit) to z. Otherwise, i.e., if j is already in C, we simply continue with the next number read. For each $i = 1, \ldots, k$, if the query algorithm is about to append the $(4t+1)$'st bit to z, we terminate the procedure for that i and continue with the next PROPORTIONALSAMPLING. Also, if the i-th PROPORTIONALSAMPLING operation terminates before $4t$ bits have been appended to z, we pad with 0s such that a total of $4t$ bits are always appended to z. Letting Y denote the number of 1s in z, the next part of the encoding consists of $\log B$ bits specifying Y, followed by $\log \binom{4tk}{Y}$ bits specifying z (there are $\binom{4tk}{Y}$ strings of length $|z| = 4tk$ having Y 1s).

   We note that the reason why we maintain C and only encode each X_j at most once, is that this forces Y to be proportional to the number of distinct 1s that we have seen. Since each distinct 1 reveals much information about X, this will eventually give our contradiction.

3. Finally, collect the set D containing all indices i for which $X_i = 1$ and where either $i \in C$ (it was read by one of the PROPORTIONALSAMPLING queries), or the corresponding index was returned as the result of one of the PROPORTIONALSAMPLING queries that terminated without appending more than $4t$ bits to z during step 2 (the data structure might return an index without reading the corresponding input number). For each $j = 0, \ldots, B-1$ (in this order), let i_j be the index of the 1 in X which is stored in the j-th group (numbers $X_{j(n/B)}, \ldots, X_{(j+1)(n/B)-1}$). If i_j is not contained in D, we write down the offset of i_j within its group, i.e. we write down the value $i_j - j(n/B)$. Since $|D| \geqslant Y$, this part of the encoding costs at most $(B - Y)\log(n/B)$ bits.

Before analyzing the expected size of the encoding, we present the decoding procedure:

Decoding Procedure Recall we have access to the random streams $R_1, \ldots, R_k$ during the decoding, i.e., we conditioned on these variables. To recover X from the above encoding, we now do the following:

1. First initialize an empty set $\overline{C}$, which eventually will contain pairs (i, Δ_i), where i is an index into $X = X_1, \ldots, X_n$ and Δ_i is the value stored at that index, i.e. $\Delta_i = X_i$. Now for $i = 1, \ldots, k$, start running the query procedure for PROPORTIONALSAMPLING using R_i as the source of randomness. While running the i-th PROPORTIONALSAMPLING, we maintain a pointer g_i into the string z which was constructed in step 2 of the encoding procedure. When starting the i-th PROPORTIONALSAMPLING, g_i points to the first bit that was

appended by the i-th ProportionalSampling during step 2 of the encoding procedure. This bit is exactly the $((i-1)4t)$-th bit of z (counting from 0).

When running the i-th ProportionalSampling operation, the query procedure starts by requesting either some of the redundant bits or some input number. If it requests some of the redundant bits, we have those bits immediately from step 1 of the encoding procedure and can continue with the next step of the ProportionalSampling procedure. If on the other hand it requests the number X_j, we first check whether there is a pair (j, Δ_j) in $\overline{C}$ for some Δ_j. If so, Δ_j equals X_j and we can continue the procedure. If not, we know from step 2 of the encoding procedure that the bit pointed to by g_i stores the value X_j and we can again continue the procedure after incrementing $g_i \leftarrow g_i + 1$ and adding the pair (j, X_j) to $\overline{C}$. If at any step we are about to increment g_i for the $(4t+1)$'st time, we simply abandon the i-th ProportionalSampling and continue with the next. Clearly these k invocations of ProportionalSampling allow us to recover the set D.

2. From the set D recovered above, we can deduce the groups in X for which the index of the corresponding 1 is not in D. It finally follows that we can recover X from D and the bits written down during step 3 of the encoding procedure.

What remains is to analyze the size of the encoding and derive the lower bound.

Analysis The number of bits in the encoding, denoted by K, is precisely

$$\begin{aligned} K &= r + \log B + \log \binom{4tk}{Y} + (B-Y)\log(n/B) \\ &\leqslant r + \log B + Y\log(4etk/Y) + (B-Y)\log(n/B) \\ &= H(X) + r + \log B - Y\log(nY/4etkB) \\ &= H(X) + r + \log B - Y\log(nY/2etB^2). \end{aligned}$$

The only random variable in the above is Y and since

$$Y\log(nY/2etB^2) = Y\log Y + Y\log(n/2etB^2)$$

is convex, we get from Jensen's inequality that

$$\mathbb{E}[K] \leqslant H(X) + r + \log B - \mathbb{E}[Y]\log(n\mathbb{E}[Y]/2etB^2).$$

Now observe that each of the $k = B/2$ calls to ProportionalSampling returns an element not returned in any of the other ProportionalSampling operations with probability at least $1/2$. Furthermore, we get from Markov's inequality that each ProportionalSampling terminates within the first $4t$ steps with probability at least $3/4$. From a union bound, we conclude $\mathbb{E}[Y] \geqslant B/8$. Inserting this in the above, we have

$$\mathbb{E}[K] \leqslant H(X) + r + \log B - B\log(n/16etB)/8.$$

Choosing $B = 8r$, we get (using $r + \log(8r) \leqslant 4r$):

$$\mathbb{E}[K] \leqslant H(X) + 4r - r\log(n/2^5etr),$$

from which we conclude that the claimed data structure must satisfy

$$\begin{aligned} 4r &\geqslant r\log(n/2^5etr) \Rightarrow \\ 4 + \log(2^5e) &\geqslant \log(n/tr) \Rightarrow \\ tr &= \Omega(n). \end{aligned}$$

This completes the proof of Theorem 1.2.

2.4. Space Lower Bound

In this section, we show that the information theoretic minimum number of bits needed to sample from a discrete distribution is nw bits for any $1 \leqslant w = o(n)$ and n sufficiently large, proving Theorem 1.3. For this, observe that two inputs $p_1, \ldots, p_n$ and $\hat{p}_1, \ldots, \hat{p}_n$ represent the same probability distribution only if there exists a value $\alpha > 0$ such that $p_i = \alpha\hat{p}_i$ for all $1 \leqslant i \leqslant n$. We want to show that there are not too many pairs of inputs for which this is true. To prove this, define an input set of w-bit integers $p_1, \ldots, p_n$ to be *irreducible* if for all $0 < \alpha < 1$, there is at least one $i \in \{1, \ldots, n\}$ for which αp_i is not an integer. Clearly, any two distinct and irreducible inputs represent two distinct probability distributions. First, we prove that the following condition is sufficient to guarantee irreducibility:

Lemma 2.1. *An input set of w-bit integers $p_1, \ldots, p_n$ is irreducible if there are at least two distinct primes among $p_1, \ldots, p_n$.*

Proof. Assume $p_i = q$ and $p_j = q'$ for some $i \neq j$ and some primes $q \neq q'$. Assume also that $p_1, \ldots, p_n$ is not irreducible. This implies the existence of a value $0 < \alpha < 1$ such that $\alpha q = c$ and $\alpha q' = c'$ for some integers $c, c' \geqslant 1$. Now since $\alpha < 1$, we have $c < q$ and hence q is not a prime factor in c. But $c' = \alpha q' = cq'/q$ and it follows that q is not a prime factor in cq', thus c' cannot be integer, i.e., a contradiction. □

For the remaining part of the proof, consider drawing each p_i as a uniform random integer in $[2^w]$. The probability that a particular p_i is prime is $\Omega(1/w)$. Thus, for $2 \leqslant w = o(n)$ and any sufficiently large n, the number of distinct primes in the randomly chosen $p_1, \ldots, p_n$ is at least two with high probability, certainly with probability at least $3/4$. Since we chose $p_1, \ldots, p_n$ uniformly at random, we conclude that at least $(3/4)2^{nw}$ of the 2^{nw} possible inputs are irreducible. Therefore, any sampling data structure must use at least

$$\lceil \log((3/4)2^{nw}) \rceil = nw$$

bits of space. In the case $w = 1$, the result follows immediately.

3 Sorted Input and Subset Sampling

This chapter is based on [21]. I contributed more than 50% at all stages and to all parts of this paper. Moreover, I contributed the substantial improvements over the conference version that are incorporated in this chapter.

[21] K. Bringmann and K. Panagiotou. "Efficient Sampling Methods for Discrete Distributions." In: *Proc. 39th International Colloquium on Automata, Languages, and Programming (ICALP'12)*. Vol. 7391. LNCS. 2012, 133–144.

In this chapter we consider the problems ProportionalSampling and SubsetSampling on sorted and unsorted input sequences, i.e., the problem variants SortedProportionalSampling, UnsortedProportionalSampling, SortedSubsetSampling, as well as UnsortedSubsetSampling. Recall that we defined SubsetSampling as follows. On input $\mathbf{p} = (p_1, \dots, p_n)$ we consider n independent events with indicator random variables $X_1, \dots, X_n$, and $\Pr[X_i = 1] = p_i$. For shortcut we write $\mu = \mu_{\mathbf{p}} = \sum_{i=1}^{n} p_i = \mathbb{E}\big[\sum_{i=1}^{n} X_i\big]$. We want to sample the random variable $X = X_{\mathbf{p}} = \{i \in [n] \mid X_i = 1\}$, i.e., the set of all events that occurred; in particular, for any $S \subseteq [n]$ we have

$$\Pr[X = S] = \Big(\prod_{i \in S} p_i\Big) \cdot \Big(\prod_{i \in [n] \setminus S} (1 - p_i)\Big).$$

We restate our results from Section 1.6 that we will prove in this chapter.

Theorem 1.6. *Let $\beta \in \{2, \dots, n\}$.* SortedProportionalSampling *can be solved in preprocessing time $\mathcal{O}(\log_\beta n)$ and expected query time*

$$t_q^\beta(n, \mu) = \mathcal{O}\big(\min\big\{\beta, \frac{\log n}{\log\log n}\big\}\big).$$

This is optimal, as for some constant $\varepsilon > 0$, SortedProportionalSampling *has no data structure with preprocessing time $\varepsilon \log_\beta(n)$ and expected query time $\varepsilon\, t_q^\beta(n, \mu)$ for any $\mu = \mu(n)$.*

Theorem 1.7. UnsortedProportionalSampling *can be solved in preprocessing time $\mathcal{O}(n)$ and query time $\mathcal{O}(1)$. This is optimal, as for some constant $\varepsilon > 0$,*

UnsortedProportionalSampling *has no data structure with preprocessing time* εn *and expected query time* εn *for any* $\mu = \mu(n)$.

Note that the upper bound of Theorem 1.7 already follows from Walker's alias method. We prove the matching lower bound.

Theorem 1.9. *Let* $\beta \in \{2, \dots, n\}$. SortedSubsetSampling *can be solved in preprocessing time* $\mathcal{O}(\log_\beta n)$ *and expected query time*

$$t_q^\beta(n,\mu) = \begin{cases} \mathcal{O}(\mu), & \text{if } \mu \geqslant \frac{1}{2}\log n, \\ \mathcal{O}(1+\beta\mu), & \text{if } \mu < \frac{1}{\beta}\log_\beta n, \\ \mathcal{O}\big(\frac{\log n}{\log(\frac{\log n}{\mu})}\big), & \text{otherwise.} \end{cases}$$

In particular, the query time is always bounded by $\mathcal{O}(1+\beta\mu)$. *This is optimal, as for some constant* $\varepsilon > 0$, SortedSubsetSampling *has no data structure with preprocessing time* $\varepsilon \log_\beta n$ *and expected query time* $\varepsilon\, t_q^\beta(n,\mu)$ *for any* $\mu = \mu(n)$.

Theorem 1.10. UnsortedSubsetSampling *can be solved in preprocessing time* $\mathcal{O}(n)$ *and expected query time* $\mathcal{O}(1+\mu)$. *This is optimal, as for some constant* $\varepsilon > 0$, UnsortedSubsetSampling *has no data structure with preprocessing time* εn *and expected query time* εn *for any* $\mu = \mu(n)$.

This chapter is structured as follows. In Section 3.1 we present our new algorithms, proving (the upper bounds of) Theorem 1.6 in Section 3.1.1 and Theorems 1.9 and 1.10 in Section 3.1.2. In Section 3.2 we present the lower bounds, proving (the lower bounds of) Theorems 1.7 and 1.10 in Section 3.2.1, Theorem 1.6 in Section 3.2.2, and Theorem 1.9 in Section 3.2.3. We present our reduction from ProportionalSampling to SubsetSampling in Section 3.3. We discuss relaxations to our input and machine model and possible extensions in Section 3.4.

3.1. Upper Bounds

3.1.1. A Simple Algorithm for Sorted Proportional Sampling

In this section, we prove the upper bound of Theorem 1.6 by presenting an algorithm for SortedProportionalSampling with $\mathcal{O}(\beta)$ expected query time after $\mathcal{O}(\log_\beta n)$ preprocessing, where $\beta \in \{2, \dots, n\}$ is a parameter.

We note that one may reduce the query time to $\mathcal{O}\big(\min\{\beta, \frac{\log n}{\log\log n}\}\big)$ by adapting the query algorithm as follows. If $\beta > \frac{\log n}{\log\log n}$ then we rerun the preprocessing with β set to $\tilde{\beta} := \frac{\log n}{\log\log n}$. Then we run the query algorithm (with respect to $\tilde{\beta}$). This takes total expected time $\mathcal{O}(\log_{\tilde{\beta}} n + \tilde{\beta}) = \mathcal{O}\big(\frac{\log n}{\log\log n}\big)$. Hence, we only have to show a query time of $\mathcal{O}(\beta)$.

Let $p_1, \dots, p_n$ be an input sequence to SortedProportionalSampling. Consider the blocks $B_k := \{i \in [n] \mid \beta^k \leqslant i < \beta^{k+1}\}$ with $0 \leqslant k \leqslant L := \lfloor \log_\beta n \rfloor$. Note

that $B_0, \dots, B_L$ partition $[n] = \{1, \dots, n\}$. For $i \in B_k$ we set $\overline{p}_i := p_{\beta^k}$, which is an upper bound for p_i. Let $\mu := \sum_i p_i$ and $\overline{\mu} := \sum_i \overline{p}_i$. We also set for $0 \leqslant k \leqslant L$

$$q_k := \sum_{i \in B_k} \overline{p}_i = |B_k| \cdot p_{\beta^k} = \big(\min(\beta^{k+1}, n+1) - \beta^k\big) \cdot p_{\beta^k}.$$

For preprocessing, we run the preprocessing of UnsortedProportionalSampling on $q_1, \dots, q_L$. This takes time $\mathcal{O}(L) = \mathcal{O}(\log_\beta n)$ using Theorem 1.7, since q_k can be evaluated in constant time.

Our query algorithm consists of two steps. First, we sample an index i with distribution $\overline{p}_1, \dots, \overline{p}_n$. To this end, we sample a block B_k proportional to the distribution $q_1, \dots, q_L$ and then sample an index $i \in B_k$ uniformly at random. Second, with probability $1 - p_i/\overline{p}_i$ we reject i and repeat the whole process. Otherwise we return i. This culminates into Algorithm 1.

Algorithm 1 SortedProportionalSampling

Input: $p_1 \geqslant \ldots \geqslant p_n \geqslant 0$ and parameter $\beta \in \{2, \dots, n\}$

Preprocessing:
$L := \lfloor \log_\beta n \rfloor$
$q_k := (\min\{\beta^{k+1}, n+1\} - \beta^k) \cdot p_{\beta^k}$
Run preprocessing of UnsortedProportionalSampling$(q_0, \dots, q_L)$

Querying:
Repeat
$k :=$ UnsortedProportionalSampling$(q_1, \dots, q_L)$
pick i uniformly at random in $\{\beta^k, \dots, \min\{\beta^{k+1} - 1, n\}\}$
Break with probability p_i/p_{β^k}
Return i

Note that we pick index $i \in B_k$ with probability proportional to $\overline{p}_i$ and do not reject it with probability $p_i/\overline{p}_i$. Thus, the probability of returning a particular index i is proportional to $\overline{p}_i \cdot p_i/\overline{p}_i = p_i$ and we obtained an exact sampling algorithm. Moreover, in any iteration of the loop the probability r of not rejecting, i.e., of leaving the loop, is

$$r = \frac{1}{\overline{\mu}} \sum_{i=1}^{n} \overline{p}_i \cdot p_i/\overline{p}_i.$$

In this equation, note the first step of sampling with respect to $\overline{p}_1, \dots, \overline{p}_n$ ($\frac{1}{\overline{\mu}} \sum_{i=1}^n \overline{p}_i$) and the second step of rejection ($p_i/\overline{p}_i$). Clearly, this simplifies to $r = \mu/\overline{\mu}$. The following lemma shows that $\overline{\mu} \leqslant \beta \cdot \mu$, implying $r \geqslant 1/\beta$. Hence, the expected number of iterations of the loop is $\mathcal{O}(\beta)$, and in total querying takes expected time $\mathcal{O}(\beta)$.

Lemma 3.1. *We have* $\mu \leqslant \overline{\mu} \leqslant \beta \cdot \mu$.

Proof. The first inequality follows from $p_i \leqslant \overline{p}_i$. Note that for $i \in B_k$ we have $\lceil i/\beta \rceil \leqslant \beta^k$. Thus, $p_{\lceil i/\beta \rceil} \geqslant p_{\beta^k}$. Hence,

$$\overline{\mu} = \sum_{i=1}^{n} \overline{p}_i \leqslant \sum_{i=1}^{n} p_{\lceil i/\beta \rceil} \leqslant \beta \sum_{i=1}^{n} p_i = \beta \cdot \mu. \qquad \square$$

3.1.2. Subset Sampling

In this section we consider SortedSubsetSampling and UnsortedSubsetSampling and prove the upper bounds of Theorems 1.9 and 1.10. An interesting interplay between both of these problem variants will be revealed on the way.

We begin with an algorithm for unsorted probabilities that has a quite large preprocessing time, but will be used as a base case later. The algorithm uses Theorem 1.7.

Lemma 3.2. UnsortedSubsetSampling *can be solved in preprocessing time* $\mathcal{O}(n^2)$ *and expected query time* $\mathcal{O}(1+\mu)$.

Proof. For $i \in [n]$ let us denote by S_i the smallest sampled element that is at least i, or ∞, if no such element is sampled. Then S_i is a random variable such that

$$\Pr[S_i = j] = p_j \prod_{i \leqslant k < j} (1 - p_k) \quad \text{and} \quad \Pr[S_i = \infty] = \prod_{i \leqslant k \leqslant n} (1 - p_k).$$

All these probabilities can be computed on a Real RAM in time $\mathcal{O}(n)$ for any i, i.e., in time $\mathcal{O}(n^2)$ for all i. After having computed the distribution of the S_i's, we execute, for each $i \in [n]$, the preprocessing of Theorem 1.7, which allows us to quickly sample S_i later on. This preprocessing takes time $\mathcal{O}(n^2)$.

For querying, we start at $i = 1$ and iteratively sample the smallest element $j \geqslant i$ (i.e., sample S_i), output j, and start over with $i = j + 1$. This is done until $j = \infty$ or $i = n + 1$. Note that any sample of S_i can be computed in $\mathcal{O}(1)$ time with our preprocessing, so that sampling $S \subseteq [n]$ will be done in time $\mathcal{O}(1 + |S|)$. The expected running time is, thus, $\mathcal{O}(1 + \mu)$. $\square$

After having established this base case, we turn towards reductions between SortedSubsetSampling and UnsortedSubsetSampling. First, we give an algorithm for UnsortedSubsetSampling that reduces the problem to SortedSubsetSampling. For this, we roughly sort the probabilities so that we get good upper bounds for each probability. Then these upper bounds will be a sorted instance. After querying from this sorted instance, we use rejection to sample with the original probabilities.

Lemma 3.3. *Assume that* SortedSubsetSampling *can be solved in preprocessing time* $t_p(n, \mu)$ *and expected query time* $t_q(n, \mu)$*, where* t_p *and* t_q *are monotonically increasing in* n *and* μ*. Then* UnsortedSubsetSampling *can be solved in preprocessing time* $\mathcal{O}(n + t_p(n, 2\mu + 1))$ *and expected query time* $\mathcal{O}(1 + \mu + t_q(n, 2\mu + 1))$.

Proof. Let $\mathbf{p} = (p_1, \ldots, p_n)$ be an input sequence to UnsortedSubsetSampling. For preprocessing, we permute the input $\mathbf{p}$ so that it is approximately sorted, by putting it into buckets $U_k := \{i \in [n] \mid 2^{-k} \geqslant p_i \geqslant 2^{-k-1}\}$, for $k \in \{0, 1, \ldots, L-1\}$, and $U_L := \{i \in [n] \mid 2^{-L} \geqslant p_i\}$, where $L = \lceil \log n \rceil$. For each $i \in U_k$ we set $\overline{p}_i := 2^{-k}$, which is an upper bound on p_i. We sort the probabilities $\overline{p}_i$, $i \in [n]$, descendingly using bucket sort with the buckets U_k, yielding $\overline{p}'_1 \geqslant \ldots \geqslant \overline{p}'_n$. In this process we store the original index $\mathrm{ind}(i)$ corresponding to $\overline{p}'_i$, so that we can find $p_{\mathrm{ind}(i)}$ corresponding to $\overline{p}'_i$ in constant time. Then we run the preprocessing of SortedSubsetSampling on $\overline{p}'_1, \ldots, \overline{p}'_n$. Note that

$$\overline{\mu} := \sum_{i=1}^{n} \overline{p}'_i = \sum_{i=1}^{n} \overline{p}_i \leqslant \sum_{i=1}^{n} \max\left\{2p_i, \frac{1}{n}\right\} \leqslant 2\mu + 1.$$

Thus, the total preprocessing time is bounded by

$$\mathcal{O}(n) + t_p(n, \overline{\mu}) = \mathcal{O}(n + t_p(n, 2\mu + 1)),$$

establishing the first claim.

For querying, we query $\overline{p}'_1, \ldots, \overline{p}'_n$ using SortedSubsetSampling, yielding $S' \subseteq [n]$. We compute $S := \{\mathrm{ind}(i) \mid i \in S'\}$. Each $i \in S$ was sampled with probability $\overline{p}_i \geqslant p_i$. We use rejection to get this probability down to p_i. For this, we generate for each $i \in S$ a uniformly random number $r \in [0, 1]$ and check whether it is smaller than or equal to $p_i/\overline{p}_i$. If this is not the case, we delete i from S. Note that we have thus sampled i with probability p_i, and all elements are sampled independently, so S has the desired distribution. Moreover, since the expected size of S' is $\overline{\mu}$, the expected query time is bounded by

$$t_q(n, \overline{\mu}) + \mathcal{O}(1 + \mathbb{E}[|S'|]) = \mathcal{O}(1 + \mu + t_q(n, 2\mu + 1)),$$

and the second claim is also established. □

We also give a reduction in the other direction, solving SortedSubsetSampling by UnsortedSubsetSampling.

Lemma 3.4. *Let $\beta \in \{2, \ldots, n\}$. Assume that* UnsortedSubsetSampling *can be solved in preprocessing time $t_p(n, \mu)$ and expected query time $t_q(n, \mu)$, where t_p and t_q are monotonically increasing in n and μ. Then* SortedSubsetSampling *can be solved in preprocessing time $\mathcal{O}(\log_\beta n + t_p(1 + \log_\beta n, \beta\mu))$ and expected query time $\mathcal{O}(1 + \beta\mu + t_q(1 + \log_\beta n, \beta\mu))$. More precisely, our preprocessing computes a value $\overline{\mu}$ with $\mu \leqslant \overline{\mu} \leqslant \beta\mu$ and the expected query time is $\mathcal{O}(1 + \overline{\mu} + t_q(1 + \log_\beta n, \overline{\mu}))$.*

Proof. Let $p_1, \ldots, p_n$ be an input sequence to SortedSubsetSampling. As in Section 3.1.1, we consider blocks $B_k = \{i \in [n] \mid \beta^k \leqslant i < \beta^{k+1}\}$, with $k \in \{0, \ldots, L\}$ and $L := \lfloor \log_\beta n \rfloor$, and let $\overline{p}_i := p_{\beta^k}$ for $i \in B_k$. We will first sample with respect to the probabilities $\overline{p}_i$ – call the sampled elements *potential* – and then use rejection. For this, let X_k be an indicator random variable for the event that we sample *at least one* potential element in B_k. Then

$$q_k := \Pr[X_k = 1] = 1 - (1 - p_{\beta^k})^{|B_k|}.$$

Moreover, let Y_k be a random variable for the first potential element in block B_k minus β^k. Let $Y_k = \infty$, if no element in B_k is sampled as a potential element. Then $\Pr[Y_k = i] = p_{\beta^k}(1-p_{\beta^k})^i$ for $i \in \{0, \dots, |B_k|-1\}$, and $\Pr[Y_k = \infty] = \Pr[X_k = 0] = 1 - q_k$. We calculate

$$\Pr[Y_k = i \mid X_k = 1] = \frac{\Pr[Y_k = i]}{\Pr[X_k = 1]} = \frac{p_{\beta^k}}{q_k}(1-p_{\beta^k})^i, \qquad i \in \{0, \dots, |B_k|-1\}.$$

Since this is a (truncated) geometric distribution, we can sample from it in constant time on the Real RAM, see Section 1.7.2.

For preprocessing, we first compute the probabilities q_k, $k \in \{0, \dots, L\}$. This can be done in time $\mathcal{O}(L) = \mathcal{O}(\log_\beta n)$ (as $a^b = \exp(b \log a)$ can be computed in constant time on a Real RAM). Then we run the preprocessing of UnsortedSubsetSampling on them; note that the q_k's are in general not sorted. In total, the preprocessing time is at most

$$\mathcal{O}(\log_\beta n) + t_p(1 + \log_\beta n, \nu), \qquad \text{where} \qquad \nu = \sum_{i=0}^{\lfloor \log_\beta n \rfloor} q_k.$$

Using that $(1-x)^y \geqslant 1 - xy$ for $0 < x < 1$ and $y \geqslant 1$ we obtain

$$\nu = \sum_{i=0}^{\lfloor \log_\beta n \rfloor} 1 - (1-p_{\beta^k})^{|B_k|} \leqslant \sum_{i=0}^{\lfloor \log_\beta n \rfloor} p_{\beta^k}|B_k| = \sum_{i=1}^{n} \overline{p}_i = \overline{\mu}.$$

Using Lemma 3.1 we obtain $\nu \leqslant \beta\mu$, and the bound $\mathcal{O}(\log_\beta n + t_p(1 + \log_\beta n, \beta\mu))$ for the total preprocessing time follows immediately.

For querying, we query the blocks B_k that contain potential elements using the query algorithm for UnsortedSubsetSampling. Then, for each block B_k that contains a potential element, we sample all potential elements in this block. Note that the first of the potential elements in B_k is distributed as $\Pr[Y_k = i \mid X_k = 1]$, which is geometric, so we can sample from it in constant time, while all further potential elements are distributed as Y_k (but only on the remainder of the block), which is still geometric. Then, after having sampled a set $\overline{S}$ of potential elements, we keep each $i \in \overline{S}$ independently with probability $p_i/\overline{p}_i$. This yields a random sample $S \subseteq \overline{S}$ with the desired distribution. The overall query time is then at most

$$t_q(1 + \log_\beta n, \nu) + \mathcal{O}(1 + |\overline{S}|) \leqslant t_q(1 + \log_\beta n, \overline{\mu}) + \mathcal{O}(1 + |\overline{S}|)$$

As the expected value of $|\overline{S}|$ is $\overline{\mu} \leqslant \beta\mu$ the proof is completed. □

Next, we put the above three lemmas together to prove the upper bounds of Theorems 1.9 and 1.10.

Proof of Theorem 1.10, upper bound. To solve UnsortedSubsetSampling, we use the reduction Lemma 3.3 and then Lemma 3.4 (where we set $\beta = 2$), followed by the base case Lemma 3.2. This reduces the instance size from n to $\mathcal{O}(\log n)$, so that preprocessing costs $\mathcal{O}(n)$ for the invocation of the first lemma, $\mathcal{O}(\log n)$ for

the second, and $\mathcal{O}(\log^2 n)$ for the third. Note that μ is increased only by constant factors, so that we indeed get the a query time of $\mathcal{O}(1+\mu)$. □

For SortedSubsetSampling we first prove a weaker statement than Theorem 1.9, which follows from simply putting together the reductions of this section.

Lemma 3.5. *Let $\beta \in \{2, \dots, n\}$. Then* SortedSubsetSampling *can be solved in preprocessing time $\mathcal{O}(\log_\beta n)$ and expected query time $\mathcal{O}(1+\beta\mu)$. More precisely, our preprocessing computes a value $\overline{\mu}$ with $\mu \leqslant \overline{\mu} \leqslant \beta\mu$ and the expected query time is $\mathcal{O}(1+\overline{\mu})$.*

Proof. To solve SortedSubsetSampling, we use (in this order) the reductions presented in Lemma 3.4, Lemma 3.3, and then again Lemma 3.4 (where this time we set $\beta = 2$), followed by the base case Lemma 3.2. This reduces the instance size from n to $\mathcal{O}(\log_\beta n)$ and further down to $\mathcal{O}(\log\log_\beta n)$, while μ is increased to $\overline{\mu}$ with $\overline{\mu} = \mathcal{O}(1+\beta\mu)$. For precomputation this yields a running time of $\mathcal{O}(\log_\beta n)$ from Lemmas 3.4 and 3.3, $\mathcal{O}(\log\log_\beta n)$ from the second invocation of Lemma 3.4, and $\mathcal{O}(\log^2\log_\beta n)$ from the base case Lemma 3.2, summing up to a total $\mathcal{O}(\log_\beta n)$. The expected query time is $\mathcal{O}(1+\overline{\mu}) \leqslant \mathcal{O}(1+\beta\mu)$. □

Proof of Theorem 1.9, upper bound. Assume that we are allowed preprocessing time $\mathcal{O}(\log_{\tilde{\beta}} n)$ for some $\tilde{\beta} \in \{2, \dots, n\}$. Our algorithm for SortedSubsetSampling simply runs the preprocessing of Lemma 3.5 with $\beta = \tilde{\beta}$ to satisfy the preprocessing time constraint.

For querying, we improve upon the running time of Lemma 3.5 as follows. For any $\beta \in \{2, \dots, n\}$, let $\overline{\mu}(\beta)$ be the upper bound on μ computed by Lemma 3.5 given $\mathcal{O}(\log_\beta n)$ preprocessing time. Initially, we set $\beta := \tilde{\beta}$ so that $\overline{\mu}(\beta) = \overline{\mu}(\tilde{\beta})$ was computed by our preprocessing. If $1 + \overline{\mu}(\tilde{\beta}) \leqslant \log_{\tilde{\beta}} n$ then we run the query algorithm of Lemma 3.5 and are done. Otherwise, we repeatedly set $\beta := \lceil \beta^{1/2} \rceil$ and rerun the preprocessing of Lemma 3.5, until $\beta = 2$ or $1 + \overline{\mu}(\beta) \leqslant \log_\beta n$. Then we run the query algorithm of Lemma 3.5.

It remains to analyze the running time of this query algorithm. We consider three cases. (1) If $1 + \overline{\mu}(\tilde{\beta}) \leqslant \log_{\tilde{\beta}} n$ then the β-decreasing loop does not start and the query time is $\mathcal{O}(1+\overline{\mu}(\tilde{\beta})) \leqslant \mathcal{O}(1+\tilde{\beta}\mu)$. (2) If the β-decreasing loop breaks at $\beta = 2$, then since it did not stop at $\beta \in \{3, 4\}$ we have $1 + 4\mu > \log_4 n$, or $\mu = \Omega(\log n)$. In this case, the total query time is $\mathcal{O}(1 + \mu + \log n) = \mathcal{O}(\mu)$. (3) Otherwise the β-decreasing loop stopped at some β^* with $1 + \overline{\mu}(\beta^*) \leqslant \log_{\beta^*} n$. Using $\overline{\mu}(\beta) \leqslant \beta\mu$ and that we decrease β by taking its square root, we obtain $\beta^* \geqslant \gamma^{1/2}$, where $\gamma \geqslant 2$ satisfies

$$1 + \gamma\mu = \log_\gamma n.$$

The above equation solves to $\gamma = \Theta\big(\frac{\log n}{\mu} \log\big(\frac{\log n}{\mu}\big)\big)$. This yields a total query time of $\mathcal{O}(\log_{\beta^*} n) = \mathcal{O}(\log_\gamma n) = \mathcal{O}\big(\frac{\log n}{\log(\frac{\log n}{\mu})}\big)$, which proves the claimed query time. □

3.2. Lower Bounds

We prove most of our lower bounds by reducing the various sampling problems to the following fact, that searching in an unordered array of length m takes time $\Omega(m)$. A notable exception is Lemma 3.9.

Fact 3.6. *Consider problem* ArraySearch*: Given m and query access to an array $A \in \{0,1\}^m$ consisting of m bits, with exactly one bit set to 1, find the position of this bit. Any randomized algorithm for* ArraySearch *needs $\Omega(m)$ accesses to A in expectation.*

3.2.1. Proportional Sampling on Unsorted Probabilities

The lower bound for Theorem 1.7 is provided by the following lemma that reduces ArraySearch to UnsortedProportionalSampling. Moreover, the same proof yields the lower bound of Theorem 1.10 for UnsortedSubsetSampling.

Lemma 3.7. *Any single-sample algorithm for* UnsortedProportionalSampling *has expected time $\Omega(n)$. Moreover, any single-sample algorithm for* UnsortedSubsetSampling *has expected time $\Omega(n)$.*

Proof. Let A be an instance of ArraySearch of size n, say with 1-bit at position ℓ^*. We consider the instance

$$\mathbf{p} = \mathbf{p}^A = (p_1^A, \dots, p_n^A) \quad \text{with} \quad p_i^A = A[i].$$

Any sampling algorithm for UnsortedProportionalSampling returns ℓ^* on instance $\mathbf{p}^A$ with probability 1. Thus, simulating any algorithm for UnsortedProportionalSampling (by computing p_i^A on the fly) we obtain an algorithm for finding the 1-bit of array A. Hence, by Fact 3.6, any algorithm for UnsortedProportionalSampling takes expected time $\Omega(n)$. With varying μ, no better bound is possible, either: simply set $\tilde{p}_i^A = \mu \cdot p_i^A$.

Observe that on the same instance any sampling algorithm for UnsortedSubsetSampling returns the set $\{\ell^*\}$ with probability 1. This needs expected time $\Omega(n)$ for the same reasons. With varying μ, no better bound is possible, either: Consider an ArraySearch instance A of length $n - s$, where $s := \lceil \mu - 1 \rceil$. Let $p_i^A = A[i]$ for $1 \leqslant i \leqslant n - s$ and set the last s probabilities p_i^A to values that sum up to $\mu - 1$. Then we still need running time $\Omega(n - \mu)$ by Fact 3.6. As we also need running time $\Omega(\mu)$ for outputting the result, the claim follows. □

3.2.2. Proportional Sampling on Sorted Probabilities

In this subsection, we present the proof of the lower bound of Theorem 1.6 for SortedProportionalSampling.

Proof of Theorem 1.6, lower bound. Let $n \in \mathbb{N}$, $\beta \in \{2, \dots, n\}$, and $\alpha > 0$. Let $s_i := \sum_{j=0}^{i-1} \beta^j = (\beta^i - 1)/(\beta - 1)$. For ease of readability, assume that $n = s_L$ for some

$L \in \mathbb{N}$. Then $L = \Theta(\log_\beta n)$. We consider blocks $B_i := \{s_i, s_i+1, \dots, s_i+\beta^{i-1}-1\}$, for $i = 1, \dots, L$, that partition $\{1, \dots, n\}$.

Let A be an instance of ArraySearch of size L, say with 1-bit at position ℓ^*. To construct the instance $\mathbf{p} = \mathbf{p}^A = (p_1^A, \dots, p_n^A)$ we set for any $\ell \in \{1, \dots, L\}$ and $j \in B_\ell$

$$p_j^A := \alpha \cdot \beta^{-\ell + A[\ell]}.$$

As block B_ℓ has size β^ℓ, the total probability mass of B_ℓ is $\sum_{j \in B_\ell} p_j^A = \alpha \cdot \beta^{A[\ell]}$, i.e., it is $\alpha\beta$ for $A[\ell] = 1$, and α otherwise. Observe that

$$\mu = \sum_{i=1}^{n} p_i^A = \alpha(L + \beta - 1),$$

since block B_{ℓ^*} contributes $\alpha \cdot \beta$ and each of the other $L-1$ blocks contributes α as total probability mass. Furthermore, note that $p_1^A, \dots, p_n^A$ is indeed sorted, as the probability of an element in block B_ℓ is smaller by a factor of (at least) β than the probability of an element in $B_{\ell-1}$, except if $\ell = \ell^*$, in which case these probabilities coincide.

In the following we will prove that for any $\beta \leqslant \log_\beta n$ there is no sampling algorithm where the preprocessing reads at most εL input values and the querying reads at most $\varepsilon\beta$ input values in expectation, for a sufficiently small constant $\varepsilon > 0$, which proves the claim as $L = \Theta(\log_\beta n)$. Assume, for the sake of contradiction, that such an algorithm exists. On $\mathbf{p}^A$ we run the preprocessing and then K times the query algorithm, sampling K numbers $X_1, \dots, X_K \in \{1, \dots, n\}$. Denote by Y_k the block of X_k, i.e., $X_k \in B_{Y_k}$. If $A[Y_k] = 1$ for some $1 \leqslant k \leqslant K$ then we return Y_k, otherwise we linearly search for the 1-bit of A.

This yields an algorithm for ArraySearch, let us analyze its expected number of accesses to A. Since the total probability mass of block B_{ℓ^*} is $\alpha \cdot \beta$, we have

$$\Pr[Y_k = \ell^*] = \frac{\alpha \cdot \beta}{\mu} = \frac{\beta}{L + \beta - 1} = \Omega\Big(\frac{\beta}{L}\Big).$$

Thus, $\Pr[\nexists k\colon A[Y_k] = 1] = (1 - \Omega(\beta/L))^K = \exp(-\Omega(K\beta/L))$. Setting $K = \Theta(\log(1/\varepsilon)L/\beta)$ (with sufficiently large hidden constant), this probability is at most ε. Hence, the expected number of accesses to A of the constructed algorithm is (counting preprocessing, K queries, and a possible linear search through A)

$$\varepsilon L + K \cdot \varepsilon\beta + \Pr[\nexists k\colon A[Y_k] = 1] \cdot L \leqslant \mathcal{O}(\log(1/\varepsilon)\varepsilon L).$$

For sufficiently small $\varepsilon > 0$ this contradicts Fact 3.6. Note that this lower bound holds restricted to any $\mu = \mu(n)$, since we still have the freedom of choosing α. □

Note that the same proof also works for single-sample algorithms. In this case the preprocessing reads no input values, and the only restriction is $\beta \leqslant L$. Setting $\beta = \Theta(\log(n)/\log\log(n))$ yields a lower bound of $\Omega(\log(n)/\log\log(n))$ on the expected running time of any single-sample algorithm for SortedProportionalSampling.

3.2.3. Subset Sampling on Sorted Probabilities

We first prove two lemmas proving lower bounds for SortedSubsetSampling in different situations. Then we show how the lower bound of Theorem 1.9 follows from these lemmas.

Lemma 3.8. *Let $\beta, \gamma \in \{2, \dots, n\}$ with $\gamma \leqslant \beta$. Consider any data structure for* SortedSubsetSampling *with preprocessing time $\varepsilon \log_\beta n$ (where $\varepsilon > 0$ is a sufficiently small constant) and query time $t_q(n, \mu)$. Then for any $\mu = \mu(n)$ with $\gamma(1 + \mu) = \mathcal{O}(\log_\gamma n)$ we have $t_q(n, \mu) = \Omega(\gamma\mu)$.*

Proof. We closely follow the proof of the lower bound of Theorem 1.6 (Section 3.2.2). Let $s_i := \sum_{j=0}^{i-1} \gamma^j = (\gamma^i - 1)/(\gamma - 1)$. For ease of readability, assume that $n = s_L$ for some $L \in \mathbb{N}$. Then $L = \Theta(\log_\gamma n)$. We consider blocks $B_i := \{s_i, s_i + 1, \dots, s_i + \gamma^{i-1} - 1\}$, for $i = 1, \dots, L$, that partition $\{1, \dots, n\}$.

Let A be an instance of ArraySearch of size L, say with 1-bit at position ℓ^*. To construct the instance $\mathbf{p} = \mathbf{p}^A = (p_1^A, \dots, p_n^A)$ we set for any $\ell \in \{1, \dots, L\}$ and $j \in B_\ell$ the input to $p_j^A := \alpha \cdot \gamma^{-\ell + A[\ell]}$, for some $\alpha > 0$. As block B_ℓ has size γ^ℓ, the total probability mass of B_ℓ is $\sum_{j \in B_\ell} p_j^A = \alpha \cdot \gamma^{A[\ell]}$. Observe that $\mu = \sum_{i=1}^n p_i^A = \alpha(L + \gamma - 1)$. Furthermore, note that $p_1^A, \dots, p_n^A$ is indeed sorted.

Assume for the sake of contradiction that there is a data structure for SortedSubsetSampling where the preprocessing reads at most $\varepsilon \log_\beta n \leqslant \varepsilon \log_\gamma n$ input values and the querying reads at most $\varepsilon\gamma\mu$ input values in expectation, for a sufficiently small constant $\varepsilon > 0$.

On $\mathbf{p}^A$ we run the preprocessing and then K times the query algorithm, sampling K sets $X_1, \dots, X_K \subseteq \{1, \dots, n\}$. For every $x \in \bigcup_{k=1}^K X_k$ we determine its block B_y and check whether $A[y] = 1$. If so, we have found the 1-bit of A. Otherwise we linearly search for the 1-bit of A.

This yields an algorithm for ArraySearch, let us analyze its expected number of accesses to A. Let ℓ^* be the position of the 1-bit in A. The probability of not sampling any $i \in B_{\ell^*}$ in any of the K queries is

$$\prod_{i \in B_{\ell^*}} (1 - p_i)^K = (1 - \alpha \cdot \gamma^{-\ell^* + 1})^{K\gamma^{\ell^*}} \leqslant \exp(-K\alpha\gamma).$$

This probability becomes at most ε by setting $K = \lceil \ln(1/\varepsilon)/(\alpha\gamma) \rceil = \Theta(1 + \log(1/\varepsilon)/(\alpha\gamma))$. Hence, the expected number of accesses to A of the constructed algorithm is (counting preprocessing, K queries, and a linear search through A with probability at most ε)

$$\mathcal{O}(\varepsilon L + K \cdot \varepsilon\gamma\mu + \varepsilon \cdot L) \leqslant \mathcal{O}(\varepsilon(L + \gamma\mu + \log(1/\varepsilon)\mu/\alpha)) \leqslant \mathcal{O}(\varepsilon(\log(1/\varepsilon)(L + \gamma) + \gamma\mu)),$$

using $\mu = \alpha(L + \gamma - 1)$. Because of the conditions $\gamma(1 + \mu) = \mathcal{O}(\log_\gamma n)$ and $\gamma \leqslant \beta$ we can further bound the expected number of accesses to A by $\mathcal{O}(\log(1/\varepsilon)\varepsilon L)$, which contradicts Fact 3.6 for sufficiently small $\varepsilon > 0$. □

Lemma 3.9. *Consider any data structure for* SortedSubsetSampling *with preprocessing time $t_p(n)$ and expected query time $t_q(n,\mu)$. For any $\mu = \mu(n) \leqslant \mathcal{O}(1)$ we have*

$$t_p(n) + t_q(n,\mu) = \Omega\Big(\frac{\log n}{\log \frac{\log n}{\mu}}\Big).$$

Proof. Let (P, Q) be a preprocessing and a query algorithm, and let $\mathbf{p}$ be an instance. Let $D = P(\mathbf{p})$ be the result of the precomputation. By definition we have for any $S \subseteq [n]$

$$\Pr[Q(\mathbf{p}, D) = S] = \left(\prod_{i\in S} p_i\right)\left(\prod_{i\in[n]\setminus S}(1-p_i)\right) =: P_{\mathbf{p}}(S),$$

meaning that we sample a set with the right probability (independent of the possible random choices in the preprocessing).

Let $\mathcal{P} \subseteq [n]$ be the positions $i \in [n]$ at which the preprocessing reads the value p_i during the computation of D, note that $|\mathcal{P}| \leqslant t_p = t_p(n)$. Without loss of generality, we can assume that $1, n \in \mathcal{P}$, i.e., that the preprocessing reads p_1 and p_n, as this adjustment of the algorithm does not increase its running time asymptotically. Furthermore, without loss of generality, we can assume that the query algorithm reads all positions i in its return set $S = Q(\mathbf{p}, D)$.

For an instance $\mathbf{p}$ and $S \subseteq \mathcal{Q} \subseteq [n]$, let $P_{\mathbf{p}}(\mathcal{Q}, S)$ be the probability that algorithm $Q(\mathbf{p}, D)$ reads exactly the values p_i with $i \in \mathcal{Q}$ and returns the set S. We clearly have

$$\sum_{\mathcal{Q}\supseteq S} P_{\mathbf{p}}(\mathcal{Q}, S) = P_{\mathbf{p}}(S). \tag{3.1}$$

Furthermore, if we assume an expected query time of at most $t_q = t_q(n,\mu)$, then there is a set $S^* \subseteq [n]$ with

$$\sum_{\substack{\mathcal{Q}\supseteq S^* \\ |\mathcal{Q}|\leqslant 2t_q}} P_{\mathbf{p}}(\mathcal{Q}, S^*) \geqslant \frac{1}{2} P_{\mathbf{p}}(S^*), \tag{3.2}$$

since otherwise

$$\Pr[Q(\mathbf{p}, D) \text{ runs for time} \leqslant 2t_q] \leqslant \sum_{\substack{\mathcal{Q}, S\subseteq[n] \\ \mathcal{Q}\supseteq S \\ |\mathcal{Q}|\leqslant 2t_q}} P_{\mathbf{p}}(\mathcal{Q}, S) < \frac{1}{2}\sum_{S\subseteq[n]} P_{\mathbf{p}}(S) = \frac{1}{2},$$

in contrast to Markov's inequality. Here, we used that $|\mathcal{Q}|$ is a lower bound on the running time of $Q(\mathbf{p}, D)$.

From (3.2) we infer, using the maximum-arithmetic mean inequality, that there is a set $\mathcal{Q}^* \supseteq S^*$ with

$$P_{\mathbf{p}}(\mathcal{Q}^*, S^*) \geqslant P_{\mathbf{p}}(S^*) \Big/ 2\binom{n}{2t_q}. \tag{3.3}$$

Now we fix the instance $\mathbf{p} = (p_1, \ldots, p_n)$ by setting

$$p_i := \frac{\alpha}{i},$$

for a parameter $0 < \alpha \leqslant 1/2$ chosen such that $\sum_{i=1}^{n} p_i = \alpha H_n = \mu = \mu(n)$, so that $\alpha = \Theta(\mu / \log n)$. Fixing sets $S^*, \mathcal{Q}^*$ as above for this instance $\mathbf{p}$, we define a second instance $\mathbf{p}' = (p'_1, \ldots, p'_n)$ by setting

$$p'_i := \min\{p_j \mid i \geqslant j \in \mathcal{Q}^* \cup \mathcal{P}\}.$$

That is, $\mathbf{p}$ and $\mathbf{p}'$ agree on the read positions $\mathcal{Q}^*$ and $\mathcal{P}$, and at all other positions p'_i is as large as possible with $\mathbf{p}'$ still being sorted. This means that the preprocessing and the query algorithm cannot distinguish between both instances, implying a critical property we will use,

$$P_{\mathbf{p}'}(\mathcal{Q}^*, S^*) = P_{\mathbf{p}}(\mathcal{Q}^*, S^*).$$

With this, we obtain

$$P_{\mathbf{p}'}(S^*) \overset{(3.1)}{\geqslant} P_{\mathbf{p}'}(\mathcal{Q}^*, S^*) = P_{\mathbf{p}}(\mathcal{Q}^*, S^*) \overset{(3.3)}{\geqslant} \frac{1}{2\binom{n}{2t_q}} P_{\mathbf{p}}(S^*). \tag{3.4}$$

We next bound $P_{\mathbf{p}}(S^*)$ and $P_{\mathbf{p}'}(S^*)$. For the former we get

$$P_{\mathbf{p}}(S^*) = \left(\prod_{i \in S^*} p_i\right)\left(\prod_{i \in [n] \setminus S^*} (1 - p_i)\right) = W \prod_{i=1}^{n} (1 - p_i),$$

where $W := \prod_{i \in S^*} \frac{p_i}{1 - p_i}$. Since $p_i \leqslant \alpha \leqslant 1/2$ we have $1 - p_i \geqslant 4^{-p_i}$ for all $i \in [n]$, which yields, as $\mu = \mu(n) \leqslant \mathcal{O}(1)$,

$$P_{\mathbf{p}}(S^*) \geqslant W \cdot 4^{-\mu} = \Omega(W). \tag{3.5}$$

Denote the read positions by $\mathcal{Q}^* \cup \mathcal{P} = \{i_1, \ldots, i_k\}$ with $i_1 \leqslant \ldots \leqslant i_k$, and note that $k \leqslant t_p + t_q$. By assumption, we have $i_1 = 1$, $i_k = n$, and we define $i_{k+1} := n + 1$. For $P_{\mathbf{p}'}(S^*)$ we now get

$$\begin{aligned} P_{\mathbf{p}'}(S^*) &= \left(\prod_{i \in S^*} p'_i\right)\left(\prod_{i \in [n] \setminus S^*} (1 - p'_i)\right) \\ &= \left(\prod_{i \in S^*} p_i\right)\left(\prod_{i \in (\mathcal{Q}^* \cup \mathcal{P}) \setminus S^*} (1 - p_i)\right)\left(\prod_{\ell=1}^{k} (1 - p_{i_\ell})^{i_{\ell+1} - i_\ell - 1}\right), \end{aligned}$$

which simplifies to

$$P_{\mathbf{p}'}(S^*) = W \prod_{\ell=1}^{k} (1 - p_{i_\ell})^{i_{\ell+1} - i_\ell}.$$

Using $1 - x \leqslant e^{-x}$ for $x \geqslant 0$ this yields

$$P_{\mathbf{p}'}(S^*) \leqslant W \cdot \exp\left(-\sum_{\ell=1}^{k} p_{i_\ell}(i_{\ell+1} - i_\ell)\right) = W \cdot \exp\left(-\alpha \sum_{\ell=1}^{k} \left(\frac{i_{\ell+1}}{i_\ell} - 1\right)\right).$$

Using the arithmetic-geometric mean inequality we obtain

$$\frac{1}{k}\sum_{\ell=1}^{k} \frac{i_{\ell+1}}{i_\ell} \geqslant \left(\prod_{\ell=1}^{k} \frac{i_{\ell+1}}{i_\ell}\right)^{1/k} \geqslant n^{1/k},$$

which yields $P_{\mathbf{p}'}(S^*) \leqslant W \cdot \exp\big(-\alpha k(n^{1/k} - 1)\big)$. Combining this with (3.4) and (3.5),

$$\exp\Big(-k\alpha(n^{1/k} - 1)\Big) \geqslant \Omega(n^{-(2t_q+1)}).$$

Taking the logarithm twice and rearranging yields

$$k \geqslant \frac{\log n}{\log(1 + (2t_q + 1)\ln n/(k\alpha))}.$$

Using $t_p + t_q \geqslant k \geqslant 1$, $\alpha = \Theta(\mu/\log n)$, and $t_q = \mathcal{O}(\log n)$ (otherwise the claim follows directly), we obtain

$$t_p + t_q \geqslant \frac{\log n}{\log(\mathcal{O}(\log^3(n)/\mu))} = \Omega\Big(\frac{\log n}{\log(\log(n)/\mu)}\Big). \qquad \square$$

A tedious case distinction now shows that the lower bound of Theorem 1.9 follows from the above two lemmas.

Proof of Theorem 1.9, lower bound. We prove that any data structure for SORTED-SUBSETSAMPLING with $\varepsilon \log_\beta n$ preprocessing time (where $\varepsilon > 0$ is a sufficiently small constant) needs query time $\Omega(t_q^\beta(n, \mu))$ for any $\mu = \mu(n)$, where

$$t_q^\beta(n,\mu) = \begin{cases} \mathcal{O}(\mu), & \text{if } \mu \geqslant \frac{1}{2}\log n, \\ \mathcal{O}(1 + \beta\mu), & \text{if } \mu < \frac{1}{\beta}\log_\beta n, \\ \mathcal{O}\big(\frac{\log n}{\log(\frac{\log n}{\mu})}\big), & \text{otherwise.} \end{cases}$$

We consider six cases depending on μ and β, in each case reducing the claim to Lemma 3.8 or 3.9.

Case 1, $\mu \geqslant \frac{1}{2}\log n$: As the expected output size is μ, the expected query time is always $\Omega(\mu)$, which is tight in this case.

Case 2, $1 \leqslant \mu < \frac{1}{\beta}\log_\beta n$: These inequalities imply $\beta \leqslant \beta\mu \leqslant \log_\beta n$. Thus, Lemma 3.8 with $\gamma := \beta$ applies, showing that the query time is $\Omega(\beta\mu)$. As any algorithm takes time $\Omega(1)$, the query time is also bounded by $\Omega(1 + \beta\mu)$, as desired.

Case 3, $\mu \geqslant 1$ and $\frac{1}{\beta}\log_\beta n \leqslant \mu < \frac{1}{2}\log n$: In this case, we can choose $2 \leqslant \gamma \leqslant \beta$ such that $\mu = \Theta(\frac{1}{\gamma}\log_\gamma n)$. Solving for γ yields $\gamma = \Theta\big(\frac{\log n}{\mu} \big/ \log\frac{\log n}{\mu}\big)$. We have

$\gamma \leqslant \gamma\mu \leqslant \mathcal{O}(\log_\gamma n)$, so Lemma 3.8 is applicable, yielding a lower bound of $\Omega(\gamma\mu) = \Omega\big(\frac{\log n}{\log \frac{\log n}{\mu}}\big)$.

Case 4, $\mu < 1$ and $\frac{1}{\beta}\log_\beta n \leqslant \mu < \frac{1}{2}\log n$: Note that $\mu \geqslant \frac{1}{\beta}\log_\beta n$ implies $\beta^2 \geqslant \beta\log\beta \geqslant \frac{1}{\mu}\log n$ so that $\log\beta = \Omega\big(\log\frac{\log n}{\mu}\big)$. Hence, the preprocessing time is $\varepsilon\log_\beta n = \mathcal{O}\big(\varepsilon\frac{\log n}{\log\frac{\log n}{\mu}}\big)$. For sufficiently small $\varepsilon > 0$, Lemma 3.9 now implies $t_q(n,\mu) = \Omega\big(\frac{\log n}{\log\frac{\log n}{\mu}}\big)$, as desired.

Case 5, $\mu < 1$ and $\mu < \frac{1}{\beta^3}\log n$: Note that $\mu < \frac{1}{\beta^3}\log n$ implies $\mu < \frac{1}{\beta}\log_\beta n$. Thus, if $\beta\mu < 1$ then our query time is $\mathcal{O}(1)$, which is clearly optimal. Hence, assume $\beta\mu \geqslant 1$. Together with $\mu < \frac{1}{\beta^3}\log n$ this implies $\beta \leqslant \sqrt{\log n}$. Hence,

$$\log_\beta n \geqslant \Omega(\tfrac{\log n}{\log\log n}) \gg \mathcal{O}(\sqrt{\log n}) \geqslant \beta \geqslant \Omega(\beta(1+\mu)),$$

where the last inequality uses $\mu < 1$. Thus, Lemma 3.8 is applicable with $\gamma := \beta$ and we obtain a lower bound of $t_q(n,\mu) = \Omega(\beta\mu) = \Omega(1+\beta\mu)$, as desired.

Case 6, $\mu < 1$ and $\frac{1}{\beta^3}\log n \leqslant \mu < \frac{1}{\beta}\log_\beta n$: Then $\log\beta = \Omega\big(\log\frac{\log n}{\mu}\big)$ and $\log_\beta n = \mathcal{O}\big(\frac{\log n}{\log\frac{\log n}{\mu}}\big)$. Hence, with $\varepsilon\log_\beta n$ preprocessing time and sufficiently small $\varepsilon > 0$, Lemma 3.9 implies that $t_q(n,\mu) = \Omega\big(\frac{\log n}{\log\frac{\log n}{\mu}}\big) \geqslant \Omega(\log_\beta n) \geqslant \Omega(\beta\mu)$, where the last inequality follows from $\mu < \frac{1}{\beta}\log_\beta n$. Since any algorithm takes time $\Omega(1)$, this yields a lower bound of $\Omega(1+\beta\mu)$, as desired. □

3.3. Reduction: Proportional to Subset Sampling

In this section, we present a reduction from (Sorted or Unsorted) ProportionalSampling to (Sorted or Unsorted) SubsetSampling. This yields an alternative proof of the upper bounds for ProportionalSampling (Theorems 1.6 and 1.7) using the upper bounds for SubsetSampling (Theorems 1.9 and 1.10). Moreover, it shows that the classic ProportionalSampling problem is easier than SubsetSampling (or the former can be seen as a special case of the latter).

We present our reduction for the special case of $1/\beta \leqslant \mu \leqslant 1$ first, where $\beta \geqslant 1$ is a parameter. Then we reduce the general case with arbitrary μ to the special case.

3.3.1. Special Case $1/\beta \leqslant \mu \leqslant 1$

Let $\mathbf{p}$ be an instance to SortedProportionalSampling or UnsortedProportionalSampling with μ in the range $[1/\beta, 1]$. Instead of $\mathbf{p}$ we consider $\mathbf{p}' = (p'_1, \dots, p'_n)$ with $p'_i := p_i/(1+p_i)$. Note that if $\mathbf{p}$ is sorted then $\mathbf{p}'$ is also sorted. Moreover, $\mu' := \sum_{i=1}^n p'_i$ is in the range $[\mu/2, \mu]$, thus in the range $[1/2\beta, 1]$.

Let $Y = \text{ProportionalSampling}(\mathbf{p})$ be the random variable denoting proportional sampling on input $\mathbf{p}$, and $X = \text{SubsetSampling}(\mathbf{p}')$ be the random variable denoting subset sampling on input $\mathbf{p}'$. Then conditioned on sampling exactly one

element $X = \{i\}$, this element i is distributed exactly as Y, as formulated by the following lemma.

Lemma 3.10. *We have for all $i \in [n]$*

$$\Pr[X = \{i\} \mid |X| = 1] = \Pr[Y = i].$$

Proof. By applying Bayes' rule we infer that

$$\begin{aligned}\Pr\left[X = \{i\} \mid |X| = 1\right] &= \Pr[X = \{i\}] / \Pr[|X| = 1] \\ &= \left(\frac{p'_i}{1-p'_i}\prod_{k=1}^{n}(1-p'_k)\right) / \left(\sum_{j=1}^{n}\frac{p'_j}{1-p'_j}\prod_{k=1}^{n}(1-p'_k)\right) \\ &= \left(\frac{p'_i}{1-p'_i}\right) / \left(\sum_{j=1}^{n}\frac{p'_j}{1-p'_j}\right)\end{aligned}$$

Plugging in the definition of p'_i yields

$$\Pr[X = \{i\} \mid |X| = 1] = \frac{p_i}{\sum_{j=1}^{n} p_j} = \Pr[Y = i],$$

and the statement is shown. □

Moreover, the probability of sampling exactly one element is not too small, as shown in the following lemma. This bound is not best possible but sufficient for our purposes.

Lemma 3.11. *With the definitions and assumptions of this section we have*

$$\Pr[|X| = 1] \geqslant \mu/4.$$

Proof. First, observe that by Markov's inequality

$$\Pr[|X| \geqslant 2] \leqslant \mathbb{E}[|X|]/2 = \mu'/2 \leqslant 1/2,$$

and thus, $\Pr[|X| \in \{0,1\}] \geqslant 1/2$. Moreover, the definition of X implies that

$$\Pr[|X| = 0] = \prod_{k=1}^{n}(1-p'_k)$$

and

$$\Pr[|X| = 1] = \sum_{j=1}^{n}\frac{p'_j}{1-p'_j}\prod_{k=1}^{n}(1-p'_k) = \mu \cdot \Pr[|X| = 0].$$

By putting everything together we obtain that $\Pr[|X| = 1](1 + \frac{1}{\mu}) \geqslant 1/2$, and thus

$$\Pr[|X| = 1] \geqslant \mu \cdot \frac{1}{2(1+\mu)} \geqslant \frac{\mu}{4},$$

as claimed. □

We put these facts together to show the following result.

Lemma 3.12. *Let $\beta \geqslant 1$. Assume that (*Sorted *or* Unsorted*)* SubsetSampling *can be solved in preprocessing time $t_p(n,\mu)$ and expected query time $t_q(n,\mu)$, where t_p and t_q are monotonically increasing in n and μ. Then (*Sorted *or* Unsorted*, respectively)* ProportionalSampling *on instances with $1/\beta \leqslant \mu \leqslant 1$ can be solved in preprocessing time $\mathcal{O}(t_p(n,\mu))$ and expected query time $\mathcal{O}(\frac{1}{\mu} \cdot t_q(n,\mu))$.*

Proof. For preprocessing, given input $\mathbf{p}$, we run the preprocessing of SubsetSampling on input $\mathbf{p}'$. This does not mean that we compute the vector $\mathbf{p}'$ explicitly, but if the preprocessing algorithm of SubsetSampling reads the i-th input value, we compute $p'_i = p_i/(1+p_i)$ on the fly, so that preprocessing needs running time $\mathcal{O}(t_p(n,\mu))$ (recall that $\mu' \leqslant \mu$). It allows to sample X later on in expected running time $\mathcal{O}(t_q(n,\mu))$ using the same trick of computing $\mathbf{p}'$ on the fly.

For querying, we repeatedly sample X until we sample a set S of size one. Returning the unique element of S results in a proper sample according to SortedProportionalSampling by Lemma 3.10. Moreover, by Lemma 3.11 and the fact that sampling X needs expected time $\mathcal{O}(t_q(n,\mu))$ after our preprocessing, the total expected query time is $\mathcal{O}(\frac{1}{\mu} \cdot t_q(n,\mu))$. □

3.3.2. General Case

In this subsection, we reduce the general case with arbitrary μ to the special case $1/\beta \leqslant \mu \leqslant 1$. In the unsorted case, we simply compute μ exactly in time $\mathcal{O}(n)$, which shows the following proposition. In the sorted case, we approximate μ using an idea of Section 3.1.1, see Proposition 3.14.

Proposition 3.13. *Assume that* UnsortedSubsetSampling *can be solved in preprocessing time $t_p(n,\mu)$ and expected query time $t_q(n,\mu)$, where t_p and t_q are monotonically increasing in n and μ. Then* UnsortedProportionalSampling *can be solved in preprocessing time $\mathcal{O}(n + t_p(n,1))$ and expected query time $\mathcal{O}(t_q(n,1))$.*

Note that plugging Theorem 1.10 into the above proposition yields the upper bound of Theorem 1.7.

Proof. In the preprocessing we compute μ in time $\mathcal{O}(n)$, and set $\tilde{p}_i := p_i/\mu$ for $i \in [n]$. Then we run the algorithm guaranteed by Lemma 3.12 on $\tilde{p}_1, \ldots, \tilde{p}_n$. □

Proposition 3.14. *Let $\beta \in \{2, \ldots, n\}$. Assume that* SortedSubsetSampling *can be solved in preprocessing time $t_p(n,\mu)$ and expected query time $t_q(n,\mu)$, where t_p and t_q are monotonically increasing in n and μ. Then* SortedProportionalSampling *can be solved in preprocessing time $\mathcal{O}(\log_\beta n + t_p(n,1))$ and expected query time $\mathcal{O}(\max_{1/\beta \leqslant \nu \leqslant 1} \frac{1}{\nu} t_q(n,\nu))$.*

Note that plugging Theorem 1.9 into the above proposition yields the upper bound of Theorem 1.6 (to see the bound on the query time, note that we can set $t_q(n,\mu) = \mathcal{O}(1+\beta\mu)$ so that $\max_{1/\beta \leqslant \nu \leqslant 1} \frac{1}{\nu} t_q(n,\nu) = \mathcal{O}(\max_{1/\beta \leqslant \nu \leqslant 1} \frac{1}{\nu}(1+\beta\nu)) = \mathcal{O}(\beta)$).

Proof. Let $\mathbf{p}$ be an instance of SortedProportionalSampling with $\mu = \sum_{i=1}^{n} p_i$. As in Section 3.1.1 we consider the blocks $B_k := \{i \in [n] \mid \beta^k \leqslant i < \beta^{k+1}\}$ with $0 \leqslant k \leqslant L := \lfloor \log_\beta n \rfloor$ and set $\overline{p}_i := p_{\beta^k}$ for $i \in B_k$. Then for $\overline{\mu} := \sum_{i=1}^{n} \overline{p}_i$ we have $\mu \leqslant \overline{\mu} \leqslant \beta \cdot \mu$ by Lemma 3.1. Note that we can compute $\overline{\mu}$ in time $\mathcal{O}(\log_\beta n)$, as

$$\overline{\mu} = \sum_{k=0}^{L} p_{\beta^k} \cdot \big(\min(\beta^{k+1}, n+1) - \beta^k\big).$$

With these observations at hand, for preprocessing, we compute $\overline{\mu}$ and consider $\mathbf{p}' = (p'_1, \ldots, p'_n)$ with $p'_i := p_i/\overline{\mu}$. Since $\mu \leqslant \overline{\mu} \leqslant \beta \cdot \mu$ we have $\mu' := \sum_{i=1}^{n} p'_i$ in the range $[1/\beta, 1]$. Thus, we can run the preprocessing of SortedProportionalSampling on $\mathbf{p}'$; Lemma 3.12 is applicable since $\mathbf{p}'$ has $\mu' \in [1/\beta, 1]$. We do this without computing the whole vector $\mathbf{p}'$. Instead, if the preprocessing algorithm reads the i-th input value, we compute p'_i on the fly. This way we need a total running time for preprocessing of $\mathcal{O}(\log_\beta n + t_p(n, 1))$.

For querying, Lemma 3.12 allows us to query according to $\mathbf{p}'$ in expected running time $\mathcal{O}(\frac{1}{\mu'} t_q(n, \mu')) \leqslant \mathcal{O}(\max_{1/\beta \leqslant \nu \leqslant 1} \frac{1}{\nu} t_q(n, \nu))$, where we again compute values of $\mathbf{p}'$ on the fly as needed. As we want to sample proportionally to the input distribution, a sample with respect to $\mathbf{p}'$ has the same distribution as a sample with respect to $\mathbf{p}$, so that we simply return the sampled number. □

3.4. Relaxations

In this section, we describe some natural relaxations for the input and machine model studied so far in this chapter.

Large Deviations for the Running Times The query running times in Theorems 1.6, 1.9, and 1.10 are, in fact, not only small in expectation, but they are also concentrated, i.e., they satisfy large deviation estimates in the following sense. Let t be the expected running time bound and T the actual running time. Then

$$\Pr[T > kt] = e^{-\Omega(k)},$$

where the asymptotics are with respect to k. This is shown rather straightforwardly along the lines of our proofs of these theorems. The fundamental reason for this is that the size of the random set X is concentrated. Indeed, let X_i be an indicator random variable for the i-th element as above. Then for any $a > 1$ we obtain along the lines of the proof of the Chernoff bound

$$\Pr[|S| > k(\mu+1)] = \Pr\left[a^{\sum_{i=1}^{n} X_i} > a^{k(\mu+1)}\right] \leqslant \mathbb{E}\left[a^{\sum_{i=1}^{n} X_i}\right] a^{-k(\mu+1)}.$$

The independence of the X_i's implies that

$$\Pr[|S| > k(\mu+1)] \leqslant \prod_{i=1}^{n} \mathbb{E}[a^{X_i}] \cdot a^{-k(\mu+1)} = \prod_{i=1}^{n} (ap_i + (1-p_i)) \cdot a^{-k(\mu+1)},$$

and we obtain

$$\Pr[|S| > k(\mu+1)] \leqslant \exp\left\{(a-1)\mu - k(\mu+1)\ln a\right\}.$$

Setting $a = k+1$ yields

$$\Pr[|S| > k(\mu+1)] \leqslant \exp\left\{k\mu - k(\mu+1)\ln(k+1)\right\} \leqslant (k+1)^{-k},$$

for any $k \geqslant 2$, as claimed.

Partially Sorted Input The condition of sorted input for SortedSubsetSampling and SortedProportionalSampling can easily be relaxed, as long as we have sorted upper bounds of the probabilities. Given input $\mathbf{p}$ and sorted $\overline{\mathbf{p}}$ with $p_i \leqslant \overline{p}_i$ for all $i \in [n]$, we simply sample according to $\overline{\mathbf{p}}$ and use rejection to get down to the probabilities $\mathbf{p}$. This allows for the optimal query time $\mathcal{O}(1+\mu)$ as long as $\overline{\mu} = \sum_{i=1}^{n} \overline{p}_i = \mathcal{O}(1+\mu)$, where $\mu = \sum_{i=1}^{n} p_i$.

Unimodular Input Many natural distributions $\mathbf{p}$ are not sorted, but unimodular, meaning that p_i is monotonically increasing for $1 \leqslant i \leqslant m$ and monotonically decreasing for $m \leqslant i \leqslant n$ (or the other way round). Knowing m, we can run the algorithms developed in this chapter on both sorted halfs, and combine the return values, which gives an optimal query algorithm for unimodular inputs. Alternatively, if we have strong monotonicity, we can search for m in time $\mathcal{O}(\log n)$ using ternary search.

This can be naturally generalized to k-modular inputs, where the monotonicity changes k times.

Approximate Input In some applications it may be costly to compute the probabilities p_i exactly, but we are able to compute approximations $\overline{p}_i(\varepsilon) \geqslant p_i \geqslant \underline{p}_i(\varepsilon)$, with relative error at most ε, where the cost of computing these approximations depends on ε. We can still guarantee optimal query time, if the costs of computing these approximations are small enough, see e.g. [87].

Indeed, we can surely sample a superset $\overline{S}$ with respect to the probabilities $\overline{p}_i(\frac{1}{2})$. Then we want to use rejection, i.e., for each element $i \in \overline{S}$ we want to compute a uniformly random number $r \in [0,1]$ and delete i from $\overline{S}$ if $r \cdot \overline{p}_i(\frac{1}{2}) > p_i$, to get a sample set S. This check can be performed as follows. We initialize $k := 1$. If $r \cdot \overline{p}_i(\frac{1}{2}) > \overline{p}_i(2^{-k})$ we delete i from $\overline{S}$. If $r \cdot \overline{p}_i(\frac{1}{2}) \leqslant \underline{p}_i(2^{-k})$ we keep i and are done. Otherwise, we increase k by 1. This method needs an expected number of $\mathcal{O}(1)$ rounds of increasing k; the probability of needing k rounds is $\mathcal{O}(2^{-k})$. Hence, if the cost of computing $\overline{p}_i(\varepsilon)$ and $\underline{p}_i(\varepsilon)$ is $\mathcal{O}(\varepsilon^{-c})$ with $c < 1$, the expected overall cost is constant, and we get an optimal expected query time of $\mathcal{O}(1+\mu)$.

Word RAM Throughout this chapter we worked in the Real RAM model of computation. In the more realistic Word RAM model each cell consists of $w = \Omega(\log n)$ bits and any reasonable operation on two words can be performed in constant time, see Section 1.1.2. In this dissertation, we prove that on the Word RAM Bernoulli and geometric random variates can be drawn in constant time (Chapter 4) and the classic alias method for UnsortedProportionalSampling still works (see Section 1.2). This already allows one to translate large parts of the algorithms of this chapter to the Word RAM. Unfortunately, terms like $\prod_{1 \leqslant k \leqslant n}(1 - p_k)$ (see Section 3.1.2) cannot be evaluated exactly on the Word RAM, as the result would need at least n bits. This difficulty can be solved by working with $\mathcal{O}(\log n)$ bit approximations and increasing the precision as needed, similarly to the generalization to *approximate input* that we discussed in the last paragraph. This way one can obtain a complete translation of our algorithms to the Word RAM. We omit the details.

Our lower bounds hold for both models since we bound the number of probed inputs p_i, and in both models in unit time we can only read a single input p_i.

4 Sampling from Special Distributions

This chapter is based on [14] and a small part of [5]. I contributed at least 50% at all stages and to all parts of [14]. Moreover, I contributed the part of [5] presented in Section 4.7.

[5] K. Bringmann, F. Kuhn, K. Panagiotou, U. Peter, and H. Thomas. "Internal DLA: Efficient Simulation of a Physical Growth Model." In: *Proc. 41th International Colloquium on Automata, Languages, and Programming (ICALP'14)*. Vol. 8572. LNCS. 2014, 247–258.

[14] K. Bringmann and T. Friedrich. "Exact and efficient generation of geometric random variates and random graphs." In: *Proc. 40th International Colloquium on Automata, Languages, and Programming (ICALP'13)*. Vol. 7965. LNCS. 2013, 267–278.

In this chapter, we present Word RAM algorithms for sampling from several special distributions. In Section 4.1 we fix notation and our input model. We consider Bernoulli random variates in Section 4.2. For geometric random variates, in Section 4.3 we translate the Real RAM algorithm to the Word RAM via multi-precision arithmetic and show that this yields sub-optimal running time, and in Section 4.4 we present a lower bound and our optimal algorithm. We adapt this algorithm for bounded geometric random variates in Section 4.5. The applications to sampling random graphs are presented in Section 4.6. Finally, we show how to sample binomial random variates in Section 4.7. We restate the theorems from Section 1.7 that we prove in this chapter in the respective sections.

4.1. Preliminaries

Word RAM model Typically, on a Word RAM it is assumed that a pointer consists of one cell and one may address only cells for which a pointer fits into a cell. However, since a geometric random variate can be arbitrary large, we have to adapt our machine model to allow potentially unbounded space usage. To this

end, we could assume that any number of consecutive cells can form a pointer P, and accessing the memory cell pointed at takes time proportional to the number of memory cells forming P. Our results would remain valid for even larger access times, as long as the cost of accessing the i-th memory cell is $\mathcal{O}((2-\varepsilon)^{i/2^w})$ for some $\varepsilon > 0$.

For simplicity, we will instead assume that accessing any memory cell still takes time $\mathcal{O}(1)$.

Probability distributions Let $p \in (0, 1]$. The *Bernoulli* distribution $\mathrm{Ber}(p)$ takes values in $\{0, 1\}$ such that $\Pr[\mathrm{Ber}(p) = 1] = 1 - \Pr[\mathrm{Ber}(p) = 0] = p$. The *geometric* distribution $\mathrm{Geo}(p)$ takes values in $\mathbb{N}_0$ such that for any $i \in \mathbb{N}_0$, we have $\Pr[\mathrm{Geo}(p) = i] = p(1-p)^i$. For $n \in \mathbb{N}_0$, we define the *bounded geometric* distribution $\mathrm{Geo}(p, n)$ to be $\min\{n, \mathrm{Geo}(p)\}$. This means that $\mathrm{Geo}(p, n)$ takes values in $\{0, \ldots, n\}$ such that for any $i \in \mathbb{N}_0$, $i < n$, we have $\Pr[\mathrm{Geo}(p, n) = i] = p(1-p)^i$, and $\Pr[\mathrm{Geo}(p, n) = n] = (1-p)^n$. The *uniform* distribution $\mathrm{Uni}[0, 1]$ takes values in $[0, 1]$ with uniform probability. For $n \in \mathbb{N}$, we define the uniform distribution $\mathrm{Uni}(n)$ to be the uniform distribution over $\{0, \ldots, n-1\}$. The *binomial* distribution $\mathrm{Bin}(n, p)$ takes values in $\{0, \ldots, n\}$ and has $\Pr[\mathrm{Bin}(n, p) = i] = \binom{n}{i} 2^{-n}$.

Random graph models In the *Erdős-Rényi* [70] random graph model $G(n, p)$, each edge of an n vertex graph is independently present with probability p. This yields a binomial degree distribution and approaches a Poisson distribution in the limit. As many real-world networks have power-law degree distributions, we also study inhomogeneous random graphs. We consider *Chung-Lu* [76] graphs $G(n, W)$ with n vertices and weights $W = (W_1, W_2, \ldots, W_n) \in \mathbb{R}^n_{>0}$. In this model, an edge between two vertices i and j is independently present with probability $p_{i,j} := \min\{W_i W_j / \sum_k W_k, 1\}$. We will assume that the expected number of edges m is $\Theta(\sum_i W_i)$. This is known to hold if the W_i follow a *power-law*. The related definitions of generalized random graph [88] with $p_{ij} = W_i W_j / (\sum_k W_k + W_i W_j)$ and Norros-Reittu random graphs [89] with $p_{i,j} = 1 - \exp(-W_i W_j / \sum_k W_k)$ can be handled in a similar way. However, we will focus on Chung-Lu random graphs.

Input model In all sections except for Section 4.2, we assume that we can compute arbitrary precision floating point approximations of the input p. In particular, we can compute an *exponent* $k \in \mathbb{N}_0$ such that $2^{-k} \geqslant p > 2^{-k-2}$. Moreover, for any $i \in \mathbb{N}$ we can compute a number $p_i \leqslant 1$ such that $|p_i - 2^k p| \leqslant 2^{-i}$. We can assume that p_i has at most $i + 1$ bits (otherwise take the first $i + 1$ bits of p_{i+1}, which are a 2^{-i}-approximation of $2^k p$). Since we assumed $w = \Omega(\log \log(1/p))$, the exponent k fits into $\mathcal{O}(1)$ words; this resembles the usual assumption that we can compute with numbers as large as the input or output size in constant time. Furthermore, we want to assume that p_i can be computed in time $i^{\mathcal{O}(1)}$. This means that p can be approximated efficiently. However, it would be sufficient even if the running time was $\mathcal{O}((2-\varepsilon)^i)$ for some constant $\varepsilon > 0$. All numbers other than the input parameter p will be encoded as simple strings of words or floating point numbers, as discussed in Section 1.1.2.

Notation For integer division we write $a \operatorname{div} b := \lfloor a/b \rfloor$ for $a, b \in \mathbb{Z}$.

4.2. Bernoulli Random Variates

We prove the following theorem from Section 1.7. Note that here we have a slightly weaker assumption on the input p than for all other results in this chapter.

Theorem 1.11. *Let $p \in [0,1]$ and assume that for any $i \in \mathbb{N}$ we can compute a number $p_i \in [p - 2^{-i}, p + 2^{-i}]$ in time $i^{\mathcal{O}(1)}$. Then the Bernoulli random variate* $\mathrm{Ber}(p)$ *can be sampled in expected running time $\mathcal{O}(1)$ on a Word RAM.*

Note that the running time is independent of p. Moreover, note that we can assume that p_i has at most $i+1$ bits; otherwise take the first $i+1$ bits b of p_{i+1}, which fulfill $|b - p| \leqslant 2^{-i}$, since $|p_{i+1} - p| \leqslant 2^{-i-1}$ and $|b - p_{i+1}| \leqslant 2^{-i-1}$.

We can generate $\mathrm{Ber}(p)$ by generating a uniformly random real $r \in [0,1]$ and returning 1 if $r \leqslant p$ and 0, otherwise. We now describe how this can be efficiently simulated without having to cope with the real number r at once.

Let $r \sim \mathrm{Uni}[0,1]$. We get an approximation r_i of r by sampling i random bits. Then $r_i \leqslant r < r_i + 2^{-i}$. Comparing r_i and p_i, it can happen that the intervals $[r_i, r_i + 2^{-i}]$ and $[p_i - 2^{-i}, p_i + 2^{-i}]$ are non-intersecting. In this case, $r_i < p_i$ implies $r < p$, and similarly $r_i > p_i$ implies $r > p$, so we are done. Otherwise, we can increase the precision i by 1 and repeat this process (remembering the former random choices we made for r). Note that the probability that these intervals are non-intersecting is at most $4 \cdot 2^{-i}$; there are at most 4 choices for r_i such that they intersect. Hence, the probability that we need precision at least i is at most $4 \cdot 2^{-i}$. Since we can compute p_i in time $i^{\mathcal{O}(1)}$, and can clearly sample r_i and compare both intervals also in this time bound, we get an expected time of at most

$$\sum_{i=1}^{\infty} i^{\mathcal{O}(1)} \cdot 4 \cdot 2^{-i} = \mathcal{O}(1).$$

Note that we did not use any parallelism provided by the Word RAM in this section. Regarding concentration, this has $\Pr[T > t] \leqslant \exp(-t^{\Omega(1)})$, since we take time at least $i^{\mathcal{O}(1)}$ with probability at most $2^{-\Omega(i)}$.

4.3. Multi-Precision Approach

In this section we illustrate the simple approach of translating a Real RAM algorithm to the Word RAM via multi-precision arithmetic. We use geometric random variates as an example. This shows the following sub-optimal result.

Theorem 1.13. *On a Word RAM with word size $w = \Omega(\log\log(1/p))$, a geometric random variate* $\mathrm{Geo}(p)$ *with parameter $p \in (0,1]$ can be sampled in expected running time $\mathcal{O}(1 + \log(1/p) \operatorname{poly}\log\log(1/p)/w)$.*

Let us first consider the following situation on a Real RAM. Let $X = X_p$ be a random variable with some parameter p, that we omit in the following discussion as a subscript. Let X have cumulative distribution function $F(x) = \Pr[X \leqslant x]$ and let U be a uniform real in $[0, 1]$. Then X can be sampled as $X \sim F^{-1}(U)$; this is called the inversion method (see, e.g., Devroye [45]). On a Real RAM this method is applicable as long as F^{-1} is efficiently computable; the uniform variable U can be sampled in unit time by assumption. It is easy to see that for a geometric random variable $X \sim \mathrm{Geo}(p)$ we have $F^{-1}(x) = \lfloor \log_{1-p}(1-x) \rfloor$. Thus, since a logarithm of a real number can be computed in unit time by assumption, $\mathrm{Geo}(p)$ can be sampled in unit time on a Real RAM, i.e., in time that is independent of the parameter p. Note that typical implementations for sampling geometric random variables use the above algorithm (e.g. in C++11), but with the usual floating point precision, although the algorithm is only exact if used with infinite precision.

The situation changes when we consider a bounded precision machine like the Word RAM instead. We focus on the case where $F^{-1}(x)$ can be written as $\lfloor g(x) \rfloor$, with g being a smooth and monotonically increasing function; note that this is the case for geometric random variables. Suppose that we sample only an n-bit approximation $\tilde{U}$ of U, i.e., we sample the first n bits of U, so that $\tilde{U} \leqslant U < \tilde{U} + 2^{-n}$. Suppose further that we compute an n-bit approximation $\tilde{G}$ of $g(\tilde{U})$, such that $\tilde{G} \leqslant g(\tilde{U}) \leqslant (1 + 2^{-n})\tilde{G}$. $\lfloor \tilde{G} \rfloor$ is not exactly distributed as X is; we make a certain amount of error. So let us also compute an n-bit approximation $\tilde{G}'$ of $g(\tilde{U} + 2^{-n})$, such that $\tilde{G}' \geqslant g(\tilde{U} + 2^{-n}) \geqslant (1 - 2^{-n})g(\tilde{U} + 2^{-n})$. Then $\tilde{G} \leqslant g(U) \leqslant \tilde{G}'$. Thus, if $\tilde{G}$ and $\tilde{G}'$ lie in the same interval $[k, k + 1)$, for some $k \in \mathbb{N}$, we have correctly identified $F^{-1}(U) = \lfloor g(U) \rfloor$ as the number k. Of course, it may be that both values do not lie in the same interval. In this case we can increase n (e.g., double it) and repeat the process, until at some point we have approximated $g(U)$ well enough to identify its integral part.

This method terminates with probability 1: Since $g(U) \notin \mathbb{N}$ with probability 1, an approximation of $g(U)$ with (strictly positive) additive error $\min_{m \in \mathbb{N}} |g(U) - m|$ suffices. Since $\tilde{G}, \tilde{G}' \to g(U)$ for $n \to \infty$ (as g is smooth), we reach the necessary precision after a finite number of incrementations of n. Thus, we have described an exact algorithm for sampling X.

Let us now turn to the question of whether the above method is also efficient. More precisely, let us bound its expected running time (asymptotically in terms of p). We will first bound the final n in terms of $g(U), g'(U)$, and $\delta := \min_{m \in \mathbb{N}} |g(U) - m|$.

Lemma 4.1. *Let $U \in [0, 1]$ with $g(U) \notin \mathbb{N}$ be fixed and consider the final n of the sampling algorithm. Then $n = \mathcal{O}(1 + \log(g(U) + g'(U)) + \log(1/\delta))$.*

Proof. Note that the smoothness of g implies $g(\tilde{U} + 2^{-n}) - g(\tilde{U}) = \mathcal{O}(2^{-n} g'(U))$. This yields $g(\tilde{U}) = g(U) - \mathcal{O}(2^{-n} g'(U))$, and since $(1 + 2^{-n})\tilde{G} \geqslant g(\tilde{U})$, we have $(1+2^{-n})\tilde{G} + \mathcal{O}(2^{-n} g'(U)) \geqslant g(U)$. Using $\tilde{G} \leqslant g(U)$, we get $\tilde{G} \geqslant g(U) - \mathcal{O}(2^{-n}(g(U) + g'(U)))$. Analogously, we have $\tilde{G}' \leqslant g(U) + \mathcal{O}(2^{-n}(g(U) + g'(U)))$. Hence, it suffices to have $n = c + \log(g(U) + g'(U)) + \log(1/\delta)$, for c large enough, to have $\tilde{G} \geqslant g(U) - \delta$ and $\tilde{G}' < g(U) + \delta$, i.e., the method terminates for some $n = \mathcal{O}(1 + \log(g(U) + g'(U)) + \log(1/\delta))$. □

Now that we have bounded n, let the running time of one iteration be $f(n) = f_w(n)$. We have $f(n) = \Omega(n)$, since for sampling the first n bits of U we need running time $\Theta(n/w)$. Thus, doubling n after each iteration, the running time of the algorithm is bounded by the running time of the last iteration. Moreover, assume that f is at most polynomial in n, i.e., $f(n) \leqslant \mathcal{O}(n^c)$ for some $c > 0$. Then plugging in the above bound on n, we get a running time of at most

$$\mathcal{O}(1 + f(\log(g(U) + g'(U))) + f(\log(1/\delta))),$$

since for polynomial $q(x)$ we have $q(x + y) = \mathcal{O}(q(x) + q(y))$. Now, if the expected value of this term is small, then the above method is efficient.

We calculate this expected value for the geometric random variable $\mathrm{Geo}(p)$.

Lemma 4.2. *For geometric random variables, i.e., for $g(x) = \log_{1-p}(1 - x)$, we have in the situation of this section*

$$\mathbb{E}[1 + f(\log(g(U) + g'(U))) + f(\log(1/\delta))] = \mathcal{O}(1 + f(\log(1/p))).$$

Proof. In this case, $g(x) = \log_{1-p}(1 - x)$ and $g'(x) = -1/((1 - x)\log(1 - p))$. Since $1/(1 - x) \geqslant -\log(1 - x)$ holds for all $x \in [0, 1]$, we have $g(U) + g'(U) = \Theta(g'(U))$. Because $\log(1 - p)$ is asymptotically equal to $-p$, we even have $\log(g(U) + g'(U)) = \mathcal{O}(1) + \log(1/(1-U)) + \log(1/p)$, so we can bound the running time of the algorithm by $\mathcal{O}(1 + f(\log(1/(1 - U))) + f(\log(1/p)) + f(\log(1/\delta)))$. We first analyze δ. Since $\Pr[\delta \leqslant x] = g^{-1}(x) + \sum_{k\in\mathbb{N}} g^{-1}(k + x) - g^{-1}(k - x)$, the density function of δ is $1/g'(g^{-1}(x)) + \sum_{k\in\mathbb{N}} 1/g'(g^{-1}(k + x)) - 1/g'(g^{-1}(k - x))$. Since, furthermore, g' is monotonically increasing we have $1/g'(g^{-1}(k + x)) - 1/g'(g^{-1}(k - x)) \leqslant 0$, so the density is bounded by $1/g'(g^{-1}(\delta)) \leqslant 1/g'(0) = -1/\log(1 - p) \leqslant 1/p$. Thus, δ is a random variable of the following form (with $q = 1/p$): Let Y be a random variable with values in $[0, 1]$ and density bounded from above by $q \geqslant 1$. Then we show that $\mathbb{E}[f(\log(1/Y))] \leqslant \mathcal{O}(f(\log(q)) + 1)$. The expected value is maximized if Y is uniform in $[0, 1/q]$, and then it is equal to $\int_0^{1/q} q \cdot f(\log(1/y))dy$. Setting $x = \frac{1}{qy}$, this is equal to $\int_1^\infty f(\log(xq))x^{-2}dx \leqslant \mathcal{O}(\int_1^\infty f(\log(q))x^{-2}dx + \int_1^\infty \log^c(x)x^{-2}dx)$ using our assumption on f. This yields the desired bound. Note that this bound yields not only an upper bound of $\mathcal{O}(f(\log(1/p)) + 1)$ for the expected value of $f(\log(1/\delta))$, but also an upper bound of $\mathcal{O}(1)$ for the expected value of $f(\log(1/(1 - U)))$. □

Hence, the geometric random variable $\mathrm{Geo}(p)$ can be sampled in time $\mathcal{O}(1 + f(\log(1/p)))$. Note that $f(n)$, the running time of one iteration, is dominated by the time for approximating a logarithm of an n-bit number up to precision 2^{-n}. This can be done in time $\mathcal{O}(M(n)\log(n))$ (see, e.g., [90]), where $M(n)$ is the time to multiply two n-bit numbers. By using fast Fourier transforms, $M(n) = \mathcal{O}(\frac{n}{w}\log(\frac{n}{w})\log\log(\frac{n}{w}))$ [91]. The best known bound is $M(n) = \mathcal{O}(\frac{n}{w}\log(\frac{n}{w})2^{\log^*(\frac{n}{w})})$ [92, 93]. It is conjectured that $M(n) = \Omega(\frac{n}{w}\log(\frac{n}{w}))$ [91]. This gives $f(n) = \mathcal{O}(1 + \frac{n}{w}\operatorname{poly}\log(n))$, which is efficient. A geometric random variable can thus be sampled in expected time $\mathcal{O}(1 + \frac{\log(1/p)}{w}\operatorname{poly}\log\log(1/p))$ which proves Theorem 1.13. Despite its simplicity, to the best of our knowledge this approach has not been formalized yet. Note that this approach does not, however, give linear running time $\mathcal{O}(1 + \log(1/p)/w)$.

4.4. Geometric Random Variates

In this section, we prove the following two theorems from Section 1.7, i.e., a lower bound for sampling geometric random variates and an algorithm matching the lower bound.

Theorem 1.12. *On a Word RAM, any algorithm sampling a geometric random variate* $\mathrm{Geo}(p)$ *with parameter* $p \in (0,1]$ *has expected running time* $\Omega(1+\log(1/p)/w)$.

Theorem 1.14. *On a Word RAM with word size* $w = \Omega(\log\log(1/p))$*, a geometric random variate* $\mathrm{Geo}(p)$ *with parameter* $p \in (0,1]$ *can be sampled in expected running time* $\mathcal{O}(1+\log(1/p)/w)$*, which is optimal.*

We assume that we can compute arbitrary precision floating point approximations of the parameter p, as discussed in Section 4.1. In particular, we know an exponent k with $2^{-k} \geqslant p > 2^{-k-2}$.

We first prove that the expected output size is $\Theta(\log(1/p))$, which gives a lower bound of $\Omega(1+\log(1/p)/w)$ for the expected running time of any algorithm sampling $\mathrm{Geo}(p)$ on the Word RAM since (at most) w bits can be processed in parallel. This proves Theorem 1.12.

Lemma 4.3. *For any* $p \in (0,1]$*, we have* $\mathbb{E}[\log(1+\mathrm{Geo}(p))] = \Theta(\log(1/p))$*, where the lower bound holds for* $1/p$ *large enough.*

Proof. For the upper bound we can use

$$\mathbb{E}[\log(1+\mathrm{Geo}(p))] \leqslant \log(1+\mathbb{E}[\mathrm{Geo}(p)]) = \log(1/p).$$

For the lower bound we have

$$\begin{aligned}
\mathbb{E}[\log(1+\mathrm{Geo}(p))] &= \sum_{i=1}^{\infty} \log(i)\cdot p(1-p)^{i-1} \\
&\geqslant \sum_{i=\lceil 1/p\rceil}^{\infty} \log(i)\cdot p(1-p)^{i-1} \\
&\geqslant p\cdot\log(1/p)\cdot\sum_{i=\lceil 1/p\rceil}^{\infty}(1-p)^{i-1} \\
&= p\cdot\log(1/p)\cdot(1-p)^{\lceil 1/p\rceil-1}\cdot 1/p \\
&\geqslant \Omega(\log(1/p)),
\end{aligned}$$

since e.g. for $p \leqslant 1/2$ we have $(1-p)^{\lceil 1/p\rceil-1} \geqslant (1-p)^{1/p} \geqslant (1-1/2)^2 = 1/4$. □

We now present an algorithm achieving this optimal expected running time. The main trick is that we split up $\mathrm{Geo}(p)$ into $\mathrm{Geo}(p)\operatorname{div} 2^k$ and $\mathrm{Geo}(p)\bmod 2^k$. It is easy to see that both parts are independent random variables. Now, $\mathrm{Geo}(p)\operatorname{div} 2^k$ has constant expected value, so we can iteratively check whether it equals $0, 1, 2, \ldots$.

On the other hand, $\mathrm{Geo}(p) \bmod 2^k$ is sufficiently well approximated by the uniform distribution over $\{0, \ldots, 2^k - 1\}$; the rejection method suffices for fast sampling. These ideas are brought together in Algorithm 2.

Algorithm 2 GenGeo(p) samples Geo(p).

```
1: D ← 0
2: while Ber((1 − p)^{2^k}) do
3:     D ← D + 1
4: repeat
5:     M ←(lazy)— Uni(2^k)
6: until Ber((1 − p)^M)
7: fill up M with random bits
8: return 2^k D + M
```

Here, D represents $\mathrm{Geo}(p) \operatorname{div} 2^k$, initialized to 0. It is increased by 1 as long as a Bernoulli random variate $\mathrm{Ber}((1-p)^{2^k})$ turns out to be 1. Then M, corresponding to $\mathrm{Geo}(p) \bmod 2^k$, is chosen uniformly from the interval $\{0, \ldots, 2^k - 1\}$, but rejected with probability $(1-p)^M$. We sample M lazily, i.e., a bit of M is sampled only if needed by the test $\mathrm{Ber}((1-p)^M)$. After we leave the loop, M is filled up with random bits, so that we return the same value as if we had sampled M completely inside of the second loop. The result is, naturally, $2^k D + M$.

We will next discuss correctness of this algorithm, describe the details of how to implement it efficiently, and analyze its running time. We postpone the issue of how to sample $\mathrm{Ber}((1-p)^n)$ to the end of this section. For the moment we will just assume that this can be done in expected constant time, looking at the first expected constant many bits of p and n.

Correctness Let $n \geqslant 0$. The probability of outputting $n = 2^k D + M$ should be $p(1-p)^n$, i.e., it should be proportional to $(1-p)^n$. Following the algorithm step by step we see that the probability is

$$\underbrace{\left((1-p)^{2^k}\right)^D \cdot \left(1 - (1-p)^{2^k}\right)}_{\text{first loop}} \cdot \underbrace{\sum_{t \geqslant 0} \left(1 - \sum_{i=0}^{2^k-1} 2^{-k}(1-p)^i\right)^t 2^{-k}(1-p)^M}_{\text{second loop}},$$

where t is the number of iterations of the second loop; note that $2^{-k}(1-p)^i$ is the probability of outputting i in the first iteration of the second loop, so that $\sum_{i=0}^{2^k-1} 2^{-k}(1-p)^i$ is the probability of leaving the second loop after the first iteration. Collecting the factors dependent on D and M we see that this probability is proportional to $(1-p)^{2^k D + M} = (1-p)^n$, showing correctness of the algorithm.

Implementation Details. We encode D and M as strings of words, which is the easiest way of representing large integers. This way, incrementing the counter D can be done in amortized constant time. Also, computing $2^k D + M$ can be done by shifting D by k and overwriting the last k bits of $2^k D$ (which are all 0) with M

(which is smaller than 2^k), which has the cost of reading the output once. Note that these operations can even be avoided, if we store D and M at the right positions in a string of words right away.

Moreover, we want to sample M uniformly in the interval $\{0, \dots, 2^k - 1\}$. This can be done by generating $\lceil k/w \rceil$ uniformly random words and truncating the first one to $k \bmod w$ bits, in case w does not divide k. Thus, this can be done in time $\mathcal{O}(1 + \log(1/p)/w)$. However, for our purposes it actually makes more sense to sample bits of M on demand: Sampling $\mathrm{Ber}((1-p)^M)$ requires an expected number of $\mathcal{O}(1)$ bits of M, so we can sample these bits on demand, storing already sampled bits, and filling up the rest of the bits of M after we leave the second loop. This has the advantage that we sample a large number of random bits exactly once, not an expected number of $\mathcal{O}(1)$ times, so that the running time of the algorithm is more concentrated.

Running Time We show that the expected running time of Algorithm 2 is $\mathcal{O}(1 + \log(1/p)/w)$. Again, assume that we can sample $\mathrm{Ber}((1-p)^n)$ in expected constant time. By the last section, incrementing the counter D can be done in amortized constant time, and we only need an expected constant number of bits of M during the second loop, after which we fill up M with random bits in time $\mathcal{O}(1+\log(1/p)/w)$. Hence, if we show that the two loops run in expected constant time, then Algorithm 2 runs in expected time $\mathcal{O}(1 + \log(1/p)/w)$.

We consider the probabilities of dropping out of the two loops. Since $2^{-k} \geqslant p > c2^{-k}$, for the first loop this is

$$1 - (1-p)^{2^k} \geqslant 1 - (1-p)^{c/p} \geqslant 1 - e^{-c}, \tag{4.1}$$

so we have constant probability to drop out of this loop in every iteration. Moreover, the second loop terminates immediately if $k = 0$; otherwise we have

$$(1-p)^M \geqslant (1-p)^{2^k} \geqslant (1 - 2^{-k})^{2^k} \geqslant (1 - 1/2)^2 = 1/4, \tag{4.2}$$

so for the second loop we also have constant probability of dropping out.

To show that each loop runs in expected constant time, let T be a random variable denoting the number of iterations of the loop; note that $\mathbb{E}[T] = \mathcal{O}(1)$, since the probability of dropping out of each loop is $\Omega(1)$. Furthermore, let X_i be the running time of the i-th iteration of the loop; note that by assumption we can sample $\mathrm{Ber}((1-p)^n)$ in expected constant time, so that $\mathbb{E}[X_i \mid T \geqslant i] = \mathcal{O}(1)$. The total running time of the loop is $X_1 + \ldots + X_T$. Thus, the following lemma shows that the expected running time of the loop is $\mathcal{O}(1)$. This finishes the proof of Theorem 1.14, aside from sampling $\mathrm{Ber}((1-p)^n)$.

Lemma 4.4. *Let T be a random variable with values in $\mathbb{N}_0$ and X_i, $i \in \mathbb{N}$, be random variables with values in $\mathbb{R}$; we assume no independence. Let $\alpha \in \mathbb{R}$ with $\mathbb{E}[X_i \mid T \geqslant i] \leqslant \alpha$ for all $i \in \mathbb{N}$. Then we have $\mathbb{E}[X_1 + \ldots + X_T] \leqslant \alpha \cdot \mathbb{E}[T]$.*

Proof. The variable X_i is part of the sum if and only if $T \geqslant i$. Hence, we have

$$\mathbb{E}[X_1 + \ldots + X_T] = \sum_{i \geqslant 1} \mathbb{E}[X_i \mid T \geqslant i] \cdot \Pr[T \geqslant i].$$

Using $\mathbb{E}[X_i \mid T \geqslant i] \leqslant \alpha$ and the definition of $\mathbb{E}[T]$, this yields

$$\mathbb{E}[X_1 + \ldots + X_T] \leqslant \alpha \sum_{i \geqslant 1} \Pr[T \geqslant i] = \alpha \cdot \mathbb{E}[T]. \qquad \square$$

We remark that the above lemma is an easy special case of Wald's equation.

Note that the only points where this algorithm is using the Word RAM parallelism are when we fill up M and when we compute with exponents. The generation of $\mathrm{Ber}((1-p)^n)$, discussed in the remainder of this section, will use Word RAM parallelism only for working with exponents. The filling of M can be done in time $\mathcal{O}(1 + \log(1/p)/w)$ as we assumed that we can generate random words in unit time. Also note that given P processors, each one capable of performing Word RAM operations, we can trivially further parallelize this algorithm to run in expected time $\mathcal{O}\big(1 + \frac{\log(1/p)}{wP}\big)$.

Sampling $\mathbf{Ber((1-p)^n)}$ It is left to show how to sample a Bernoulli random variable with parameter $(1-p)^n$. We can use the fact that we know k with $2^{-k} \geqslant p > 2^{-k-2}$ and can approximate $2^k p$ by p_i, and that $n \in \mathbb{N}$, $n \leqslant 2^k$. Note that we can easily get an approximation n_i of n of the form $|2^{-k}n - n_i| \leqslant 2^{-i}$ in the situation of Algorithm 2: In the first loop we have $n = 2^k$, then simply pick $n_i = 1$; in the second loop $n = M$ is uniform in $\{0, \ldots, 2^k - 1\}$, so that we get n_i by determining (i.e. flipping) the highest i bits of n. In this situation we can show the following lemma.

Lemma 4.5. *In the above situation for $n = 2^k$ or for uniformly random n in $\{0, \ldots, 2^k - 1\}$, we can sample $\mathrm{Ber}((1-p)^n)$ in expected constant time.*

Recall that in Theorem 1.11 the only thing we need to efficiently sample $\mathrm{Ber}(q)$ is to be able to compute an approximation q_i of q with $|q - q_i| \leqslant 2^{-i}$ in time $i^{\mathcal{O}(1)}$. To get such an approximation for $(1-p)^n$, we make use of the binomial theorem $(1-p)^n = \sum_{j=0}^{n} \binom{n}{j}(-p)^j$. Noting that $\binom{n}{j} \leqslant \frac{n^j}{j!}$ and $n \leqslant 1/p$, we see that the j-th summand is absolutely bounded by $1/j!$. Moreover, the absolute value of the summands is monotonically decreasing in j, and their sign is $(-1)^j$, implying

$$\left| \sum_{j=i+2}^{n} \binom{n}{j} (-p)^j \right| \leqslant 1/(i+2)! \leqslant 2^{-i-1}. \tag{4.3}$$

Thus, by summing up only the first $i+2$ summands we get a good approximation of $(1-p)^n$.

Moreover, we have

$$\sum_{j=0}^{i+1} \binom{n}{j} (-p)^j = \frac{1}{(i+1)!} \sum_{j=0}^{i+1} (-p)^j \Bigg(\prod_{h=j+1}^{i+1} h \Bigg) \prod_{h=0}^{j-1} (n-h). \tag{4.4}$$

We will compute the right-hand side of this with working precision r. This means that we work with floating point numbers, with an exact exponent encoded by a string of words, and a mantissa which is a string of $\lceil r/w \rceil$ words. We get p and n up to working precision r by plugging in $2^{-k}p_r$ and $2^k n_r$. Then we calculate the numerator and denominator of the right-hand side independently with working precision r. Note that adding or multiplying the floating point numbers takes time $\mathcal{O}(\text{poly}(r))$ for adding/multiplying the mantissas (even using the school method for multiplication is fine for this), and $\mathcal{O}(1 + \log(i))$ for subtracting/adding the exponents, as all exponents in equation (4.4) are absolutely bounded by $\mathcal{O}(\text{poly}(i) \cdot k)$ and k fits in $\mathcal{O}(1)$ words.

Regarding running time, noting that there are $\mathcal{O}(\text{poly}(i))$ operations to carry out in computing the right-hand side of equation (4.4), we see that we can compute the latter with working precision r in time $\mathcal{O}(\text{poly}(r, i))$. If we choose r large enough so that this yields an approximation of equation (4.4) with absolute error at most 2^{-i-1}, then combined with the error analysis from using only the first $i + 2$ terms (equation (4.3)), we get a running time of $\mathcal{O}(\text{poly}(r, i))$ to compute an approximation of $(1 - p)^n$ with absolute error 2^{-i}. Now, as long as we can choose $r = \text{poly}(i)$, this running time is small enough to use Theorem 1.11, since we only needed an approximation of $(1 - p)^n$ with absolute error 2^{-i} in time $i^{\mathcal{O}(1)}$ for some $\varepsilon > 0$. Under this assumption on r, we are done proving Lemma 4.5. The following lemma shows that $r = \text{poly}(i)$ is indeed sufficient.

Lemma 4.6. *The absolute error of computing equation* (4.4) *with working precision* $r = i + \alpha(1 + \log(i))$ *is at most* 2^{-i-1}*, for a large enough constant* α*.*

Proof of Lemma 4.6. Consider computing a product $c = a \cdot b$ of floating point numbers. Note that we work with precision r, so already a and b should have some relative error $1 + \varepsilon_a$ and $1 + \varepsilon_b$ relative to their correct values. Then what is the relative error of c? We compute c to be the product of a and b rounded to the next floating point number with precision r, so c has relative error at most $(1 + \varepsilon_a)(1 + \varepsilon_b)(1 + \mathcal{O}(2^{-r}))$. As long as $\varepsilon_a, \varepsilon_b \leqslant 2^{-r/2}$ this product is less than $1 + \varepsilon_a + \varepsilon_b + \mathcal{O}(2^{-r})$. Thus, computing a product like p^j with working precision r we get a relative error of $1 + \mathcal{O}(j2^{-r})$, since we plugged in p with relative error at most $\mathcal{O}(2^{-r})$. This assumes $j \leqslant 2^{r/2}$. Similarly, we get a relative error of $\mathcal{O}(i2^{-r})$ for $(i + 1)!$ and $\prod_{h=j+1}^{i+1} h$, assuming that $i + 1 < 2^r$ (so that each factor can be represented exactly).

The subtraction $n - h$ can be performed with relative error $1 + \mathcal{O}(i2^{-r})$ if $h \leqslant i + 1 < 2^r$, since in this case the floating point representation of h is actually exact, and the exponent of $n - h$ can drop from the exponent of n by at most $\log(i)$. Thus, the product $\prod_{h=0}^{j-1}(n - h)$ can be computed with relative error at most $1 + \mathcal{O}(i^2 2^{-r})$

In general, for an addition $c = a + b$ we cannot hope for a relative approximation of the resulting number c, at least not if a and b can have opposing signs. However, if e_a and e_b are the exponents of a and b, we can hope for an approximation relative to $2^{\max\{e_a, e_b\}}$. So assume that we computed a and b with relative error $1 + \varepsilon_a$, $1 + \varepsilon_b$ relative to some $2^{e'_a} \geqslant 2^{e_a}$, $2^{e'_b} \geqslant 2^{e_b}$. Then their sum will have relative error at most $1 + \varepsilon_a + \varepsilon_b + \mathcal{O}(2^{-r}))$ relative to $2^{\max\{e'_a, e'_b\}}$. Observe that this is sufficient for the sum of equation (4.4): Each summand is bounded by $(i + 1)!$, so

we get approximations relative to $2^{e'} = \mathcal{O}((i+1)!)$. Since we start with a relative approximation error of $1 + \mathcal{O}(i^2 2^{-r})$ for each summand, this adds up to an error of $1 + \mathcal{O}(i^3 2^{-r})$ for the numerator, assuming $i^3 \leqslant 2^{r/2}$. Since this error is relative to $2^{e'} = \mathcal{O}((i+1)!)$, and the denominator is $(i+1)!$, we end up with an approximation of equation (4.4) with *absolute* error at most $\mathcal{O}(i^3 2^{-r})$. Thus, to get an absolute error of at most 2^{-i-1} it suffices to set $r = i + \mathcal{O}(\log(i))$ with a large enough hidden constant. Moreover, the above construction is explicit enough to allow for computing the involved constants. □

4.5. Bounded Geometric Random Variates

Generating a bounded geometric random variate $\mathrm{Geo}(p, n)$ in the obvious way, $\min\{n, \mathrm{Geo}(p)\}$, takes expected time $\mathcal{O}(1 + \log(1/p)/w)$. We show in this section how to reduce the expected running time to $\mathcal{O}(1 + \log(\min\{1/p, n\})/w)$.

Theorem 1.16. *On a Word RAM with word size $w = \Omega(\log\log(1/p))$, a bounded geometric random variate $\mathrm{Geo}(p, n)$ with parameters $p \in (0, 1]$ and $n \in \mathbb{N}$ can be sampled in expected running time $\mathcal{O}(1 + \log(\min\{1/p, n\})/w)$, which is optimal.*

If p is a rational number with numerator and denominator fitting in $\mathcal{O}(1)$ words, then our algorithm needs $\mathcal{O}(n)$ space in the worst case.

This is again optimal, as shown by the following theorem.

Theorem 1.15. *On a Word RAM, any algorithm sampling a bounded geometric random variate $\mathrm{Geo}(p, n)$ with parameters $p \in (0, 1]$ and $n \in \mathbb{N}$ has expected running time $\Omega(1 + \log(\min\{1/p, n\})/w)$.*

Theorem 1.15 follows from Lemma 4.7 which bounds the expected output size.

Lemma 4.7. *For any $p \in (0, 1]$ and $n \in \mathbb{N}_0$, we have $\mathbb{E}[\log(1 + \mathrm{Geo}(p, n))] = \Theta(\log(\min\{1/p, n + 1\}))$, where the lower bound holds for $1/p$ and n large enough.*

Proof. We have

$$\mathbb{E}[\log(1 + \mathrm{Geo}(p, n))] = \log(n+1) \cdot (1-p)^n + \sum_{i=0}^{n-1} \log(i+1) \cdot p(1-p)^i.$$

Thus, for n large enough and $n \leqslant 1/p$, we have

$$\begin{aligned}
\mathbb{E}[\log(1 + \mathrm{Geo}(p, n))] &\geqslant \log(n+1) \cdot (1-p)^n \\
&\geqslant \log(n+1) \cdot (1 - 1/n)^n \\
&\geqslant \log(n+1) \cdot (1 - 1/2)^2 = \Omega(\log(n+1)).
\end{aligned}$$

On the other hand, for $1/p$ large enough and $1/p < n$, we have

$$\begin{aligned}\mathbb{E}[\log(1+\mathrm{Geo}(p,n))] &\geqslant \sum_{i=\lfloor 1/(2p)\rfloor}^{\lfloor 1/p\rfloor} \log(i+1)\cdot p(1-p)^i \\ &\geqslant \log(1/2p)\, p \sum_{i=\lfloor 1/(2p)\rfloor}^{\lfloor 1/p\rfloor} (1-p)^i,\end{aligned}$$

In this interval, $(1-p)^i \geqslant (1-p)^{1/p} = \Omega(1)$, and the interval is of length $\Omega(1/p)$, yielding

$$\mathbb{E}[\log(1+\mathrm{Geo}(p,n))] = \Omega(\log(1/p)).$$

For the upper bound we may use

$$\mathbb{E}[\log(1+\mathrm{Geo}(p,n))] \leqslant \log(1+\mathbb{E}[\mathrm{Geo}(p,n)]).$$

As $\mathrm{Geo}(p,n) = \min\{n, \mathrm{Geo}(p)\}$, we have

$$\mathbb{E}[\mathrm{Geo}(p,n)] \leqslant \min\{n, \mathbb{E}[\mathrm{Geo}(p)]\} = \min\{n, 1/p-1\},$$

which finishes the proof. □

Again we assume that we can compute arbitrary precision floating point approximations of the parameter p, as discussed in Section 4.1. In particular, we know an exponent k with $2^{-k} \geqslant p > 2^{-k-2}$. Moreover, we assume that we are given n as a string of words together with an exponent m with $2^m \geqslant n = \Omega(2^m)$.

Algorithm 3 BoundedGeo(p,n) samples $\mathrm{Geo}(p,n)$.

```
1: D ← 0
2: while Ber((1−p)^{2^k}) do
3:     D ← D + 1
4:     if 2^k D ⩾ 2^m then return n
5: repeat
6:     M ←(lazy)— Uni(2^k)
7: until Ber((1−p)^M)
8: sample remaining bits of M (highest to lowest) until one of the bits is 1,
9: D and the current bits of M form a 2-approximation X of 2^k D + M
10: if X ⩾ 2^m then return n
11: fill up M with random bits
12: return min{n, 2^k D + M}
```

We now slightly change Algorithm 2 and obtain Algorithm 3. Again, we sample $D = \mathrm{Geo}(p) \operatorname{div} 2^k$ and $M = \mathrm{Geo}(p) \bmod 2^k$, the latter lazily as in Algorithm 2. However, now we ensure that whenever we know that the output will be larger than n, we exit and return n. To this end, we check after each incrementation of D whether $2^k D \geqslant 2^m$ (note that this is a cheaper test than $2^k D \geqslant n$), returning

n if true. Moreover, after having sampled D and exiting the second loop (so that we have sampled some initial bits of M already), we sample more bits of M from highest to lowest until we find a 1-bit, or until M is completely sampled. At this point, we have a 2-approximation X of $2^k D + M$, since for a number $N \in \mathbb{N}$ with highest bit j we have $2^j \leqslant N < 2^{j+1}$. Since $2^k D + M$ is at least X, we can return n if $X \geqslant 2^m$.

Correctness of this algorithm is clear, since Algorithm 2 was correct and we return n only if we are sure that the output will be at least n.

Implementation Details It is easy to argue that the tests $2^k D \geqslant 2^m$ and $X \geqslant 2^{m+1}$ are implementable in constant time, again assuming that the exponents k and m fit in $\mathcal{O}(1)$ words: We only have to know the highest bit j of the left hand side. Then we have $2^k D \geqslant 2^m$ if and only if $2^j \geqslant 2^m$, which is trivial to test. However, the highest bit of the counter D is easy to remember, and we can remember the highest bit of X during its construction.

We can also argue that computing $\min\{n, Y\}$, with $Y = 2^k D + M$, in the last step of the algorithm works in time $\mathcal{O}(1 + \log(Y)/w)$: First, we scan Y to determine its exponent. If this is sufficiently smaller than m, we know that $Y \leqslant n$ and return Y. Otherwise, n is not much larger than Y, so we can afford to compare them and return the minimum. Observe that at this point of the algorithm Y cannot be much larger than n, otherwise we would have aborted and returned n earlier. In fact, we have $Y = \mathcal{O}(\min\{n, Y\})$. Thus, the running time of this step is $\mathcal{O}(1 + \log(1 + Y)/w) = \mathcal{O}(1 + \log(1 + \min\{n, Y\})/w)$, where $\min\{n, Y\}$ is the output. Thus, the second to last line takes asymptotically only as much time as the return statement in the last line.

Running Time Consider Algorithm 3 without the second to last line and without the return statements. Observe that the remaining parts have an expected running time of $\mathcal{O}(1)$, as shown by the analysis of Algorithm 2: The tests $2^k D \geqslant 2^m$ and $X \geqslant 2^m$ are one difference to Algorithm 2, but we argued that they can be implemented in $\mathcal{O}(1)$ worst-case time. Another difference is the sampling of further bits of M until we see the first 1-bit, but this clearly takes expected time $\mathcal{O}(1)$.

We also argued in the last section that the second-to-last line and the computation of $\min\{n, 2^k D + M\}$ can be done in the same asymptotic running time as returning the result. Thus, the lines we did not consider so far take time $\mathcal{O}(1 + \log(1 + Z)/w)$, where Z is the returned value in this case. The expected value of any of the return statements is thus bounded by $\mathcal{O}(1 + \mathbb{E}[\log(1 + \mathrm{Geo}(p, n))]/w)$, which is itself bounded by $\mathcal{O}(1 + \log(\min\{n, 1/p\})/w)$ by Lemma 4.7.

Space Usage Since $\mathrm{Geo}(p, n)$ is bounded, we could hope for a sampling algorithm with bounded space usage, in contrast to $\mathrm{Geo}(p)$. In this section we show that Algorithm 3 can be adapted to need $\mathcal{O}(n)$ words in the worst case. This is, of course, possibly exponentially large in the input size, but it will help for generating $\mathcal{G}(n, p)$'s.

Again, we delay the discussion of the tests $\mathrm{Ber}((1-p)^n)$. Observe that apart from this, Algorithm 3 can be made to use $\mathcal{O}(1 + \log(n)/w)$ words in the worst case: We

have $D \leqslant 2^m = \mathcal{O}(n)$, otherwise we exit. Moreover, if we throw away initial 0 bits of $2^k D + M$ in the process of computing X, we need only $\mathcal{O}(1)$ words in the worst case for storing D and the initial bits of M when we arrive at X. At this point, either we exit, or $X < 2^m$, so that we have $2^k D + M \leqslant 2X \leqslant 2^{m+2} = \mathcal{O}(n)$, implying that the remainder of the algorithm uses, again, only $\mathcal{O}(1+\log(n)/w)$ words in the worst case. This proves that Algorithm 3 has a space usage of $\mathcal{O}(1 + \log(n)/w)$ words in the worst case, apart from the tests $\mathrm{Ber}((1-p)^n)$.

We cannot hope for the test $\mathrm{Ber}((1-p)^n)$ to run in bounded space if p is an arbitrary real. However, we can achieve this if p is a rational with given numerator a and denominator b fitting into a constant number of words:

Lemma 4.8. *Let p be a given rational number with numerator and denominator fitting in $\mathcal{O}(1)$ words, and let $2^{-k} \geqslant p \geqslant 2^{-k-2}$. For $n = 2^k$ or for uniformly random n in $\{0, \ldots, 2^k - 1\}$, we can sample $\mathrm{Ber}((1-p)^n)$ using $\mathcal{O}(n)$ words in the worst case. The algorithm runs in expected constant time using an expected constant number of words.*

Proof. To obtain a worst-case $\mathcal{O}(n)$ space bound, we can afford to compute and store the numerator and denominator of $(1-p)^n = (b-a)^n/b^n$ explicitly in $\mathcal{O}(n)$ words (and $\mathcal{O}(\mathrm{poly}(n))$ time); call them A and B. We now show that $\mathrm{Ber}(A/B)$ can be sampled using $\mathcal{O}(n)$ space in worst-case and expected time $\mathcal{O}(\mathrm{poly}(n))$. Later we discuss how to reduce the expected time and space.

In order to sample $\mathrm{Ber}(A/B)$, we can take a random real number $r \in [0,1)$ and return 1, if $r \leqslant A/B$ and 0 otherwise, as usual. Such an r can be written as $\sum_{i \geqslant 1} b_i 2^{-i}$ with $b_i \in \{0,1\}$ uniformly at random and independent. Let $s_k := \sum_{i>k} b_i 2^{-i+k}$; s_k is again a uniform random number in $[0,1)$. Then we have $r = s_0$ and $s_k = \frac{1}{2}(b_{k+1} + s_{k+1})$. Moreover, $s_k \leqslant A'/B$, if and only if

$$s_{k+1} \leqslant \frac{2A' - b_{k+1}B}{B}.$$

Thus, we get a recursion for numerators A_k such that $r \leqslant A/B$ if and only if $s_0 \leqslant A_0/B$ if and only if $s_k \leqslant A_k/B$. The recursion is $A_0 = A$ and $A_k = 2A_{k-1} - b_k B$ for $k \geqslant 1$. Note that we are done with this test as soon as $A_k/B < 0$, since then $s_k \geqslant 0 > A_k/B$, or $A_k/B \geqslant 1$, since then $s_k \leqslant 1 \leqslant A_k/B$. Also note that before we are done, we always have $0 \leqslant A_k \leqslant B$, so we can store A_k/B using $\mathcal{O}(n)$ words. Thus, this algorithm uses $\mathcal{O}(n)$ words in the worst case.

We show that this algorithm aborts after an expected number of $\mathcal{O}(1)$ incrementations of k. Observe that we can solve the recursion to

$$A_k/B = 2^k A/B - 2^k r_k,$$

for any $k \geqslant 0$, where $r_k = \sum_{i=1}^{k} b_i 2^{-i}$ is an approximation of r by its first k bits. Hence, we have $A_k/B \in [0,1]$ if and only if $A/B - r_k \in [0, 2^{-k}]$, which can only hold if $A/B - r \in [-2^{-k}, 2^{-k}]$. Since A/B is fixed, this happens with probability at most

2^{1-k}. Thus, the expected running time of the algorithm is bounded by

$$\sum_{k \geqslant 1} \mathcal{O}(\text{poly}(n, k)) \cdot 2^{1-k} = \mathcal{O}(\text{poly}(n)).$$

The above algorithm can be combined with the algorithm from Lemma 4.5: As long as it needs $\mathcal{O}(n)$ storage we run the latter algorithm, but if we reach the point where it would need more than n words, we switch to the new algorithm. This way we have constant expected running time and space, while using $\mathcal{O}(n)$ words in the worst case. □

4.6. Generating Random Graphs

In this section we show that Erdős-Rényi and Chung-Lu random graphs can be efficiently generated. For this we simply take the efficient generation on the Real RAM from [71, 72] and replace the generation of bounded geometric variables by our algorithm from the last section. This yields the following results from Section 1.7.

Theorem 1.17. *On a Word RAM with word size $w = \Omega(\log n)$, the Erdős-Rényi random graph $\mathcal{G}(n, p)$ can be sampled in expected time $\mathcal{O}(n+m)$, where $m = \Theta(pn^2)$ is the expected number of edges. This is optimal if $w = \mathcal{O}(\log n)$. If p is a rational number with numerator and denominator fitting into $\mathcal{O}(1)$ words, then the worst-case space complexity of our algorithm is asymptotically equal to the size of the output graph, which is optimal.*

Theorem 1.18. *Let $W = (W_1, \ldots, W_n)$ be rationals with common denominator, where all numerators and the common denominator fit into $\mathcal{O}(1)$ words. Then on a Word RAM with word size $w = \Omega(\log n)$, the Chung-Lu random graph $\mathcal{G}(n, W)$ can be sampled in expected time $\Theta(n+m)$, where m is the expected number of edges. This is optimal if $w = \mathcal{O}(\log n)$. The worst-case space complexity of the algorithm is asymptotically equal to the size of the output graph, which is optimal.*

Consider the original efficient generation algorithm of Erdős-Rényi random graphs described in [71], which is essentially the following. For each vertex $u \in [n]$ we want to sample its neighbors $v \in [u-1]$ in decreasing order. Defining $v_0 := u$, the first neighbor v_1 of u is distributed as $v_1 \sim v_0 - 1 - \text{Geo}(p, v_0 - 1)$, where the event $v_1 = 0$ represents that u has no neighbor. Then the next neighbor is distributed as $v_2 \sim v_1 - 1 - \text{Geo}(p, v_1 - 1)$ and so on. Sampling the graph in this way, we use $m + n$ bounded geometric variables, where m is the number of edges in the final graph (which is a random variable).

In this algorithm we have to cope with indices of vertices, thus, it is natural to assume $w = \Omega(\log n)$. Under this assumption, all single operations of the original algorithm can be performed in worst-case constant time on a Word RAM, except for the generation of bounded geometric variables $\text{Geo}(p, k)$, with $k \leqslant n$. The latter, however, can be done in expected time $\mathcal{O}(1 + \log(\min\{n, 1/p\})/w) = \mathcal{O}(1)$ using our algorithm from Theorem 1.16. Hence, the expected running time of the modified

algorithm (with replaced sampling of bounded geometric variables) should be the same as that of the original algorithm. To prove this, consider the running time the modified algorithm spends on sampling bounded geometric variables. This random variable can be written as $X_1 + \ldots + X_T$, where T is a random variable denoting the number of bounded geometric variables sampled by the algorithm, and X_i is the time spent on sampling the i-th such variable. Note that $\mathbb{E}[X_i \mid T \geqslant i] = \mathcal{O}(1)$. Thus, by Lemma 4.4 we can bound $\mathbb{E}[X_1 + \ldots + X_T]$ by $\mathcal{O}(\mathbb{E}[T])$. Since the original algorithm spends time $\Omega(T)$, the total expected running time of the modified algorithm is asymptotically the same as the expected running time of the original algorithm, $\mathcal{O}(n + pn^2)$. This running time is optimal, as writing down a graph takes time $\Omega(n+m)$ (each index needs $\Theta(1)$ words; this depends, however, on the representation of the graph). Noting that the space requirements of the algorithm are met by Theorem 1.16, this proves Theorem 1.17.

A similar result applies to Chung-Lu random graphs $\mathcal{G}(n, W)$. Again we assume $w = \Omega(\log n)$. Let us further assume, for simplicity, that all given weights W_u, $u \in V$ are rational numbers with the same denominator, with each numerator and the common denominator fitting in $\mathcal{O}(1)$ words. In this case, the sum $S = \sum_{u \in V} W_u$ has the same denominator as all W_u and numerator bounded by n times the numerator of the largest W_u. Since $w = \Omega(\log n)$, the numerator of S fits in $\mathcal{O}(1)$ more words than used for the largest W_u. Hence, numerator and denominator of S fit in $\mathcal{O}(1)$ words and can be computed in $\mathcal{O}(n)$ time. Moreover, the edge probabilities $p_{u,v} = \min\{W_u W_v / S, 1\}$ are also rationals with numerator and denominator fitting in $\mathcal{O}(1)$ words that can be computed in constant time if S is available.

Carefully examining the efficient sampling algorithm for Chung-Lu random graphs, Algorithm 2 of Miller and Hagberg [72], we see that now every step can be performed in the same deterministic time bound as on a Real RAM, except for the generation of bounded geometric variables and Bernoulli variables. Note that for any $p \in (0, 1)$ we have $\mathrm{Ber}(p) \sim \mathrm{Geo}(1-p, 1)$, so Theorem 1.16 shows that the bounded geometric as well as the Bernoulli random variables can be sampled in expected constant time and bounded space (for $w = \Omega(\log n)$). Thus, we can bound the expected running time of the modified generation for Chung-Lu graphs analogously to the Erdős-Rényi case, proving Theorem 1.18.

4.7. Binomial Random Variates

In this section we consider binomial random variates, more precisely, the number of heads in a sequence of n coin flips.

Theorem 1.19. *On the Word RAM with $w = \Omega(\log n)$, a binomial random variable $\mathrm{Bin}(n, \frac{1}{2})$ can be sampled in expected time $\mathcal{O}(1)$. Moreover, it can be sampled in a running time that is larger than $t > 0$ with probability $\exp(-t^{\Omega(1)})$.*

We may assume that n is even, otherwise we split into $\mathrm{Bin}(n-1, \frac{1}{2}) + \mathrm{Bin}(1, \frac{1}{2})$ and sample the latter in constant time. Thus, from now on we want to sample $\mathrm{Bin}(2n, \frac{1}{2})$

for some $n \in \mathbb{N}$. We may also assume $n \geqslant 2$, otherwise we sample $\mathrm{Bin}(2n, \frac{1}{2})$ as a sum of $2n$ random bits in constant time.

Upper bound for binomial coefficients. Let $m \in \mathbb{N}$ be an approximation of $\sqrt{2n}$ with $m \in [\sqrt{2n}, \sqrt{2n} + 3]$. We obtain such an approximation by using any (rough) approximation of the function $\sqrt{x}$, yielding $m' \in [\sqrt{2n} - 1, \sqrt{2n} + 1]$, and considering $m := \lceil m' \rceil + 1$.

Lemma 4.9. *In the above situation we have for any $k \in \mathbb{Z}$*

$$\binom{2n}{n + k \cdot m} \leqslant 2^{2n} \frac{4}{2^{|k|} m}.$$

Proof. Since $\binom{2n}{n+i} = \binom{2n}{n-i}$ we may assume $k \geqslant 0$. For $0 \leqslant i \leqslant n$, standard calculations yield

$$\begin{aligned}\binom{2n}{n+i} \Big/ \binom{2n}{n} &= \prod_{j=1}^{i} \frac{n+1-j}{n+j} \leqslant \prod_{j=1}^{i} \left(1 - \frac{j}{n+1}\right) \\ &\leqslant \exp\left(-\sum_{j=1}^{i} \frac{j}{n+1}\right) = \exp\left(-\tfrac{i(i+1)}{2(n+1)}\right) \leqslant \exp\left(-\tfrac{i^2}{2n}\right).\end{aligned}$$

A well-known bound following from Stirling's approximation is

$$\binom{2n}{n} \leqslant \frac{2^{2n}}{\sqrt{\pi n}},$$

so that we get

$$\binom{2n}{n+i} \leqslant \frac{2^{2n}}{\sqrt{\pi n}} \exp\left(-\tfrac{i^2}{2n}\right).$$

For $i = k \cdot m$ and $m \geqslant \sqrt{2n}$ we have

$$\exp\left(-\tfrac{i^2}{2n}\right) \leqslant \exp(-k^2) \leqslant 2^{1-k}.$$

Finally, for $m \leqslant \sqrt{2n} + 3$ and $n \geqslant 2$ we have $\sqrt{\pi n} \geqslant m/2$, which yields the claim. □

Partition $\mathbb{Z}$ into buckets of m consecutive numbers, $B_k := \{km, km+1, \ldots, km+m-1\}$ for $k \in \mathbb{Z}$. Let

$$f(i) := \frac{4}{2^{\max\{k,-k-1\}} m}$$

for any $i \in B_k$, $k \in \mathbb{Z}$. Moreover, set

$$p(i) := \binom{2n}{n+i} / 2^{2n},$$

with $p(i) := 0$ for $|i| > n$. Note that we have $\Pr[\mathrm{Bin}(2n, \frac{1}{2}) = n + i] = p(i)$.

Lemma 4.10. *We have*

1. $f(i) \geqslant p(i)$ *for all* $i \in \mathbb{Z}$ *and*
2. $\sum_{i\in\mathbb{Z}} f(i) = 16$.

Proof. The first statement follows from Lemma 4.9 and monotonicity of binomial coefficients: For $i \in B_k$ with $k \geqslant 0$ we have $i \geqslant km$ so that $\binom{2n}{n+i} \leqslant \binom{2n}{n+km}$, which yields $p(i) \leqslant \binom{2n}{n+km}/2^{2n}$. Together with Lemma 4.9 this proves $p(i) \leqslant f(i)$. For $k < 0$ we argue similarly using $i \leqslant (k+1)m \leqslant 0$.

For the second statement we use symmetry of f around 0 and $|B_k| = m$ to obtain

$$\sum_{i<0} f(i) = \sum_{i\geqslant 0} f(i) = \sum_{k\geqslant 0} |B_k| \frac{4}{2^k m} = 4 \sum_{k\geqslant 0} 2^{-k} = 8.$$

□

Note that the above Lemma implies that $\overline{f}(i) := f(i)/16$ gives a probability distribution.

Rejection sampling. We use rejection sampling with the function f to sample from $\mathrm{Bin}(2n, \frac{1}{2})$ as follows. We first sample $i \in \mathbb{Z}$ with probability distribution $\overline{f}$ (where $\overline{f}(i) = f(i)/16$). Then we sample a Bernoulli random variate $\mathrm{Ber}(p(i)/f(i))$ (which is 1 with probability $p(i)/f(i)$). If it turns out 1 then we return $n + i$ and are done. Otherwise we reject i, i.e., we throw away i and repeat the whole process.

1. Sample $i \in \mathbb{Z}$ with probability distribution $\overline{f}$.
2. With probability $p(i)/f(i)$: Return $n + i$.
3. Otherwise: Reject i and goto 1.

Let us first argue about correctness and running time of this algorithm and then fill in the details of how to implement steps 1 and 2. Note that the probability of returning $n+i$ in a particular iteration of this algorithm is $\overline{f}(i) \cdot \frac{p(i)}{f(i)} = p(i)/16$. Thus, the probability of returning $n + i$ is proportional to $p(i) = \Pr[\mathrm{Bin}(2n, \frac{1}{2}) = n + i]$, so that we have an exact sampling algorithm. Moreover, since $p(i)$ is a probability distribution we have $\sum_{i\in\mathbb{Z}} p(i)/16 = 1/16 = \Omega(1)$. Hence, in every iteration the stopping probability of the algorithm is constant, so that the expected number of iterations of this algorithm is constant, and the number of iterations has an exponential tail.

To implement step 1 of this algorithm, we first flip a random bit whether $i \geqslant 0$ or $i < 0$ (making use of the fact $\sum_{i<0} f(i) = \sum_{i\geqslant 0} f(i)$). Without loss of generality let $i \geqslant 0$ so that $f(i) = \frac{4}{2^k m}$ for $i \in B_k$, $k \geqslant 0$. The block B_k containing i is distributed geometrically, so we can draw random bits $X_1, X_2, \ldots$ and let $k \geqslant 0$ maximal with $X_1 = \ldots = X_k = 1$. Finally, we pick i uniformly at random in block B_k. This runs in expected constant time (with exponential tail).

To implement step 2, we have to sample $\mathrm{Ber}(p(i)/f(i))$ in expected constant time. In general, to sample a Bernoulli random variate $\mathrm{Ber}(p)$ it suffices to be able to

compute an additive 2^{-L}-approximation of p in time $L^{\mathcal{O}(1)}$ by Theorem 1.11. The total running time of this is larger than t with probability $\exp(-t^{\Omega(1)})$, as promised.

Note that since $0 \leqslant p(i)/f(i) \leqslant 1$ it suffices to compute a *relative* $2^{-\Omega(L)}$-approximation of $p(i)/f(i)$ in time $L^{\mathcal{O}(1)}$.

Recall from Section 1.1.2 that on the Word RAM we can use floating-point approximations of reals by storing both mantissa and exponent as long integers. This allows one to perform typical operations on two floating-point numbers with L-bit mantissas and E-bit exponents in time $\mathcal{O}(1 + ((L+E)/w)^{\mathcal{O}(1)})$. For the numbers that we will consider, the exponents are $\mathcal{O}(L + \log n)$-bits numbers. If $L \leqslant \log(n)$ the usual Word RAM assumption of $w \geqslant \Omega(\log n)$ implies that we can compute with these exponents in constant time. In any case, the running time for handling the exponents is bounded by $L^{\mathcal{O}(1)}$. For this reason, we will only discuss the mantissa in the following.

Observe that once we have a floating-point approximation of $p(i)$ with precision $2^{-\Theta(L)}$, we easily obtain a floating point approximation of $p(i)/f(i)$ with precision $2^{-\Theta(L)}$, since $f(i)$ is build of elementary functions. Moreover, to approximate $p(i) = \binom{2n}{n+i}/2^{2n}$ it suffices to be able to approximate factorials.

Hence, we are left with the following problem. Given n and L, compute a floating-point approximation of $n!$ with precision 2^{-L} (i.e., with relative error 2^{-L}). Note that the standard Stirling's approximation only allows one to approximate $n!$ with fixed precision (depending on n). Classic formulas that yield arbitrary precision approximations of $n!$ are Lanczos approximation [94] and Spouge's approximation [95]. A fixed precision version of the former is, for instance, implemented in the GNU Scientific Library[1]. These classic formulas allow one to approximate $n!$ up to precision 2^{-L} in time $L^{\mathcal{O}(1)}$. The only possible obstacle for us using these approximations is that they are typically analyzed on the Real RAM model of computation, where we can compute with real numbers in constant time and do not have to worry about floating-point approximations. In the following we go through Spouge's approximation to check that it indeed works on the Word RAM.

Spouge's approximation [95] states that for any $n, L \in \mathbb{N}$, $L > 2$,

$$n! \approx (n+L)^{n+1/2} \mathrm{e}^{-(n+L)} \left[c_0 + \sum_{k=1}^{L-1} \frac{c_k}{n+k} \right],$$

where

$$c_0 = \sqrt{2\pi}, \quad c_k = \frac{(-1)^{k-1}}{(k-1)!} (L-k)^{k-1/2} \mathrm{e}^{L-k},$$

with a relative error that is bounded by

$$L^{-1/2} (2\pi)^{-(L+1/2)} \leqslant 2^{-L-1}.$$

To evaluate this formula, it suffices to be able to compute (floating-point approximations of) π and the functions e^x and $\sqrt{x}$. The standard algorithms for this also work on the Word RAM.

[1] http://www.gnu.org/software/gsl/

It remains to show that it suffices to compute all terms of the formula with precision $2^{-\Theta(L)}$ in order to obtain $n!$ with precision $2^{-L} = 2^{-L-1} + 2^{-L-1}$ (the error from Spouge's approximation plus the error of floating-point approximation). Note that this requires an argument, since the terms $c_k/(n+k)$ change in sign and could cancel, making very high precision necessary. However, it is not hard to bound

$$\left|\frac{c_k}{n+k}\right| \leqslant 2^{\ell_1}$$

with $\ell_1 = \mathcal{O}(L)$. Moreover, using Spouge's approximation guarantee and the easy approximation $n! = \Theta((n/\mathrm{e})^{n+1/2})$ we obtain

$$c_0 + \sum_{k=1}^{L-1} \frac{c_k}{n+k} \geqslant (1 - 2^{-L-1}) \frac{n!}{(n+L)^{n+1/2}\mathrm{e}^{-(n+L)}} \geqslant 2^{-\ell_2},$$

with $\ell_2 = \mathcal{O}(L)$. Together, these inequalities show that $\ell_1 + \ell_2 + C \cdot L = \mathcal{O}(L)$ bits of precision for a summand $\frac{c_k}{n+k}$ (or c_0) yield a precision of $C \cdot L$ bits *relative to the sum* $c_0 + \sum_{k=1}^{L-1} \frac{c_k}{n+k}$. For C a sufficiently large constant, this precision suffices for the sum to have a precision of $L+1$ bits so that we compute a relative 2^{-L-1}-approximation of $n!$, as desired. Since there are $L^{\mathcal{O}(1)}$ terms in Spouge's approximation, we obtain a total running time of $L^{\mathcal{O}(1)}$ to compute $n!$ with precision 2^{-L}. This finishes the proof of Theorem 1.19.

5 Application: Sampling a Model from Physics

This chapter is based on [5]. Improved algorithms with runtime $o(n^2)$ were independently obtained by myself and the remaining authors. The final algorithm in the paper improves upon both of our independent results. The work of writing the paper was equally shared by Ueli Peter, Henning Thomas, and myself.

[5] K. Bringmann, F. Kuhn, K. Panagiotou, U. Peter, and H. Thomas. "Internal DLA: Efficient Simulation of a Physical Growth Model." In: *Proc. 41th International Colloquium on Automata, Languages, and Programming (ICALP'14)*. Vol. 8572. LNCS. 2014, 247–258.

In this chapter, we consider the internal diffusion limited aggregation (IDLA) model from physics and present an improved algorithm to sample from this model. Recall that IDLA is a random process defined as follows. IDLA places n particles on the two-dimensional integer grid $\mathbb{Z}^2$. Let $A(i) \subset \mathbb{Z}^2$ denote the set of occupied grid points after placing i particles. The first particle is placed on the origin, i.e., $A(1) = \{(0,0)\}$. From there on, $A(i+1)$ is constructed from $A(i)$ by adding the first grid point in $\mathbb{Z}^2 \setminus A(i)$ that is reached by a random walk on $\mathbb{Z}^2$ starting at the origin. Recall that we want to prove the following theorem.

Theorem 1.20. *IDLA can be simulated in $\mathcal{O}(n \log^2 n)$ time and $\mathcal{O}(n^{1/2} \log n)$ space, both in expectation and with high probability.*

As sketched in the introduction, see Section 1.8, we want to combine many steps of the random walk to a single jump that we can perform quickly. We present a general framework that utilizes jumps to efficiently simulate IDLA in Section 5.2. Different jump procedures are discussed in Section 5.3, our best jump procedure follows in Section 5.4. We close with a discussion of generalizations in Section 5.5.

5.1. Preliminaries

Notation For $z = (x, y) \in \mathbb{Z}^2$ we let $|z| = \sqrt{x^2 + y^2}$ be its 2-norm. For $z \in \mathbb{Z}^2$ and $r > 0$ we define the ball with radius r around z as $B_z(r) := \{w \in \mathbb{Z}^2 \mid |z - w| \leqslant r\}$.

We write $\Gamma(z)$ for the set of grid neighbors of $z \in \mathbb{Z}^2$, and for an arbitrary set $S \subseteq \mathbb{Z}^2$ we write ∂S for the set of all position that can be reached from S, i.e.

$$\partial S := \left\{ z \in \mathbb{Z}^2 \setminus S \mid \Gamma(z) \cap S \neq \emptyset \right\} \quad \text{and} \quad \overline{S} := S \cup \partial S.$$

Whenever it is clear from the context, which particle we are simulating, we will write A for $A(i)$. For an IDLA shape A let $r_I = r_I(A)$ and $r_O = r_O(A)$ be its in- and outradius (rounded for technical reasons), i.e.,

$$r_I := \left\lfloor \min_{x \in \mathbb{Z}^2 \setminus A} |x| \right\rfloor \quad \text{and} \quad r_O := \left\lfloor \max_{x \in A} |x| \right\rfloor + 1.$$

Moreover, we say that $B_0(r_I)$ is the *in-* and $B_0(r_O)$ the *out-circle* of A.

The Shape of IDLA Recently, Jerison, Levine and Sheffield proved a long open conjecture which stated that $A(n) = B_0(\sqrt{n/\pi}) \pm \mathcal{O}(\log n)$ with high probability.

Theorem 5.1 (Theorem 1 in [83])**.** *For every $\gamma > 0$ exists a constant $\alpha = \alpha(\gamma) < \infty$ such that for sufficiently large r*

$$\Pr\left[B_0(r - \alpha \log r) \subset A(\lfloor \pi r^2 \rfloor) \subset B_0(r + \alpha \log r) \right] \geqslant 1 - r^{-\gamma}. \tag{5.1}$$

Additionally using $r_O \leqslant n$, this theorem implies that $r_O - r_I = \mathcal{O}(\log n)$, both in expectation and with high probability.

Drift Analysis Let Ω be some state space, $Y_k \in \Omega$ ($k \in \mathbb{N}$) a stochastic process and $g : \Omega \to \mathbb{R}_{\geqslant 0}$ a function on Y_k. Let the hitting time τ be the smallest k such that $g(Y_k) = 0$. We say that $g(Y_k)$ has an additive drift of at least ε if for all $0 \leqslant k < \tau$

$$\mathbb{E}\left[g(Y_{k+1}) - g(Y_k) \mid Y_k \right] < -\varepsilon. \tag{5.2}$$

The following theorem bounds the expected hitting time by the inverse of the additive drift.

Theorem 5.2 ([96])**.** *In the situation of this paragraph we have $\mathbb{E}[\tau] \leqslant \frac{g(Y_0)}{\varepsilon}$.*

5.1.1. Random Walks on $\mathbb{Z}$ and $\mathbb{Z}^2$

Let $z = z_0, z_1, z_2, \ldots$ be a random walk starting in $z \in \mathbb{Z}^2$. Here, we always consider the standard random walk on $\mathbb{Z}^2$ that chooses each adjacent grid point with probability $1/4$. We write $\mathrm{RW}_T(z) = z_T$ for the outcome of a random walk of length T starting in z and abbreviate $\mathrm{RW}_T(0) = \mathrm{RW}_T$. Note that $\mathrm{RW}_T(z) \sim z + \mathrm{RW}_T$.

We also reach each adjacent grid point with probability $1/4$ by flipping two coins $c_1, c_2 \in \{1, -1\}$ and choosing the next position to be

$$z + c_1 \cdot (1/2, 1/2) + c_2 \cdot (-1/2, 1/2).$$

This yields the following reformulation of a 2-dimensional random walk as a linear combination of two independent 1-dimensional random walks.

Lemma 5.3. *Let $z \in \mathbb{Z}^2$ and $T \in \mathbb{N}$. Let S_T be the sum of T independent uniform $\{1, -1\}$ random variables, $S_T \sim 2\,\mathrm{Bin}(T, 1/2) - T$, and let X, Y be independent copies of S_T. Then*

$$\mathrm{RW}_T(z) \sim z + X \cdot (1/2, 1/2) + Y \cdot (1/2, -1/2).$$

In particular, the above lemma allows us to quickly sample $\mathrm{RW}_T(z)$, even on a Word RAM, since we presented an algorithm for sampling binomial random variables in Section 4.7 (confer Theorem 1.19).

Lemma 5.4. *For any $T \leqslant n^{\mathcal{O}(1)}$ and $z \in \mathbb{Z}^2$, we can sample $\mathrm{RW}_T(z)$ in expected time $\mathcal{O}(1)$. Moreover, our algorithm has a running time that is larger than $t > 0$ with probability $\exp(-t^{\Omega(1)})$.*

We remark that Lemma 5.4 implies that m samples from $\mathrm{Bin}(T, \frac{1}{2})$ take total time $\mathcal{O}(m)$, both in expectation and with high probability in m (we will show the latter in Lemma 5.12).

Note that our random walks are "bipartite" in the sense that in even time steps one can reach only the "even" positions of the grid $\{(x, y) \in \mathbb{Z}^2 \mid x + y \equiv 0 \,(\mathrm{mod}\, 2)\}$, and similarly for odd time steps. We write $z \equiv_T x$ if z can be reached from x by a walk of length $\ell \in \mathbb{N}$ with $\ell \equiv T \,(\mathrm{mod}\, 2)$.

In the remainder of this section, we present several easy or known facts about random walks that are used throughout this chapter. For some of these facts we need explicit constants in the error terms. Thus, although variants incorporating $\mathcal{O}$-notation could be found, e.g, in [97], we need to reprove them with explicit constants.

First, we need bounds for the probability of RW_T to end up in a particular point $z \in \mathbb{Z}^2$. A proof of this statement amounts to Stirling's approximation on binomial coefficients with explicitly bounding the error term constants.

Lemma 5.5. *Let $T \in \mathbb{N}$ and $z \in \mathbb{Z}^2$ with $|z| \leqslant \frac{1}{2}T$ and $z \equiv_T (0, 0)$. Then we have*

$$\Pr[\mathrm{RW}_T = z] \leqslant \frac{2}{T} \exp\Big(- \frac{|z|^2}{2T}\Big(1 - \frac{2|z|}{T}\Big)\Big),$$
$$\Pr[\mathrm{RW}_T = z] \geqslant \frac{1}{3T} \exp\Big(- \frac{|z|^2}{2T}\Big(1 + \frac{|z|}{T}\Big)\Big).$$

Proof. Let $z = x \cdot (1/2, 1/2) + y \cdot (1/2, -1/2)$. Without loss of generality we can assume that $x, y \geqslant 0$. With the notation of Lemma 5.3 we have

$$\Pr[\mathrm{RW}_T = z] = \Pr[X = x] \cdot \Pr[Y = y].$$

Counting the number of paths on $\mathbb{Z}$ from 0 to x (or y) yields

$$\Pr[\mathrm{RW}_T = z] = 2^{-T} \binom{T}{\frac{T+x}{2}} \cdot 2^{-T} \binom{T}{\frac{T+y}{2}}. \tag{5.3}$$

Using Stirling's approximation

$$n! = r_1 \sqrt{2\pi n} \Big(\frac{n}{\mathrm{e}}\Big)^n \quad \text{with } 1 \leqslant r_1 \leqslant 1.1$$

yields after some simplifications

$$2^{-T}\binom{T}{\frac{T+x}{2}} = r_2\sqrt{\frac{2T}{\pi(T^2-x^2)}}\Big(1-\frac{x^2}{T^2}\Big)^{-T/2}\Big(1+\frac{x}{T}\Big)^{-x/2}\Big(1-\frac{x}{T}\Big)^{x/2},$$

with $(1.1)^{-2} \leqslant r_2 \leqslant 1.1$. Note that by assumption $x, y \leqslant |z| \leqslant T/2$. This allows us to bound $T^2 \geqslant T^2 - x^2 \geqslant \frac{3}{4}T^2$. Furthermore, for $a \geqslant -\frac{1}{2}$ we have $\exp(a(1-a)) \leqslant 1 + a \leqslant \exp(a)$. Together we get

$$\begin{aligned} 2^{-T}\binom{T}{\frac{T+x}{2}} &\leqslant 1.1\sqrt{\frac{8}{3\pi T}}\exp\left(\frac{x^2}{2T}\Big(1+\frac{x^2}{T^2}\Big) - \frac{x^2}{2T}\Big(1-\frac{x}{T}\Big) - \frac{x^2}{2T}\right) \\ &\leqslant 1.1\sqrt{\frac{8}{3\pi T}}\exp\left(-\frac{x^2}{2T}\Big(1-\frac{2x}{T}\Big)\right), \end{aligned}$$

and, analogously,

$$2^{-T}\binom{T}{\frac{T+x}{2}} \geqslant (1.1)^{-2}\sqrt{\frac{2}{\pi T}}\exp\left(-\frac{x^2}{2T}\Big(1+\frac{x}{T}\Big)\right).$$

Plugging this into equation (5.3), using $|z|^2 = x^2 + y^2$ and $x^3 + y^3 \leqslant (x^2+y^2)^{3/2} = |z|^3$, and rounding the factors, in total we obtain

$$\begin{aligned} \Pr[\mathrm{RW}_T = z] &\leqslant (1.1)^2\frac{8}{3\pi T}\exp\left(-\frac{x^2+y^2}{2T} + \frac{x^3+y^3}{T^2}\right) \\ &\leqslant \frac{2}{T}\exp\left(-\frac{|z|^2}{2T}\Big(1-\frac{2|z|}{T}\Big)\right), \end{aligned}$$

and, analogously,

$$\Pr[\mathrm{RW}_T = z] \geqslant \frac{1}{3T}\exp\left(-\frac{|z|^2}{2T}\Big(1+\frac{|z|}{T}\Big)\right). \qquad \square$$

Second, the outcome of a one-dimensional random walk of length T has standard deviation $\Theta(\sqrt{T})$. Intuitively, this implies that with at least constant probability the two-dimensional random walk RW_T is further than $\sqrt{T}$ away from the origin. Moreover, in any direction ξ the expected jump length is large, as shown by the following lemma.

Lemma 5.6. *For any $T \in \mathbb{N}$ we have $\Pr[|\mathrm{RW}_T| \geqslant \sqrt{T}] \geqslant \Omega(1)$.*

Moreover, let τ be a symmetric stopping time, i.e., for all $z \in \mathbb{Z}^2$ we have $\Pr[\mathrm{RW}_\tau = z] = \Pr[\mathrm{RW}_\tau = z']$ where z' is obtained from z by rotating it by $90°$. Then for any $\xi \in \mathbb{R}^2$ with $|\xi| = 1$ and any $T \in \mathbb{N}$ we have

$$\mathbb{E}[|\xi \cdot \mathrm{RW}_\tau|] = \Omega(\Pr[|\mathrm{RW}_\tau| \geqslant \sqrt{T}] \cdot \sqrt{T}).$$

Proof. For any small constant T the claim clearly holds. In the remaining cases we proceed as follows. For any $d \geqslant 1$ the slice $S := \{z \in \mathbb{Z}^2 \mid d \leqslant |z| < 2d\}$ contains

$\Omega(d^2)$ grid points. Using Lemma 5.5 this yields

$$\Pr[\mathrm{RW}_T \geqslant d] \geqslant \Pr[\mathrm{RW}_T \in S] \geqslant \Omega\left(\frac{d^2}{T}\exp\left(-\frac{(2d)^2}{2T}\Big(1+\frac{2d}{T}\Big)\right)\right).$$

For $d = \sqrt{T}$ the right hand side is $\Omega(1)$.

For the second statement, let $w_1, \dots, w_4$ be symmetric points around the origin, i.e., $w_1, \dots, w_4$ form a square with midpoint the origin. Then there is a point w_i which forms an angle of at most 45 degrees with ξ, so that we have $|\xi \cdot w_i| = \Omega(|w_i|)$. Thus, we have

$$\sum_{i=1}^{4} |\xi \cdot w_i| = \Omega(|w_i|) = \Omega\Big(\sum_{i=1}^{4} |w_i|\Big).$$

As the stopping time is symmetric, we can use the above symmetry argument as follows,

$$\mathbb{E}[|\xi \cdot \mathrm{RW}_\tau|] = \sum_{w\in\mathbb{Z}^2} \Pr[\mathrm{RW}_\tau = w] \cdot |\xi \cdot w| = \sum_{w\in\mathbb{Z}^2} \Pr[\mathrm{RW}_\tau = w] \cdot \Omega(|w|).$$

A rough upper bound now yields the claim,

$$\mathbb{E}[|\xi \cdot \mathrm{RW}_\tau|] \geqslant \Omega\Big(\sum_{\substack{w\in\mathbb{Z}^2\\ |w|\geqslant\sqrt{T}}} \Pr[\mathrm{RW}_\tau = w] \cdot \sqrt{T}\Big) = \Omega\big(\Pr[|\mathrm{RW}_\tau| \geqslant \sqrt{T}] \cdot \sqrt{T}\big). \quad \square$$

Finally, the Chernoff bound yields a tail bound for the probability of RW_T to end up too far away from the origin. Together with the reflection principle (see [97]) this yields a tail bound for the probability of being too far away from the origin at any time $0 \leqslant t \leqslant T$.

Lemma 5.7. *For any $T, k \in \mathbb{N}$ we have*

$$\Pr[|\mathrm{RW}_T| > k] \leqslant 4\mathrm{e}^{-\frac{k^2}{2T}}.$$

Moreover, let $0 = z_0, z_1, \dots$ be a random walk and set $\tau := \min_{t\geqslant 0}\{|z_t| > k\}$. Then

$$\Pr[\tau \leqslant T] \leqslant 8\mathrm{e}^{-\frac{k^2}{2T}}.$$

Proof. Let $X_1, X_2, \dots$ be independent copies of a uniform $\{1, -1\}$ random variable and let $S_i := \sum_{j=1}^{i} X_i$. By Lemma 5.3 we have $\mathrm{RW}_T \sim X \cdot (1/2, 1/2) + Y \cdot (1/2, -1/2)$, where X, Y are independent copies of S_T. Since $|\mathrm{RW}_T| = (X^2 + Y^2)^{1/2}/\sqrt{2}$, if $|\mathrm{RW}_T| > k$ then in particular one of $|X|$ and $|Y|$ is larger than k. Hence, we have

$$\Pr[|\mathrm{RW}_T| > k] \leqslant 2\Pr[|S_T| > k] = 2\Pr[|\operatorname{Bin}(T, 1/2) - T/2| > k/2].$$

With a Chernoff bound we obtain

$$\Pr[|\mathrm{RW}_T| > k] \leqslant 4\mathrm{e}^{-\frac{k^2}{2T}}.$$

The second claim now follows from the reflection principle (see [97]) which states that

$$\Pr[\tau \geqslant T] \leqslant 2\Pr[|\mathrm{RW}_T| > k]. \qquad \square$$

5.2. A General Framework

The main idea of our algorithms is to combine many steps of a particle's random walk to a jump as long as the current particle $n+1$ is in the in-circle of $A = A(n)$. In this section, we first formalize the notion of a jump. After that we provide a framework that yields an IDLA simulation algorithm for any given jump procedure. Throughout this chapter, a *step* refers to a single step in a particle's random walk and a *jump* refers to several steps at once.

5.2.1. The Concept of a Jump

Ideally, a *jump* does multiple steps of a random walk at once to save the effort of simulating every single step. Jumps should be concatenable to form longer portions of a random walk. More formally, let $z = z_0, z_1, \dots$ be a random walk starting in z and $\tau = \tau(A, z)$ a stopping time of this random walk. Then $z \mapsto z_\tau$ defines a jump procedure, and the concatenation of two such jumps is again the outcome of a random walk at a certain stopping time. This concatenation property allows us to add up jumps until we finally hit the boundary ∂A. A jump should make at least one single step of the random walk in order to have guaranteed progress, i.e., we require $\tau \geqslant 1$ (with probability 1). Moreover, in order to have a correct simulation of IDLA, jumps must stop at the latest when the random walk leaves A, since then the particle's simulation is complete. Additionally, all jump procedures considered in this chapter are symmetric around z.

There are two important goals for the design of a jump procedure. First, the (expected) *running time* to compute the outcome of a jump should be as small as possible. In particular, it should be faster than simulating the random walk step-by-step. Second, intuitively a jump should be the combination of as many single steps as possible. This can be formalized by requiring the *expected jumping distance* to be large. The following definition captures this concept of a jump.

Definition 5.8. *A* jump procedure *is a randomized algorithm J with input (an IDLA structure) $A \subset \mathbb{Z}^2$ and a point $z \in A$ and output $J(A, z) = z_\tau$, where $z = z_0, z_1, \dots$ is a random walk and $\tau = \tau_J(A, z)$ is any stopping time. We require the jump to make at least one single step of a random walk and to stop at the latest when leaving A for the first time, i.e., $\Pr[1 \leqslant \tau \leqslant \tau_{\partial A}] = 1$, where $\tau_{\partial A} = \min\{t \mid z_t \in \partial A\}$ is the hitting time of ∂A. Additionally, J shall be symmetric around z, i.e., $\Pr[J(A, z) = z + w] = \Pr[J(A, z) = z - w]$ for all $w \in \mathbb{Z}^2$.*

We say that J has running time bound $t_J = t_J(n)$ *if $J(A, z)$ has running time T_J with* $\Pr[T_J > k\, t_J] \leqslant \mathcal{O}(k^{-c})$ *for any $k > 0$ and a sufficiently large constant $c > 0$ (where* Pr *goes over the internal randomness of J). In particular, $J(A, z)$ needs expected time $\mathcal{O}(t_J)$. Moreover, we define the* expected jumping distance *as*

$$\Delta_J(A, z) := \min_{|\xi|=1} \mathbb{E}[|\xi \cdot (J(A, z) - z)|].$$

When A is clear from the context we also write $J(z)$ for $J(A, z)$.

5.2.2. From Jumps to IDLA

Any jump procedure can be iterated to find the point where the random walk first leaves the IDLA structure A. Let $z_0 := (0, 0)$ and $z_{i+1} := J(A, z_i)$, for every $i = 0, 1, 2, \ldots$ as long as z_i is still in A. Moreover, let $\tau^* = \tau^*(J, A) := \min\{i \mid z_i \in \partial A\}$ and $J^* = J^*(A) := z_{\tau^*}$. Note that since J is a randomized algorithm, J^* and τ^* are random variables. Clearly, J^* is distributed exactly as the endpoint of an IDLA particle. This way, any jump procedure gives rise to a simulation algorithm for IDLA.

The following theorem gives an upper bound on the running time of an IDLA simulation with jump procedure J.

Theorem 5.9. *Let J be a jump procedure with running time bound t_J. Let Δ_J be its expected jumping distance, $c_J > 0$ some constant, $B_I := B_0(r_I - c_J \ln n)$, set*

$$\delta_J(A) := \max_{z \in B_I} \frac{r_O - |z|}{\Delta_J(A, z)},$$

and assume that for some $\overline{\delta}_J = \overline{\delta}_J(n)$ we have $\delta_J(A) \leqslant \overline{\delta}_J$ in expectation and with high probability (over the randomness of $A = A(n)$). Then we can construct an algorithm for simulating IDLA with running time

$$\mathcal{O}(n \cdot t_J \cdot \log n \cdot (\overline{\delta}_J^2 + \log n))$$

and space usage[1] $\mathcal{O}(n^{1/2} \log n)$, both in expectation and with high probability.

To see that $\mathcal{O}(n^{1/2} \log n)$ bits are sufficient (in expectation) to store $A(n)$, note that by Theorem 5.1 we have with high probability $B_0(\sqrt{n} - \mathcal{O}(\log n)) \subseteq A(n) \subseteq B_0(\sqrt{n} + \mathcal{O}(\log n))$, and $B_0(\sqrt{n} + \mathcal{O}(\log n)) \setminus B_0(\sqrt{n} - \mathcal{O}(\log n))$ contains $\mathcal{O}(n^{1/2} \log n)$ grid cells, for each of which we can store whether it is occupied in 1 bit.

In Section 5.4 we present a jump procedure with $t_J = \mathcal{O}(1)$ and $\overline{\delta}_J^2 = \mathcal{O}(\log n)$, and thereby provide a proof for Theorem 1.20.

In the remainder of this section we prove Theorem 5.9. First, in Section 5.2.3 we analyze the expected number of jumps that we need to simulate. Then, in Section 5.2.4, we describe and analyze our data structures and finally, in Section 5.2.5, we combine everything to prove Theorem 5.9.

[1] Not including the space used by the jump function.

5.2.3. Number of Jumps

To bound the expected running time of the simulation of a particle using a jump procedure J, we only have to bound the hitting time τ^* of ∂A. The following lemma provides such a bound.

Lemma 5.10. *With the notation of Section 5.2.2, for any IDLA structure A and $k > 0$ we have* $\Pr[\tau^* \geqslant k\,(\delta_J^2(A)\log n + (r_O - r_I + \log n)^2)] \leqslant \exp(-\Omega(k))$.

In particular, we have $\mathbb{E}[\tau^*] \leqslant \mathcal{O}\left(\delta_J^2(A)\log n + (r_O - r_I + \log n)^2\right)$.

Proof. Consider again the stochastic process $z_0 = (0,0)$, and $z_{k+1} = J(A, z_k)$ for $k > 0$. Set $\sigma := \sqrt{2}(r_O - r_I + c_J \ln n)$. We analyze this process in phases. The process starts in phase 1 and changes to phase 2 the first time it reaches a position $z_k \notin B_I$. For the next σ^2 jumps the process stays in phase 2. After that it returns to phase 1, except if we are again outside B_I, then we directly start another phase 2. This repeats until we hit ∂A. For these phases we prove the following.

(1) Starting phase 1 anywhere in B_I, we stay in phase 1 for at most $\mathcal{O}(\delta_J^2(A)\log n)$ jumps in expectation.

(2) Starting phase 2 anywhere outside B_I, the probability of hitting ∂A before the end of the phase is $\Omega(1)$.

Using Markov's inequality, (1) implies that after at most $\mathcal{O}(\delta_J^2(A)\log n)$ jumps we leave phase 1 with probability $\Omega(1)$. Together with (2) we obtain that, wherever we start, within $\mathcal{O}(\delta_J^2(A)\log n + \sigma^2)$ jumps we hit ∂A with probability $\Omega(1)$. Hence, within $\mathcal{O}(k(\delta_J^2(A)\log n + \sigma^2))$ jumps we hit ∂A with probability $1 - \exp(-\Omega(k))$, yielding both expectation and concentration of the hitting time.

We first prove (2). Assume that phase 2 starts in $z_k \notin B_I$. We show that with probability $\Omega(1)$ the random walk hits ∂A within the next σ^2 *steps*; since each jump consists of at least one step this proves the claim. Consider the boundary of a ball with radius σ around z_k, $S := \overline{B}_{z_k}(\sigma)$. By Lemma 5.6, starting in z_k we hit S in σ^2 steps with constant probability. Note that at least a quarter of S lies outside A (even outside $B_0(r_O)$), specifically the set $S' := \{x \in S \mid \sphericalangle(z_k, x - z_k) \in [-\frac{\pi}{4}, \frac{\pi}{4}]\}$, i.e., the set of points $x \in S$ such that the angle between z_k and $x - z_k$ is at most $\pi/4$. This is because any such point has a distance to $(0,0)$ of

$$\begin{aligned} |x| \geqslant \frac{x \cdot z_k}{|z_k|} = |z_k| + \frac{(x - z_k)\cdot z_k}{|z_k|} &\geqslant |z_k| + \frac{1}{\sqrt{2}}|x - z_k| \\ &\geqslant (r_I - c_J \ln n) + (r_O - r_I + c_J \ln n) = r_O. \end{aligned}$$

By symmetry of the random walk, we not only hit S, but even hit S' in σ^2 steps with constant probability. This means, however, that within σ^2 steps we leave A with constant probability, proving the claim.

To show (1) we apply additive drift analysis to prove that the stochastic process $z_0, z_1, \ldots, z_\tau$ (for $z_0 \in B_I$ and $\tau := \min\{k \mid z_k \notin B_I\}$) has an expected hitting time

as claimed. In order to apply Theorem 5.2 we need a suitable distance function $g : \mathbb{Z}^2 \to \mathbb{R}_{\geqslant 0}$. We let

$$g(z) := \begin{cases} \ln(r_O + 2 - |z|), & z \in B_I \\ 0, & z \in \mathbb{Z}^2 \setminus B_I. \end{cases}$$

In the following we will show that g has an additive drift of $\min_{z \in B_I} \frac{(\Delta_J(A,z))^2}{2(r_O+2-|z|)^2}$ for all $0 \leqslant k < \tau$, i.e., for any $z_k \in B_I$

$$\mathbb{E}\left[g(z_{k+1}) - g(z_k) \mid z_k\right] \leqslant -\min_{z \in B_I} \frac{(\Delta_J(A,z))^2}{2(r_O + 2 - |z|)^2}\,. \tag{5.4}$$

Applying Theorem 5.2 together with $g(z) \leqslant \mathcal{O}(\log n)$ then yields an expected hitting time of $\mathbb{Z}^2 \setminus B_I$ of $\mathcal{O}(\delta_J^2(A) \log n)$.

Whenever $z_k \in A$ we know that $z_{k+1} \in \overline{A} \subseteq B_0(r_O+1)$. In this case we can bound $g(z_{k+1}) \leqslant \ln(r_O + 2 - |z_{k+1}|)$. To shorten notation we let $L(x) := \ln(r_O + 2 - x)$ for any $x \in \mathbb{R}$ in the remainder of this proof. Hence, the expectation of $g(z_{k+1})$ conditioned on z_k, $z_k \in B_I$, is at most[2]

$$\sum_{x \in \mathbb{Z}^2} \Pr\left[z_{k+1} = x \mid z_k\right] \cdot L(|x|) \leqslant \sum_{x \in \mathbb{Z}^2} \Pr\left[z_{k+1} = x \mid z_k\right] \cdot L\left(x \frac{z_k}{|z_k|}\right),$$

since the length of the projection of x is bounded by $|x|$ in any direction. Using the transformation $y_x := x - z_k$ and the symmetry of jump procedures we can rewrite this as

$$\begin{aligned} &\sum_{x \in \mathbb{Z}^2} \Pr\left[z_{k+1} = x | z_k\right] \cdot L\left(|z_k| - y_x \frac{z_k}{|z_k|}\right) \\ &= \frac{1}{2} \sum_{x \in \mathbb{Z}^2} \Pr\left[z_{k+1} = x | z_k\right] \cdot \left(L\left(|z_k| - y_x \frac{z_k}{|z_z|}\right) + L\left(|z_k| + y_x \tfrac{z_k}{|z_k|}\right)\right), \end{aligned} \tag{5.5}$$

where $|y_x \frac{z_k}{|z_k|}| \leqslant r_O + 1 - |z_k|$ for all x with $\Pr\left[z_{k+1} = x | z_k\right] > 0$.

Now we use the following estimate that holds for any $a, b \in \mathbb{R}$ with $a > 0$ and $|b| \leqslant a$:

$$\ln(a+b) + \ln(a-b) \leqslant 2\ln(a) - \frac{b^2}{a^2}. \tag{5.6}$$

Combining (5.5) and (5.6) yields

$$\mathbb{E}\left[g(z_{k+1}) | z_k\right] \leqslant \frac{1}{2} \sum_{x \in \mathbb{Z}^2} \Pr\left[z_{k+1} = x | z_k\right] \cdot \left(2L(|z_k|) - \frac{(y_x \cdot z_k/|z_k|)^2}{(r_O + 2 - |z_k|)^2}\right),$$

[2]Here, we define the corresponding summand to be 0 whenever the ln is undefined.

which we further bound as

$$\mathbb{E}\left[g(z_{k+1})|z_k\right] \leqslant g(z_k) - \frac{\mathbb{E}\left[(y_{z_{k+1}} \cdot z_k/|z_k|)^2 \middle| z_k\right]}{2(r_O + 2 - |z_k|)^2} = g(z_k) - \frac{\mathbb{E}\left[|(z_{k+1} - z_k)\frac{z_k}{|z_k|}|^2 \middle| z_k\right]}{2(r_O + 2 - |z_k|)^2}$$
$$\leqslant g(z_k) - \frac{\mathbb{E}\left[|(z_{k+1} - z_k)\frac{z_k}{|z_k|}| \middle| z_k\right]^2}{2(r_O + 2 - |z_k|)^2} \tag{5.7}$$

where the last inequality follows from Jensen's inequality. Considering the definition of the expected jumping distance Δ_J (Definition 5.8) with $\xi = \frac{z_k}{|z_k|}$ we obtain

$$\mathbb{E}\left[g(z_{k+1})|z_k\right] \leqslant g(z_k) - \frac{(\Delta_J(A, z_k))^2}{2(r_O + 2 - |z_k|)^2}$$

which proves the drift inequality (5.4) and, thus, the lemma. □

5.2.4. Data Structure

Jump procedures need access to the IDLA structure $A = A(n)$ in some way. In this section we describe a data structure for storing A that fits the needs of our jump procedures.

The most natural solution for storing the shape of A is a $2n \times 2n$ matrix in which each element contains the information whether the corresponding position is occupied. The size of this matrix is $\Theta(n^2)$ and already its initialization would therefore exceed our desired running time. As we have $r_O - r_I = \mathcal{O}(\log n)$ with high probability, an $\mathcal{O}(\sqrt{n}) \times \mathcal{O}(\sqrt{n})$ matrix is sufficient (in most simulations). However, in this case A contains the whole center $B_0(r_I)$, so most of the information stored in the matrix is still redundant.

To save space we split the grid in slices $S_0, S_1, \ldots$, where slice S_i contains all $z \in \mathbb{Z}^2$ with $i \leqslant |z| < i + 1$. We store r_I and r_O, as well as $A \cap S_i$ for all $r_I \leqslant i < r_O$. This is sufficient information to reconstruct A, as $A \cap S_i = S_i$ for $i < r_I$ and $A \cap S_i = \emptyset$ for $i \geqslant r_O$.

We would like to store each set $A \cap S_i$ by a bit array, however, the natural index set S_i is not a range of integers. As a workaround, we first construct a perfect hash function $h_i \colon S_i \to [n_i]$ to map S_i to a range of integers of length $n_i = \mathcal{O}(|S_i|) = \mathcal{O}(i)$ without collision. See, e.g., [98] for a construction of a perfect hash function with $\mathcal{O}(n_i)$ construction time, $\mathcal{O}(n_i)$ space usage and $\mathcal{O}(1)$ evaluation time[3]. Additionally, we store a bit array $B_i[1..n_i]$ with $B_i[h_i(z)] = [z \in A]$, i.e., for each $z \in S_i$ the bit $B_i[h_i(z)]$ stores whether z is in A. In total this yields $\mathcal{O}(1)$ time to access/modify $[z \in A]$ with space requirement $\mathcal{O}(i)$. Note the similarities to storing a bit array for $A \cap S_i$, only that this is not directly possible.

To allow for efficient updates of our data structure after placing a particle, we also store $|S_i|$ and $|A \cap S_i|$ for each $r_I \leqslant i < r_O$. Then whenever a new grid point z is occupied (in slice S_i), we do the following. If slice S_i is not yet built, i.e., if r_O was

[3]We remark that it is an easy exercise to explicitly construct a perfect hash function in our situation, simply based on the angles of the grid points inside a slice.

at most i, then we increase r_O and initialize the data structure for $A \cap S_i = \{z\}$. Specifically, we enumerate S_i, compute $|S_i|$ and $|A \cap S_i| = |\{z\}| = 1$, build the perfect hash function h_i and initialize the bit array B_i. Otherwise, if the slice S_i was already built, then we simply update $B_i[h_i(z)] := 1$ and $|A \cap S_i|$. Whenever the slice S_{r_I} becomes full (i.e., $|A \cap S_{r_I}| = |S_{r_I}|$) we trash everything we stored for this slice $(h_{r_I}, B_{r_I}, |S_{r_I}|, |A \cap S_{r_I}|)$ and increase r_I. We repeat this as long as S_{r_I} is full. This way we satisfy an invariant of S_{r_I} being the innermost non-full slice and S_{r_O} being the lowest empty slice. In particular, r_I and r_O are always the correct in- and out-radius.

It is not hard to see that, when starting from the empty set, the updates of our data structure take time $\mathcal{O}(r_O^2)$ in total, and that it uses $\mathcal{O}(r_O(r_O - r_I))$ bits of space. Since $r_O - r_I$ (and, thus, also r_O) is bounded both in expectation and with high probability, we get the following lemma.

Lemma 5.11. *We can construct a data structure for A that allows to*

- *query r_I and r_O in $\mathcal{O}(1)$ time,*
- *check $z \in A$ in $\mathcal{O}(1)$ time, and*
- *add $z \in \mathbb{Z}^2$ to A.*

Adding the n particles of an IDLA simulation one-by-one to this data structure overall needs $\mathcal{O}(n)$ time and $\mathcal{O}(n^{1/2}\log(n))$ space, both in expectation and with high probability.

5.2.5. Proof of Theorem 5.9

Theorem 5.9 is a consequence of Theorem 5.1, Lemma 5.10 and Lemma 5.11.

Proof of Theorem 5.9. We simulate the first $\sqrt{n}$ particles in the naive step-by-step way in expected time $\mathcal{O}(n)$. For all the remaining particles we use the jump procedure. Since Theorem 5.1 together with $r_O \leqslant n$ implies $\mathbb{E}[(r_O - r_I)^2] \leqslant \mathcal{O}(\log^2 n)$, we can read off Lemma 5.10 an expected number of jumps of $\mathcal{O}(\log n(\bar{\delta}_J^{-2} + \log n))$, yielding an expected time of $\mathcal{O}(t_J \log n(\bar{\delta}_J^{-2} + \log n))$ to simulate the i-th particle. Adding the running time of our data structure (Lemma 5.11), in total over all particles we obtain an expected time as claimed. Lemma 5.11 also directly implies the statement about space usage.

Arguing about the concentration of the running time requires some more work. Note that $r_O - r_I = \mathcal{O}(\log n)$ for all $\sqrt{n} < i \leqslant n$ with high probability. If this is the case then the number of jumps needed for particle $\sqrt{n} < i \leqslant n$ is more than $k \log n(\bar{\delta}_J^{-2} + \log n)$ with probability $\exp(-\Omega(k))$ by Lemma 5.10. By the following lemma, the total number of jumps for all these particles is $\mathcal{O}(n \log n(\bar{\delta}_J^{-2} + \log n))$ with high probability. Moreover, the running time of the jump procedure is larger than $k\, t_J$ with probability at most $\mathcal{O}(k^{-c})$, therefore the following lemma shows that these $\mathcal{O}(n \log n(\bar{\delta}_J^{-2} + \log n))$ jumps in total take time $\mathcal{O}(t_J n \log n(\bar{\delta}_J^{-2} + \log n))$ also with high probability. In total, the claimed running time $\mathcal{O}(t_J n \log n(\bar{\delta}_J^{-2} + \log n))$ also

holds with high probability for the particles $\sqrt{n} < i \leqslant n$. For the naive simulation of the first $\sqrt{n}$ particles we are in a similar situation. With high probability $A(i)$ is contained in $B_0(\mathcal{O}(n^{1/4}))$ for all $1 \leqslant i \leqslant \sqrt{n}$ (Theorem 5.1). If this is the case, then the simulation of one of the first $\sqrt{n}$ particles takes time larger than $k\sqrt{n}$ with probability $\exp(-\Omega(k))$ for any $k \geqslant 1$, since after each block of $\Theta(\sqrt{n})$ steps we leave the ball with radius $\Theta(n^{1/4})$ with constant probability (Lemma 5.6). Again using the following lemma, we obtain that this naive simulation takes time $\mathcal{O}(n)$ in total with high probability, finishing the proof. □

Lemma 5.12. *Let $X_1, \ldots, X_m$ be independent positive random variables with $\Pr[X_i > kv] \leqslant \mathcal{O}(k^{-c})$ for some $v > 0$, constant $c > 0$, and all $k \geqslant 1$. Then we have*

$$\sum_{i=1}^{m} X_i \leqslant \mathcal{O}(mv)$$

with probability at least $1 - \mathcal{O}(m^{-(c/3-1)})$.

Proof. Without loss of generality we can assume $v = 1$. Assume that all X_i are at most $M > 0$ to be fixed later. By a union bound this happens with probability at least $1 - \mathcal{O}(mM^{-c})$. This assumption means that we instead consider bounded random variables Y_i with $\Pr[Y_i \leqslant k] = \Pr[X_i \leqslant k] / \Pr[X_i \leqslant M]$ for any $k \leqslant M$. Using the Azuma-Hoeffding bound (see, e.g., [99, Chapter 5]) we obtain

$$\Pr\Big[\sum_{i=1}^{m}(Y_i - \mathbb{E}[Y_i]) > t\Big] \leqslant \exp\left(\frac{-t^2}{2mM^2}\right).$$

Hence, for sufficiently large constant $\alpha > 0$ we have

$$\Pr\Big[\sum_{i=1}^{m} Y_i > \alpha m\Big] \leqslant \exp(-\Omega(m/M^2)),$$

and in total we get

$$\Pr\Big[\sum_{i=1}^{m} X_i > \alpha m\Big] \leqslant \exp(-\Omega(m/M^2)) + \mathcal{O}(mM^{-c}).$$

Choosing $M = m^{1/3}$ this error is $\mathcal{O}(m^{-(c/3-1)})$ as claimed. □

5.3. Implementing Jumps

Here, we discuss two approaches for jump procedures. In Section 5.3.1, we analyze jumps that combine d steps, where d is the current distance of the particle to the boundary of the in-circle. In Section 5.3.2 we discuss jumping directly to the boundary of a bounding circle. Note that we present our best jump procedure in Section 5.4.

5.3.1. Time Jumps

Observe that, starting at some point z in the in-circle, a random walk cannot leave $A(n)$ within $r_I(A) - |z|$ steps. Building on this observation, the *time jump* J_{time} simply simulates $r_I(A) - |z|$ steps of a random walk at once, i.e., $J_{\text{time}}(z) = \text{RW}_\tau$ with $\tau = \tau(A, z) := \max\{\lfloor r_I(A) - |z| \rfloor, 1\}$.

Clearly, this is a stopping time and J_{time} is symmetric. Since in this number of steps we cannot leave $\overline{A}(n)$, we also have $\Pr[1 \leqslant \tau \leqslant \tau_{\partial A}] = 1$. For the expected jumping distance, Lemma 5.6 yields a bound of $\Delta_{J_{\text{time}}}(z) \geqslant \Omega(\sqrt{\max\{r_I - |z|, 1\}})$, This is not very much: When the initial distance to the in-circle is $\Theta(n^{1/2})$, the first jump has expected length only $\Theta(n^{1/4})$. However, J_{jump} can be computed efficiently with running time bound $t_{J_{\text{time}}} = \mathcal{O}(1)$, since we can sample RW_τ in expected constant time by Lemma 5.4. Note that in order to determine τ we need to keep track of r_I at all times, but this is done by our data structure for $A(n)$ (see Lemma 5.11). Plugging J_{time} into Theorem 5.9 yields the following result. Note that this is already faster than the naive simulation.

Corollary 5.13. *Using time jumps we can simulate IDLA in $\mathcal{O}(n^{3/2} \log n)$ time and $\mathcal{O}(n^{1/2} \log n)$ space, both in expectation and with high probability.*

5.3.2. Shape Jumps

Alternatively, we take some shape S (think of a ball or square) around the particle's current position and directly jump to the first position where the particle leaves the shape S. More precisely, let $S \subseteq B_z(r_I - |z|)$ (symmetric around z). Then $S \subseteq A$, so we leave A after leaving S. Thus, a valid jump is to directly sample $J_{shape}(z) := z_{\tau_{\partial S}}$, where $\tau_{\partial S}$ is the hitting time of ∂S of a random walk $z_0 = z, z_1, z_2, \ldots$. Once the particle is outside $B_0(r_I)$ we just make single steps, i.e., $J_{shape}(z) := z_1$.

The expected jumping distance of a shape jump is automatically very large, if $B_z(d) \subseteq S$ then $\Delta_{J_{shape}}(z) = \Omega(d)$. The crux lies in efficiently sampling $J_{shape}(z)$. For this we precompute the probabilities $P_S(w) := \Pr[J_{shape}(z) = w]$ for all $w \in \partial S$. After this precomputation we can sample $J_{shape}(z)$ in constant time. In the following we sketch the details of this.

For any $v \in \overline{S}$ and $w \in \partial S$ let $P_S(w, v)$ be the probability that a random walk starting in v leaves S (for the first time) at w. Clearly,

$$P_S(w, v) = \begin{cases} \sum_{v' \in \Gamma(v)} \frac{1}{4} P_S(w, v') & \text{if } v \in S, \\ 1 & \text{if } v = w, \\ 0 & \text{if } v \in \partial S \setminus \{w\}. \end{cases}$$

Solving this system yields $P_S(w, z) = P_S(w)$, so solving this system for all $w \in \partial S$ computes the desired probability distribution.

How much time does it take to solve this linear system? It is not hard to see that the resulting probabilities are rationals with numerator and denominator bounded by $2^{\mathcal{O}(|\partial S|)}$. Hence, Gaussian Elimination takes time $\mathcal{O}(|S|^3 |\partial S|)$. Even if we could solve the linear system faster (which seems possible, as this is a sparse system), this

would still take time proportional to the output size, which is $\Omega(|S||\partial S|)$. The total time to compute all values $P_S(w)$ is then between $\mathcal{O}(|S|^3|\partial S|^2)$ and $\Omega(|S||\partial S|^2)$.

Once we have computed the probability distribution $P_S(w)$ we bring them to a common denominator of $\mathcal{O}(|\partial S|^2)$ bits. Then we use a data structure for sampling from discrete distributions, e.g., Walker's alias method. After some small preprocessing this data structure allows to sample from the input distribution in expected time $\mathcal{O}(1)$ plus the time to compare a random real in $[0,1]$ with a rational of $\mathcal{O}(|\partial S|^2)$ bits. The latter can be done in expected constant time if we additionally store approximations of this rational with a denominator of 2^i bits for $i = 1, 2, \ldots$. In total we get that in time between $\mathcal{O}(|S|^3|\partial S|^2)$ and $\Omega(|S||\partial S|^2)$ we can build a data structure that allows to sample from $P_S(w)$ in expected constant time.

To get a simulation algorithm for IDLA we now do the following. We build a data structure for sampling from $P_S(w)$ for S being a ball of radius $2^1, 2^2, \ldots, 2^k$ with $2^k = \Theta(n^\varepsilon)$. When the particle is at position z we take the largest of these balls (translated by z) that still lies in $B_0(r_I)$ and directly jump to its boundary. This sampling can be done in time $t_{J_{shape}} = \mathcal{O}(1)$, the precomputation takes time between $\mathcal{O}(n^{8\varepsilon})$ and $\Omega(n^{4\varepsilon})$. The expected jumping distance now is

$$\Delta_{J_{shape}}(z) = \Theta(\min\{n^\varepsilon, \max\{r_I - |z|, 1\}\}).$$

This allows to bound $\overline{\delta}^2_{J_{shape}} = \mathcal{O}(n^{1-2\varepsilon})$. Setting $\varepsilon = 1/5$ and plugging this into Theorem 5.9 yields an algorithm for IDLA with expected running time $\mathcal{O}(n^{8/5} \log n)$, which is worse than for time jumps. If the running time for solving the linear system is near to the lower bound, the best possible from this approach is achieved for $\varepsilon = 1/3$ with a running time of $\mathcal{O}(n^{4/3} \log n)$. While this is faster than time jumps, it is much worse than the long jumps that we present in the next section.

5.4. Long Jumps

In the previous section, we simulated the random walk either by a fixed amount of steps, which we choose small enough to be safe to stay in $A(n)$, or by the hitting point of a surrounding shape, which has a complicated distribution that takes much time to compute. In this section, we combine these two approaches to obtain jumps that are both long and efficiently computable.

To this end, consider a particle at position $z \in B_I = B_0(r_I - c_J \ln n)$ (for some sufficiently large constant $c_J > 0$) and consider the ball $S := B_z(\sigma)$ with midpoint z and radius $\sigma := r_I - |z|$, so that S is contained in $B_0(r_I) \subseteq A$. Let $z_0, z_1, \ldots$ be a random walk starting in $z_0 = z$, let $\tau_{\partial S} := \min\{i \mid z_i \in \partial S\}$ be its hitting time of the boundary of S, and similarly let $\tau_{\partial A} := \min\{i \mid z_i \in \partial A\}$. Note that a shape jump as in Section 5.3.2 would return $z_{\tau_{\partial S}}$, while a time jump as in Section 5.3.1 would return z_σ. In the current section we will directly jump to $J_{\text{long}}(z) := z_\tau$ with

$$\tau := \min\{\tau_{\partial S}, T\},$$

where

$$T := \left\lfloor \frac{\sigma^2}{c_J \ln(n/\mathrm{e})} \right\rfloor = \Theta\Big(\frac{\sigma^2}{\log n}\Big).$$

Whenever $z \notin B_I$, we simply make one step of the random walk, i.e., $\tau := 1$. This way we make sure that $\tau \geqslant 1$ (for all $z \in A$). Note that here we use $\tau_{\partial S}$ to ensure $\tau \leqslant \tau_{\partial S} \leqslant \tau_{\partial A}$, meaning that we stop at the latest when leaving A. Since τ is a stopping time and J_{long} is symmetric, this is a valid jump procedure according to Definition 5.8. It is not clear at first sight that J_{long} can be sampled efficiently for all $z \in B_I$. We present an algorithm with constant expected running time in the next section and then analyze its correctness and running time. Note that in this analysis we can assume that $z \in B_I$, as otherwise we do single steps. After that, we determine the expected jumping distance of J_{long}. Overall, we obtain the following result.

Lemma 5.14. *The jump procedure J_{long} has running time bound $t_{J_{\text{long}}} = \mathcal{O}(1)$ and for any $z \in B_I$ an expected jumping distance of $\Delta_{J_{\text{long}}}(A, z) = \Omega(\sqrt{T}) = \Omega\left(\frac{r_I(A)-|z|}{\sqrt{\log n}}\right)$. Furthermore, it has a space usage of $\mathcal{O}(1)$ memory cells (in expectation and with high probability).*

Plugging this into Theorem 5.9 proves the main result of this chapter, specifically that we can construct an algorithm for simulating IDLA with running time $\mathcal{O}(n \log^2 n)$ and space $\mathcal{O}(n^{1/2} \log n)$, both in expectation and with high probability.

Proof of Theorem 1.20. By Lemma 5.14 we can bound $\delta_{J_{\text{long}}}$ (as defined in Theorem 5.9) by

$$\max_{z \in B_I} \mathcal{O}\left(\sqrt{\log n} \cdot \frac{r_O - |z|}{r_I - |z|}\right) = \mathcal{O}\left(\frac{r_O - r_I + \log n}{\sqrt{\log n}}\right),$$

since the term on the left hand is is maximized at the boundary of B_I, where $|z| = r_I - c_J \ln n$. Since we have by Theorem 5.1 that $r_O - r_I = \mathcal{O}(\log n)$ in expectation and with high probability, we can set $\overline{\delta}_{J_{\text{long}}} = \mathcal{O}(\sqrt{\log n})$. Plugging this into Theorem 5.9 yields the desired running time. □

5.4.1. An Algorithm for Sampling Long Jumps

Observe that with high probability a random walk of length T starting in z does not leave S. Hence, the minimum of $\tau_{\partial S}$ and T is typically obtained at T. We will design an algorithm that samples the position of z_T (restricted to a certain subset) very efficiently. Additionally, we have to patch this approximate algorithm by a second (slow) algorithm that is executed only with small probability and that compensates for any mistakes we might make by sampling only z_T.

First consider Algorithm 4, which does not yet correctly sample a jump according to the distribution of $J_{\text{long}}(z)$. It simply draws a point $z' = \mathrm{RW}_T(z)$ (see Lemma 5.4) and rejects as long as $z' \notin \frac{1}{2}S$ (where $\frac{1}{2}S$ is the ball with midpoint z and radius $\frac{1}{2}\sigma$).

Algorithm 4 Algorithm Long-Jump-Incomplete

1: **repeat**
2: $z' := \mathrm{RW}_T(z)$
3: **until** $z' \in \frac{1}{2}S$
4: **return** z'.

For $w \in \mathbb{Z}^2$ let $P_J(w) := \Pr[J_{\text{long}}(z) = w]$ and denote the probability of Algorithm 4 to return w by $P_{Alg4}(w)$. To patch Algorithm 4 we choose a failure probability p_{fail} (to be fixed later). Then, with probability $1 - p_{\text{fail}}$ we run Algorithm 4, but with probability p_{fail} we patch the algorithm by exhaustively computing the probabilities $P_J(w)$ and $P_{Alg4}(w)$ for all $w \in \overline{S}$ and returning $w \in \overline{S}$ with probability $P_{rest}(w)$, where

$$(1 - p_{\text{fail}}) \cdot P_{Alg4}(w) + p_{\text{fail}} \cdot P_{rest}(w) = P_J(w). \tag{5.8}$$

The above equation ensures that overall we draw $w \in \mathbb{Z}^2$ according to the right probability distribution P_J. The approach is summarized in Algorithm 5.

Algorithm 5 Algorithm Long-Jump-Complete

1: choose p uniformly at random from $[0, 1]$.
2: **if** $p < p_{\text{fail}}$ **then**
3: *// fail compensate*
4: calculate $P_J(w)$ and $P_{Alg4}(w)$ for all $w \in \overline{S}$
5: compute $P_{rest}(w)$ according to equation (5.8)
6: return $w \in \overline{S}$ drawn according to the distribution $P_{rest}(w)$
7: **else**
8: run Algorithm 4

Correctness This algorithm is correct if p_{fail} can be chosen in such a way that P_{rest} is a probability distribution. We analyze for which values of p_{fail} this is the case.

Lemma 5.15. *The values $P_{rest}(w)$ for $w \in \overline{S}$ form a probability distribution if we choose $p_{\text{fail}} \geqslant 28\mathrm{e}^{c_J/2} n^{-\min\{c_J/8, 5c_J/16-1\}}$.*

Because of equation (5.8) and $\sum_w P_J(w) = \sum_w P_{Alg4}(w) = 1$ we easily obtain the equality $\sum_w P_{rest}(w) = 1$. However, we have to prove that we can choose p_{fail} (and c_J) in such a way that $P_{rest}(w)$ is non-negative for all $w \in \overline{S}$. It suffices to choose p_{fail} such that for all $w \in \overline{S}$

$$(1 - p_{\text{fail}}) P_{Alg4}(w) \leqslant P_J(w),$$

since then $P_{rest}(w) \geqslant 0$ according to equation (5.8). Without loss of generality we consider $w \in \frac{1}{2}S$ with $w \equiv_T z$, as otherwise we have $P_{Alg4}(w) = 0$.

We abbreviate $P_{RW}(w) := \Pr[\mathrm{RW}_T(z) = w]$. Since Algorithm 4 returns the endpoint of a random walk of length T conditioned on being contained in $\frac{1}{2}S$, we clearly have

$$P_{Alg4}(w) = \frac{P_{RW}(w)}{1 - \Pr[\mathrm{RW}_T(z) \notin \frac{1}{2}S]}. \tag{5.9}$$

Since $\Pr[\mathrm{RW}_T(z) \notin \frac{1}{2}S] = \Pr[|\mathrm{RW}_T| > \frac{1}{2}\sigma]$ and $T \leqslant \frac{\sigma^2}{c_J \ln(n/\mathrm{e})}$ we can apply Lemma 5.7 to get

$$P_{Alg4}(w) \leqslant \frac{P_{RW}(w)}{1 - 4(n/\mathrm{e})^{-c_J/8}}. \tag{5.10}$$

For $P_J(w)$, any walk of length T starting in z and ending in w is counted, except if it hits ∂S. Hence, we have

$$P_J(w) \geqslant P_{RW}(w) - \Pr[\tau_{\partial S} \leqslant T].$$

We want to write the right hand side of this as $P_{RW}(w) \cdot (1 - \rho)$, so that combined with equation (5.10) we obtain $P_J(w) \geqslant P_{Alg4}(w) \cdot (1 - \rho')$. For this we need an upper bound for $\Pr[\tau_{\partial S} \leqslant T]$ (provided by Lemma 5.7) and a lower bound for $P_{RW}(w)$ (provided by Lemma 5.5). Combining the two yields (after some technical simplifications that we postpone)

$$\frac{\Pr[\tau_{\partial S} \leqslant T]}{P_{RW}(w)} \leqslant 24\mathrm{e}^{c_J/2} n^{1-5c_J/16}. \tag{5.11}$$

Thus,

$$P_J(w) \geqslant P_{RW}(w)(1 - 24\mathrm{e}^{c_J/2} n^{1-5c_J/16}),$$

and together with equation (5.10) we obtain

$$P_J(w) \geqslant (1 - 24\mathrm{e}^{c_J/2} n^{1-5c_J/16} - 4(n/\mathrm{e})^{-c_J/8}) P_{Alg4}(w).$$

Thus, we can safely set $p_{\text{fail}} \geqslant 28\mathrm{e}^{c_J/2} n^{-\min\{c_J/8, 5c_J/16-1\}}$.

In the following we show inequality (5.11). First note that by the definition of T and $\sigma \geqslant c_J \ln n$ we have

$$T \geqslant \frac{\sigma^2}{c_J \ln(n/\mathrm{e})} - 1 \geqslant \frac{\sigma^2}{c_J \ln n} \cdot \left(1 + \frac{1}{\ln n}\right) - 1 \geqslant \frac{\sigma^2}{c_J \ln n} + c_J - 1 \geqslant \frac{\sigma^2}{c_J \ln n},$$

for $c_J \geqslant 1$. Thus, $|w| \leqslant \frac{1}{2}\sigma$, $|w|/T \leqslant \frac{1}{2}$ and $\frac{\sigma^2}{T} \leqslant c_J \ln n$. Using this, we can combine Lemmas 5.7 and 5.5 to obtain

$$\begin{aligned}\frac{\Pr[\tau_{\partial S} \leqslant T]}{P_{RW}(w)} \leqslant \frac{8(n/\mathrm{e})^{-c_J/2}}{\frac{1}{3T}\exp\big(-\frac{|w|^2}{2T}\big(1 + \frac{|w|}{T}\big)\big)} &\leqslant 24T(n/\mathrm{e})^{-c_J/2} \exp\left(\frac{3\sigma^2}{16T}\right) \\ &\leqslant 24\mathrm{e}^{c_J/2} T n^{-5c_J/16}.\end{aligned}$$

Note that as S is completely filled with particles we have $n \geqslant \sigma^2 \geqslant T$, finally yielding

$$\frac{\Pr[\tau_{\partial S} \leqslant T]}{P_{RW}(w)} \leqslant 24\mathrm{e}^{c_J/2} n^{1-5c_J/16}.$$

Running Time In the fail compensation part of our algorithm we have to compute P_{Alg4} and P_J exactly. In this section we discuss how to do this efficiently, which yields a bound on the running time of our algorithm.

Observe that for $P_{RW}(w) := \Pr[\mathrm{RW}_T(z) = w]$ we have for all $w \in \frac{1}{2}S$ that

$$P_{Alg4}(w) = P_{RW}(w) / \sum_{w \in \frac{1}{2}S} P_{RW}(w).$$

This reduces the calculation of P_{Alg4} to the calculation of $P_{RW}(w)$ for all $w \in \frac{1}{2}S$. For $w \not\equiv_T z$ we have $P_{RW}(w) = 0$, so let $w \equiv_T z$. Then we can write $w = x \cdot (1/2, 1/2) + y \cdot (1/2, -1/2)$ with $x, y \in \mathbb{Z}$. With the notation of Lemma 5.3 we have

$$P_{RW}(w) = \Pr[X = x] \cdot \Pr[Y = y] = 2^{-T} \binom{T}{\frac{T+x}{2}} \cdot 2^{-T} \binom{T}{\frac{T+y}{2}}.$$

Note that this probability has denominator 4^T, so it can be stored using $\mathcal{O}(T)$ bits. Moreover, as $\binom{T}{i}$ can be computed in $\mathcal{O}(T)$ multiplications and divisions of a $\mathcal{O}(T)$ bit number by a $\mathcal{O}(\log T)$ bit number, we can calculate $P_{RW}(W)$ in time $\mathcal{O}(T^2 \log T)$. The total running time for calculating P_{Alg4} is therefore $\mathcal{O}(\sigma^2 T^2 \log T)$ and the occupied space is $\mathcal{O}(\sigma^2 T)$.

For computing P_J we use a simple iterative scheme. We recursively define X_w^t for $0 \leqslant t \leqslant T$ and $w \in \overline{S}$. For $t = 0$ we set

$$X_w^0 = \begin{cases} 1 & \text{if } w = z, \\ 0 & \text{otherwise,} \end{cases}$$

while for $t > 0$ we set

$$X_w^t = \begin{cases} \sum_{v \in \Gamma(w) \cap S} \frac{1}{4} X_v^{t-1} & \text{if } w \in S, \\ X_w^{t-1} + \sum_{v \in \Gamma(w) \cap S} \frac{1}{4} X_v^{t-1} & \text{if } w \in \partial S. \end{cases}$$

Observe that X_w^T is equal to $P_J(w)$ for every $w \in \overline{S}$, and each probability X_w^t can be stored using $\mathcal{O}(T)$ bits. The total running time to calculate P_J is therefore $\mathcal{O}(\sigma^2 T^2)$ and the space usage is $\mathcal{O}(\sigma^2 T)$ bits.

As the ball S is completely filled with particles, we have $n \geqslant \sigma^2$. Using $T = \Theta(\sigma^2 / \log n)$ we get a running time of $\mathcal{O}(n^3)$ and a space usage of $\mathcal{O}(n^2)$ for computing P_J and P_{Alg4}.

Thus, the fail compensation is called with probability p_{fail} and takes time $\mathcal{O}(n^3)$. The remaining part of the algorithm, Algorithm 4, has a loop for sampling z', which takes time $\mathcal{O}(1)$ in expectation and with high probability, as the probability of $\mathrm{RW}_T \notin \frac{1}{2}S$ is small (smaller than p_{fail}, as chosen in the last section). One

iteration of this loop samples from the binomial distribution and takes time T with $\Pr[T > t] \leqslant \exp(-t^{\Omega(1)})$, by Lemma 5.4. In total, the expected running time is

$$\mathcal{O}(1 + p_{\text{fail}} \cdot n^3) = \mathcal{O}(1),$$

for sufficiently large constant c_J, so that Lemma 5.15 allows us to choose p_{fail} sufficiently small. Moreover, the total running time T_J satisfies $\Pr[T_J > k] \leqslant \mathcal{O}(k^{-c})$. The contribution of the fail compensation part is $\mathcal{O}(p_{\text{fail}})$ for any $k \leqslant \mathcal{O}(n^3)$, which is small enough. Hence, we obtain a running time bound of $t_{J_{\text{long}}} = \mathcal{O}(1)$. This proves the first part of Lemma 5.14.

Expected Jumping Distance In this section we analyze the expected jumping distance $\Delta_{J_{\text{long}}}(z)$ of long jumps, proving the second part of Lemma 5.14. Recall that the expected jumping distance at $z \in B_I$ is defined as

$$\Delta_{J_{\text{long}}}(A, z) = \min_{|\xi|=1} \mathbb{E}[|\xi^T (J_{\text{long}}(A, z) - z)|].$$

Since the stopping time τ of J_{long} is symmetric, we can use the second part of Lemma 5.6 to obtain $\Delta_{J_{\text{long}}}(A, z) = \Omega(\Pr[|J_{\text{long}}(A, z) - z| \geqslant \sqrt{T}] \cdot \sqrt{T})$. Observe that we have $\Pr[|J_{\text{long}}(A, z) - z| \geqslant \sqrt{T}] \geqslant \Pr[|\text{RW}_T| \geqslant \sqrt{T}]$, where the inequality comes from some walks in $\text{RW}_{\min\{\tau_{\partial S}, T\}}(z)$ ending prematurely (if $\tau_{\partial S} \leqslant T$). Together with the first part of Lemma 5.6, this shows $\Delta_{J_{\text{long}}}(A, z) \geqslant \Omega(\sqrt{T}) = \Omega((r_I(A) - |z|)/\sqrt{\log n})$.

5.5. Generalizations

A common generalization of the random walks considered in this chapter works as follows. We have a "stencil", which is a probability distribution p on $\mathbb{Z}^2$ with finite support $S = \{x_1, \ldots, x_m\}$. In each step of the random walk we sample a point x according to the distribution p and go from the current point z_k to $z_{k+1} = z_k + x$. It is not known whether for this generalization a sufficiently smooth ball emerges, too. However, one might conjecture that a statement analogous to Theorem 5.1 holds whenever the stencil is fixed (in particular the "width" $\max_i |x_i|$ of the stencil is constant) and symmetric ($p(x) = p(-x)$, so that the random walk has no drift) and maybe some more conditions are fulfilled. If this is the case, then the main results of this chapter generalize. One statement that does not generalize is that random walks can be split into two independent 1-dimensional random walks (Lemma 5.3). This necessitates a different method for sampling RW_T. One way to still sample this in constant time is to first sample how many times the random walks takes a step in direction x_1, this is $T_1 \sim \text{Bin}(T, p(x_1))$, then to sample how many times we go to x_2, this is $T_2 \sim \text{Bin}(T - T_1, p(x_2))$, and so on. Apart from this, the generalizations are straightforward, at least once analogous lemmas to Section 5.1.1 are shown. Unfortunately, it is very tedious to get explicit bounds on the involved constants, which are needed to specify an explicit failure probability p_{fail} for the long jump

procedure. This means that it is relatively easy to show that an exact sampling algorithm with expected running time $\mathcal{O}(n \log^2 n)$ *exists*, but it is tedious to show which of the algorithms, parameterized by p_{fail}, has these guarantees.

PART II

Computing Fréchet Distances

6 Introduction

Intuitively, the (continuous) Fréchet distance of two curves P, Q is the minimal length of a leash required to connect a dog to its owner, as they walk along P or Q, respectively, without backtracking. The Fréchet distance is a very popular measure of similarity of two given curves. In contrast to distance notions such as the Hausdorff distance, it takes into account the order of the points along the curve, and thus better captures the similarity as perceived by human observers [100].

Alt and Godau introduced the Fréchet distance to computational geometry in 1991 [101, 102]. For polygonal curves P and Q with n and m vertices[1], respectively, they presented an $\mathcal{O}(nm \log(nm))$ algorithm. Since Alt and Godau's seminal paper, Fréchet distance has become a rich field of research, with various directions such as generalizations to surfaces (see, e.g., [103]), approximation algorithms for realistic input curves ([104–106]), the geodesic and homotopic Fréchet distance (see, e.g., [107, 108]), and many more variants (see, e.g., [109–112]). Being a natural measure for curve similarity, the Fréchet distance has found applications in various areas such as signature verification (see, e.g., [113]), map-matching tracking data (see, e.g., [114]), and moving objects analysis (see, e.g., [115]).

A particular variant that we will also discuss in this dissertation is the *discrete* Fréchet distance. Here, intuitively the dog and its owner are replaced by two frogs, and in each time step each frog can jump to the next vertex along its curve or stay at its current vertex. Defined in [116], the original algorithm for the discrete Fréchet distance has running time $\mathcal{O}(nm)$.

Quadratic time complexity? Recently, improved algorithms have been found for some variants. Agarwal et al. [117] showed how to compute the discrete Fréchet distance in (mildly) sub-quadratic time $\mathcal{O}\big(nm\frac{\log\log n}{\log n}\big)$. Buchin et al. [118] gave algorithms for the continuous Fréchet distance with runtime $\mathcal{O}(n^2\sqrt{\log n}(\log\log n)^{3/2})$ on the Real RAM and $\mathcal{O}(n^2(\log\log n)^2)$ on the Word RAM. However, the problem remains open whether there is a *strongly sub-quadratic*[2] algorithm for the Fréchet distance, i.e., an algorithm with running time $\mathcal{O}(n^{2-\delta})$ for any $\delta > 0$. For a particular variant, the discrete Fréchet distance with shortcuts, strongly sub-quadratic algorithms have been found recently [119], however, this seems to have no implications for the classical continuous or discrete Fréchet distance.

[1] We always assume that $m \leqslant n$.

[2] We use the term *strongly sub-quadratic* to differentiate between this running time and the *(mildly) sub-quadratic* $\mathcal{O}(n^2 \log\log n/\log n)$ algorithm from [117].

The only known lower bound shows that the Fréchet distance takes time $\Omega(n \log n)$ (in the algebraic decision tree model) [120]. The typical way of proving (conditional) quadratic lower bounds for geometric problems is via 3SUM [121], in fact, Alt conjectured that the Fréchet distance is 3SUM-hard. Buchin et al. [118] argued that the Fréchet distance is unlikely to be 3SUM-hard, because it has strongly sub-quadratic decision trees. However, their argument breaks down in light of a recent result showing strongly sub-quadratic decision trees also for 3SUM [122]. Hence, it is completely open whether the Fréchet distance is 3SUM-hard and whether it has strongly sub-quadratic algorithms.

Realistic Input Curves In attempts to break the apparent quadratic time barrier at least for realistic inputs, various restricted classes of curves have been considered, such as backbone curves [104], κ-bounded and κ-straight curves [105], and ϕ-low density curves [106]. The most popular model of realistic inputs are *c-packed curves*. A curve π is c-packed if for any point $z \in \mathbb{R}^d$ and any radius $r > 0$ the total length of π inside the ball $B(z, r)$ is at most cr, where $B(z, r)$ is the ball of radius r around z. This model is well motivated from a practical point of view. Examples of classes of c-packed curves are boundaries of convex polygons and γ-fat shapes as well as algebraic curves of bounded maximal degree (see [106]). The model has been used for several generalizations of the Fréchet distance, such as map matching [123], the mean curve problem [124], a variant of the Fréchet distance allowing shortcuts [110], and Fréchet matching queries in trees [125]. Driemel et al. [106] introduced c-packed curves and presented a $(1 + \varepsilon)$-approximation for the continuous Fréchet distance in time $\mathcal{O}(cn/\varepsilon + cn \log n)$, which works in any $\mathbb{R}^d$, $d \geqslant 2$. Thus, near-linear approximation algorithms exist for realistic input curves.

We formally define the Fréchet distance in the next section. In the remainder of this chapter, we discuss our new results.

6.1. Variants of the Fréchet Distance

In this section, we formally define the continuous and the discrete variant of the Fréchet distance.

A (polygonal) curve P is defined by its vertices $p_1, \ldots, p_n$. We view P as a continuous function $P\colon [0, n] \to \mathbb{R}^d$ with $P(i + \lambda) = (1 - \lambda)p_i + \lambda p_{i+1}$ for $i \in [n - 1]$, $\lambda \in [0, 1]$. We write $|P| = n$ for the number of vertices of P and $\|P\|$ for its total length $\sum_{i=1}^{n-1} \|p_i - p_{i+1}\|$, where $\|.\|$ denotes the Euclidean distance.

Let Φ_n be the set of all continuous and non-decreasing functions ϕ from $[0, 1]$ onto $[0, n]$. The *continuous Fréchet distance* between two curves P_1, P_2 with $|P_1| = n$, $|P_2| = m$ is defined as

$$d_{\mathrm{F}}(P_1, P_2) := \inf_{\substack{\phi_1 \in \Phi_n \\ \phi_2 \in \Phi_m}} \max_{t \in [0,1]} \|P_1(\phi_1(t)) - P_2(\phi_2(t))\|.$$

We call (ϕ_1, ϕ_2) a (continuous) *traversal* of (P_1, P_2), and say that it has *width* D if $\max_{t\in[0,1]} \|P_1(\phi_1(t)) - P_2(\phi_2(t))\| \leqslant D$.

In the discrete case, we let Δ_n be the set of all non-decreasing functions ϕ from $[0, 1]$ onto $[n]$. The *discrete Fréchet distance* between two curves P_1, P_2 with $|P_1| = n$, $|P_2| = m$ is then defined as

$$d_{\mathrm{dF}}(P_1, P_2) := \inf_{\substack{\phi_1\in\Delta_n \\ \phi_2\in\Delta_m}} \max_{t\in[0,1]} \|P_1(\phi_1(t)) - P_2(\phi_2(t))\|.$$

We obtain an analogous notion of a (discrete) *traversal* and its *width*. Note that any $\phi \in \Delta_n$ is a staircase function attaining all values in $[n]$. Hence, $(\phi_1(t), \phi_2(t))$ changes only at finitely many points in time t. At any such *time step* we jump to the next vertex in P_1 or P_2 or both.

It is known that for any curves P_1, P_2 we have $d_{\mathrm{F}}(P_1, P_2) \leqslant d_{\mathrm{dF}}(P_1, P_2)$ [116].

For $r \geqslant 0$ and $z \in \mathbb{R}^d$ we denote by $B(z, r)$ the ball or radius r around z.

6.2. Lower Bounds Based On SETH

The results in this section are proven in Chapter 7. Instead of relating the Fréchet distance to 3SUM, we consider the Strong Exponential Time Hypothesis.

Strong Exponential Time Hypothesis Exponential Time Hypothesis (ETH) and Strong Exponential Time Hypothesis (SETH), both introduced by Impagliazzo, Paturi, and Zane [126, 127], provide ways of proving conditional lower bounds. ETH asserts that 3-SAT has no $2^{o(N)}$ algorithm, where N is the number of variables, and can be used to prove matching lower bounds for a wealth of problems, see [128] for a survey. However, since this hypothesis does not specify the exact exponent, it is not suited for proving polynomial time lower bounds, where the exponent is important.

The stronger hypothesis SETH asserts that there is no $\delta > 0$ such that k-SAT has an $\mathcal{O}((2 - \delta)^N)$ algorithm for all k. Here, we will use the following weaker variant, which has also been used in [129, 130].

Hypothesis SETH′: *There is no $\mathcal{O}^*((2 - \delta)^N)$ algorithm for CNF-SAT for any $\delta > 0$. Here, $\mathcal{O}^*$ hides polynomial factors in the number of variables N and the number of clauses M.*

While SETH deals with formulas of width k, SETH′ deals with CNF-SAT, i.e., unbounded width clauses. Thus, it is a weaker assumption and more likely to be true. Note that exhaustive search takes time $\mathcal{O}^*(2^N)$, and the fastest known algorithms for CNF-SAT are only slightly faster than that, namely of the form $\mathcal{O}^*(2^{N(1-C/\log(M/N))})$ for some positive constant C [131, 132]. Thus, SETH′ is a reasonable assumption that can be considered unlikely to fail. It has been observed that one can use SETH and SETH′ to prove lower bounds for polynomial time problems such as k-Dominating Set and others [129], the diameter of sparse graphs [130],

and dynamic connectivity problems [133]. However, it seems to be applicable only for few problems, e.g., it seems to be a wide open problem to prove that 3SUM has no strongly sub-quadratic algorithms unless SETH fails, similarly for matching, maximum flow, edit distance, and other classic problems.

Main lower bound Our main result of this section gives strong evidence that the Fréchet distance may have no strongly sub-quadratic algorithms by relating it to the Strong Exponential Time Hypothesis.

Theorem 6.1. *There is no $\mathcal{O}(n^{2-\delta})$ algorithm for the (continuous or discrete) Fréchet distance for any $\delta > 0$, unless* SETH′ *fails.*

Since SETH and its weaker variant SETH′ are reasonable hypotheses, by this theorem one can consider it unlikely that the Fréchet distance has strongly sub-quadratic algorithms. In particular, any strongly sub-quadratic algorithm for the Fréchet distance would not only give improved algorithms for CNF-SAT that are much faster than exhaustive search, but also for various other problems such as Hitting Set, Set Splitting, and NAE-SAT via the reductions in [134]. Alternatively, in the spirit of [129], one can view the above theorem as a possible attack on CNF-SAT, as algorithms for the Fréchet distance now could provide a route to faster CNF-SAT algorithms. In any case, anyone trying to find strongly sub-quadratic algorithms for the Fréchet distance should be aware that this is as hard as finding improved CNF-SAT algorithms, which might be impossible.

We remark that all our lower bounds (unless stated otherwise) hold in the Euclidean plane, and thus also in $\mathbb{R}^d$ for any $d \geqslant 2$.

Extensions We extend our main lower bound in two important directions: We show approximation hardness and we prove that the lower bound still holds for restricted classes of curves.

First, it would be desirable to have good approximation algorithms in strongly sub-quadratic time, say a near-linear time approximation scheme. We exclude such algorithms by proving that there is no 1.001-approximation for the Fréchet distance in strongly sub-quadratic time unless SETH′ fails. Hence, within $n^{o(1)}$-factors any 1.001-approximation takes as much time as an exact algorithm. We did not try to optimize the constant 1.001, but only to find the asymptotically largest possible approximation ratio, which seems to be a constant. We leave it as an open problem whether there is a strongly sub-quadratic $\mathcal{O}(1)$-approximation. The literature so far contains no strongly sub-quadratic approximation algorithms for general curves at all.

Second, it might be conceivable that if one curve has much fewer vertices than the other, i.e., $m \ll n$, then after some polynomial preprocessing on the smaller curve we can compute the Fréchet distance of the two curves quickly, e.g., in total time $\mathcal{O}((n + m^3) \log n)$. Note that such a running time is not ruled out by the trivial argument that any algorithm needs time $\Omega(n + m)$ for reading the input, and is also not ruled out by Theorem 6.1, since the running time is not sub-quadratic for $n = m$. We rule out such running times by proving that there is no $\mathcal{O}((nm)^{1-\delta})$ algorithm "for any m", unless SETH′ fails. More precisely, we prove this lower bound

for the "special case" $m \approx n^\gamma$ for any constant $0 \leqslant \gamma \leqslant 1$. To make this formal, for any input parameter α and constants $\gamma_0 < \gamma_1$ in $\mathbb{R} \cup \{-\infty, \infty\}$, we say that a statement holds *for any polynomial restriction of* $n^{\gamma_0} \leqslant \alpha \leqslant n^{\gamma_1}$ if it holds restricted to instances with $n^{\gamma-\delta} \leqslant \alpha \leqslant n^{\gamma+\delta}$ for any constants $\delta > 0$ and $\gamma_0 + \delta \leqslant \gamma \leqslant \gamma_1 - \delta$. We obtain the following extension of the main lower bound, Theorem 6.1, which yields tight lower bounds for any behavior of m and any $(1+\varepsilon)$-approximation with $0 \leqslant \varepsilon \leqslant 0.001$.

Theorem 6.2. *There is no* 1.001*-approximation with running time* $\mathcal{O}((nm)^{1-\delta})$ *for the (continuous or discrete) Fréchet distance for any* $\delta > 0$*, unless* SETH′ *fails. This holds for any polynomial restriction of* $1 \leqslant m \leqslant n$*.*

Realistic input curves We also consider realistic input assumptions, specifically the popular model of c-packed curves. Recall that an algorithm by Driemel et al. [106] yields a $(1+\varepsilon)$-approximation for the continuous Fréchet distance on c-packed curves in time $\mathcal{O}(cn/\varepsilon + cn \log n)$. This works in any $\mathbb{R}^d$, $d \geqslant 2$. While this algorithm is near-linear for small c and $1/\varepsilon$, is is not clear whether its dependence on c and $1/\varepsilon$ is optimal for c and $1/\varepsilon$ that grow with n. We give strong evidence that the algorithm of [106] has optimal dependence on c for any constant $0 < \varepsilon \leqslant 0.001$.

Theorem 6.3. *There is no* 1.001*-approximation with running time* $\mathcal{O}((cn)^{1-\delta})$ *for the (continuous or discrete) Fréchet distance on* c*-packed curves for any* $\delta > 0$*, unless* SETH′ *fails. This holds for any polynomial restriction of* $1 \leqslant c \leqslant n$*.*

Since we prove this claim for any polynomial restriction $c \approx n^\gamma$, the above result also excludes 1.001-approximations with running time, say, $\mathcal{O}(c^2 + n)$.

Regarding the dependence on ε, in any dimension $d \geqslant 5$ we can prove a conditional lower bound that matches the dependency on ε of [106] up to a polynomial.

Theorem 6.4. *In* $\mathbb{R}^d$*,* $d \geqslant 5$*, there is no* $(1+\varepsilon)$*-approximation for the (continuous or discrete) Fréchet distance on* c*-packed curves running in time* $\mathcal{O}(\min\{cn/\sqrt{\varepsilon}, n^2\}^{1-\delta})$ *for any* $\delta > 0$*, unless* SETH′ *fails. This holds for sufficiently small* $\varepsilon > 0$ *and any polynomial restriction of* $1 \leqslant c \leqslant n$ *and* $\varepsilon \leqslant 1$*.*

Outline of the main lower bound To prove the main result of this section, we present a reduction from CNF-SAT to the Fréchet distance. Given a CNF-SAT instance φ, we partition its variables into sets V_1, V_2 of equal size. In order to find a satisfying assignment of φ we have to choose (partial) assignments a_1 of V_1 and a_2 of V_2. We will construct curves P_1, P_2 where P_k is responsible for choosing a_k. To this end, P_k consists of *assignment gadgets*, one for each assignment of V_k. Assignment gadgets are built of *clause gadgets*, one for each clause. The assignment gadgets of assignments a_1 of V_1 and a_2 of V_2 are constructed such that they have Fréchet distance at most 1 if and only if (a_1, a_2) forms a satisfying assignment of φ. In P_1 and P_2 we connect these assignment gadgets with some additional curves to implement an OR-gadget, which forces any traversal of (P_1, P_2) to walk along two assignment gadgets in parallel. If φ is not satisfiable, then any pair of assignment gadgets has Fréchet distance larger than 1, so that P_1, P_2 have Fréchet distance larger than 1. If, on the other hand, a satisfying assignment (a_1, a_2) of φ exists,

then we ensure that there is a traversal of P_1, P_2 that essentially only traverses the assignment gadgets of a_1 and a_2 in parallel, so that it always stays in distance 1.

To argue about the running time, since P_k contains an assignment gadget for every assignment of one half of the variables, and every assignment gadget has polynomial size in M, there are $n = \mathcal{O}^*(2^{N/2})$ vertices on each curve. Thus, any $\mathcal{O}(n^{2-\delta})$ algorithm for the Fréchet distance would yield an $\mathcal{O}^*(2^{(1-\delta/2)N})$ algorithm for CNF-SAT, contradicting SETH′.

Remark: Orthogonal Vectors Let Orthog be the problem of "finding a pair of orthogonal vectors": given two sets $S_1, S_2 \subseteq \{0,1\}^d$ of n vectors each, determine if there are $u \in S_1$ and $v \in S_2$ with $\langle u, v\rangle = \sum_{i=1}^{d} u_i v_i = 0$, where the sum is computed over the integers, see [135, 136]. Clearly, Orthog can be solved in time $\mathcal{O}(n^2 d)$. However, Orthog has no strongly sub-quadratic algorithms unless SETH′ fails. More precisely, in [135] it was shown that SETH′ implies the following statement.

OrthogHypothesis: *There is no algorithm for Orthog with running time $\mathcal{O}(n^{2-\delta} d^{\mathcal{O}(1)})$ for any $\delta > 0$.*

All known conditional lower bounds based on SETH′ implicitly go through Orthog or some variant of this problem. In fact, this is also the case for our results, as is easily seen by going through the proof in [135] and noting that we use the same tricks. Specifically, given a CNF-SAT instance ϕ on variables $x_1, \ldots, x_N$ and clauses $C_1, \ldots, C_M$ we split the variables into two halves V_1, V_2 of equal size and enumerate all assignments A_k of true and false to V_k. Then every clause C_i specifies sets $B_k^i \subseteq A_k$ of partial assignments that do not make C_i become true. Clearly, a satisfying assignment $(a_1, a_2) \in A_1 \times A_2$ has to evade $B_1^i \times B_2^i$ for all i. This problem is equivalent to an instance of Orthog with $d = M$ and $n = 2^{N/2}$, where S_k contains a vector for every partial assignment $a_k \in A_k$ and the i-th position of this vector is 1 or 0, depending on whether $a_k \in B_k^i$ or not. In our proof, we could replace this instance by an arbitrary instance of Orthog, yielding a reduction from Orthog to the Fréchet distance.

Hence, in Theorems 6.1, 6.3, and 6.4 we could replace the assumption "unless SETH′ fails" by the weaker assumption "unless OrthogHypothesis fails". This is a stronger statement, since there is only more reason to believe that Orthog has no strongly sub-quadratic algorithms than that there is for believing that CNF-SAT takes time $2^{N-o(N)}$. Moreover, it shows a relation between two polynomial time problems, Orthog and the Fréchet distance.

For Theorem 6.2 we would need an imbalanced version of the OrthogHypothesis, where the two sets S_1, S_2 have different sizes n_1, n_2. Then unless SETH′ fails there is no $\mathcal{O}((n_1 n_2)^{1-\delta} d^{\mathcal{O}(1)})$ algorithm for any $\delta > 0$, and this holds for any polynomial restriction of $1 \leqslant n_1 \leqslant n_2$, which follows from a slight generalization of [135]. If we state this implication of SETH′ as a hypothesis OrthogHypothesis*, then in Theorem 6.2 we could replace "unless SETH′ fails" by the weaker assumption "unless OrthogHypothesis* fails".

6.3. Improved Approximation for Realistic Inputs

The results in this section are proven in Chapter 8.

For c-packed curves, the conditional lower bounds from the last section leave two particularly interesting open questions, asking for new algorithms or improved lower bounds. Here, we use $\tilde{\mathcal{O}}$ to ignore any polylogarithmic factors in n, c, and $1/\varepsilon$.

1. Is there a $(1+\varepsilon)$-approximation with running time $\tilde{\mathcal{O}}(cn/\sqrt{\varepsilon})$ for the Fréchet distance on c-packed curves?
2. In any dimension $d \in \{2,3,4\}$, is there a $(1+\varepsilon)$-approximation with running time $\tilde{\mathcal{O}}(cn)$ for the Fréchet distance on c-packed curves? Or is there even an exact algorithm with running time $\tilde{\mathcal{O}}(cn)$?

Here, we positively answer the first question, i.e., we improve upon the algorithm by Driemel et al. [106] and present an algorithm that matches the conditional lower bound of Theorem 6.4. The second question is left as an open problem.

Theorem 6.5. *For any $0 < \varepsilon \leqslant 1$ we can compute a $(1+\varepsilon)$-approximation for the continuous and discrete Fréchet distance on c-packed curves in time $\tilde{\mathcal{O}}(cn/\sqrt{\varepsilon})$.*

Specifically, our running time is $\mathcal{O}(\frac{cn}{\sqrt{\varepsilon}}\log(1/\varepsilon) + cn\log n)$ for the discrete variant and $\mathcal{O}(\frac{cn}{\sqrt{\varepsilon}}\log^2(1/\varepsilon) + cn\log n)$ for the continuous variant.

We want to highlight that in general dimensions (specifically, $d \geqslant 5$) this running time is *optimal* (apart from lower order terms of the form $n^{o(1)}$ unless SETH fails by Theorem 6.4). Moreover, we obtained our new algorithm by investigating why the conditional lower bound of Theorem 6.4 cannot be improved and exploiting the discovered properties. Thus, the above theorem is the outcome of a synergistic effect of algorithms and lower bounds. In particular, this shows one more reason why conditional lower bounds such as Theorems 6.2, 6.3, and 6.4 should be studied, as they can show tractable cases and suggest properties that make these cases tractable.

We remark that the same algorithm also yields improved running time guarantees for other models of realistic input curves, like κ-bounded and κ-straight curves, where we are also able to essentially replace ε by $\sqrt{\varepsilon}$ in the running time bound. In contrast to c-packed curves, it is not clear how far these bounds are from being optimal. See Section 8.2.2 for details.

Outline of the Algorithm We give an improved algorithm that approximately decides whether the Fréchet distance of two given curves π, σ is at most δ. Using a construction of [106] to search over possible values of δ, this yields an improved approximation algorithm. We partition our curves into sub-curves, each of which is either a *long segment*, i.e., a single segment of length at least $\Lambda = \Theta(\sqrt{\varepsilon}\delta)$, or a *piece*, i.e., a sub-curve staying in the ball of radius Λ around its initial vertex. Now we run the usual algorithm that explores the reachable free-space (see Section 8.1 for definitions), however, we treat regions spanned by a piece π' of π and a piece σ' of σ in a special way. Typically, if π', σ' consist of n', m' segments then their

free-space would be resolved in time $\mathcal{O}(n'm')$. Our overall speedup comes from reducing this running time to $\tilde{\mathcal{O}}(n' + m')$, which is our first main contribution. To this end, we consider the line through the initial vertices of the pieces π', σ', and project π', σ' onto this line to obtain curves $\hat{\pi}, \hat{\sigma}$. Since π', σ' are *pieces*, i.e., they stay within distance $\Lambda = \Theta(\sqrt{\varepsilon}\delta)$ of their initial vertices, this projection does not change distances from π to σ significantly (it follows from the Pythagorean theorem that any distance of approximately δ is changed, by the projection, by less than $\varepsilon\delta$). Thus, we can replace π', σ' by $\hat{\pi}, \hat{\sigma}$ without introducing too much error. Note that $\hat{\pi}, \hat{\sigma}$ are *one-dimensional* curves; without loss of generality we can assume that they lie on $\mathbb{R}$. Moreover, we show how to ensure that $\hat{\pi}, \hat{\sigma}$ are *separated*, i.e., all vertices of $\hat{\pi}$ lie above 0 and all vertices of $\hat{\sigma}$ lie below 0. Hence, we reduced our problem to resolving the free-space region of one-dimensional separated curves.

It is known[3] that the Fréchet distance of one-dimensional separated curves can be computed in near-linear time, essentially since we can walk along π and σ with *greedy steps* to either find a feasible traversal or bottleneck sub-curves. However, we face the additional difficulty that we have to resolve the *free-space region* of one-dimensional separated curves, i.e., given entry points on $\hat{\pi}$ and $\hat{\sigma}$, compute all exits on $\hat{\pi}$ and $\hat{\sigma}$. Our second main contribution is that we present an extension of the known result to handle this much more complex problem.

[3]We thank Wolfgang Mulzer for pointing us to this result by Matias Korman and Sergio Cabello (personal communication). To the best of our knowledge this result is not published.

7 Lower Bounds Based On SETH

This chapter is based on the single author paper [2].

[2] K. Bringmann. "Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails." In: *Proc. 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS'14)*. To appear. 2014.

In this chapter, we prove conditional lower bounds for algorithms computing the Fréchet distance. All of our lower bounds assume a weaker variant of the Strong Exponential Time Hypothesis (SETH'). We first restate the results from Section 6.2 that we will prove in this chapter.

Theorem 6.1. *There is no $\mathcal{O}(n^{2-\delta})$ algorithm for the (continuous or discrete) Fréchet distance for any $\delta > 0$, unless SETH' fails.*

Theorem 6.2. *There is no 1.001-approximation with running time $\mathcal{O}((nm)^{1-\delta})$ for the (continuous or discrete) Fréchet distance for any $\delta > 0$, unless SETH' fails. This holds for any polynomial restriction of $1 \leqslant m \leqslant n$.*

Theorem 6.3. *There is no 1.001-approximation with running time $\mathcal{O}((cn)^{1-\delta})$ for the (continuous or discrete) Fréchet distance on c-packed curves for any $\delta > 0$, unless SETH' fails. This holds for any polynomial restriction of $1 \leqslant c \leqslant n$.*

Theorem 6.4. *In $\mathbb{R}^d$, $d \geqslant 5$, there is no $(1+\varepsilon)$-approximation for the (continuous or discrete) Fréchet distance on c-packed curves running in time $\mathcal{O}(\min\{cn/\sqrt{\varepsilon}, n^2\}^{1-\delta})$ for any $\delta > 0$, unless SETH' fails. This holds for sufficiently small $\varepsilon > 0$ and any polynomial restriction of $1 \leqslant c \leqslant n$ and $\varepsilon \leqslant 1$.*

This chapter is structured as follows. We start by defining notation and basic properties of CNF-SAT in Section 7.1. Section 7.2 deals with general curves. We prove the main result for the discrete Fréchet distance on 3 pages in Section 7.2.1. This construction also already proves inapproximability. We generalize the proof to the continuous Fréchet distance in Section 7.2.2 (which is more tedious than in the discrete case) and to $m \ll n$ in Section 7.2.3 (which is an easy trick). Section 7.3 deals with c-packed curves. In Section 7.3.1 we present a new OR-gadget that

generates less packed curves; plugging in the curves constructed in the main result proves Theorem 6.3. In Section 7.3.2 we make use of the fact that in $\geqslant 4$ dimensions there are point sets Q_1, Q_2 of arbitrary size with each pair of points (q_1, q_2) having distance exactly 1. This allows to construct less packed curves that we plug into the OR-gadget from the preceding section to prove Theorem 6.4.

7.1. Preliminaries

Recall that a (polygonal) curve P is defined by its vertices $p_1, \ldots, p_n$ and we can view P as a continuous function $P\colon [0, n] \to \mathbb{R}^d$ with $P(i + \lambda) = (1 - \lambda)p_i + \lambda p_{i+1}$ for $i \in [n - 1]$, $\lambda \in [0, 1]$. We write $|P| = n$ for the number of vertices of P. For two curves P_1, P_2 we let $P_1 \circ P_2$ be the curve on $|P_1| + |P_2|$ vertices that first follows P_1, then walks along the segment from $P_1(|P_1|)$ to $P_2(0)$, and then follows P_2. In particular, for two points $p, q \in \mathbb{R}^d$ the curve $p \circ q$ is the segment from p to q, and any curve P on vertices $p_1, \ldots, p_n$ can be written as $P = p_1 \circ \ldots \circ p_n$.

Consider a curve P and two points $p_1 = P(\lambda_1)$, $p_2 = P(\lambda_2)$ with $\lambda_1, \lambda_2 \in [0, n]$. We say that p_1 *is within distance D of p_2 along P* if the length of the sub-curve of P between $P(\lambda_1)$ and $P(\lambda_2)$ is at most D.

Realistic input curves Recall that a curve P is *c-packed* if for any point $q \in \mathbb{R}^d$ and any radius $r > 0$ the total length of P inside the ball $B(q, r)$ is at most cr. We say that a curve P is *$\Theta(c)$-packed*, if there are constants $\alpha > \beta > 0$ such that P is αc-packed but not βc-packed.

Satisfiability In CNF-SAT we are given a formula φ on variables $x_1, \ldots, x_N$ and clauses $C_1, \ldots, C_M$ in conjunctive normal form with unbounded clause width. Let V be any subset of the variables of φ. Let a be any assignment of T (true) or F (false) to the variables of V. We call a a *partial assignment* and say that *a satisfies* a clause $C = \bigvee_{i \in I} x_i \vee \bigvee_{i \in J} \neg x_i$ if for some $i \in I \cap V$ we have $a(x_i) = \mathsf{T}$ or for some $i \in J \cap V$ we have $a(x_i) = \mathsf{F}$. We denote by $\mathsf{sat}(a, C)$ whether partial assignment a satisfies clause C. Note that assignments a of V and a' of the remaining variables V' form a satisfying assignment (a, a') of φ if and only if we have $\mathsf{sat}(a, C_i) \vee \mathsf{sat}(a', C_i) = \mathsf{T}$ for all $i \in \{1, \ldots, M\}$.

All bounds that we prove in this chapter assume the hypothesis SETH' (see Section 6.2), which asserts that CNF-SAT has no $\mathcal{O}^*((2-\delta)^N)$ algorithm for any $\delta > 0$. Here, $\mathcal{O}^*$ hides polynomials factors in N and M. The following is an easy corollary of SETH'.

Lemma 7.1. *There is no $\mathcal{O}^*((2 - \delta)^N)$ algorithm for CNF-SAT restricted to formulas with N variables and $M \leqslant 2^{\delta' N}$ clauses for any $\delta, \delta' > 0$, unless SETH' fails.*

Proof. Any such algorithm would imply an $\mathcal{O}^*((2 - \delta)^N)$ algorithm for CNF-SAT (without restrictions on the input), since for $M \leqslant 2^{\delta' N}$ we can run the given algorithm, while for $M > 2^{\delta' N}$ we can decide satisfiability in time $\mathcal{O}(M2^N) = \mathcal{O}(M^{1+1/\delta'}) = \mathcal{O}^*(1)$. □

7.2. General Curves

We first present a reduction from CNF-SAT to the Fréchet distance and show that it proves Theorem 6.1 for the discrete Fréchet distance. In Section 7.2.2 we then show that the same construction also works for the continuous Fréchet distance. Finally, in Section 7.2.3 we generalize these results to curves with imbalanced numbers of vertices n, m to show Theorem 6.2.

7.2.1. The Basic Reduction, Discrete Case

Let φ be a given CNF-SAT instance with variables $x_1, \dots, x_N$ and clauses $C_1, \dots, C_M$. We split the variables into two halves $V_1 := \{x_1, \dots, x_{N/2}\}$, $V_2 := \{x_{N/2+1}, \dots, x_N\}$. For $k \in \{1, 2\}$ let A_k be all assignments[1] of T or F to the variables in V_k, so that $|A_k| = 2^{N/2}$. In the whole section we let $\varepsilon := 1/1000$.

We will construct two curves P_1, P_2 such that $d_{\mathrm{dF}}(P_1, P_2) \leqslant 1$ if and only if φ is satisfiable. In the construction we will use gadgets as follows.

Clause gadgets This gadget encodes whether a partial assignment satisfies a clause. We set for $i \in \{0, 1\}$

$$c^i_{1,\mathsf{T}} := \big(i/3, \tfrac{1}{2} - \varepsilon\big), \qquad c^i_{1,\mathsf{F}} := \big(i/3, \tfrac{1}{2} + \varepsilon\big),$$
$$c^i_{2,\mathsf{T}} := \big(i/3, -\tfrac{1}{2} + \varepsilon\big), \quad c^i_{2,\mathsf{F}} := \big(i/3, -\tfrac{1}{2} - \varepsilon\big),$$

see Figure 7.1a. Let $k \in \{1, 2\}$. For any partial assignment $a_k \in A_k$ and clause C_i, $i \in [M]$, we construct a clause gadget consisting of a single point,

$$CG(a_k, i) := c^{i \bmod 2}_{k,\mathsf{sat}(a_k, C_i)}.$$

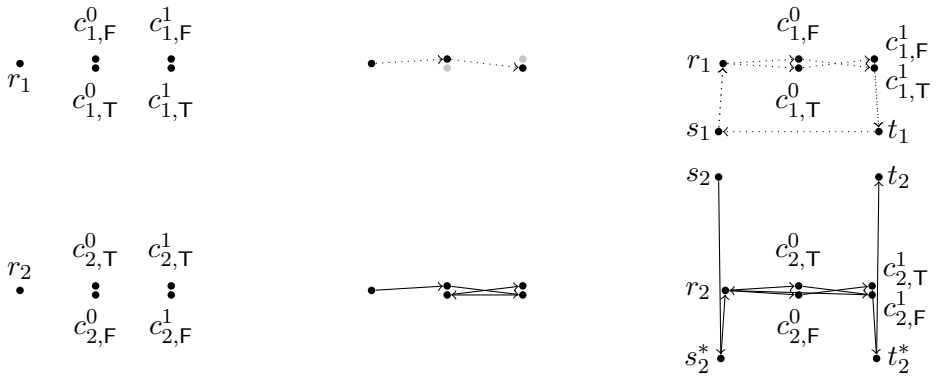
Thus, if assignment a_k satisfies clause C_i then the corresponding clause gadget is nearer to the clause gadgets associated with A_{3-k}. Explicitly calculating all pairwise distances of these points, we obtain the following lemma.

Lemma 7.2. *Let $a_k \in A_k$, $k \in \{1, 2\}$, and $i, j \in [M]$. If $i \equiv j \pmod 2$ and $\mathsf{sat}(a_1, C_i) \vee \mathsf{sat}(a_2, C_j) = \mathsf{T}$ then $\|CG(a_1, i) - CG(a_2, j)\| \leqslant 1$. Otherwise $\|CG(a_1, i) - CG(a_2, j)\| \geqslant 1 + 2\varepsilon$.*

Assignment gadgets This gadget consists of clause gadgets and encodes the set of satisfied clauses for an assignment. We set

$$r_1 := (-\tfrac{1}{3}, \tfrac{1}{2}), \quad r_2 := (-\tfrac{1}{3}, -\tfrac{1}{2}).$$

[1] In later sections we will replace V_1, V_2 by different partitionings and A_1, A_2 by subsets of all assignments. The lemmas in this section are proven in a generality that allows this extension.



(a) Clause gadgets. **(b)** Assignment gadgets. **(c)** Final curves P_1 (dotted) and P_2 (solid).

Figure 7.1: Construction of clause gadgets, assignment gadgets, and the final curves.

The assignment gadget for any $a_k \in A_k$ consists the starting point r_k followed by all clause gadgets of a_k,

$$AG(a_k) := r_k \circ \bigcirc_{i\in[M]} CG(a_k, i),$$

(recall the definition of $\circ$ in Section 7.1). Figure 7.1b shows an assignment gadget on $M = 2$ clauses at the top and an assignment gadget on $M = 4$ clauses at the bottom. The arrows indicate the order in which the segments are traversed.

Lemma 7.3. *Let $a_k \in A_k$, $k \in \{1, 2\}$. If (a_1, a_2) is a satisfying assignment of φ then $d_{\mathrm{dF}}(AG(a_1), AG(a_2)) \leqslant 1$. If (a_1, a_2) is not satisfying then $d_{\mathrm{dF}}(AG(a_1), AG(a_2)) > 1+\varepsilon$, and we even have $d_{\mathrm{dF}}(AG(a_1)\circ\pi_1, AG(a_2)\circ\pi_2) > 1+\varepsilon$ for any curves π_1, π_2.*

Proof. If (a_1, a_2) is satisfying then the parallel traversal

$$(r_1, r_2), (CG(a_1, 1), CG(a_2, 1)), \ldots, (CG(a_1, M), CG(a_2, M))$$

has width 1 by Lemma 7.2.

Assume for the sake of contradiction that (a_1, a_2) is not satisfying but there is a traversal of $(AG(a_1)\circ\pi_1, AG(a_2)\circ\pi_2)$ with width $1+\varepsilon$. Observe that $\|r_1 - r_2\| = 1$ and $\|r_k - c^i_{3-k,x}\| \geqslant 1 + 2\varepsilon$ for any $k \in \{1, 2\}, i \in \{0, 1\}, x \in \{\mathsf{T}, \mathsf{F}\}$. Thus, the traversal has to start at positions (r_1, r_2) and then step to positions $(CG(a_1, 1), CG(a_2, 1))$, as advancing in only one of the curves leaves us in distance larger than $1 + \varepsilon$. Inductively and using Lemma 7.2, the same argument shows that in the i-th step we are at positions $(CG(a_1, i), CG(a_2, i))$ for any $i \in [M]$. Since there is an unsatisfied clause C_i, so that $\|CG(a_1, i) - CG(a_2, i)\| \geqslant 1 + 2\varepsilon$ by Lemma 7.2, we obtain a contradiction. □

Construction of the curves The curve P_k will consist of all assignment gadgets for assignments A_k, $k \in \{1, 2\}$, plus some additional points. The additional points

implement an OR-gadget over the assignment gadgets, by enforcing that any traversal of (P_1, P_2) with width $1 + \varepsilon$ has to traverse two assignment gadgets in parallel, and traversing one pair of assignment gadgets in parallel suffices.

We define the following control points,

$$s_1 := (-\tfrac{1}{3}, \tfrac{1}{5}), \quad t_1 := (\tfrac{1}{3}, \tfrac{1}{5}),$$
$$s_2 := (-\tfrac{1}{3}, 0), \quad t_2 := (\tfrac{1}{3}, 0), \quad s_2^* := (-\tfrac{1}{3}, -\tfrac{4}{5}), \quad t_2^* := (\tfrac{1}{3}, -\tfrac{4}{5}).$$

Finally, we set

$$P_1 := \bigcirc_{a_1 \in A_1} \big(s_1 \circ AG(a_1) \circ t_1\big),$$
$$P_2 := s_2 \circ s_2^* \circ \Big(\bigcirc_{a_2 \in A_2} AG(a_2)\Big) \circ t_2^* \circ t_2.$$

Figure 7.1c shows these final curves in an example with $M = 2$ clauses and (unrealistically) only two assignments.

Let Q_k be the points that may appear in P_k, i.e., $Q_1 = \{s_1, t_1, r_1, c^0_{1,\mathsf{F}}, c^0_{1,\mathsf{T}}, c^1_{1,\mathsf{F}}, c^1_{1,\mathsf{T}}\}$ and $Q_2 = \{s_2, t_2, r_2, s_2^*, t_2^*, c^0_{2,\mathsf{F}}, c^0_{2,\mathsf{T}}, c^1_{2,\mathsf{F}}, c^1_{2,\mathsf{T}}\}$. Explicitly calculating all pairwise distances of all points, we obtain the following lemma.

Lemma 7.4. *No pair $(q_1, q_2) \in Q_1 \times Q_2$ has $\|q_1 - q_2\| \in (1, 1 + \varepsilon]$. Moreover, the set $\{(q_1, q_2) \in Q_1 \times Q_2 \mid \|q_1 - q_2\| \leqslant 1\}$ consists of the following pairs:*

$(q, s_2), (q, t_2)$ *for any* $q \in Q_1$,
(s_1, q) *for any* $q \in Q_2 \setminus \{t_2^*\}$,
(t_1, q) *for any* $q \in Q_2 \setminus \{s_2^*\}$,
(r_1, r_2),
$(c^i_{1,x}, c^i_{2,y})$ *for* $x \vee y = \mathsf{T}$ *where* $i \in \{0, 1\}, x, y \in \{\mathsf{T}, \mathsf{F}\}$.

Correctness We show that if φ is satisfiable then $d_{\mathrm{dF}}(P_1, P_2) \leqslant 1$, while otherwise $d_{\mathrm{dF}}(P_1, P_2) > 1 + \varepsilon$.

Lemma 7.5. *If $d_{\mathrm{dF}}(P_1, P_2) \leqslant 1 + \varepsilon$ then $A_1 \times A_2$ contains a satisfying assignment.*

Proof. By Lemma 7.4 any traversal with width $1 + \varepsilon$ also has width 1. Consider any traversal of (P_1, P_2) with width 1. Consider any time step T at which we are at position s_2^* in P_2. The only point in P_1 that is within distance 1 of s_2^* is s_1, say we are at the copy of s_1 that comes right before assignment gadget $AG(a_1)$, $a_1 \in A_1$. Following time step T, we have to start traversing $AG(a_1)$, so consider the first time step T' where we are at the point r_1 in $AG(a_1)$. The only points in P_2 within distance 1 of r_1 are s_2, t_2, and r_2. Note that we already passed s_2^* in P_2 by time T, so we cannot be in s_2 at time T'. Moreover, in between T and T' we are only at s_1 and r_1 in P_1, which have distance larger than 1 to t_2^*. Thus, we cannot pass t_2^*, and we cannot be at t_2 at time T'. Hence, we are at r_2, say at the copy of r_2 in assignment gadget $AG(a_2)$ for some $a_2 \in A_2$. The yet untraversed remainder of P_k is of the form $AG(a_k) \circ \pi_k$ for $k \in \{1, 2\}$. Since our traversal of (P_1, P_2) has width 1, we obtain $d_{\mathrm{dF}}(AG(a_1) \circ \pi_1, AG(a_2) \circ \pi_2) \leqslant 1$. By Lemma 7.3, (a_1, a_2) forms a satisfying assignment of φ. □

Lemma 7.6. *If $A_1 \times A_2$ contains a satisfying assignment then $d_{\mathrm{dF}}(P_1, P_2) \leqslant 1$.*

Proof. Let $(a_1, a_2) \in A_1 \times A_2$ be a satisfying assignment of φ. We describe a traversal through P_1, P_2 with width 1. We start at $s_2 \in P_2$ and the first point of P_1. We stay at s_2 and follow P_1 until we arrive at the copy of s_1 that comes right before $AG(a_1)$ (note that s_2 has distance 1 to any point in P_1). Then we stay at s_1 and follow P_2 until we arrive at the copy of r_2 in $AG(a_2)$ (note that the only point that is too far away from s_1 is t_2^*, but this point comes after all assignment gadgets in P_2). In the next step we go to positions (r_1, r_2) (in $AG(a_1), AG(a_2)$). Then we follow the clause gadgets $(CG(a_1, i), CG(a_2, i))$ in parallel, always staying within distance 1 by Lemma 7.2. In the next step we stay at $CG(a_2, M)$ and go to t_1 in P_1 (which has distance 1 to any point in P_2 except for s_2^*, which we will never encounter again). We stay at t_1 in P_1 and follow P_2 completely until we arrive at its endpoint t_2. Since t_2 has distance 1 to any point in P_1, we can now stay at t_2 in P_2 and follow P_1 to its end. □

Proof of Theorem 6.1, discrete case Note that we have

$$n = \max\{|P_1|, |P_2|\} = \mathcal{O}(M) \cdot \max\{|A_1|, |A_2|\} = \mathcal{O}(M \cdot 2^{N/2}).$$

Moreover, the instance (P_1, P_2) can be constructed in time $\mathcal{O}(NM2^{N/2})$. Any $(1+\varepsilon)$-approximation can decide whether $d_{\mathrm{dF}}(P_1, P_2) \leqslant 1$ or $d_{\mathrm{dF}}(P_1, P_2) > 1 + \varepsilon$, which by Lemmas 7.5 and 7.6 yields an algorithm that decides whether φ is satisfiable. If such an algorithm runs in time $\mathcal{O}(n^{2-\delta})$ for any small $\delta > 0$, then the resulting CNF-SAT algorithm runs in time $\mathcal{O}(M^2 2^{(1-\delta/2)N})$, contradicting SETH'.

7.2.2. Continuous Case

The construction from the last section also works for the continuous Fréchet distance. However, for unsatisfiable formulas it becomes tedious to argue that continuous traversals are not much better than discrete traversals. For instance, we have to argue that we cannot stay at a fixed point between the clause gadgets $c^0_{1,\mathsf{T}}$ and $c^1_{1,\mathsf{T}}$ while traversing more than one clause gadget in P_2.

We adapt the proof from the last section on the same curves P_1, P_2 to work for the continuous Fréchet distance. To this end, we have to reprove Lemmas 7.5 and 7.6. We will make use of the following property. Here, we set $\mathrm{sym}(CG(a_1, i)) := CG(a_2, i)$ and $\mathrm{sym}(r_1) := r_2$ and interpolate linearly between them to obtain a symmetric point in $AG(a_2)$ for every point in $AG(a_1)$ (for any fixed $a_1 \in A_1$, $a_2 \in A_2$). We also set $\mathrm{sym}(\mathrm{sym}(p_1)) := p_1$, to obtain a symmetric point in $AG(a_1)$ for every point in $AG(a_2)$.

Lemma 7.7. *Consider any points p_k in $AG(a_k)$, $k \in \{1, 2\}$, with $\|p_1 - p_2\| \leqslant 1 + \varepsilon$. Then we have $\|p_2 - \mathrm{sym}(p_1)\| \leqslant \frac{1}{9}$ and $\|\mathrm{sym}(p_2) - p_1\| \leqslant \frac{1}{9}$.*

Proof. Let $p_k = (x_k, y_k)$ and note that we have $|y_1 - y_2| \geqslant 1 - 2\varepsilon$. Thus, if $|x_1 - x_2| > \frac{1}{9} - 2\varepsilon$ then we have (recall that $\varepsilon = 1/1000$)

$$\|p_1 - p_2\| > \sqrt{(\tfrac{1}{9} - 2\varepsilon)^2 + (1 - 2\varepsilon)^2} > 1 + \varepsilon,$$

a contradiction. Since $\mathrm{sym}(p_1) = (x_1, y_1')$ with $|y_1' - y_2| \leqslant 2\varepsilon$, we obtain

$$\|p_2 - \mathrm{sym}(p_1)\| \leqslant \sqrt{(\tfrac{1}{9} - 2\varepsilon)^2 + (2\varepsilon)^2} \leqslant \tfrac{1}{9}.$$

and the same bound holds for $\|\mathrm{sym}(p_2) - p_1\|$. □

Lemma 7.8. (Analogue of Lemma 7.5) *If $d_{\mathrm{F}}(P_1, P_2) \leqslant 1 + \varepsilon = 1.001$ then $A_1 \times A_2$ contains a satisfying assignment.*

Proof. In this proof, we say that two points $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$ have *y-distance D* if $|y_1 - y_2| \leqslant D$.

Consider any traversal of (P_1, P_2) with width $1 + \varepsilon$. Consider any time step T where we are at position s_2^* in P_2. The only points in P_1 that are within distance $1 + \varepsilon$ of s_2^* are within distance $1/20$ and y-distance ε of s_1 (since no point in P_1 has lower y-value than s_1 and $\sqrt{1 + (1/20)^2} > 1 + \varepsilon$). Say we are near the copy of s_1 that comes right before assignment gadget $AG(a_1)$, $a_1 \in A_1$. Following time step T, we have to start traversing $AG(a_1)$, so consider the first time step T' where we are at the point r_1 in $AG(a_1)$. The only points in P_2 within distance $1 + \varepsilon$ of r_1 are near s_2, t_2, or r_2. Note that we already passed s_2^* in P_2 by time T, so we cannot be near s_2 at time T'. Moreover, in between T and T' we are always near s_1 or between s_1 and r_1 in P_1, so we are always above and to the left of $s_1 + (1/20, 0)$, which has distance larger than $1 + \varepsilon$ to t_2^*. Thus, we cannot pass t_2^*, and we cannot be near t_2 at time T'. Hence, we are near r_2, more precisely, we are in distance $1/20$ and y-distance ε of r_2 (this is the same situation as for s_1 and s_2^*). After that, the traversal has to further traverse $AG(a_1)$ and/or $AG(a_2)$. Consider the first time step at which we are at $CG(a_1, 1)$ or $CG(a_2, 1)$, say we reach $CG(a_1, 1)$ first. By Lemma 7.7, we are within distance $1/9$ of $CG(a_2, 1)$. Since we were near r_2 at time T', we now passed r_2, and since we did not pass $CG(a_2, 1)$ yet, we are even within distance $1/9$ of $CG(a_2, 1)$ *along the curve P_2*. This proves the induction base of the following inductive claim.

Claim 7.9. *Let T_i be the first step in time at which the traversal is at $CG(a_1, i)$ or $CG(a_2, i)$, $i \in [M]$. At time T_i the traversal is within distance 1/9 of $CG(a_k, i)$ along the curve P_k for both $k \in \{1, 2\}$.*

Proof. Note that at all times T_i (and in between) Lemma 7.7 is applicable, so we clearly are within distance $1/9$ of $CG(a_k, i + 1)$ at time T_{i+1} for any $i \in [M]$, $k \in \{1, 2\}$. Since $\|CG(a_k, i) - CG(a_k, i + 1)\| \geqslant 1/3$, points within distance $1/9$ of $CG(a_k, i)$ are not within distance $1/9$ of $CG(a_k, i + 1)$. Hence, if we are within distance $1/9$ of $CG(a_k, i)$ along P_k for both $k \in \{1, 2\}$ at time T_i, then at time T_{i+1} we passed $CG(a_k, i)$ and did not pass $CG(a_k, i + 1)$ yet (by definition of T_{i+1}), so that we are within distance $1/9$ of $CG(a_k, i + 1)$ along P_k for both $k \in \{1, 2\}$. □

Finally, we show that the above claim implies that (a_1, a_2) is a satisfying assignment. Assume for the sake of contradiction that some clause C_i is not satisfied by both a_1 and a_2. Say at time T_i we are at $CG(a_1, i)$ (if we are at $CG(a_2, i)$ instead, then a symmetric argument works). At the same time we are at some point p in $AG(a_2)$. By the above claim, p is within distance $1/9$ of $CG(a_2, i)$ along P_2. Note that p lies on any of the line segments $c^0_{2,\mathsf{T}} \circ c^1_{2,\mathsf{F}}$, $c^0_{2,\mathsf{F}} \circ c^1_{2,\mathsf{T}}$, $c^0_{2,\mathsf{F}} \circ c^1_{2,\mathsf{F}}$, or $r_2 \circ c^0_{2,\mathsf{F}}$, since $\mathsf{sat}(a_2, C_i) = \mathsf{F}$. In any case, the current distance $\|p - CG(a_1, i)\|$ is at least the distance from the point $c^0_{1,\mathsf{F}}$ to the line through $c^0_{2,\mathsf{F}}$ and $c^1_{2,\mathsf{T}}$. We compute this distance as

$$\frac{\frac{1}{3}(1+2\varepsilon)}{\sqrt{(\frac{1}{3})^2 + (2\varepsilon)^2}} > 1 + \varepsilon,$$

which contradicts the traversal having width $1 + \varepsilon$. □

Lemma 7.10. (Analogue of Lemma 7.6) *If $A_1 \times A_2$ contains a satisfying assignment then $d_{\mathrm{F}}(P_1, P_2) \leqslant 1$.*

Proof. Follows from Lemma 7.6 and the general inequality $d_{\mathrm{F}}(P_1, P_2) \leqslant d_{\mathrm{dF}}(P_1, P_2)$. □

7.2.3. Generalization to Imbalanced Numbers of Vertices

Assume that the input curves P_1, P_2 have different numbers of vertices $n = |P_1|$, $m = |P_2|$ with $n \geqslant m$. We show that there is no $\mathcal{O}((nm)^{1-\delta})$ algorithm for the Fréchet distance for any $\delta > 0$, even for any polynomial restriction of $1 \leqslant m \leqslant n$. More precisely, for any $\delta \leqslant \gamma \leqslant 1-\delta$ we show that there is no $\mathcal{O}((nm)^{1-\delta})$ algorithm for the Fréchet distance restricted to instances with $n^{\gamma-\delta} \leqslant m \leqslant n^{\gamma+\delta}$.

To this end, given a CNF-SAT instance φ we partition its variables $x_1, \ldots, x_N$ into[2] $V'_1 := \{x_1, \ldots, x_\ell\}$ and $V'_2 := \{x_{\ell+1}, \ldots, x_N\}$ and let A'_k be all assignments of V'_k, $k \in \{1, 2\}$. Note that $|A'_1| = 2^{|V'_1|} = 2^\ell$ and $|A'_2| = 2^{N-\ell}$. Now we use the same construction as in Section 7.2.1 but replace V_k by V'_k and A_k by A'_k. Again we obtain that any 1.001-approximation for the Fréchet distance of the constructed curves P_1, P_2 decides satisfiability of φ. Observe that the constructed curves contain a number of points of

$$n = |P_1| = \Theta(M \cdot |A'_1|), \quad m = |P_2| = \Theta(M \cdot |A'_2|).$$

Hence, any 1.001-approximation with running time $\mathcal{O}((nm)^{1-\delta})$ for any small $\delta > 0$ for the Fréchet distance yields an algorithm for CNF-SAT with running time $\mathcal{O}(M^2(2^\ell 2^{N-\ell})^{1-\delta}) = \mathcal{O}(M^2 2^{(1-\delta)N})$, contradicting SETH'.

Finally, we set $\ell := N/(\gamma+1)$ (rounded in any way) so that $|A'_1| = \Theta(2^{N/(\gamma+1)})$ and $|A'_2| = \Theta(2^{N\gamma/(\gamma+1)})$. Using Lemma 7.1 we can assume that $1 \leqslant M \leqslant 2^{\delta N/4}$. Hence, we have

$$\Omega(2^{N/(\gamma+1)}) \leqslant n \leqslant \mathcal{O}(2^{N/(\gamma+1)+\delta N/4}),$$
$$\Omega(2^{N\gamma/(\gamma+1)}) \leqslant m \leqslant \mathcal{O}(2^{N\gamma/(\gamma+1)+\delta N/4}),$$

[2] For the impatient reader: we will set $\ell := N/(\gamma+1)$ (rounded in any way).

which implies $\Omega(n^{\gamma-\delta/2}) \leqslant m \leqslant \mathcal{O}(n^{\gamma+\delta/2})$. For sufficiently large n, we obtain the desired polynomial restriction $n^{\gamma-\delta} \leqslant m \leqslant n^{\gamma+\delta}$. This proves Theorem 6.2.

7.3. Realistic Inputs: c-Packed Curves

7.3.1. Constant Factor Approximations

The curves constructed in Section 7.2.1 are highly packed, since all assignment gadgets lie roughly in the same area. Specifically, they are not $o(n)$-packed. In this section we want to construct c-packed instances and show that there is no 1.001-approximation with running time $\mathcal{O}((cn)^{1-\delta})$ for any $\delta > 0$ for the Fréchet distance unless SETH′ fails, not even restricted to instances with $n^{\gamma-\delta} \leqslant c \leqslant n^{\gamma+\delta}$ for any $\delta \leqslant \gamma \leqslant 1-\delta$. This proves Theorem 6.3.

To this end, we again consider a CNF-SAT instance φ, partition its variables $x_1, \dots, x_N$ into two sets V_1, V_2 of size $N/2$, and consider the set A_k of all assignments of T and F to the variables in V_k. Now we partition A_k into sets $A_k^1, \dots, A_k^\ell$ of size $\Theta(2^{N/2}/\ell)$, where we fix $1 \leqslant \ell \leqslant 2^{N/2}$ later. Formula φ is satisfiable if and only if for some pair $(j_1, j_2) \in [\ell]^2$ the set $A_1^{j_1} \times A_2^{j_2}$ contains a satisfying assignment. This suggests to use the construction of Section 7.2.1 after replacing A_1 by $A_1^{j_1}$ and A_2 by $A_2^{j_2}$, yielding a pair of curves $(P_1^{j_1 j_2}, P_2^{j_1 j_2})$. Now, φ is satisfiable if and only if $d_{\mathrm{F}}(P_1^{j_1 j_2}, P_2^{j_1 j_2}) \leqslant 1$ for some $(j_1, j_2) \in [\ell]^2$. For the sake of readability, we rename the constructed curves slightly so that we have curves (P_1^j, P_2^j) for $j \in [\ell^2]$.

OR-gadget In the whole section we let $\rho := 1/\sqrt{2}$. We present an OR-construction over the gadgets (P_1^j, P_2^j) that is not too packed, in contrast to the OR-construction over assignment gadgets that we used in Section 7.2.1. We start with two building blocks, where for any $j \in \mathbb{N}$ we set

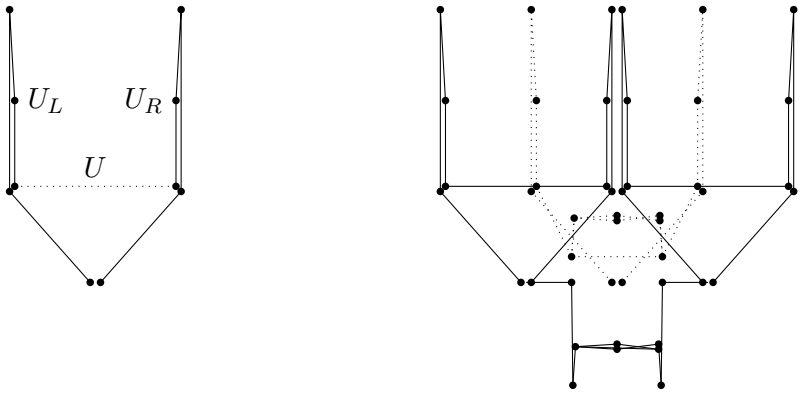
$$U_L(j) := (j\rho, 0) \circ ((j-1)\rho, \rho) \circ ((j-1)\rho, 3\rho) \circ ((j-1)\rho, 2\rho) \circ ((j-1)\rho, \rho),$$
$$U_R(j) := ((j+1)\rho, \rho) \circ ((j+1)\rho, 2\rho) \circ ((j+1)\rho, 3\rho) \circ ((j+1)\rho, \rho) \circ (j\rho, 0),$$

see Figure 7.2a. Moreover, we set $U(j) := U_L(j) \circ U_R(j)$. For a curve π and $z \in \mathbb{R}$ we let $\mathrm{tr}_z(\pi)$ be the curve π translated by z in x-direction. The OR-gadget now consists of the following two curves,

$$R_1 := \bigcirc_{j=1}^{\ell^2} \big(U_L(2j) \circ \mathrm{tr}_{2j\rho}(P_1^j) \circ U_R(2j)\big),$$
$$R_2 := U(1) \circ \bigcirc_{j=1}^{\ell^2} \big(\mathrm{tr}_{2j\rho}(P_2^j) \circ U(2j+1)\big),$$

see Figures 7.2b and 7.3.

We denote by R_1^j the j-th "summand" of R_1, i.e., $R_1^j = U_L(2j) \circ \mathrm{tr}_{2j\rho}(P_1^j) \circ U_R(2j)$. Informally, we will use the term *U-shape* for the sub-curves R_1^j and $U(2j+1)$, since they resemble the letter U. Moreover, we consider "summands" of R_2, namely $R_2^j := U(2j-1) \circ \mathrm{tr}_{2j\rho}(P_2^j) \circ ((2j+1)\rho, 0)$ and $\tilde{R}_2^j := ((2j-1)\rho, 0) \circ \mathrm{tr}_{2j\rho}(P_2^j) \circ U(2j+1)$.



(a) A U-shape showing the building blocks of the OR-gadget.

(b) The OR-gadget consisting of the curves R_1 (dotted) and R_2 (solid) for $\ell^2 = 1$.

Figure 7.2: This figure illustrates the construction of the OR-gadget.

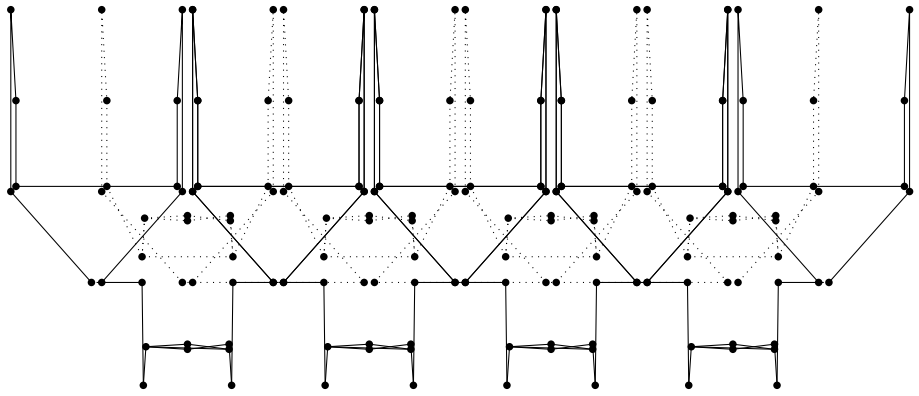


Figure 7.3: The OR-gadget consisting of the curves R_1 (dotted) and R_2 (solid) for $\ell^2 = 4$.

Intuition Considering traversals that stay within distance 1, we can traverse one U-shape in R_1 and one neighboring U-shape in R_2 together. Such traversals can be stitched together to a traversal of any number j of neighboring U-shapes in both curves. So far we can only traverse the same number of U-shapes in both curves, but R_2 has one more U-shape than R_1. We will show that we can traverse two U-shapes in R_2 while traversing only one U-shape in R_1, if these parts contain a satisfying assignment.

In the unsatisfiable case, essentially we show that we cannot traverse two U-shapes in R_2 while traversing only one U-shape in R_1, which implies a contradiction since the number of U-shapes in R_2 is larger than in R_1. We make this intuition formal in the remainder of this section.

Analysis In order to be able to replace the curves P_1^j, P_2^j constructed above by other curves in the next section, we analyze the OR-gadget in a rather general way. To this end, we first specify a set of properties and show that the curves P_1^j, P_2^j constructed above satisfy these properties. Then we analyze the OR-gadget using only these properties of P_1^j, P_2^j.

Property 7.11. *(i) If φ is satisfiable then for some $j \in [\ell^2]$ we have $d_{\mathrm{dF}}(P_1^j, P_2^j) \leqslant 1$.*

(ii) If φ is not satisfiable then for all $j \in [\ell^2]$ and curves $\sigma_1, \sigma_2, \pi_1, \pi_2$ such that σ_1 stays to the left and above $(-\rho, \rho)$ and π_1 stays to the right and above (ρ, ρ), we have $d_{\mathrm{F}}(\sigma_1 \circ P_1^j \circ \pi_1, \sigma_2 \circ P_2^j \circ \pi_2) > \beta$, for some $\beta > 1$.

(iii) P_k^j is $\Theta(c)$-packed for some $c \geqslant 1$ for all $j \in [\ell^2]$, $k \in \{1, 2\}$.

(iv) $(0, \rho)$ is within distance 1 of any point in P_1^j for all $j \in [\ell^2]$.

(v) $(0, 0)$ is within distance 1 of any point in P_2^j for all $j \in [\ell^2]$.

Lemma 7.12. *The curves (P_1^j, P_2^j) constructed above satisfy Property 7.11 with $\beta = 1.001$ and $c = \Theta(M \cdot 2^{N/2}/\ell)$. Moreover, we have $|P_k^j| = \Theta(M \cdot 2^{N/2}/\ell)$ for all $j \in [\ell^2]$, $k \in \{1, 2\}$.*

Proof. Property 7.11.(i) follows from Lemma 7.6, since at least one pair $(A_1^{j_1}, A_2^{j_2})$ contains a satisfying assignment. Properties (iv) and (v) can be verified by considering all points in the construction in Section 7.2.1.

Observe that $|P_k^j| = \Theta(M \cdot 2^{N/2}/\ell)$, since P_k^j consists of $|A_k^j| = \Theta(2^{N/2}/\ell)$ assignment gadgets of size $\Theta(M)$. The upper bound of (iii) follows since any polygonal curve with at most m segments is m-packed. The lower bound of (iii) follows from P_k^j being contained in a ball of radius 1 (by (iv) and (v)) and every segment of P_k^j having constant length.

For (ii), note that from any traversal of $(\sigma_1 \circ P_1^j \circ \pi_1, \sigma_2 \circ P_2^j \circ \pi_2)$ with width 1.001 one can extract a traversal of (P_1^j, P_2^j) with width 1.001, by mapping any point in σ_k to the starting point s_k of P_k^j and any point in π_k to the endpoint t_k of P_k^j, $k \in \{1, 2\}$. This does not increase the width, since (1) s_2 and t_2 are within distance 1 to all points in P_1^j, and (2) s_1 has smaller distance to any point in P_2^j than any point in σ_1 has, since σ_1 stays above and to the left of s_1 while all points of P_1^j lie below

and to the right of s_1. A similar statement holds for t_1 and π_1. Property (ii) now follows from Lemma 7.8. □

In the following lemma we analyze the OR-gadget.

Lemma 7.13. *For any curves (P_1^j, P_2^j), $j \in [\ell^2]$, that satisfy Property 7.11, the OR-gadget (R_1, R_2) satisfies:*

(i) $|R_k| = \Theta\big(\sum_{j=1}^{\ell^2} |P_k^j|\big)$ *for* $k \in \{1, 2\}$.

(ii) R_1 *and* R_2 *are* $\Theta(c)$*-packed,*

(iii) If φ *is satisfiable then* $d_{\mathrm{F}}(R_1, R_2) \leqslant d_{\mathrm{dF}}(R_1, R_2) \leqslant 1$,

(iv) If φ *is not satisfiable then* $d_{\mathrm{dF}}(R_1, R_2) \geqslant d_{\mathrm{F}}(R_1, R_2) > \min\{\beta, 1.2\}$.

Proof. (i) Precisely, we have $|R_k| = \sum_{j=1}^{\ell^2}(|P_k^j| + 10) + 10(k-1)$ for $k \in \{1, 2\}$.

(ii) Let $k \in \{1, 2\}$ and consider any ball $B = B(q, r)$. If $r \leqslant 1$ then B hits $\mathcal{O}(1)$ of the curves P_k^j. Since these curves are c-packed, their contribution to the total length of R_k in B is at most $\mathcal{O}(cr)$. Moreover, B hits $\mathcal{O}(1)$ segments of U or U_L, U_R, and the connecting segments to P_k^j. Each of these segments has length at most $2r$ inside B. This yields a total length of R_k in B of $\mathcal{O}((c+1)r)$.

Similarly, if $r > 1$ then B hits $\mathcal{O}(r)$ of the curves P_k^j. Note that the total length of P_k^j is at most c, since the curve is c-packed and contained in a ball of radius 1 around $(0, 0)$ or $(0, \rho)$ by Property 7.11. Hence, the total length of of the curves P_k^j in B is $\mathcal{O}(cr)$. Moreover, B hits $\mathcal{O}(r)$ segments of U, U_L, U_R, and the connectors to P_k^j, each of constant length. This yields a total length of R_k in B of $\mathcal{O}((c+1)r)$.

In total, the curve R_k is $\mathcal{O}(c+1)$-packed. As $c \geqslant 1$, it is also $\mathcal{O}(c)$-packed. Since for some $\alpha > 0$ the curve P_k^j is not αc-packed, also R_k is not αc-packed, so R_k is even $\Theta(c)$-packed.

(iii) Note that $d_{\mathrm{F}}(R_1, R_2) \leqslant d_{\mathrm{dF}}(R_1, R_2)$ holds in general, so we only have to show that if φ is satisfiable then $d_{\mathrm{dF}}(R_1, R_2) \leqslant 1$. First we show that we can traverse one U-shape in R_1 and one neighboring U-shape in R_2 together.

Claim 7.14. *For any $j \in [\ell^2]$, we have $d_{\mathrm{dF}}(R_1^j, U(2j-1)) \leqslant 1$ and $d_{\mathrm{dF}}(R_1^j, U(2j+1)) \leqslant 1$.*

Proof. We only show the first inequality, the second is similar. We start by traversing $U_L(2j)$ and the left half of $U(2j-1)$ in parallel, being at the i-th point of $U_L(2j)$ and $U(2j-1)$ at the same time. At any point in time we are within distance ρ. Now we step to $(2j\rho, \rho)$ in $U(2j-1)$. We stay there while traversing $\mathrm{tr}_{2j\rho}(P_1^j)$ in R_1^j, staying within distance 1 by Property 7.11.(iv). Finally, we traverse $U_R(2j)$ and the second half of $U(2j-1)$ in parallel, where again the largest encountered distance is ρ. □

We can stitch these traversals together so that we traverse any number j of neighboring U-shapes in both curves together, because the parts in between the U-shapes are near to a single point, as shown by the following claim. Note that $(2j\rho, 0) \circ ((2j+2)\rho, 0)$ is the connecting segment in R_1 between $U_R(2j)$ and $U_L(2j+2)$,

while $((2j-1)\rho, 0) \circ \mathrm{tr}_{2j\rho}(P_2^j) \circ ((2j+1)\rho, 0)$ is the part in R_2 between $U(2j-1)$ and $U(2j+1)$.

Claim 7.15. *For any $j \in [\ell^2]$,*

$$d_{\mathrm{dF}}((2j\rho, 0) \circ ((2j+2)\rho, 0), ((2j+1)\rho, 0)) \leqslant 1,$$
$$d_{\mathrm{dF}}((2j\rho, 0), ((2j-1)\rho, 0) \circ \mathrm{tr}_{2j\rho}(P_2^j) \circ ((2j+1)\rho, 0)) \leqslant 1.$$

Proof. The first claim is immediate. The second follows from Property 7.11.(v). □

Thus, we can stitch together traversals of U-shapes in both curves. However, so far we can only traverse the same number of U-shapes in both curves, but R_2 has one more U-shape than R_1. Consider $J \in [\ell^2]$ with $d_{\mathrm{dF}}(P_1^J, P_2^J) \leqslant 1$, which exists since φ is satisfiable, see Property 7.11.(i). Consider the two sub-curves (see also Figure 7.2b)

$$R_1' := R_1^J = U_L(2J) \circ \mathrm{tr}_{2J\rho}(P_1^J) \circ U_R(2J),$$
$$R_2' := U(2J-1) \circ \mathrm{tr}_{2J\rho}(P_2^J) \circ U(2J+1).$$

We show that $d_{\mathrm{dF}}(R_1', R_2') \leqslant 1$, i.e., we can traverse two U-shapes in R_2 while traversing only one U-shape in R_1, using $d_{\mathrm{dF}}(P_1^J, P_2^J) \leqslant 1$. Adding simple traversals of U-shapes before and after (R_1', R_2'), we obtain a traversal of (R_1, R_2) with width 1, proving $d_{\mathrm{dF}}(R_1, R_2) \leqslant 1$. It is left to show the following claim.

Claim 7.16. $d_{\mathrm{dF}}(R_1', R_2') \leqslant 1$.

Proof. We traverse $U_L(2J)$ and $U(2J-1)$ in parallel until we are at point $((2J-1)\rho, 2\rho)$ in $U_L(2J)$. We stay in this point and follow $U(2J-1)$ until its second-to-last point. In the next step we can finish traversing $U_L(2J)$ and $U(2J-1)$. In the next step we go to the first positions of (the translated) P_1^J and P_2^J. We follow any traversal of (P_1^J, P_2^J) with width 1. Finally, we use a traversal symmetric to the one of $(U_L(2J), U(2J-1))$ to traverse $(U_R(2J), U(2J+1))$. □

(iv) Note that the inequality $d_{\mathrm{dF}}(R_1, R_2) \geqslant d_{\mathrm{F}}(R_1, R_2)$ holds in general, so we only have to show that if φ is not satisfiable then $d_{\mathrm{F}}(R_1, R_2) > \min\{\beta, 1.2\}$. Assume for the sake of contradiction that there is a traversal of (R_1, R_2) with width $\min\{\beta, 1.2\}$. Essentially we show that it cannot traverse 2 U-shapes in R_2 while traversing only one U-shape in R_1, which implies a contradiction since the number of U-shapes in R_2 is larger than in R_1.

Let Y_ρ be the line $\{(x, y) \in \mathbb{R}^2 \mid y = \rho\}$. We inductively prove the following claims.

Claim 7.17. *(i) For any $0 \leqslant j \leqslant \ell^2$, when the traversal is in R_2 at the left highest point $(2j\rho, 3\rho)$ of $U(2j+1)$, then in R_1 we fully traversed R_1^j and are above the line Y_ρ.*

(ii) For any $1 \leqslant j \leqslant \ell^2$, when the traversal is in R_1 at the right highest point $((2j+1)\rho, 3\rho)$ of R_1^j, then in R_2 it is in $U(2j-1)$.

Note that claim (i) for $j = \ell^2$ yields the desired contradiction, since after traversing $R_1^{\ell^2}$ the curve R_1 has ended (at the point $(2\ell^2\rho, 0)$), so that we cannot go above the line Y_ρ anymore.

Proof. (i) Note that we have to be above the line Y_ρ because all points below Y_ρ have distance at least $2\rho > 1.2$ to the point $(2j\rho, 3\rho)$. For $j = 0$, claim (i) holds immediately, since there is no sub-curve R_1^0 (so this part of the statement disappears). In general, claim (i) for any $1 \leqslant j \leqslant \ell^2$ follows from claim (ii) for j: When we are at $z_1 := ((2j+1)\rho, 3\rho)$ in R_1^j, we are still in $U(2j-1)$. Once we reach the endpoint $z_2 := ((2j-1)\rho, 0)$ of $U(2j-1)$, in R_1 we are at a point p_1 below the line Y_ρ, since all points in R_2 that follow z_1 and lie above Y_ρ have distance more than $2\rho > 1.2$ to z_2. Now we follow R_2 until we reach $p_2 := (2j\rho, 3\rho)$ in $U(2j+1)$. At this point we have to be above the line Y_ρ in R_1, but all points in R_1^j following p_1 lie below Y_ρ. Thus, at this point we have fully traversed R_1^j (and have to be in R_1^{j+1}).

(ii) This claim for any $1 \leqslant j \leqslant \ell^2$ follows from claim (i) for $j-1$. Assume for the sake of contradiction that claim (ii) for some j does not hold. Consider the sub-curve R_1' of R_1^j between (the first occurrence of) $((2j-1)\rho, \rho)$ and $((2j+1)\rho, 3\rho)$. Let R_2' be the sub-curve of R_2 that the traversal traverses together with R_1'. Since (R_1', R_2') forms a sub-traversal of the traversal of (R_1, R_2), which has width $\min\{\beta, 1.2\}$, we have $d_{\mathrm{F}}(R_1', R_2') \leqslant \min\{\beta, 1.2\}$ (*). By claim (i) for $j-1$, the starting point of R_2' lies before $\mathrm{tr}_{2j\rho}(P_2^j)$ along R_2, since we reach $((2j-2)\rho, 3\rho)$ in $U(2j-1)$ only after being in the starting point of R_1'. Moreover, the endpoint of R_2' lies after $\mathrm{tr}_{2j\rho}(P_2^j)$ along R_2. Indeed, while being at the endpoint $((2j+1)\rho, 3\rho)$ of R_1', we cannot be in $U(2j-1)$ since we assumed that claim (ii) is wrong for j. We can also not be in $\mathrm{tr}_{2j\rho}(P_2^j)$, since by Property 7.11.5 all points in this curve lie in a ball of radius 1 around $(2j\rho, 0)$, so their distance to $((2j+1)\rho, 3\rho)$ is at least $\|((2j+1)\rho, 3\rho) - (2j\rho, 0)\| - 1 = \sqrt{5} - 1 > 1.2$. Hence, we already passed $\mathrm{tr}_{2j\rho}(P_2^j)$, and R_2' is of the form $\sigma_2 \circ \mathrm{tr}_{2j\rho}(P_2^j) \circ \pi_2$ for any curves σ_2, π_2. Note that R_1' is of the form $\sigma_1 \circ \mathrm{tr}_{2j\rho}(P_1^j) \circ \pi_1$ with σ_1 staying above and to the left of $((2j-1)\rho, \rho)$ and π_1 staying above and to the right of $((2j+1)\rho, \rho)$. Thus, after translation Property 7.11.(ii) applies, proving $d_{\mathrm{F}}(R_1', R_2') > \beta$, a contradiction to (*). □

Proof of Theorem 6.3 Finally, we use the OR-gadget (Lemma 7.13) together with the curves P_1^j, P_2^j we obtained from Section 7.2.1 (Lemma 7.12) to prove a running time bound for c-packed curves: Any 1.001-approximation for the (discrete or continuous) Fréchet distance of (R_1, R_2) decides satisfiability of φ. Note that R_1 and R_2 are c-packed with

$$c = \Theta(M \cdot 2^{N/2}/\ell), \quad n = \max\{|R_1|, |R_2|\} = \Theta(\ell^2 M \cdot 2^{N/2}/\ell).$$

Thus, any $\mathcal{O}((cn)^{1-\delta})$ algorithm for the Fréchet distance implies a $\mathcal{O}(M^2 2^{(1-\delta)N})$ algorithm for CNF-SAT, contradicting SETH'. Moreover, using Lemma 7.1 we can assume that $1 \leqslant M \leqslant 2^{\delta N/4}$. Setting $\ell := \Theta(2^{\frac{1-\gamma}{1+\gamma} N/2})$ for any $0 \leqslant \gamma \leqslant 1$ we obtain

$$\Omega(2^{\frac{2}{1+\gamma} N/2}) \leqslant n \leqslant \mathcal{O}(2^{(\frac{2}{1+\gamma} + \delta/2)N/2}), \qquad \Omega(2^{\frac{2\gamma}{1+\gamma} N/2}) \leqslant c \leqslant \mathcal{O}(2^{(\frac{2\gamma}{1+\gamma} + \delta/2)N/2}).$$

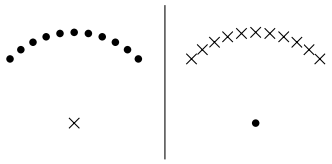


Figure 7.4: Point sets Z_1, Z_2 in $\mathbb{R}^4$ with distance 1 between any pair of points $(z_1, z_2) \in Z_1 \times Z_2$. The left picture shows the projection onto the first two dimensions and the right picture shows the projection onto the last two dimensions. Here, Z_1 (circles) is placed along a quarter-circle in the $(1,2)$-plane and Z_2 (crosses) is placed along a quarter-circle in the $(3,4)$-plane.

From this it follows that $\Omega(n^{\gamma-\delta/2}) \leqslant c \leqslant \mathcal{O}(n^{\gamma+\delta/2})$, which implies the desired polynomial restriction $n^{\gamma-\delta} \leqslant c \leqslant n^{\gamma+\delta}$ for sufficiently large n.

7.3.2. Approximation Schemes

In this section, we consider the dependence on ε of the running time of a $(1+\varepsilon)$-approximation for the Fréchet distance on c-packed curves. We show that in $\mathbb{R}^d$ with $d \geqslant 5$ there is no such algorithm with running time $\mathcal{O}(\min\{cn/\sqrt{\varepsilon}, n^2\}^{1-\delta})$ for any $\delta > 0$ unless SETH' fails (Theorem 6.4). This matches the dependence on ε of the fastest known algorithm up to a polynomial. The result holds for sufficiently small $\varepsilon > 0$ and any polynomial restriction of $1 \leqslant c \leqslant n$ and $\varepsilon \leqslant 1$.

We will reuse the OR-gadget from the last section, embedded into the first two dimensions of $\mathbb{R}^5$. Specifically, we will reuse Lemma 7.13. However, we adapt the curves P_1^j, P_2^j, essentially by embedding the same set of points in a different way. In this new embedding we make use of the fact that in $\mathbb{R}^4$ there are point sets Z_1, Z_2 of arbitrary size such that any pair of points $(z_1, z_2) \in Z_1 \times Z_2$ has distance 1. For an example, see Figure 7.4.

Construction As usual, consider a CNF-SAT instance φ, partition its variables $x_1, \ldots, x_N$ into two sets V_1, V_2 of size $N/2$, and consider any set A_k of assignments of T and F to the variables in V_k. Fix any enumeration $\{a_k^1, \ldots, a_k^{|A_k|}\}$ of A_k. Again set $\rho := 1/\sqrt{2}$. For $h \in [|A_k|]$ and $i \in \{0, \ldots, M+1\}$ let

$$\begin{aligned}
\mathrm{rot}(a_1^h, i) &:= \big(\rho \sin\big(\tfrac{\pi}{4} + \tfrac{\pi}{2}\tfrac{h(M+2)+i}{|A_1|\cdot(M+2)}\big), \rho\cos\big(\tfrac{\pi}{4} + \tfrac{\pi}{2}\tfrac{h(M+2)+i}{|A_1|\cdot(M+2)}\big), 0, 0, 0\big),\\
\mathrm{rot}(a_2^h, i) &:= \big(0, 0, \rho \sin\big(\tfrac{\pi}{4} + \tfrac{\pi}{2}\tfrac{h(M+2)+i}{|A_2|\cdot(M+2)}\big), \rho\cos\big(\tfrac{\pi}{4} + \tfrac{\pi}{2}\tfrac{h(M+2)+i}{|A_2|\cdot(M+2)}\big), 0\big).
\end{aligned}$$

Note that these points are placed along a quarter-circle in the $(1,2)$-plane or $(3,4)$-plane, respectively, as in Figure 7.4. In particular, $\|\mathrm{rot}(a_1^h, i) - \mathrm{rot}(a_2^{h'}, i')\| = 1$ for all h, h', i, i'. Moreover, let e_5 be the vector $(0,0,0,0,\rho)$. For $a_k \in A_k$ and $i \in [M]$ we set

$$CG(a_k, i) := \begin{cases} (1-2\varepsilon)\,\mathrm{rot}(a_k, i) + (i \bmod 2)\cdot 8\sqrt{\varepsilon}\, e_5, & \text{if } \mathsf{sat}(a_k, C_i) = \mathsf{T} \\ (1+\varepsilon)\,\mathrm{rot}(a_k, i) + (i \bmod 2)\cdot 8\sqrt{\varepsilon}\, e_5, & \text{if } \mathsf{sat}(a_k, C_i) = \mathsf{F} \end{cases}$$

Thus, we align the clause gadgets of A_1 roughly along a quarter-circle in the $(1,2)$-plane, and similarly the clause gadgets of A_2 roughly along a quarter-circle in the

$(3,4)$-plane. Moreover, for $a_k \in A_k$ we set

$$\begin{aligned}
r_k(a_k) &:= \operatorname{rot}(a_k, 0) - 8\sqrt{\varepsilon}\, e_5,\\
s_1(a_1) &:= (1-400\varepsilon)\operatorname{rot}(a_1, 0) + 10\sqrt{\varepsilon}\, e_5,\\
t_1(a_1) &:= (1-400\varepsilon)\operatorname{rot}(a_1, M+1) - 10\sqrt{\varepsilon}\, e_5,\\
s_2 = t_2 &:= (0,0,0,0,0),\\
s_2^* &:= (1+9\sqrt{\varepsilon})e_5, \quad t_2^* := -(1+9\sqrt{\varepsilon})e_5.
\end{aligned}$$

We define assignment gadgets and the curves P_1, P_2 as in Section 7.2.1, i.e.,

$$\begin{aligned}
AG(a_k) &:= r_k(a_k) \circ \bigcirc_{i\in[M]} CG(a_k, i),\\
P_1 &:= \bigcirc_{a_1\in A_1} \big(s_1(a_1) \circ AG(a_1) \circ t_1(a_1)\big),\\
P_2 &:= s_2 \circ s_2^* \circ \Big(\bigcirc_{a_2\in A_2} AG(a_2)\Big) \circ t_2^* \circ t_2.
\end{aligned}$$

Analysis Again, we split the considered points into Q_1, Q_2, depending on whether they may appear on P_1 or P_2, i.e., $Q_1 := \{s_1(a_1), t_1(a_1), r_1(a_1), CG(a_1, i) \mid a_1 \in A_1, i \in [M]\}$ and $Q_2 := \{s_2, t_2, s_2^*, t_2^*, r_2(a_2), CG(a_2, i) \mid a_2 \in A_2, i \in [M]\}$. It is easy, but tedious to verify that the constructed points behave as follows.

Lemma 7.18. *The following pairs of points have distance at most 1 for any $a_k \in A_k$:*

$(q, s_2), (q, t_2)$ *for any* $q \in Q_1$,
$(s_1(a_1), q)$ *for any* $q \in Q_2 \setminus \{t_2^*\}$,
$(t_1(a_1), q)$ *for any* $q \in Q_2 \setminus \{s_2^*\}$,
$(r_1(a_1), r_2(a_2))$,
$(CG(a_1, i), CG(a_2, i))$ *if assignment* (a_1, a_2) *satisfies clause* C_i.

Moreover, the following pairs of points have distance more than $1+\varepsilon$ for any $a_k \in A_k$:

(q, s_2^*) *for any* $q \in Q_1 \setminus \{s_1\}$,
(q, t_2^*) *for any* $q \in Q_1 \setminus \{t_1\}$,
$(r_1(a_1), CG(a_2, i))$ *for any* $i \in [M]$,
$(CG(a_1, i), r_2(a_2))$ *for any* $i \in [M]$,
$(CG(a_1, i), CG(a_2, j))$ *for any* $i, j \in [M], i \not\equiv j \bmod 2$,
$(CG(a_1, i), CG(a_2, i))$ *if assignment* (a_1, a_2) *does not satisfy clause* C_i.

Proof. Using that ε is sufficiently small, we only have to compute the largest order term of ε for all distances. E.g., for all $a_k \in A_k$

$$\|s_1(a_1) - r_2(a_2)\| = \sqrt{\rho^2((1-400\varepsilon)^2 + 1 + (18\sqrt{\varepsilon})^2)} = \sqrt{1 - 476\varepsilon + \mathcal{O}(\varepsilon^2)} \leqslant 1.$$

□

Now we use these curves in the OR-gadget from the last section. To this end, again partition the set of all assignments of V_k into sets $A_k^1, \ldots, A_k^\ell$ of size $\Theta(2^{N/2}/\ell)$, where

we fix $1 \leqslant \ell \leqslant 2^{N/2}$ later. Use the above construction of P_1, P_2 after replacing A_1 by $A_1^{j_1}$ and A_2 by $A_2^{j_2}$ for any $j_1, j_2 \in [\ell]$ to obtain curves $P_1^{j_1 j_2}, P_2^{j_1 j_2}$. Slightly rename these curves so that we have curves (P_1^j, P_2^j) for $j \in [\ell^2]$. Then these curves satisfy Property 7.11.

Lemma 7.19. *The curves P_1^j, P_2^j satisfy Property 7.11 with $c = \Theta(1 + \sqrt{\varepsilon}M|A_k|)$ and $\beta = 1 + \varepsilon$. Moreover, $|P_k^j| = \Theta(M2^{N/2}/\ell)$ for any $j \in [\ell^2]$, $k \in \{1, 2\}$.*

Proof. Using Lemma 7.18, we can follow the proof in Section 7.2.1, since everything that we used about P_1, P_2 is captured by this lemma. This proves that if φ is satisfiable then $d_{\mathrm{dF}}(P_1^j, P_2^j) \leqslant 1$ for some $j \in [\ell^2]$, and if φ is not satisfiable then $d_{\mathrm{dF}}(P_1^j, P_2^j) > 1 + \varepsilon$ for all $j \in [\ell^2]$, i.e., Properties 7.11.(i) and (ii) in the discrete case. The same adaptations as in Section 7.2.2 allow to prove correctness in the continuous case, we omit the details.

It is easy to see that all constructed points lie within distance 1 of $(0, 0, 0, 0, 0)$, showing (iv). For (v) we use that we placed the points along the upper quarter-circle, and not the full circle. This way, all points in P_1^j have a distance to $(0, \rho, 0, 0, 0)$ of at most $\|(0, \rho) - (\frac{1}{2}, \frac{1}{2})\| + \mathcal{O}(\sqrt{\varepsilon}) < 1$, for sufficiently small ε.

For (iii) observe that all segments of P_k^j (except for the finitely many segments incident to s_2^*, t_2^*) have length $\Theta(\sqrt{\varepsilon} + 1/(M|A_k^j|))$, $k \in \{1, 2\}$. Moreover, the $\Theta(M|A_k^j|)$ segments of P_k^j are spread along a quarter-circle. Hence, any ball $B(q, r)$ intersects $\mathcal{O}(1 + \min\{1, r\}M|A_k^j|)$ segments of P_k^j. Since each of these segments has length $\mathcal{O}(\min\{r, \sqrt{\varepsilon} + 1/(M|A_k^j|)\})$ in $B(q, r)$, the total length of P_k^j in $B(q, r)$ is $\mathcal{O}(r(1 + \sqrt{\varepsilon}M|A_k^j|))$. Thus, P_k^j is $\mathcal{O}(1 + \sqrt{\varepsilon}M|A_k^j|)$-packed. It is also $\Theta(1 + \sqrt{\varepsilon}M|A_k^j|)$-packed, since all $\Theta(M|A_k^j|)$ segments of length $\Theta(\sqrt{\varepsilon} + 1/(M|A_k^j|))$ lie in a ball of radius 1 around $(0, 0, 0, 0, 0)$ or $(0, \rho, 0, 0, 0)$ by (iv) and (v). Finally, note that $|A_k^j| = 2^{N/2}/\ell$ so that $|P_k^j| = \Theta(M2^{N/2}/\ell)$. □

Proof of Theorem 6.4 The above Lemma 7.19 allows to apply Lemma 7.13, which constructs curves R_1, R_2 such that any $(1+\varepsilon)$-approximation for the Fréchet distance of (R_1, R_2) decides satisfiability of φ. Since R_1 and R_2 are c-packed with

$$c = \Theta(1 + \sqrt{\varepsilon}M2^{N/2}/\ell), \quad n = \max\{|R_1|, |R_2|\} = \Theta(\ell M 2^{N/2}),$$

we obtain that any $(1+\varepsilon)$-approximation for the Fréchet distance with running time $\mathcal{O}((cn/\sqrt{\varepsilon})^{1-\delta})$ yields an algorithm for CNF-SAT with running time $\mathcal{O}(M^2 2^{(1-\delta)N})$, as long as $\ell = \mathcal{O}(\sqrt{\varepsilon}M2^{N/2})$. This contradicts SETH′.

Moreover, using Lemma 7.1 we can assume that $1 \leqslant M \leqslant 2^{\delta N/4}$. Setting $\ell := \Theta(\varepsilon^{\frac{1}{2(1+\gamma)}} 2^{\frac{1-\gamma}{1+\gamma}N/2})$ for any $0 \leqslant \gamma \leqslant 1$, we obtain

$$\begin{aligned}
\varepsilon^{\frac{1}{2(1+\gamma)}} 2^{\frac{2}{1+\gamma}N/2} \leqslant n &\leqslant \varepsilon^{\frac{1}{2(1+\gamma)}} 2^{(\frac{2}{1+\gamma}+\delta/2)N/2}, \\
\varepsilon^{\frac{\gamma}{2(1+\gamma)}} 2^{\frac{2\gamma}{1+\gamma}N/2} \leqslant c &\leqslant \varepsilon^{\frac{\gamma}{2(1+\gamma)}} 2^{(\frac{2\gamma}{1+\gamma}+\delta/2)N/2}.
\end{aligned}$$

From this it follows that $\Omega(n^{\gamma-\delta/2}) \leqslant c \leqslant \mathcal{O}(n^{\gamma+\delta})$, which implies the desired polynomial restriction $n^{\gamma-\delta} \leqslant c \leqslant n^{\gamma+\delta}$ for sufficiently large n. Note that this works

as long as

$$1 \leqslant \ell \leqslant \mathcal{O}(\sqrt{\varepsilon} M 2^{N/2}).$$

Since $\ell = \Theta\big(\big(\sqrt{\varepsilon} n/c\big)^{1/2}\big)$, the first inequality is equivalent to $cn/\sqrt{\varepsilon} \leqslant n^2$, which is a natural condition, since otherwise the exact algorithm for general curves is faster. Plugging in the definition of $\ell = \Theta(\varepsilon^{\frac{1}{2(1+\gamma)}} 2^{\frac{1-\gamma}{1+\gamma} N/2})$, the second inequality becomes $1/\varepsilon \leqslant \big(2^N M^{(1+\gamma)/\gamma}\big)^2$. Since $(1+\gamma)/\gamma \geqslant 2$, $n = \mathcal{O}(\ell M 2^{N/2}) \leqslant \mathcal{O}(M^2 2^N)$, and $c \geqslant 1$, this is implied by the first condition $cn/\sqrt{\varepsilon} \leqslant n^2$. Hence, we may choose any sufficiently small $\varepsilon = \varepsilon(n)$ with $cn/\sqrt{\varepsilon} \leqslant n^2$.

8 Improved Approximation for Realistic Inputs

This chapter is based on [1]. I contributed approximately 50% to all parts of the algorithm and at least 40% to the work of writing the paper.

[1] K. Bringmann and M. Künnemann. "Improved approximation for Fréchet distance on c-packed curves matching conditional lower bounds." Submitted. 2014. arXiv: 1408.1340 [cs.CG].

In this chapter, we present an improved $(1+\varepsilon)$-approximation for the Fréchet distance on c-packed curves, i.e., we prove the following theorem from Section 6.3.

Theorem 6.5. *For any $0 < \varepsilon \leqslant 1$ we can compute a $(1+\varepsilon)$-approximation for the continuous and discrete Fréchet distance on c-packed curves in time $\tilde{\mathcal{O}}(cn/\sqrt{\varepsilon})$.*

This chapter is structured as follows. We start with basic definitions and techniques borrowed from [106] in Section 8.1. In Section 8.2 we present our approximate decision procedure which reduces the problem to one-dimensional separated curves. We solve the latter in Section 8.3. In the whole chapter, we focus on the continuous Fréchet distance. It is straightforward to obtain a similar algorithm for the discrete variant, in fact, then Section 8.3.1 becomes obsolete, which is why we save a factor of $\log 1/\varepsilon$ in the running time.

8.1. Preliminaries

For $i, j \in \mathbb{N}$, $i \leqslant j$, we let $[i..j] := \{i, i+1, \dots, j\}$, which is not to be confused with the real interval $[i, j] = \{x \in \mathbb{R} \mid i \leqslant x \leqslant j\}$. Throughout the chapter we fix the dimension $d \geqslant 2$. Recall that a (polygonal) curve π is defined by its vertices $(\pi_1, \dots, \pi_n)$ with $\pi_p \in \mathbb{R}^d$, $p \in [1..n]$, and we can also view π as a continuous function $\pi\colon [1, n] \to \mathbb{R}^d$ with $\pi_{p+\lambda} = (1-\lambda)\pi_p + \lambda\pi_{p+1}$ for $p \in [1..n-1]$ and $\lambda \in [0, 1]$. We let $|\pi| = n$ be the number of vertices of π and $\|\pi\|$ be its total length $\sum_{i=1}^{n-1} \|p_i - p_{i+1}\|$. We write $\pi_{p..b}$ for the sub-curve $(\pi_p, \pi_{p+1}, \dots, \pi_b)$. Similarly, for an interval $I = [p..b]$ we write $\pi_I = \pi_{p..b}$. For the second curve $\sigma = (\sigma_1, \dots, \sigma_m)$ we will use indices of the form $\sigma_{q..d}$ for the reader's convenience.

Free-space diagram The discrete *free-space* of curves π, σ is defined as $\mathcal{D}^d_{\leqslant\delta}(\pi,\sigma) := \{(p,q) \in [1..n] \times [1..m] \mid \|\pi_p - \sigma_q\| \leqslant \delta\}$. Note that any discrete traversal of π, σ of width at most δ corresponds to a monotone sequence of points in the free-space where at each point in time we increase p or q or both. Because of this property, the free-space is a standard concept used in many algorithms for the Fréchet distance.

The continuous free-space is defined as $\mathcal{D}_{\leqslant\delta}(\pi,\sigma) := \{(p,q) \in [1,n] \times [1,m] \mid \|\pi_p - \sigma_q\| \leqslant \delta\}$. Again, a monotone path from $(1,1)$ to (n,m) in $\mathcal{D}_{\leqslant\delta}(\pi,\sigma)$ corresponds to a traversal of width at most δ. It is well-known [101, 102] that each *free-space cell* $C_{i,j} := \{(p,q) \in [i,i+1] \times [j,j+1] \mid \|\pi_p - \sigma_q\| \leqslant \delta\}$ (for $i \in [1..n-1], j \in [1..m-1]$) is convex, specifically it is the intersection of an ellipsoid with $[i,i+1] \times [j,j+1]$. In particular, the intersection of the free-space with any interval $[i,i+1] \times \{j\}$ (or $\{i\} \times [j,j+1]$) is an interval $I^h_{i,j}$ (or $I^v_{i,j}$), and for any such interval the subset that is reachable by a monotone path from $(1,1)$ is an interval $R^h_{i,j}$ (or $R^v_{i,j}$). Moreover, in constant time one can solve the following *free-space cell problem*: Given intervals $R^h_{i,j} \subseteq [i,i+1] \times \{j\}, R^v_{i,j} \subseteq \{i\} \times [j,j+1]$, determine the intervals $R^h_{i,j+1} \subseteq [i,i+1] \times \{j+1\}, R^v_{i+1,j} \subseteq \{i+1\} \times [j,j+1]$ consisting of all points that are reachable from a point in $R^h_{i,j} \cup R^v_{i,j}$ by a monotone path within the free-space cell $C_{i,j}$. Solving this problem for all cells from lower left to upper right we determine whether (n,m) is reachable from $(1,1)$ by a monotone path and thus decide whether the Fréchet distance is at most δ.

From approximate deciders to approximation algorithms An *approximate decider* is an algorithm that, given curves π, σ and $\delta > 0, 0 < \varepsilon \leqslant 1$, returns one of the outputs (1) $d_{\mathrm{F}}(\pi,\sigma) > \delta$ or (2) $d_{\mathrm{F}}(\pi,\sigma) \leqslant (1+\varepsilon)\delta$. In any case, the returned answer has to be correct. In particular, if $\delta < d_{\mathrm{F}}(\pi,\sigma) \leqslant (1+\varepsilon)\delta$ the algorithm may return either of the two outputs.

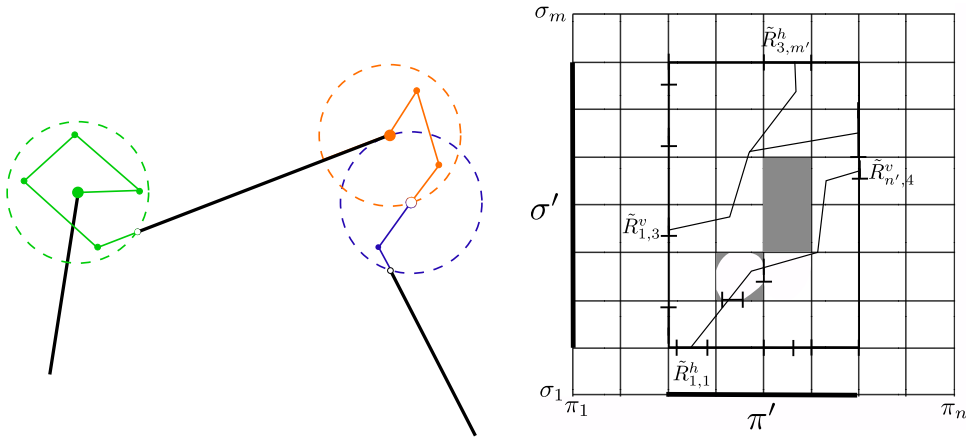
Let $D(\pi,\sigma,\delta,\varepsilon)$ be the running time of an approximate decider and set $D(\pi,\sigma,\varepsilon) := \max_{\delta>0} D(\pi,\sigma,\delta,\varepsilon)$. We assume polynomial dependence on ε, in particular, that there are constants $0 < c_1 < c_2 < 1$ such that for any $1 < \varepsilon \leqslant 1$ we have $c_1 D(\pi,\sigma,\varepsilon/2) \leqslant D(\pi,\sigma,\varepsilon) \leqslant c_2 D(\pi,\sigma,\varepsilon/2)$. Driemel et al. [106] gave a construction of a $(1+\varepsilon)$-approximation for the Fréchet distance given an approximate decider. (This follows from [106, Theorem 3.15] after replacing their concrete approximate decider with running time "$\mathcal{O}(N(\varepsilon,\pi,\sigma))$" by any approximate decider with running time $D(\pi,\sigma,\varepsilon)$.)

Lemma 8.1. *Given an approximate decider with running time $D(\pi,\sigma,\varepsilon)$ we can construct a $(1+\varepsilon)$-approximation for the Fréchet distance with running time*

$$\mathcal{O}\big(D(\pi,\sigma,\varepsilon) + D(\pi,\sigma,1)\log n\big).$$

8.2. The Approximate Decider

Let π, σ be curves for which we want to (approximately) decide whether $d_{\mathrm{F}}(\pi,\sigma) > \delta$ or $d_{\mathrm{F}}(\pi,\sigma) \leqslant (1+\varepsilon)\delta$. We modify the curve π by introducing new vertices as follows.



(a) This figure illustrates our partitioning of a curve into *pieces* (contained in dashed circles) and *long segments* (bold edges).

(b) The free-space problem for pieces π' and σ' in the free-space diagram of π and σ. Given entry intervals on the lower and left boundary of the region, compute exit intervals on the upper and right boundary.

Figure 8.1: Definition and treatment of pieces.

Start with the initial vertex π_1 as current vertex. If the segment following the current vertex has length at least $\Lambda = \Lambda_{\varepsilon,\delta} := \min\{\frac{1}{2}\sqrt{\varepsilon}, \frac{1}{4}\} \cdot \delta$ then mark this segment as *long* and set the next vertex as the current vertex. Otherwise follow π from the current vertex π_x to the first point π_y such that $\|\pi_x - \pi_y\| = \Lambda$ (or until we reach the last vertex of π). If π_y is not a vertex, but lies on some segment of π, then introduce a new vertex at π_y. Mark $\pi_{x..y}$ as a *piece* of π and set π_y as current vertex. Repeat until π is completely traversed. Since this procedure introduces at most $|\pi|$ new vertices and does not change the shape of π, with slight abuse of notation we call the resulting curve again π and set $n := |\pi|$. This partitions π into sub-curves $\pi^1, \ldots, \pi^k$, with $\pi^s = \pi_{p_s..b_s}$, where every part π^s is either (see also Figure 8.1a)

- a *long segment*: $b_s = p_s + 1$ and $\|\pi_{p_s} - \pi_{b_s}\| \geqslant \Lambda$, or
- a *piece*: $\|\pi_{p_s} - \pi_{b_s}\| = \Lambda$ and $\|\pi_{p_s} - \pi_x\| < \Lambda$ for all $x \in [p_s, b_s)$.

Note that the last piece actually might have distance $\|\pi_{p_s} - \pi_{b_s}\|$ less than Λ, however, for simplicity we assume equality for all pieces (in fact, a special handling of the last piece would only be necessary in Lemma 8.7). Similarly, we introduce new vertices on σ and partition it into sub-curves $\sigma^1, \ldots, \sigma^\ell$, with $\sigma^t = \sigma_{q_t..d_t}$, each of which is a long segment or a piece. Let $m := |\sigma|$.

We do not want to resolve each free-space cell on its own, as in the standard decision algorithm for the Fréchet distance. Instead, for any pair of pieces we want to consider the free-space region spanned by the two pieces at once, see Figure 8.1b. This is made formal by the following sub-problem.

Problem 8.2 (Free-space region problem). Given $\delta > 0$, $0 < \varepsilon \leqslant 1$, curves π, σ with n and m vertices, and *entry intervals* $\tilde{R}^h_{i,1} \subseteq [i, i+1] \times \{1\}$ for $i \in [1..n)$ and

$\tilde{R}^v_{1,j} \subseteq \{1\} \times [j, j+1]$ for $j \in [1..m)$, compute *exit intervals* $\tilde{R}^h_{i,m} \subseteq [i, i+1] \times \{m\}$ for $i \in [1..n)$ and $\tilde{R}^v_{n,j} \subseteq \{n\} \times [j, j+1]$ for $j \in [1..m)$ such that (1) the exit intervals contain all points reachable from the entry intervals by a monotone path in $\mathcal{D}_{\leqslant\delta}(\pi, \sigma)$ and (2) all points in the exit intervals are reachable from the entry intervals by a monotone path in $\mathcal{D}_{\leqslant(1+\varepsilon)\delta}(\pi, \sigma)$.

To stress that we work with approximations, we denote reachable intervals by $\tilde{R}$ instead of R in the remainder of this chapter.

The standard solution to the free-space region problem would split it up into $n \cdot m$ free-space cells and resolve each cell in constant time, resulting in an $\mathcal{O}(n \cdot m)$ algorithm (this solves the problem even exactly, i.e., for $\varepsilon = 0$). Restricted to pieces, we will show the following improvement, which will yield the desired overall speedup of a factor of $\sqrt{\varepsilon}$.

Lemma 8.3. *If π and σ are pieces then the free-space region problem can be solved in time $\mathcal{O}((n+m)\log^2 1/\varepsilon)$.*

We will prove this lemma in Sections 8.2.3 and 8.3.

Algorithm 8.4. Using an algorithm for the free-space region problem on pieces as in Lemma 8.3, we obtain an approximate decider for the Fréchet distance a follows. We create a directed graph which has a node $v_{s,t}$ for every region $[p_s, b_s] \times [q_t, d_t]$ spanned by pieces π^s and σ^t, and a node $u_{i,j}$ for every remaining region $[i, i+1] \times [j, j+1]$ (which is not contained in any region spanned by two pieces), $i \in [1..n)$, $j \in [1..m)$. We add edges between two nodes whenever their regions touch (i.e., have a common interval I on their boundary), and direct this edge from the region that is to the left or below I to the other one. With each node $u_{i,j}$ we store the entry intervals $\tilde{R}^h_{i,j}$ and $\tilde{R}^v_{i,j}$, and with each node $v_{s,t}$ we store the entry intervals $\tilde{R}^h_{i,q_t} \subseteq [i, i+1] \times \{q_t\}$ for $i \in [p_s..b_s)$ and $\tilde{R}^v_{p_s,j} \subseteq \{p_s\} \times [j, j+1]$ for $j \in [q_t..d_t)$. After correctly initializing the outer reachability intervals $\tilde{R}^h_{i,1}$ and $\tilde{R}^v_{1,j}$, we follow any topological ordering of this graph. For any node $u_{i,j}$, we resolve its region by solving the corresponding free-space *cell* problem in constant time. For any node $v_{s,t}$, we solve the corresponding free-space *region* problem on $\pi' = \pi^s, \sigma' = \sigma^t$ (and $\delta' = \delta, \varepsilon' = \varepsilon$) using Lemma 8.3. Finally, we return $d_{\mathrm{F}}(\pi, \sigma) \leqslant (1+\varepsilon)\delta$ if $(n, m) \in \tilde{R}^h_{n-1,m}$ and $d_{\mathrm{F}}(\pi, \sigma) > \delta$ otherwise.

Lemma 8.5. *Algorithm 8.4 is a correct approximate decider.*

Proof. Observe that if $(n, m) \in \tilde{R}^h_{n-1,m}$ then there exists a monotone path from $(1, 1)$ to (n, m) in $\mathcal{D}_{\leqslant(1+\varepsilon)\delta}(\pi, \sigma)$, which implies $d_{\mathrm{F}}(\pi, \sigma) \leqslant (1+\varepsilon)\delta$. If $d_{\mathrm{F}}(\pi, \sigma) \leqslant \delta$ then there is a monotone path from $(1, 1)$ to (n, m) in $\mathcal{D}_{\leqslant\delta}(\pi, \sigma)$, implying $(n, m) \in \tilde{R}^h_{n-1,m}$. □

In the above algorithm we can ignore *unreachable* nodes, i.e., nodes where all stored entry intervals would be empty. To this end, we fix a topological ordering by mapping a node corresponding to a region $[x_1, x_2] \times [y_1, y_2]$ to $x_2 + y_2$ and sorting by this value ascendingly. This yields $n + m$ layers of nodes, where the order within each layer is arbitrary. For each layer we build a dictionary data structure (a hash table), in which we store only the reachable nodes of this layer. This allows to

quickly enumerate all reachable nodes of a layer. The total overhead for managing the $n+m$ dictionaries is $\mathcal{O}(n+m)$.

Let us analyze the running time of the obtained approximate decider. Let S be the set of non-empty free-space cells $C_{i,j}$ of $\mathcal{D}_{\leqslant(1+\varepsilon)\delta}(\pi,\sigma)$ such that i or j is not contained in a piece. Moreover, let T be the set of all pairs (s,t) such that π^s, σ^t are pieces with initial vertices within distance $(1+\varepsilon)\delta+2\Lambda$. Define $N(\pi,\sigma,\delta,\varepsilon) := |S| + \sum_{(s,t)\in T}(|\pi^s|+|\sigma^t|)$ and set $N(\pi,\sigma,\varepsilon) := \max_{\delta>0} N(\pi,\sigma,\delta,\varepsilon)$. Since the algorithm considers only *reachable* cells and any reachable cell is also non-empty, the cost over all free-space cell problems solved by our approximate decider is bounded by $\mathcal{O}(|S|)$. Since every reachable (thus non-empty) region spanned by two pieces has initial points within distance $(1+\varepsilon)\delta+2\Lambda$, the second term bounds the cost over all free-space region problems on pieces (apart from the $\log^2 1/\varepsilon$ factor). Hence, we obtain the following.

Lemma 8.6. *The approximate decider has running time* $D(\pi,\sigma,\varepsilon) = \mathcal{O}(N(\pi,\sigma,\varepsilon)\cdot \log^2 1/\varepsilon)$.

8.2.1. Free-Space Complexity of c-Packed Curves

Recall that a curve π is c-packed if for any point $z\in\mathbb{R}^d$ and any radius $r>0$ the total length of π inside the ball $B(z,r)$ is at most cr.

Lemma 8.7. *Let π,σ be c-packed curves with n vertices in total and $\varepsilon>0$. Then $N(\pi,\sigma,\varepsilon) = \mathcal{O}(cn/\sqrt{\varepsilon})$.*

Proof. Our proof uses a similar argument as [110, Lemma 4.4]. Let $\delta>0$ be arbitrary. First consider the set S of non-empty free-space cells $C_{i,j}$ of $\mathcal{D}_{\leqslant(1+\varepsilon)\delta}(\pi,\sigma)$ such that i or j is not contained in a piece. Then one of the segments $\pi_{i..i+1}$ and $\sigma_{j..j+1}$ is long, i.e., of length at least $\Lambda = \min\{\frac{1}{2}\sqrt{\varepsilon}, \frac{1}{4}\}\cdot\delta$. We charge the cell $C_{i,j}$ to the shorter of the two segments. Let us analyze how often any segment $v=\pi_{i..i+1}$ can be charged. Consider the ball B of radius $r := \frac{1}{2}\|v\| + (1+\varepsilon)\delta + \max\{\|v\|,\Lambda\}$ centered at the midpoint of v. Every segment $u=\sigma_{j..j+1}$ with $(i,j)\in S$, which charges v, is of length at least $\mu := \max\{\|v\|,\Lambda\}$ (since it is longer than v and a long segment) and contributes at least μ to the total length of σ in B. Since σ is c-packed, the number of such charges is at most

$$\frac{\|\sigma\cap B\|}{\mu} \leqslant \frac{cr}{\mu} \leqslant \frac{c(\frac{1}{2}\|v\|+(1+\varepsilon)\delta+\mu)}{\max\{\|v\|,\Lambda\}} \leqslant \tfrac{3}{2}c + \frac{c(1+\varepsilon)\delta}{\min\{\frac{1}{2}\sqrt{\varepsilon},\frac{1}{4}\}\cdot\delta} = \mathcal{O}\Big(\frac{c}{\sqrt{\varepsilon}}\Big).$$

Thus, the contribution of $|S|$ to the free-space complexity $N(\pi,\sigma,\varepsilon)$ is $\mathcal{O}(cn/\sqrt{\varepsilon})$.

Let T be the set of all pairs (s,t) such that π^s,σ^t are pieces of π,σ with initial vertices within distance $(1+\varepsilon)\delta+2\Lambda$, and consider $\Sigma := \sum_{(s,t)\in T}(|\pi^s|+|\sigma^t|)$. We distribute Σ over the segments of π,σ by charging 1 to every segment of π^s and σ^t for any pair $(s,t)\in T$. Let us analyze how often any segment v of a piece π^s can be charged. Consider the ball B' of radius $r' := (1+\varepsilon)\delta+3\Lambda$ around the initial vertex π_{p_s} of π^s. Since $\|\sigma^t\|\geqslant\Lambda$, for any $(s,t)\in T$ the piece σ^t contributes at least Λ to

the total length of σ in B'. Since σ is c-packed, the number of such charges to v is at most

$$\frac{\|\sigma \cap B'\|}{\Lambda} \leqslant \frac{cr'}{\Lambda} = \frac{c(1+\varepsilon+\frac{3}{2}\sqrt{\varepsilon})}{\min\{\frac{1}{2}\sqrt{\varepsilon}, \frac{1}{4}\}} = \mathcal{O}\Big(\frac{c}{\sqrt{\varepsilon}}\Big).$$

Hence, the contribution of Σ to the free-space complexity $N(\pi, \sigma, \varepsilon)$ is also at most $\mathcal{O}(cn/\sqrt{\varepsilon})$, which finishes the proof. □

Combining Lemmas 8.7, 8.6, and 8.1, we obtain an approximation algorithm for the Fréchet distance with running time $\mathcal{O}(\frac{cn}{\sqrt{\varepsilon}} \log^2 1/\varepsilon + cn \log n) = \tilde{\mathcal{O}}(\frac{cn}{\sqrt{\varepsilon}})$, as desired.

8.2.2. Free-Space Complexity of κ-Bounded and κ-Straight Curves

Definition 8.8. *Let $\kappa \geqslant 1$ be a given parameter. A curve π is* κ-straight *if for any $p, b \in [1, |\pi|]$ we have $\|\pi_{p..b}\| \leqslant \kappa\|\pi_p - \pi_b\|$. A curve π is* κ-bounded *if for all p, b the sub-curve $\pi_{p..b}$ is contained in $B(\pi_p, r) \cup B(\pi_b, r)$, where $r = \frac{\kappa}{2}\|\pi_p - \pi_b\|$.*

The following lemma from [110] allows us to transfer our speedup for c-packed curves directly to κ-straight curves.

Lemma 8.9. *A κ-straight curve is 2κ-packed.*

In the remainder of this section we consider κ-bounded curves, closely following [110, Sect. 4.2].

Lemma 8.10. *Let $\delta > 0$, $0 < \varepsilon \leqslant 1$, $\lambda > 0$, and let π be a κ-bounded curve with disjoint sub-curves $\pi^1, \ldots, \pi^k$, where $\pi^s = \pi_{p_s..b_s}$ and $\|\pi_{p_s} - \pi_{b_s}\| \geqslant \lambda$ for all s. Then for any $z \in \mathbb{R}^d$, $r > 0$ the number of sub-curves π^s intersecting $B(z, r)$ is bounded by $\mathcal{O}(\kappa^d(1 + r/\lambda)^d)$.*

Proof. Let $\pi^{s_1}, \ldots, \pi^{s_\ell}$ be the sub-curves that intersect the ball $B = B(z, r)$. Let $X = \{s_1, s_3, \ldots, \}$ be the odd indices among the intersecting sub-curves. For all $s \in X$ pick any point π_{x_s} in $\pi^s \cap B$. Between any points $\pi_{x_s}, \pi_{x_{s'}}$ there must lie an even sub-curve $\pi^{s_{2i}}$. As the endpoints of this even sub-curve have distance at least λ, we have $\|\pi_{x_s} - \pi_{x_{s'}}\| \geqslant \lambda/(\kappa + 1)$. Otherwise the even part would not fit into $B(\pi_{x_s}, r) \cup B(\pi_{x_{s'}}, r)$ which has diameter $(\kappa + 1)\|\pi_{x_s} - \pi_{x_{s'}}\|$. Hence, the balls $B(\pi_{x_s}, \lambda/2(\kappa + 1))$ are disjoint and contained in $B(z, r + \lambda)$. A standard packing argument now shows that $\ell \leqslant 2 \cdot (r + \lambda)^d/(\lambda/2(\kappa + 1))^d = \mathcal{O}(\kappa^d(1 + r/\lambda)^d)$. □

Lemma 8.11. *For any κ-bounded curves π, σ with n vertices in total, $0 < \varepsilon \leqslant 1$, we have $N(\pi, \sigma, \varepsilon) = \mathcal{O}((\kappa/\sqrt{\varepsilon})^d n)$.*

Proof. Let $\delta > 0$ and consider the partitionings into long segments and pieces $\pi^1, \ldots, \pi^k$, $\sigma^1, \ldots, \sigma^\ell$ computed by our algorithm. Then $\sigma^t = \sigma_{q_t..d_t}$ satisfies $\|\sigma_{q_t} - \sigma_{d_t}\| \geqslant \Lambda = \min\{\frac{1}{2}\sqrt{\varepsilon}, \frac{1}{4}\} \cdot \delta$ for all t. We use the same charging scheme as in Lemma 8.7. Consider any segment v of a piece π^s. The segment v can be charged by a part σ^t which is either a long segment or a piece. In both cases, σ^t intersects the ball B centered at the midpoint of $\|v\|$ with radius $r := (1 + \varepsilon)\delta + 2\Lambda$. By Lemma 8.10 with $\lambda := \Lambda$, the number of such charges is bounded by $\mathcal{O}((\kappa/\sqrt{\varepsilon})^d)$.

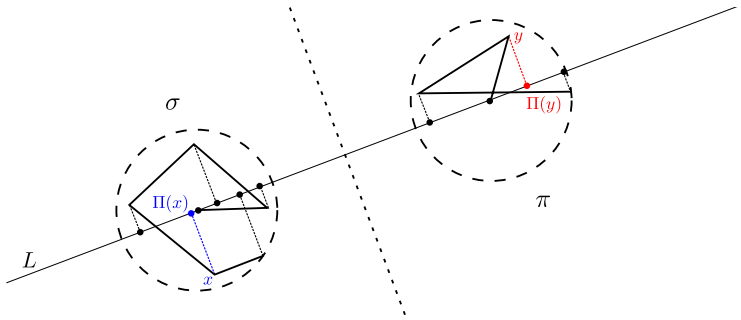


Figure 8.2: Projection of the pieces π, σ onto the line L through their initial vertices. This yields one-dimensional separated curves $\hat{\pi}, \hat{\sigma}$.

Now consider any long segment v of π. The segment v can be charged by segments of σ which are longer than v. Any such charging gives rise to a long segment σ^t intersecting the ball B centered at the midpoint of v of radius $r := (1+\varepsilon)\delta + \frac{1}{2}\|v\|$. By Lemma 8.10 with $\lambda := \|v\|$, the number of such charges is bounded by $\mathcal{O}(\kappa^d(\frac{3}{2} + (1+\varepsilon)\delta/\|v\|)^d) = \mathcal{O}((\kappa/\sqrt{\varepsilon})^d)$, since $\|v\| \geqslant \Lambda = \min\{\frac{1}{2}\sqrt{\varepsilon}, \frac{1}{4}\} \cdot \delta$.

Hence, every segment of π is charged $\mathcal{O}((\kappa/\sqrt{\varepsilon})^d)$ times; a symmetric statement holds for σ. □

Plugging the above lemma into Lemma 8.1 we obtain the following result. The best previously known running time was $\mathcal{O}((\kappa/\varepsilon)^d n + \kappa^d n \log n)$ [110].

Theorem 8.12. *For any $0 < \varepsilon \leqslant 1$ there is a $(1+\varepsilon)$-approximation for the continuous and discrete Fréchet distance on κ-bounded curves with n vertices in total in time $\mathcal{O}((\kappa/\sqrt{\varepsilon})^d n \log^2 1/\varepsilon + \kappa^d n \log n) = \tilde{\mathcal{O}}((\kappa/\sqrt{\varepsilon})^d n)$.*

8.2.3. Solving the Free-Space Region Problem on Pieces

It remains to prove Lemma 8.3. Let $(\pi, \sigma, \delta, \varepsilon)$ be an instance of the free-space region problem, where $n := |\pi|$, $m := |\sigma|$, with $\|\pi_1 - \pi_x\|, \|\sigma_1 - \sigma_y\| \leqslant \Lambda_{\varepsilon,\delta} = \Lambda$ for any $x \in [1, n]$, $y \in [1, m]$ (and entry intervals $\tilde{R}^h_{i,1} \subseteq [i, i+1] \times \{1\}$ for $i \in [1..n)$ and $\tilde{R}^v_{1,j} \subseteq \{1\} \times [j, j+1]$ for $j \in [1..m)$). We reduce this instance to the free-space region problem on *one-dimensional separated* curves, i.e., curves $\hat{\pi}, \hat{\sigma}$ in $\mathbb{R}$ such that all vertices of $\hat{\pi}$ lie above 0 and all vertices of $\hat{\sigma}$ lie below 0.

Since π and σ stay within distance Λ of their initial vertices, if their initial vertices are within distance $\|\pi_1 - \sigma_1\| \leqslant \delta - 2\Lambda$ then all pairs of points in π, σ are within distance δ. In this case, we find a translation of π making $\|\pi_1 - \sigma_1\| = \delta - 2\Lambda$ and all pairwise distances are still at most δ. This ensures that the curves π, σ are contained in disjoint balls of radius $\Lambda \leqslant \frac{1}{4}\delta$ centered at their initial vertices.

Consider the line L through the initial vertices π_1 and σ_1. Denote by $\Pi\colon \mathbb{R}^d \to L$ the projection onto L. Now, instead of the pieces π, σ we consider their projections $\hat{\pi} := \Pi(\pi) = (\Pi(\pi_1), \ldots, \Pi(\pi_n))$ and $\hat{\sigma} := \Pi(\sigma) = (\Pi(\sigma_1), \ldots, \Pi(\sigma_m))$, see Figure 8.2. Note that after rotation and translation we can assume that $\hat{\pi}$ and $\hat{\sigma}$ lie on $\mathbb{R} \subset \mathbb{R}^d$ and $\hat{\pi}$ and $\hat{\sigma}$ are separated by $0 \in \mathbb{R}$ (since π and σ are contained in disjoined balls centered on L). Now we solve the free-space region problem on $\hat{\pi}$, $\hat{\sigma}$, $\hat{\delta} := \delta$, and $\hat{\varepsilon} := \frac{1}{2}\varepsilon$ (with the same entry intervals $\tilde{R}^h_{i,j}, \tilde{R}^v_{i,j}$)).

Lemma 8.13. *Any solution to the the free-space region problem on* $(\hat{\pi}, \hat{\sigma}, \hat{\delta}, \hat{\varepsilon})$ *solves the free-space region problem on* $(\pi, \sigma, \delta, \varepsilon)$.

Proof. Let x, y be vertices of π, σ, respectively. Clearly, $\|\Pi(x) - \Pi(y)\| \leqslant \|x - y\|$. Hence, any monotone path in $\mathcal{D}_{\leqslant\delta}(\pi, \sigma)$ yields a monotone path in $\mathcal{D}_{\leqslant\delta}(\hat{\pi}, \hat{\sigma}) = \mathcal{D}_{\leqslant\hat{\delta}}(\hat{\pi}, \hat{\sigma})$, so it will be found.

Note that x and y have distance at most Λ to L. Since $\Pi(x) - \Pi(y)$ and $x - \Pi(x) - (y - \Pi(y))$ are orthogonal, we can use the Pythagorean theorem to obtain

$$\begin{aligned}\|x - y\| &= \sqrt{\|\Pi(x) - \Pi(y)\|^2 + \|x - \Pi(x) - (y - \Pi(y))\|^2} \\ &\leqslant \sqrt{\|\Pi(x) - \Pi(y)\|^2 + (2\Lambda)^2}.\end{aligned}$$

Hence, any monotone path in $\mathcal{D}_{\leqslant(1+\hat{\varepsilon})\hat{\delta}}(\hat{\pi}, \hat{\sigma})$ yields a monotone path in $\mathcal{D}_{\leqslant\alpha}(\pi, \sigma)$ with $\alpha \leqslant \sqrt{(1+\hat{\varepsilon})^2\hat{\delta}^2 + (2\Lambda)^2}$. Plugging in $\hat{\delta} = \delta$, $\hat{\varepsilon} = \frac{1}{2}\varepsilon$, and $\Lambda = \min\{\frac{1}{2}\sqrt{\varepsilon}, \frac{1}{4}\} \cdot \delta$ we obtain $\alpha \leqslant \sqrt{(1 + \frac{1}{2}\varepsilon)^2 + \varepsilon} \cdot \delta \leqslant (1 + \varepsilon)\,\delta$. Thus, the desired guarantees for the free-space region problem are satisfied. □

We prove the following lemma in Section 8.3, concluding the proof of Lemma 8.3.

Lemma 8.14. *The free-space region problem on one-dimensional separated curves can be solved in time* $\mathcal{O}((n + m) \log^2 1/\varepsilon)$.

8.3. On One-Dimensional Separated Curves

In this section, we show how to solve the free-space region problem on one-dimensional separated curves in time $\mathcal{O}((n + m) \log^2 1/\varepsilon)$, i.e., we prove Lemma 8.14.

First, in Section 8.3.1, we show how to reduce this problem to a *discrete* version, meaning that we can eliminate the continuous Fréchet distance and only consider the much simpler discrete Fréchet distance (for general curves such a reduction is not known to exist, but we only need it for one-dimensional separated curves). Moreover, we simplify our curves further by rounding the vertices. This yields a reduction to the following sub-problem. Note that we no longer ask for an approximation algorithm.

Problem 8.15 (Reduced free-space problem). Given one-dimensional separated curves π, σ with n, m vertices and all vertices being multiples of $\frac{1}{3}\varepsilon\delta$, and given an entry set $E \subseteq [1..n]$, compute the exit set $F^\pi \subseteq [1..n]$ consisting of all points f such that $d_{\mathrm{dF}}(\pi_{e..f}, \sigma) \leqslant \delta$ for some $e \in E$ and the exit set $F^\sigma \subseteq [1..m]$ consisting of all points f such that $d_{\mathrm{dF}}(\pi_{e..n}, \sigma_{1..f}) \leqslant \delta$ for some $e \in E$.

Lemma 8.16. *The reduced free-space problem can be solved in time* $\mathcal{O}((n+m) \log 1/\varepsilon)$.

As a second step, we prove the above lemma. We first consider the special case of $E = \{1\}$ and the problem of deciding whether $n \in F^\pi$, i.e., the lower left corner $(1, 1)$ of the free-space is the only entry point and we want to determine whether the upper right corner (n, m) is an exit. This is equivalent to deciding whether the discrete

Fréchet distance of π, σ is at most δ, which is known to have a near-linear time algorithm as π, σ are one-dimensional and separated (see the footnote in Section 6.3 for details). We present a greedy algorithm for this special case in Section 8.3.2. To extend this to the reduced free-space problem, we prove useful structural properties of one-dimensional separated curves in Section 8.3.3. With these, we first solve the problem of determining the exit set F^π assuming $E = \{1\}$ in Section 8.3.4. Then we show for general $E \subseteq [1..n]$ how to compute F^π (Section 8.3.4) and F^σ (Section 8.3.4).

8.3.1. Reduction from the Continuous to the Discrete Case

Essentially we use the following lemma to reduce the *continuous* free-space region problem on one-dimensional separated curves to the *discrete* reduced free-space problem.

Lemma 8.17. *Let π, σ be one-dimensional separated curves with sub-curves $\pi_{p..b}$, $\sigma_{q..d}$. Then we have $d_{\mathrm{F}}(\pi_{p..b}, \sigma_{q..d}) = d_{\mathrm{dF}}(\pi_{p..b}, \sigma_{q..d})$. In particular, assume that we subdivide any segments of π, σ by adding new vertices, which yields new curves π', σ' with sub-curves $\pi'_{p'..b'}, \sigma'_{q'..d'}$ that are subdivisions of $\pi_{p..b}, \sigma_{q..d}$. Then we have*

$$d_{\mathrm{dF}}(\pi'_{p'..b'}, \sigma'_{q'..d'}) = d_{\mathrm{dF}}(\pi_{p..b}, \sigma_{q..d}) = d_{\mathrm{F}}(\pi_{p..b}, \sigma_{q..d}).$$

Proof. It is known that $d_{\mathrm{F}}(\pi, \sigma) \leqslant d_{\mathrm{dF}}(\pi, \sigma)$ holds for all curves π, σ. Thus, we only need to show that any continuous traversal $\phi = (\phi_1, \phi_2)$ of $\pi_{p..b}, \sigma_{q..d}$ can be transformed into a discrete traversal with the same width. We adapt ϕ as follows. For any point in time $t \in [0, 1]$, if $\phi_1(t)$ is at a vertex of π we set $\phi'_1(t) := \phi_1(t)$. Otherwise $\phi_1(t)$ is in the interior of a segment $\pi_{i..i+1}$ of π. Let $j \in \{i, i+1\}$ minimize π_j. We set $\phi'_1(t) := j$. Observe that ϕ'_1 indeed is a non-decreasing function from $[0, 1]$ onto $[1..n]$. A similar construction, where we round to the value $j \in \{i, i+1\}$ maximizing σ_j, yields ϕ'_2 and we obtain a discrete traversal $\phi' = (\phi'_1, \phi'_2)$. The width of ϕ' is at most the width of ϕ since we rounded in the right way, i.e., we have $\pi(\phi'_1(t)) \leqslant \pi(\phi_1(t))$ and $\sigma(\phi'_2(t)) \geqslant \sigma(\phi_2(t))$ so that $\|\pi(\phi'_1(t)) - \sigma(\phi'_2(t))\| \leqslant \|\pi(\phi_1(t)) - \sigma(\phi_2(t))\|$ for all $t \in [0, 1]$.

Note that the discrete Fréchet distance is in general not preserved under subdivision of segments, but the continuous Fréchet distance is. Thus, the second statement follows from the first one, $d_{\mathrm{dF}}(\pi_{p..b}, \sigma_{q..d}) = d_{\mathrm{F}}(\pi_{p..b}, \sigma_{q..d}) = d_{\mathrm{F}}(\pi'_{p'..b'}, \sigma'_{q'..d'}) = d_{\mathrm{dF}}(\pi'_{p'..b'}, \sigma'_{q'..d'})$. □

The above lemma allows the following trick. Consider any finite sets $E \subseteq [1, n]$ and $F \subseteq [1, n]$. Add π_x as a vertex to π for any $x \in E \cup F$, with slight abuse of notation we say that π now has vertices at π_i, $i \in [1..n]$, and π_x, $x \in E \cup F$. Mark the vertices π_x, $x \in E$, as entries. Now solve the reduced free-space problem instance (π, σ, E). This yields the set F^π of all values $f \in F$ such that there is an $e \in E$ with $d_{\mathrm{dF}}(\pi_{e..f}, \sigma) \leqslant \delta$, which by Lemma 8.17 is equivalent to $d_{\mathrm{F}}(\pi_{e..f}, \sigma) \leqslant \delta$. Thus, we computed all exit points in F given entry points in E, with respect to the continuous Fréchet distance. This is already near to a solution of the free-space region problem, however, we have to cope with entry and exit *intervals*.

For the full reduction we need two more arguments. First, we can replace all nonempty input intervals $\tilde{R}^h_{i,1}$ by the leftmost point $(y_i, 1)$ in $\tilde{R}^h_{i,1} \cap \mathcal{D}_{\leqslant\delta}(\pi, \sigma)$, specifically, we show that any traversal starting in a point in $\tilde{R}^h_{i,1}$ can be transformed into a traversal starting in $(y_i, 1)$. Thus, we add π_{y_i} as a vertex and mark it as an entry to obtain a finite and small set of entry points. Second, for any segment $\pi_{i..i+1}$ we call a point $f \in [i, i+1]$ *reachable* if there is an $e \in E$ with $d_{\mathrm{F}}(\pi_{e..f}, \sigma) \leqslant \delta$. We show that if f is reachable then essentially all points $f' \in [i, i+1]$ with $\pi_{f'} \leqslant \pi_f$ are also reachable. Thus, the set of reachable points is an interval with one trivial endpoint, and we only need to search for the other endpoint of the interval, which can be done by binary search. Moreover, we can parallelize all these binary searches, as solving one reduced free-space problem can answer for every segment of π whether a particular point on this segment is reachable (after adding this point as a vertex). To make these binary searches finite, we round all vertices of π and σ to multiples of $\gamma := \frac{1}{4}\varepsilon\delta$ and only search for exit points that are multiples of γ. This is allowed since the free-space region problem only asks for an approximate answer. A similar procedure yields the exits on σ reachable from entries on π, and determining the exits reachable from entries on σ is a symmetric problem. Since for the binary searches we reduce to $\mathcal{O}(\log 1/\varepsilon)$ instances of the reduced free-space problem, Lemma 8.14 follows from Lemma 8.16.

In the following we present the details of this approach. Let π, σ be one-dimensional separated curves, i.e., they are contained in $\mathbb{R}$, all vertices of π lie above 0, and all vertices of σ lie below 0. Let $n = |\pi|$, $m = |\sigma|$, $\delta > 0$ and $0 < \varepsilon \leqslant 1$. Consider entry intervals $\tilde{R}^h_{i,1} \subseteq [i, i+1] \times \{1\}$ for $i \in [1..n)$ and $\tilde{R}^v_{1,j} \subseteq \{1\} \times [j, j+1]$ for $j \in [1..m)$. We reduce this instance of the free-space region problem to $\mathcal{O}(\log 1/\varepsilon)$ instances of the reduced free-space problem.

First we change π, σ as follows. (1) Let $Z \subset \mathbb{R}$ be the set of all integral multiples[1] of $\gamma := \frac{1}{4}\varepsilon\delta$. We round all vertices of π, σ to values in Z, where we round down everything in π and round up in σ, yielding curves π', σ'. (2) Let $I \subseteq [1..n)$ be the set of all i with nonempty $\tilde{R}^h_{i,1} \cap \mathcal{D}_{\leqslant\delta}(\pi', \sigma')$. For any $i \in I$ let $(y'_i, 1)$ be the leftmost point in $\tilde{R}^h_{i,1} \cap \mathcal{D}_{\leqslant\delta}(\pi', \sigma')$ and let $y_i \leqslant y'_i$ be maximal with $\pi'_{y_i} \in Z$. Add π'_{y_i} as a vertex to π' and mark it as an entry. With slight abuse of notation, we say that π' now has its vertices at π'_i, $i \in [1..n]$ and π'_{y_i}, $i \in I$. We let $E = \{y_i \mid i \in I\}$ be the indices of the entry vertices. Note that (π', σ', E) can be computed in time $\mathcal{O}(n + m)$.

For every $i \in [1..n)$ consider the multiples of γ on $\pi'_{i..i+1}$, i.e., $S_i := \{x \in [i, i+1] \mid \pi'_x \in Z\}$. Note that S_i is either $[i, i+1]$ (if $\pi_i = \pi_{i+1}$) or it forms an arithmetic progression, specifically $S_i = \{i, i + 1/t_i, i + 2/t_i, \ldots, i + 1\}$ for some $t_i \in \mathbb{N}$, since π'_i, π'_{i+1} are in Z and π'_x is a linear function in x. Thus, S_i and sub-sequences of S_i can be handled efficiently, we omit these details in the following. We want to determine the set F_i of all $f \in S_i$ such that there is an $e \in E$ with $d_{\mathrm{dF}}(\pi'_{e..f}, \sigma') \leqslant \delta$. We first argue that F_i is of an easy form.

Lemma 8.18. *If F_i is non-empty then we have $F_i = [a, b] \cap S_i$ for some $a, b \in S_i$ with $\{a, b\} \cap \{i, y_i, i+1\} \neq \emptyset$ (or $\{a, b\} \cap \{i, i+1\} \neq \emptyset$ if y_i does not exist).*

[1] Without loss of generality we assume $1/\varepsilon \in \mathbb{N}$ so that $\delta \in Z$.

Proof. We show that if any $f \in S_i$ is reachable, i.e., there is an $e \in E$ with $d_{\mathrm{dF}}(\pi'_{e..f}, \sigma') \leqslant \delta$, then any $f' \in S_i$ with $\pi'_{f'} \leqslant \pi'_f$ and $y_i \notin (f', f]$ is also reachable. This proves the claim. Let ϕ be any traversal of $\pi'_{e..f}, \sigma'$ of width at most δ. Note that $e \leqslant f'$, since $y_i \notin (f', f]$ and y_i is the only entry on the segment containing f and f'. If $f' \leqslant f$ then we change ϕ to stop at $\pi'_{f'}$ once it arrives at this point, and we traverse the remaining part of σ staying fixed at $\pi'_{f'}$. Since $\pi'_{f'} \leqslant \pi'_f$ this does not increase the width of the traversal and shows that f' is also reachable. If $f' > f$ then we append a traversal to ϕ that stays fixed at σ'_m but walks in π' from π'_f to $\pi'_{f'}$. Again since $\pi'_{f'} \leqslant \pi'_f$ this does not increase the width of the traversal and shows that f' is also reachable. □

Note that by solving the reduced free-space problem on (π', σ', E) we decide for each $f \in [n] \cup \{y_i \mid i \in I\}$ whether there is an $e \in E$ with $d_{\mathrm{dF}}(\pi'_{e..f}, \sigma') \leqslant \delta$. By the above lemma, this yields one of the endpoints of the interval F_i, say a, and we only have to determine the other endpoint, say b. In the special case $\pi'_i = \pi'_{i+1}$ we even determined both endpoints already, so from now on we can assume $\pi'_i \neq \pi'_{i+1}$ so that $|S_i| < \infty$. We search for the other endpoint of F_i using a binary search over S_i. To test whether any $z \in S_i$ is in F_i, we add π'_z as a vertex of π' and solve the reduced free-space problem on (π', σ', E). If z is in the output set F^π then it is in F_i.

Note that any vertex $\pi'_x > \delta$ on π' does not have any point of σ within distance δ, which is preserved by setting $\pi'_x := 2\delta$. Thus, we can assume that π' takes values in $[0, 2\delta]$, which implies $|S_i| \leqslant \mathcal{O}(1/\varepsilon)$, so that our binary search needs $\mathcal{O}(\log 1/\varepsilon)$ steps. Moreover, note that we can parallelize these binary searches, since we can add a vertex z_i on every sub-curve $\pi'_{i..i+1}$, so that one call to the reduced free-space problem determines for every z_i whether it is reachable. Here we use Lemma 8.17, since we need that further subdivision of some segments of π' does not change the discrete Fréchet distance. Note that since we add $\mathcal{O}(n)$ vertices to π' and since we need $\mathcal{O}(\log 1/\varepsilon)$ steps of binary search, Lemma 8.16 implies a total running time of $\mathcal{O}((n+m)\log^2 1/\varepsilon)$.

We thus computed $F_i = [a, b] \cap S_i$ with $a, b \in S_i$. We extend F_i slightly to $F'_i = [a', b'] \cap S_i$ by including the neighboring elements of a and b in S_i. Finally, we set $\tilde{R}^h_{i,m}(\pi) := [a', b'] \times \{m\}$. A similar procedure adding entries E^σ on σ' and doing a binary search over exits on π' yields an interval $\tilde{R}^h_{i,m}(\sigma)$ consisting of points $(f, m) \in [i, i+1] \times \{m\}$ such that there is an $e \in E^\sigma$ with $d_{\mathrm{dF}}(\pi'_{1..f}, \sigma'_{e..m}) \leqslant \delta$. We set $\tilde{R}^h_{i,m} := \tilde{R}^h_{i,m}(\pi) \cup \tilde{R}^h_{i,m}(\sigma)$, which will be again an interval (which follows from the proof of Lemma 8.18). A symmetric algorithm determines $\tilde{R}^v_{n,j}$ for $j \in [1..m)$.

We show that we correctly solve the given free-space region problem instance.

Lemma 8.19. *The computed intervals are a valid solution to the given free-space region instance.*

Proof. Let ϕ be any monotone path in $\mathcal{D}_{\leqslant\delta}(\pi, \sigma)$ that starts in a point $(p, 1) \in \tilde{R}^h_{j,1}$ and ends in (b, m), witnessing that $d_{\mathrm{dF}}(\pi_{p..b}, \sigma) \leqslant \delta$. After rounding down π to π' and rounding up σ to σ', ϕ is still a monotone path in $\mathcal{D}_{\leqslant\delta}(\pi', \sigma')$. Moreover, we can prepend a path from $(y'_j, 1)$ to $(p, 1)$ to ϕ, since $\tilde{R}^h_{j,1} \cap \mathcal{D}_{\leqslant\delta}(\pi', \sigma')$ is an interval containing $(y'_j, 1)$ and $(p, 1)$. We can also prepend a path from $(y_i, 1)$ to $(y'_i, 1)$, since σ_1 is a multiple of γ and π'_{y_i} is at most $\pi'_{y'_i}$ rounded up to a multiple of γ. Let r

be the value of π_b rounded down to a multiple of γ. This value r is attained at some point π_f on the same segment $\pi_{i..i+1}$ as π_b. If $f \leqslant b$ then we change ϕ to stop at π_f whenever it reaches this point. If $f > b$ then we change ϕ by appending a path from (b, m) to (f, m). In any case, this yields a monotone path in $\mathcal{D}_{\leqslant\delta}(\pi', \sigma')$ from $(y_j, 1)$ to (f, m). Since such a continuous traversal is equivalent to a discrete traversal by Lemma 8.17, we have $f \in F_i$. By the construction of F'_i, the point (b, m) will be contained in the output $\tilde{R}^h_{i,m}(\pi)$, so we find the reachable exit (b, m) as desired. A similar argument with entries on σ shows that we satisfy property (1) of the free-space region problem.

Consider any point (f, m) in the constructed output set $\tilde{R}^h_{i,m}(\pi)$. By the construction of F'_i, there is a point b on the same segment as f with $|\pi'_b - \pi'_f| \leqslant \gamma$ and there is an entry $e \in E$ with $d_{\mathrm{dF}}(\pi'_{e..b}, \sigma') \leqslant \delta$, witnessed by a traversal ϕ. By the construction of E, there is a point $y \in \tilde{R}^h_{i,1}$ with $e \leqslant y$ and $|\pi'_y - \pi'_e| \leqslant \gamma$. First assume $y \leqslant f, b$. In this case, we change ϕ so that it starts at π'_y and stays there while ϕ is at any point π'_x, $x \leqslant y$. Moreover, if $b \leqslant f$ we change ϕ so that it stops at π'_b once it reaches this point, and if $b > f$ we change ϕ by appending a path from (b, m) to (f, m). This shows $d_{\mathrm{dF}}(\pi'_{y..f}, \sigma') \leqslant \delta + \gamma$. Since π', σ' are rounded versions of π, σ where all vertices are moved by less than γ, we obtain $d_{\mathrm{dF}}(\pi_{y..f}, \sigma) \leqslant \delta + 3\gamma \leqslant (1 + \varepsilon)\delta$. In the remaining cases $e \leqslant f \leqslant y$ and $e \leqslant b \leqslant y$, we have $|\pi'_y - \pi'_x| \leqslant 2\gamma$ for all $x \in \{e, b, f\}$. Hence, a traversal staying fixed in π'_y does the job, i.e., $d_{\mathrm{dF}}(\pi'_y, \sigma') \leqslant \delta + 2\gamma$ and $d_{\mathrm{dF}}(\pi_y, \sigma) \leqslant \delta + 4\gamma = (1 + \varepsilon)\delta$. Thus, any point (f, m) in the output set is reachable from the entry sets by a monotone path in $\mathcal{D}_{\leqslant(1+\varepsilon)\delta}(\pi, \sigma)$, which together with a similar argument for entries on σ proves that we satisfy property (2) of the free-space problem. □

8.3.2. Greedy Decider for One-Dimensional Separated Curves

In the remainder of this chapter all indices of curves will be integral. Let $\pi = (\pi_1, \ldots, \pi_n)$ and $\sigma = (\sigma_1, \ldots, \sigma_m)$ be two separated polygonal curves in $\mathbb{R}$, i.e., $\pi_i \geqslant 0 \geqslant \sigma_j$. For indices $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant m$, define $\mathrm{vis}_\sigma(i, j) := \{k \mid k \geqslant j \text{ and } \sigma_k \geqslant \pi_i - \delta\}$ as the index set of vertices on σ that are later in sequence than σ_j and are still in distance δ to π_i (i.e, *seen* by π_i) and, likewise, $\mathrm{vis}_\pi(i, j) := \{k \mid k \geqslant i \text{ and } \pi_k \leqslant \sigma_j + \delta\}$. Hence, the set of points that we may reach on σ by starting in (π_i, σ_j) and staying in π_i can be defined as the longest contiguous sub-sequence $[j{+}1..j{+}k]$ such that $[j{+}1..j{+}k] \subseteq \mathrm{vis}_\sigma(i, j)$. Let $\mathrm{reach}_\sigma(i, j) := [j{+}1..j{+}k]$ denote this sub-sequence and let $\mathrm{reach}_\pi(i, j)$ be defined symmetrically. Note that $\pi_i \leqslant \pi_{i'}$ implies that $\mathrm{vis}_\sigma(i, j) \supseteq \mathrm{vis}_\sigma(i', j)$, however the converse does not necessarily hold. Also, $\mathrm{vis}_\sigma(i, j) \not\supseteq \mathrm{vis}_\sigma(i', j)$ implies that $\mathrm{vis}_\sigma(i, j) \subsetneq \mathrm{vis}_\sigma(i', j)$ and $\pi_i > \pi_{i'}$.

The visibility sets established above enable us to define a greedy algorithm for the Fréchet distance of π and σ. Let $1 \leqslant p \leqslant n$ and $1 \leqslant q \leqslant m$ be arbitrary indices on σ and π. We say that p' is a *greedy step on π from (p, q)*, written $p' \leftarrow \textsc{GreedyStep}_\pi(\pi_{p..n}, \sigma_{q..m})$, if $p' \in \mathrm{reach}_\pi(p, q)$ and $\mathrm{vis}_\sigma(i, q) \subseteq \mathrm{vis}_\sigma(p', q)$ holds for all $p \leqslant i \leqslant p'$. Symmetrically, $q' \in \mathrm{reach}_\sigma(p, q)$ is a *greedy step on σ from (p, q)*, if $\mathrm{vis}_\pi(p, i) \subseteq \mathrm{vis}_\pi(p, q')$ for all $q \leqslant i \leqslant q'$. In pseudo code, $\textsc{GreedyStep}_\pi(\pi_{p..n}, \sigma_{q..m})$ denotes a function that returns an arbitrary greedy step p' on π from (p, q) if such an index exists and returns an error otherwise (symmetrically for σ). See Figure 8.3.

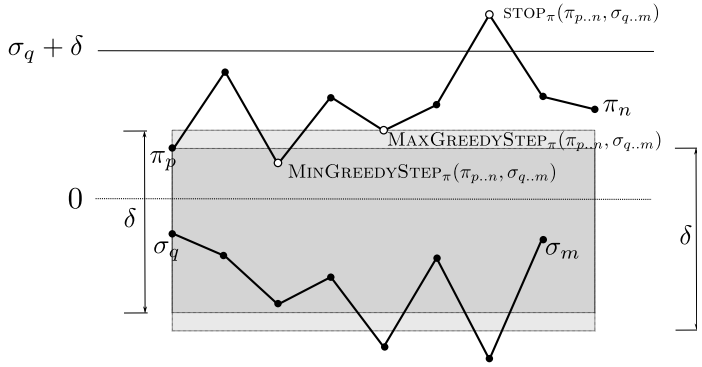


Figure 8.3: An illustration of greedy steps. For better visibility, the one-dimensional separated curves π, σ are drawn in the plane by mapping π_i to (i, π_i). In particular, the results of $\text{MinGreedyStep}_\pi(\pi_{p..n}, \sigma_{q..m})$, $\text{MaxGreedyStep}_\pi(\pi_{p..n}, \sigma_{q..m})$, and $\text{stop}_\pi(\pi_{p..n}, \sigma_{q..m})$ are shown.

Consider the following greedy algorithm:

Algorithm 6 Greedy algorithm for the Fréchet distance of separated curves $\pi_{1..n}$ and $\sigma_{1..m}$ in $\mathbb{R}$

```
1: p ← 1, q ← 1
2: repeat
3:     if p' ← GreedyStep_π(π_{p..n}, σ_{q..m}) then
4:         p ← p'
5:     if q' ← GreedyStep_σ(π_{p..n}, σ_{q..m}) then
6:         q ← q'
7: until no greedy step was found in the last iteration
8: if p = n and q = m then return d_dF(π, σ) ⩽ δ
9: else return d_dF(π, σ) > δ
```

Theorem 8.20. *Let π and σ be separated curves in $\mathbb{R}$ and $\delta > 0$. Algorithm 6 decides whether $d_{\text{dF}}(\pi, \sigma) \leqslant \delta$ in time $\mathcal{O}((n+m)\log(nm))$.*

We will first prove the correctness of the algorithm in Lemma 8.22 below and postpone the discussion how to implement the algorithm efficiently.

Correctness

Note that Algorithm 6 considers potentially only very few points of the curve explicitly during its execution. Call the indices (p, q) of point pairs considered in some iteration of the algorithm (for any choice of greedy steps, if more than one exists) *greedy (point) pairs* and all points contained in some such pair *greedy points (of π and σ).* The following useful monotonicity property holds: If some greedy point on π sees a point on σ that is yet to be traversed, all following greedy points on π will see it *until it is traversed.*

Lemma 8.21. *Let $(p_1, q_1), \ldots, (p_i, q_i)$ be the greedy point pairs considered in the iterations $1, \ldots, i$. It holds that*

1. $\mathrm{vis}_\sigma(\ell, q_i) \subseteq \mathrm{vis}_\sigma(p_i, q_i)$ *for all* $1 \leqslant \ell \leqslant p_i$, *and*

2. $\mathrm{vis}_\pi(p_i, \ell) \subseteq \mathrm{vis}_\pi(p_i, q_i)$ *for all* $1 \leqslant \ell \leqslant q_i$.

Proof. Let $k < i$. We first show that $\mathrm{vis}_\sigma(\ell, q_i) \subseteq \mathrm{vis}_\sigma(p_{k+1}, q_i)$ holds for all $p_k \leqslant \ell < p_{k+1}$. If $p_k = p_{k+1}$, the claim is immediate. Otherwise p_{k+1} is the result of a greedy step on π. By definition of visibility, we have $\mathrm{vis}_\sigma(\ell, q_i) = \mathrm{vis}_\sigma(\ell, q_k) \cap [q_i..m] \subseteq \mathrm{vis}_\sigma(p_{k+1}, q_k) \cap [q_i..m] = \mathrm{vis}_\sigma(p_{k+1}, q_i)$, where the inequality follows from p_{k+1} being a greedy step from (p_k, q_k).

For arbitrary $\ell \leqslant i$, let $k < i$ be such that $p_k \leqslant \ell < p_{k+1}$. Then $\mathrm{vis}_\sigma(\ell, q_i) \subseteq \mathrm{vis}_\sigma(p_{k+1}, q_i) \subseteq \mathrm{vis}_\sigma(p_{k+2}, q_i) \subseteq \cdots \subseteq \mathrm{vis}_\sigma(p_i, q_i)$. The second statement is symmetric. □

We will exploit this monotonicity to prove that if Algorithm 6 finds a greedy point pair that allows no further greedy steps, then no feasible traversal of π and σ exists. We derive an even stronger statement using the following notion: For a greedy point pair (p, q), define $\textsc{stop}_\pi(\pi_{p..n}, \sigma_{q..m}) := \max(\mathrm{reach}_\pi(p, q) \cup \{p\}) + 1$ as the index of the first point after π_p on π which is not seen by σ_q, or $n+1$ if no such index exists. Let $\textsc{stop}_\sigma$ be defined symmetrically.

Lemma 8.22 (Correctness of Algorithm 6)**.** *Let* (p, q) *be a greedy point of* π *and* σ, $p_{\text{stop}} := \textsc{stop}_\pi(\pi_{p..n}, \sigma_{q..m})$ *and* $q_{\text{stop}} := \textsc{stop}_\sigma(\pi_{p..n}, \sigma_{q..m})$. *If on both curves no greedy step from* (p, q) *exists then* $d_{\mathrm{dF}}(\pi, \sigma) > \delta$.

In particular, if $q_{\text{stop}} < m$ *then for all* $1 \leqslant p' \leqslant n$ *we have* $d_{\mathrm{dF}}(\pi_{1..p'}, \sigma_{1..q_{\text{stop}}}) > \delta$, *and if* $p_{\text{stop}} < n$ *then* $d_{\mathrm{dF}}(\pi_{1..p_{\text{stop}}}, \sigma_{1..q'}) > \delta$ *for all* $1 \leqslant q' \leqslant m$.

Note that the correctness of Algorithm 6 follows immediately: If the algorithm is stuck then $d_{\mathrm{dF}}(\pi, \sigma) > \delta$. Otherwise it finds a feasible traversal.

Proof of Lemma 8.22. Consider the case that no greedy step from (p, q) exists, then the following *stuckness conditions* have to hold:

1. For all $p' \in \mathrm{reach}_\pi(p, q)$, we have $\mathrm{vis}_\sigma(p', q) \subsetneq \mathrm{vis}_\sigma(p, q)$, and

2. for all $q' \in \mathrm{reach}_\sigma(p, q)$, we have $\mathrm{vis}_\pi(p, q') \subsetneq \mathrm{vis}_\pi(p, q)$.

In this case, we can extend the monotonicity property of Lemma 8.21 to include all reachable and the first unreachable point.

Claim 8.23. *If the stuckness conditions hold for* (p, q), *then we have* $\mathrm{vis}_\sigma(i, q) \subseteq \mathrm{vis}_\sigma(p, q)$ *for all* $1 \leqslant i \leqslant p_{\text{stop}}$. *In particular, if* π_p *does not see* σ_ℓ *for some* $\ell > q$, *then no vertex* π_i *with* $1 \leqslant i \leqslant p_{\text{stop}}$ *sees* σ_ℓ. *The symmetric statement holds for* σ.

Proof. By the monotonicity of the previous claim, $\mathrm{vis}_\sigma(i, q) \subseteq \mathrm{vis}_\sigma(p, q)$ holds for all $i \leqslant p$. The first of the stuckness conditions implies $\mathrm{vis}_\sigma(i, q) \subseteq \mathrm{vis}_\sigma(p, q)$ for all $p < i < p_{\text{stop}}$. If $p_{\text{stop}} = n + 1$, this already completes the proof of the claim. Otherwise, note that $\pi_{p_{\text{stop}}} > \pi_p$, since otherwise $p_{\text{stop}} \in \mathrm{reach}_\pi(p, q)$. Hence $\mathrm{vis}_\sigma(p_{\text{stop}}, q) \subseteq \mathrm{vis}_\sigma(p, q)$ holds as well. □

We distinguish the following cases that may occur under the stuckness conditions:

Case 1: $p_{\text{stop}} \leqslant n$ or $q_{\text{stop}} \leqslant m$. Without loss of generality, let $p_{\text{stop}} \leqslant n$ (the other case is symmetric). Assume for contradiction that a feasible traversal ϕ of $\pi_{1..p_{\text{stop}}}$ and $\sigma_{1..q'}$ exists for some $1 \leqslant q' \leqslant m$. In ϕ, at some point in time we have to move in π from $p_{\text{stop}} - 1$ to p_{stop} while moving in $\sigma_{1..q'}$ from some $\sigma_{\ell'}$ to σ_ℓ where $\ell' \in \{\ell - 1, \ell\}$ and σ_ℓ sees $\pi_{p_{\text{stop}}}$. Since σ_q does not see $\pi_{p_{\text{stop}}}$, the previous claim shows that $\ell > q_{\text{stop}}$. If $q_{\text{stop}} = m + 1$ or $q_{\text{stop}} < q'$, this is impossible, yielding a contradiction. Otherwise, to do this transition, in some earlier step we have to move in σ from $q_{\text{stop}} - 1$ to q_{stop} while moving in π from $\pi_{k'}$ to π_k for some $k < p_{\text{stop}}$ and $k' \in \{k - 1, k\}$. However, by definition $q_{\text{stop}} \notin \text{vis}_\sigma(p, q)$, hence Claim 8.23 implies that the transition is illegal, since π_k does not see $\sigma_{q_{\text{stop}}}$. This is a contradiction. By a symmetric argument, it holds that $d_{\text{dF}}(\pi_{1..p'}, \sigma_{1..q_{\text{stop}}}) > \delta$.

Case 2: $p_{\text{stop}} = n + 1$ and $q_{\text{stop}} = m + 1$. In this case, $\text{reach}_\pi(p, q) = [p + 1..n]$ and $\text{reach}_\sigma(p, q) = [q + 1..n]$. By stuckness conditions, there exist an index $p_{\max} > p$ such that no $\sigma_{q'}$ with $q' > q$ sees $\pi_{p_{\max}}$ and an index $q_{\min}$ such that no $\pi_{p'}$ with $p' > p$ sees $\sigma_{q_{\min}}$. Assume for contradiction that a feasible traversal ϕ exists. In ϕ, at some point in time t, we have to cross either (1) from π_p to π_{p+1} while moving in σ from $\sigma_{\ell'}$ to σ_ℓ with $\ell \leqslant q + 1 \leqslant q_{\min}$ and $\ell' \in \{\ell - 1, \ell\}$ or (2) from σ_q to σ_{q+1} while moving from $\pi_{\ell'}$ to π_ℓ with $\ell \leqslant p + 1 \leqslant p_{\max}$ and $\ell' \in \{\ell - 1, \ell\}$. In the first case, $\ell < q_{\min}$ holds, since π_{p+1} does not see $\sigma_{q_{\min}}$. For all consecutive times $t' \geqslant t$, ϕ is in a point $\pi_{p'}$ ($p' \geqslant p + 1$) that does not see $\sigma_{q_{\min}}$, which still has to be traversed, leading to a contradiction. Symmetrically, in the second case, for all times $t' \geqslant t$, ϕ is in a point $\sigma_{q'}$ ($q' \geqslant q + 1$) that does not see $\pi_{p_{\max}}$, which still has to be traversed.

This concludes the proof of Lemma 8.22. □

Implementing greedy steps

To prove Theorem 8.20, it remains to show how to implement the algorithm to run in time $\mathcal{O}((n + m) \log(nm))$. We make use of geometric range search queries. The classic technique of fractional cascading [137–139] provides a data structure D with the following properties: (i) Given n points $\mathcal{P}$ in the plane, $D(\mathcal{P})$ can be constructed in time $\mathcal{O}(n \log n)$ and (ii) given a query rectangle $Q := I_1 \times I_2$ with intervals I_1 and I_2, find and return $q \in Q \cap \mathcal{P}$ with minimal y-coordinate, or report that no such point exists, in time $\mathcal{O}(\log n)$. Here, each interval I_i may be open, half-open or closed.

By invoking the above data structure on $\mathcal{P} := \{(i, \pi_i) \mid i \in [1 \dots n]\}$ for a given curve $\pi = \pi_{1..n}$ (as well as all three rotations of $\mathcal{P}$ by multiples of 90°), we obtain a data structure D^π such that:

1. D^π can be constructed in time $\mathcal{O}(n \log n)$,
2. the query $\text{D}^\pi.\textsc{minIndex}([x_1, x_2], [p, b])$ ($\text{D}^\pi.\textsc{maxIndex}([x_1, x_2], [p, b])$) returns the minimum (maximum) index $p \leqslant i \leqslant b$ such that $x_1 \leqslant \pi_i \leqslant x_2$ in time $\mathcal{O}(\log n)$, and
3. the query $\text{D}^\pi.\textsc{minHeight}([x_1, x_2], [p, b])$ ($\text{D}^\pi.\textsc{maxHeight}([x_1, x_2], [p, b])$) returns the minimum (maximum) height $x_1 \leqslant \pi_i \leqslant x_2$ such that $p \leqslant i \leqslant b$ in time $\mathcal{O}(\log n)$.

The queries extend naturally to open and half-open intervals. If no index exists in the queried range, all of these operations return the index ∞. We will use the corresponding data structure D^σ for σ as well.

With these tools, we implement the following basic operations for arbitrary subcurves $\pi' := \pi_{p..b}$ and $\sigma' := \sigma_{q..d}$ of π and σ. See also Figure 8.3.

1. **Stopping points $\mathbf{stop}_\pi(\pi', \sigma')$.** For points p, q,

$$\textsc{stop}_\pi(\pi', \sigma') := \max(\mathrm{reach}_{\pi'}(p, q) \cup \{p\}) + 1$$

returns the index of the first point after π_p on π' which is not seen by σ_q, or $b + 1$ if no such index exists.

Algorithm 7 Finding the stopping point

```
1: function STOP_π(π_{p..b}, σ_{q..d})
2:     p_stop ← D^π.minIndex((σ_q + δ, ∞), [p, b])      ▷ First non-visible point on π
3:     if p_stop < ∞ then return p_stop
4:     else return b + 1
```

2. **Minimal greedy steps $\mathbf{MinGreedyStep}_\pi(\pi', \sigma')$.** This function returns the smallest index $p' \in \mathrm{reach}_{\pi'}(p, q)$ such that $\mathrm{vis}_{\sigma'}(p', q) \supseteq \mathrm{vis}_{\sigma'}(p, q)$ or reports that no such index exists.

Algorithm 8 Minimal greedy step

```
1: function MinGreedyStep_π(π_{p..b}, σ_{q..d})
2:     q_min ← D^σ.minHeight([π_p − δ, ∞), [q, d])      ▷ Lowest still visible point on σ
3:     p_cand ← D^π.minIndex((−∞, σ_{q_min} + δ], [p + 1, d])   ▷ If p' exists, it is p_cand
4:     p_stop ← STOP_π(π_{p..b}, σ_{q..d})              ▷ First non-visible point on π
5:     if p_cand < p_stop then return p_cand
6:     else return "No greedy step possible."           ▷ π_{p_cand} not reachable from π_p
```

3. **Maximal greedy steps $\mathbf{MaxGreedyStep}_\pi(\pi', \sigma')$.** Let $p' \in \mathrm{reach}_{\pi'}(p, q)$ be such that (i) p' is the largest index maximizing $|\mathrm{vis}_{\sigma'}(z, q)|$ among all $z \in \mathrm{reach}_{\pi'}(p, q)$ and (ii) $\mathrm{vis}_{\sigma'}(p', q) \supseteq \mathrm{vis}_{\sigma'}(p, q)$. If p' exists, $\textsc{MaxGreedyStep}_\pi$ returns this value, otherwise it reports that no such index exists. Note that if p' exists, then by definition there is no greedy step on π starting from (p', q), i.e., this step is a maximal greedy step.

4. **Arbitrary greedy steps $\mathbf{GreedyStep}_\pi(\pi', \sigma')$.** If, in some situation, it is only required to find an arbitrary index $p' \in \mathrm{reach}_{\pi'}(p, q)$ such that all $p \leqslant i \leqslant p'$ satisfy $\mathrm{vis}_{\sigma'}(i, q) \subseteq \mathrm{vis}_{\sigma'}(p', q)$ or report that no such index exists, we use the function $\textsc{GreedyStep}_\pi(\pi', \sigma')$ to denote that any such function suffices; in particular, $\textsc{MinGreedyStep}_\pi$ or $\textsc{MaxGreedyStep}_\pi$ can be used.

For σ, we define the obvious symmetric operations. Note that in these operations, it is not feasible to traverse all directly feasible points and check whether the visibility

Algorithm 9 Maximal greedy step

```
1: function MaxGreedyStep_π(π_{p..b}, σ_{q..d})
2:     q_min ← D^σ.minHeight([π_p − δ, ∞), [q, d])   ▷ Lowest still visible point on σ
3:     p_stop ← stop_π(π_{p..b}, σ_{q..d})            ▷ First non-visible point on π
4:     p_min ← D^π.minHeight((−∞, σ_{q_min} + δ], [p + 1, p_stop − 1])
5:                                  ▷ Maximizes visibility among reachable points
6:     if p_min = ∞ then        ▷ No reachable point has better visibility than π_p
7:         return "No greedy step possible."
8:     else
9:         q_min ← D^σ.minHeight([π_{p_min} − δ, ∞), [q, d])
10:                                  ▷ Lowest point on σ still seen by p_min
11:        return D^π.maxIndex((−∞, σ_{q_min} + δ], [p_min, p_stop − 1])
```

criterion is satisfied, since this would not necessarily yield a polylogarithmic running time.

Lemma 8.24. *Using $O((n+m)\log nm)$ preprocessing time,* MaxGreedyStep$_\pi$, MinGreedyStep$_\pi$ *and* stop$_\pi$ *can be implemented to run in time $\mathcal{O}(\log nm)$.*

Proof. In time $\mathcal{O}((n+m)\log nm)$, we can build the data structure D^π for π and symmetrically D^σ for σ. Algorithms 7, 8 and 9 implement the greedy steps and stop$_\pi$ using only a constant number of queries to D^π and D^σ, each with running time $\mathcal{O}(\log n)$ or $\mathcal{O}(\log m)$. □

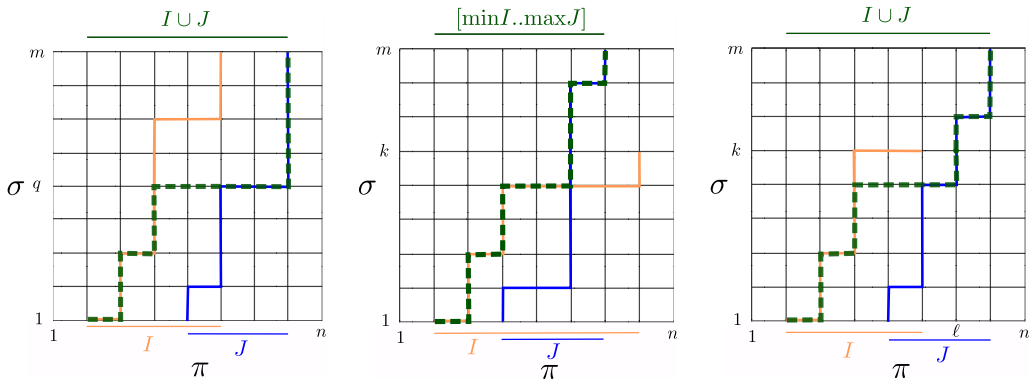
For the reduced free-space problem, these operations can be implemented even faster.

Lemma 8.25. *Let $\pi = \pi_{1..n}$ and $\sigma = \sigma_{1..m}$ be input curves of the reduced free-space problem. Using $O((n+m)\log 1/\varepsilon)$ preprocessing time,* MaxGreedyStep$_\pi$, MinGreedyStep$_\pi$ *and* stop$_\pi$ *can be implemented to run in time $\mathcal{O}(\log 1/\varepsilon)$.*

Proof. We argue that range searching can be implemented with $\mathcal{O}(\log 1/\varepsilon)$ query time and $\mathcal{O}(n \log 1/\varepsilon)$ preprocessing time. This holds since for the point set $\mathcal{P} = \{(i, \pi_i) \mid i \in [1 \dots n]\}$ (1) the x-values are $1, \dots, n$, so that we can determine the relevant pointers in the first level of the fractional-cascading tree in constant time instead of $\mathcal{O}(\log n)$ and (2) all y-values are multiples of $\gamma = \frac{1}{4}\varepsilon\delta$ and in $[-2\delta, 2\delta]$, i.e., there are only $\mathcal{O}(1/\varepsilon)$ different y-values. For the latter, note that any point $\pi_p > \delta$ sees no point in σ, and this is preserved by setting π_p to 2δ (and similarly for σ). Using these properties it is straightforward to adapt the fractional-cascading data structure, we omit the details. □

8.3.3. Composition of One-Dimensional Curves

In this subsection, we collect essential composition properties of feasible traversals of one-dimensional curves that enable us to tackle the reduced free-space problem (see Figure 8.4 for an illustration of these results). The first tool is a union lemma that states that two intersecting intervals I, J of π that each have a feasible traversal together with σ prove that also $\pi_{I \cup J}$ can be traversed together with σ.



(a) Lemma 8.26. **(b)** Lemma 8.27. **(c)** Lemma 8.28.

Figure 8.4: Composition properties of feasible traversals of one-dimensional separated curves.

Lemma 8.26. *Let $\pi = \pi_{1..n}$ and $\sigma = \sigma_{1..m}$ be one-dimensional separated curves and let $I, J \subseteq [1..n]$ be intervals with $I \cap J \neq \emptyset$. If $d_{\mathrm{dF}}(\pi_I, \sigma) \leqslant \delta$ and $d_{\mathrm{dF}}(\pi_J, \sigma) \leqslant \delta$, then $d_{\mathrm{dF}}(\pi_{I \cup J}, \sigma) \leqslant \delta$.*

Proof. If $I \subseteq J$, the claim is trivial. W.l.o.g, let $I = [a_I..b_I]$ and $J = [a_J..b_J]$, where $a_I \leqslant a_J \leqslant b_I \leqslant b_J$. Let ϕ_I (and ϕ_J) be a feasible traversal of (π_I, σ) (and (π_J, σ), respectively). By reparameterization, we can assume that $\phi_I(t) = (\psi_I(t), f(t))$ and $\phi_J(t) = (\psi_J(t), f(t))$ for suitable (non-decreasing onto) functions $\psi_I, \psi_J : [0,1] \to [1..n]$ and $f : [0,1] \to [1..m]$. One of the following cases occurs.

Case 1: There is some $0 \leqslant t \leqslant 1$ with $\psi_I(t) = \phi_J(t)$. Then we can concatenate $\phi_I(0,t)$ and $\phi_J(t,1)$ to obtain a feasible traversal of $\phi_{I \cup J}$.

Case 2: For all $0 \leqslant t \leqslant 1$, we have $\psi_I(t) < \psi_J(t)$. Let σ_q be the highest point on σ. By $d_{\mathrm{dF}}(\pi_I, \sigma) \leqslant \delta$ and $d_{\mathrm{dF}}(\pi_J, \sigma) \leqslant \delta$, the point σ_q sees all points on $\pi_{I \cup J}$. There is some $0 \leqslant t^* \leqslant 1$ with $f(t^*) = q$. We can concatenate $\phi_I(0,t)$ and the traversal of $\pi_{\psi_I(t^*)..\psi_J(t^*)}$ and σ_q to obtain a feasible traversal of $\pi_{a_I..\psi_I(t^*)}$ and $\sigma_{1..f(t^*)}$. Appending $\phi_J(t^*, 1)$ to this traversal yields $d_{\mathrm{dF}}(\pi_{a_I..b_J}, \sigma) \leqslant \delta$. □

The second result formalizes situations in which a traversal ϕ of sub-curves has to cross a traversal ψ of other sub-curves, yielding the possibility to follow ϕ up to the crossing point and to follow ψ from there on.

Lemma 8.27. *Let $\pi = \pi_{1..n}$ and $\sigma = \sigma_{1..m}$ be one-dimensional curves and consider intervals $I = [a_I..b_I]$ and $J = [a_J..b_J]$ with $J \subseteq I \subseteq [1..n]$, and $K = [1..k] \subseteq [1..m]$. If $d_{\mathrm{dF}}(\pi_I, \sigma_K) \leqslant \delta$ and $d_{\mathrm{dF}}(\pi_J, \sigma) \leqslant \delta$, then $d_{\mathrm{dF}}(\pi_{a_I..b_J}, \sigma) \leqslant \delta$.*

Proof. Let ϕ be a feasible traversal of π_I and σ_K and ψ a feasible traversal of π_J and σ. We first show that ϕ and ψ cross, i.e., there are $0 \leqslant t, t' \leqslant 1$ such that $\phi(t) = \psi(t')$. For all $k \in [1..K]$, let $[s_k^\phi..e_k^\phi]$ denote the interval of points that ϕ traverses on π while staying in σ_k. Similarly, $[s_k^\psi..e_k^\psi]$ denotes the interval of points ψ traverses on π while staying in σ_k. Assume for contradiction that $[s_k^\phi..e_k^\phi]$ and $[s_k^\psi..e_k^\psi]$ are disjoint for all $1 \leqslant k \leqslant K$. Then initially, we have $s_1^\phi = a_I \leqslant a_J = s_1^\psi$ and hence $e_1^\phi < s_1^\psi$. This implies $s_2^\phi \leqslant e_1^\phi + 1 \leqslant s_1^\psi \leqslant s_2^\psi$ and inductively we obtain

$e_k^\phi < s_k^\psi \leqslant e_k^\psi$ for all $k \in [1..K]$. This contradicts $e_K^\phi = b_I \geqslant b_J \geqslant e_K^\psi$. Hence, for some $1 \leqslant k \leqslant K$, $[s_k^\phi..e_k^\phi]$ and $[s_k^\psi..e_k^\psi]$ intersect, which gives $\phi(t) = (p,k) = \psi(t')$ for any $p \in [s_k^\phi, e_k^\phi] \cap [s_k^\psi, e_k^\psi]$ and the corresponding $0 \leqslant t, t' \leqslant 1$. By concatenating $\phi(0,t)$ with $\psi(t',1)$, we obtain a feasible traversal of $\pi_{a_I..b_J}$ and σ. □

The last result in our composition toolbox strengthens Lemma 8.26 to the case that the traversal of π_I uses only an initial sub-curve $\sigma_{1..k}$ of σ and not the complete curve.

Lemma 8.28. *Let $\pi = \pi_{1..n}$ and $\sigma = \sigma_{1..m}$ be one-dimensional separated curves and consider intervals $I = [a_I..b_I]$ and $J = [a_J..b_J]$ with $1 \leqslant a_I \leqslant a_J \leqslant b_I \leqslant b_J \leqslant n$, and $K = [1..k] \subseteq [1..m]$. If $d_{\mathrm{dF}}(\pi_I, \sigma_K) \leqslant \delta$ and $d_{\mathrm{dF}}(\pi_J, \sigma) \leqslant \delta$, then $d_F(\pi_{I \cup J}, \sigma) \leqslant \delta$.*

Proof. Let ϕ be any feasible traversal of π_J and σ. There exists $a_J \leqslant \ell \leqslant b_J$ with $\phi(t) = (\ell, k)$ for some $0 \leqslant t \leqslant 1$. Hence ϕ restricted to $[0,t]$ yields a feasible traversal of $\pi_{a_J..\ell}$ and σ_K, i.e., $d_{\mathrm{dF}}(\pi_{a_J..\ell}, \sigma_K) \leqslant \delta$. Since I and $[a_J..\ell]$ are intersecting, Lemma 8.26 yields that $d_{\mathrm{dF}}(\pi_{a_I..\ell}, \sigma_K) \leqslant \delta$. Let ψ be such a feasible traversal of $\pi_{a_I..\ell}$ and σ_K. Concatenating ψ at $\psi(1) = (\ell, k) = \phi(t)$ with $\phi(t,1)$, we construct a feasible traversal of $\pi_{a_I..b_J}$ and σ, proving the claim. □

8.3.4. Solving the Reduced Free-Space Problem

In this section, we solve the reduced free-space problems, using the structural properties derived in the previous section and the principles underlying the greedy algorithm of Section 8.3.2. Recall that the greedy steps implemented as discussed in Section 8.3.2 run in time $\mathcal{O}(\log 1/\varepsilon)$ on the input curves of the reduced free-space problem.

Single Entry

Given the separated curves $\pi = (\pi_1, \ldots, \pi_n)$ and $\sigma = (\sigma_1, \ldots, \sigma_m)$ and entry set $E = \{1\}$, we show how to compute F^σ. We present the following recursive algorithm.

Algorithm 10 Special Case: Single entry

```
1:  function FIND-σ-EXITS(π_{p..b}, σ_{q..d})
2:      if q = d then
3:          if STOP_π(π_{p..b}, σ_q) = b + 1 then
4:              return {q}                ▷ The end of π is reachable while staying in σ_q
5:          else return ∅

6:      if p' ← MAXGREEDYSTEP_π(π_{p..b}, σ_{q..d}) then
7:          return FIND-σ-EXITS(π_{p'..b}, σ_{q..d}))
8:      else if q' ← GREEDYSTEP_σ(π_{p..b}, σ_{q..d}) then
9:          return FIND-σ-EXITS(π_{p..b}, σ_{q..q'−1}) ∪ FIND-σ-EXITS(π_{p..b}, σ_{q'..d})
10:     else
11:         return FIND-σ-EXITS(π_{p..b}, σ_{q..d−1})        ▷ No greedy step possible
```

The following property establishes that a greedy step on a long curve is also a greedy step on a shorter curve. Clearly, the converse does not necessarily hold.

Proposition 8.29. *Let $1 \leqslant p \leqslant P \leqslant n$ and $1 \leqslant q \leqslant Q \leqslant m$. Any greedy step on π from (p, q) to (p', q) with $p' \leqslant P$ is also a greedy step with respect to $\tilde{\pi} := \pi_{p..P}$ and $\tilde{\sigma} := \sigma_{q..Q}$, i.e., if there is some $p' \leqslant P$ with $\mathrm{vis}_\sigma(i, q) \subseteq \mathrm{vis}_\sigma(p', q)$ for all $p \leqslant i \leqslant p'$, then also $\mathrm{vis}_{\tilde{\sigma}}(i, q) \subseteq \mathrm{vis}_{\tilde{\sigma}}(p', q)$.*

Proof. From the definition of vis_σ, we immediately derive $\mathrm{vis}_{\tilde{\sigma}}(i, q) = \mathrm{vis}_\sigma(i, q) \cap [q..Q] \subseteq \mathrm{vis}_\sigma(p', q) \cap [q..Q] = \mathrm{vis}_{\tilde{\sigma}}(p', q)$ for all $p \leqslant i \leqslant p'$. Restricting the length of π also has no influence on the greedy property, except for the trivial requirement that p' still has to be contained in the restricted curve. □

Lemma 8.30. *Algorithm 10 correctly identifies F^σ given the single entry $E = \{1\}$.*

Proof. Clearly, if FIND-σ-EXITS(π, σ) finds and returns an exit e on σ, then it is contained in F^σ, since the algorithm uses only feasible (greedy) steps. Conversely, we show that for all $I = [p..b]$ and $J = [q..d]$, where (p, q) is a greedy point pair of π and σ, and all $e \in J$ with $d_{\mathrm{dF}}(\pi_I, \sigma_{J \cap [1..e]}) \leqslant \delta$, we have $e \in$ FIND-σ-EXITS(π_I, σ_J), i.e. we find all exits.

Consider some call of FIND-σ-EXITS(π_I, σ_J) for which the precondition is fulfilled. If J consists only of a single point, then $J = \{e\}$, and a feasible traversal of π_I and σ_J exists if and only if σ_e sees all points on π_I. Let $I = [p..b]$, then this happens if and only if $\mathrm{STOP}_\pi(\pi_I, \sigma_e) = b + 1$, hence the base case is treated correctly.

Assume that $I = [p..b]$ and a maximal greedy step p' on π exists. By Property 8.29, this step is a greedy step also with respect to $\sigma_{J \cap [1..e]}$. Hence by Lemma 8.22, if there is a traversal of $\pi_{p..b}$ and $\sigma_{J \cap [1..e]}$, then a traversal of $\pi_{[p'..b]}$ and $\sigma_{J \cap [1..e]}$ also exists.

Consider the case in which $J = [q..d]$ and a greedy step q' in σ exists. If $e < q'$, then $e \in [q..q'-1]$ and $J \cap [1..e] = [q..q'-1] \cap [1..e]$. Hence, e is found in the recursive call with $J' = [q..q'-1]$. If $e \geqslant q'$, then by Property 8.29, this step is a greedy step with respect to the curves π_I and $\sigma_{J \cap [1..e]}$. Again, by Lemma 8.22, the existence of a feasible traversal of π_I and σ_J implies that also a feasible traversal of π_I and $\sigma_{J \cap [q'..e]}$ exists.

It remains to regard the case in which no greedy step exists. By Lemma 8.22, there is no feasible traversal of $\pi_{1..n}$ and $\sigma_{1..d}$. This implies $e \neq d$ and all exits are found in the recursive call with $J' = [q, d-1]$. □

Lemma 8.31. FIND-σ-EXITS$(\pi_{p..b}, \sigma_{q..d})$ *runs in time $\mathcal{O}((d - q + 1) \cdot \log 1/\varepsilon)$.*

Proof. Since the algorithm's greedy steps on π are maximal, after each greedy step on π, we split σ (by a greedy step on σ) or shorten σ (if no greedy step on σ is found). Thus, it takes at most $\mathcal{O}(\log 1/\varepsilon)$ time until σ is split or shortened. The base case is also handled in time $\mathcal{O}(\log 1/\varepsilon)$. In total, this yields a running time of $\mathcal{O}((d - q + 1) \log 1/\varepsilon)$. □

Note that by swapping the roles of π and σ, FIND-σ-EXITS can be used to determine F^π given the single entry σ_1 on σ. This is equivalent to having the single entry $E = \{1\}$ on π. Thus, we can also implement the function FIND-π-EXITS$(\pi_{1..n}, \sigma_{1..m})$ that returns F^π given the single entry $E = \{1\}$ in time $\mathcal{O}(n \log 1/\varepsilon)$.

Entries on π, Exits on π

In this section, we tackle the task of determining F^π given a set of entries E on π. It is essential to avoid computing the exits by iterating over every single entry. We show how to divide π into disjoint sub-curves that can be solved by a single call to Find-π-Exits each.

Assume we want to traverse $\pi_{p..b}$ and $\sigma_{q..d}$ starting in π_p and σ_q. Let $u(p) := \max\{p' \in [p, b] \mid \exists q \leqslant q' \leqslant d : d_{\mathrm{dF}}(\pi_{p..p'}, \sigma_{q..q'}) \leqslant \delta\}$ be the last point on π that is reachable while traversing an arbitrary sub-curve of $\sigma_{q..d}$ that starts in σ_q. This point fulfills the following properties.

Lemma 8.32. *It holds that*

1. *If there are $p \leqslant e \leqslant e' \leqslant u(p)$ with $d_{\mathrm{dF}}(\pi_{e..e'}, \sigma_{q..d}) \leqslant \delta$, then $d_{\mathrm{dF}}(\pi_{p..e'}, \sigma_{q..d}) \leqslant \delta$.*

2. *For all $p \leqslant e \leqslant u(p) < e'$, we have that $d_{\mathrm{dF}}(\pi_{e..e'}, \sigma_{q..d}) > \delta$.*

Proof. By definition of $u(p)$, there is a $q \leqslant q' \leqslant d$ with $d_{\mathrm{dF}}(\pi_{p..u(p)}, \sigma_{q..q'}) \leqslant \delta$. Since $[e, e'] \subseteq [p, u(p)]$, Lemma 8.27 proves the first statement. For the second statement, assume for contradiction that $d_{\mathrm{dF}}(\pi_{e..e'}, \sigma_{q..d}) \leqslant \delta$. Then, Lemma 8.28 yields that $d_{\mathrm{dF}}(\pi_{p..e'}, \sigma_{q..d}) \leqslant \delta$. This is a contradiction to the choice of $u(p)$, since $e' > u(p)$. □

The above lemma implies that we can ignore all entries in $[p..u(p)]$ except for p and that all exits reachable from p are contained in the interval $[p..u(p)]$. This gives rise to the following algorithm.

Algorithm 11 Given entry points E on π, compute all exits on π.

```
1:  function π-EXITS-FROM-π(π, σ, E)
2:      S ← ∅
3:      while E ≠ ∅ do
4:          p̂ ← pop minimal index from E
5:          p ← p̂, q ← 1
6:          repeat
7:              if q' ← MaxGreedyStep_σ(π_{p..n}, σ_{q..m}) then
8:                  q ← q'
9:              if p' ← GreedyStep_π(π_{p..n}, σ_{q..m}) then
10:                 p ← p'
11:         until no greedy step was found in the last iteration
12:         p̄ ← stop_π(π_{p..n}, σ_{q..m}) − 1    ▷ equals the maximal reachable point u(p̂)
13:         S ← S ∪ Find-π-Exits(π_{p̂..p̄}, σ)
14:         E ← E ∩ [p̄ + 1, n]                    ▷ drops all entries in [p̂, u(p̂)]
15:     return S
```

Lemma 8.33. *Algorithm 11 correctly computes F^π.*

Proof. We first argue that for each considered entry $\hat{p}$, the algorithm computes $\overline{p} = u(\hat{p})$. Clearly, $\overline{p} \leqslant u(\hat{p})$, since only feasible steps are used to reach $\overline{p}$. If

$\overline{p} = m$, this already implies that also $u(\hat{p}) = m$. Otherwise, let (p, q) be the greedy point pair on the curves $\pi_{\hat{p}..n}$ and σ for which no greedy step has been found. Then by Lemma 8.22, for $p_{\text{stop}} := \text{STOP}_\pi(\pi_{p..n}, \sigma_{q..m})$ and all $1 \leqslant q' \leqslant m$, we have that $d_{\text{dF}}(\pi_{\hat{p}..p_{\text{stop}}}, \sigma_{1..q'}) > \delta$. Hence, $u(\hat{p}) < p_{\text{stop}}$. Finally, note that Algorithm 11 computes $\overline{p} = p_{\text{stop}} - 1$, which proves $\overline{p} = u(\hat{p})$.

It is clear that every found exit is included in F^π. Conversely, let $e' \in F^\pi$ and $1 \leqslant e \leqslant n$ be such that $d_{\text{dF}}(\pi_{e..e'}, \sigma) \leqslant \delta$. For some $\hat{p}$ with $\hat{p} \leqslant e \leqslant u(\hat{p}) = \overline{p}$, we run FIND-$\pi$-EXITS$(\pi_{\hat{p}..\overline{p}}, \sigma)$. Hence by Lemma 8.32 (2), $e' \leqslant u(\hat{p})$ and by Lemma 8.32 (1), $d_{\text{dF}}(\pi_{\hat{p}..e'}, \sigma) \leqslant \delta$. Hence, the corresponding call FIND-π-EXITS$(\pi_{\hat{p}..\overline{p}}, \sigma)$ will find e'. □

Lemma 8.34. *Using preprocessing time $\mathcal{O}((n+m)\log 1/\varepsilon)$, Algorithm 11 runs in time $\mathcal{O}(n \log 1/\varepsilon)$.*

Proof. We first bound the cost of all calls FIND-π-EXITS(π_{I_i}, σ). Clearly, all intervals I_i are disjoint with $\bigcup I_i \subseteq [1..n]$. Hence, by Lemma 8.31, the total time spent in these calls is bounded by $\mathcal{O}(\sum_i |I_i| \log(1/\varepsilon)) = \mathcal{O}(n \log 1/\varepsilon)$. To bound the number greedy steps, let $p_1, \ldots, p_k$ be the distinct indices considered as values of p during the execution of π-EXITS-FROM-$\pi(\pi, \sigma)$. Between changing p from each p_i to p_{i+1}, we will make, by maximality, at most one call to MAXGREEDYSTEP$_\sigma$ and at most one call to GREEDYSTEP$_\pi$. Since $k \leqslant n$, the total cost of greedy calls is bounded by $\mathcal{O}(n \log 1/\varepsilon)$ as well. The total time spent in all other operations is bounded by $\mathcal{O}(n \log 1/\varepsilon)$. □

Entries on π, Exits on σ

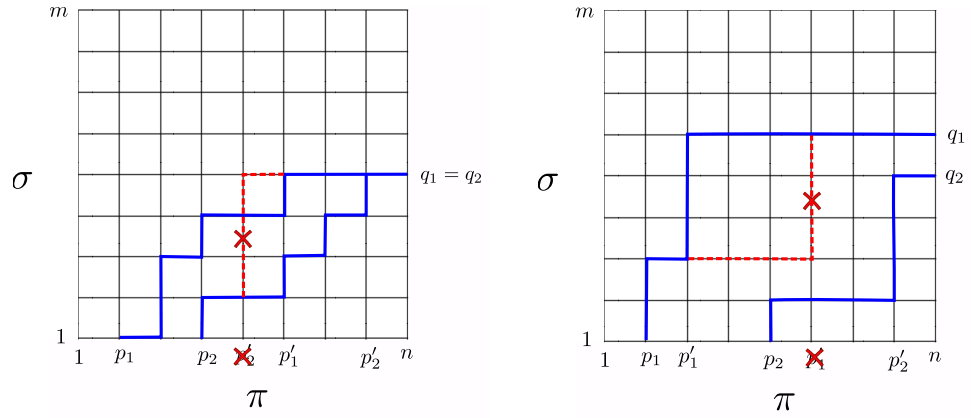
Similar to the previous section, we show how to compute the exits F^σ given entries E on π, by reducing the problem to calls of FIND-σ-EXITS on sub-curves of π and σ. This time, however, the task is more intricate. For any index p on π, let $Q(p) := \min\{q \mid d_{\text{dF}}(\pi_{p..n}, \sigma_{1..q}) \leqslant \delta\}$ be the endpoint of the shortest initial fragment of σ such that the remaining part of π can be traversed together with this fragment[2]. Let $P(p) := \min\{p' \mid d_{\text{dF}}(\pi_{p..p'}, \sigma_{1..Q(p)}) \leqslant \delta\}$ be the endpoint of the shortest initial fragment of π, such that $\sigma_{Q(p)}$ can be reached by a feasible traversal.

Note that by definition, entries p with $Q(p) = \infty$ are irrelevant for determining the exits on σ. In fact, if an entry p is *relevant*, i.e., $Q(p) < \infty$, it is easy to compute $Q(p)$ due to the following lemma.

Lemma 8.35. *Let $Q'(p) := \min\{q \mid \sigma_q \geqslant \max_{i \in [p..n]} \pi_i - \delta\}$. If $Q(p) < \infty$, then $Q(p) = Q'(p)$. Similarly, $Q(p) < \infty$ implies that $P(p) = \min\{p' \mid \pi_{p'} \leqslant \min_{i \in [q..Q(p)]} \sigma_i + \delta\} < \infty$.*

Proof. Assume that $Q(p) < Q'(p)$ holds, then no point in $\sigma_{1..Q(p)}$ sees the highest point in $\pi_{p..n}$. Hence no feasible traversal of these curves can exist, yielding a contradiction. Assume that $Q(p) > Q'(p)$ holds instead and consider the feasible traversal ϕ of the shortest initial fragment of σ that passes through all points in $\pi_{p..n}$. At some point ϕ visits $(\pi_{p'}, \sigma_{Q'(p)})$ for some $p \leqslant p' \leqslant n$. We can alter this traversal to pass through the remaining curve $\pi_{p'..n}$ while staying in $\sigma_{Q'(p)}$, since

[2] As a convention, we use $\min \emptyset = \max \emptyset = \infty$.



(a) Case $q_1 = q_2$. **(b)** Case $q_1 > q_2$.

Figure 8.5: Illustration of Lemma 8.36. For both p_i, $i \in \{1, 2\}$, a feasible traversal of the curves $\pi_{p_i..p'_i}$ and $\sigma_{1..q_i}$ is depicted as monotone paths in the free-space.

$\sigma_{Q'(p)}$ sees all points on $\pi_{p'..n}$. This gives a feasible traversal of $\pi_{p..n}$ and $\sigma_{1..Q'(p)}$, which is a contradiction to the choice of ϕ and $Q(p) > Q'(p)$.

The statement for $P(p)$ follows analogously by regarding the curves $\pi_{p..n}$ and $\sigma_{1..Q(p)}$ and switching their roles. □

Note that the previous lemma shows that for relevant entries $p_1 < p_2$, we have $Q(p_1) \geqslant Q(p_2)$, since for relevant entries, $Q(p_1) = Q'(p_1) \geqslant Q'(p_2) = Q(p_2)$. We will use the following lemma to argue that (i) if $Q(p_1) = Q(p_2)$, entry p_1 dominates p_2, and (2) if $Q(p_1) > Q(p_2)$, we have $p_2 \notin [p_1..P(p_1)]$. Hence, we can ignore all entries in $[p_1..P(p_1)]$ except for p_1 itself.

Lemma 8.36. *Let $p_1 < p_2$ be indices on π with $q_1 := Q(p_1) < \infty$ and $q_2 := Q(p_2) < \infty$. Let $p'_1 := P(p_1)$ and $p'_2 := P(p_2)$. If $q_1 = q_2$, then $p'_1 \leqslant p'_2$. Otherwise, i.e., if $q_1 > q_2$, we even have $p'_1 < p_2$.*

Proof. See Figure 8.5 for illustrations. Let $q_1 = q_2$. Assume for contradiction that $p'_1 > p'_2$, then we have $d_{\mathrm{dF}}(\pi_{p_1..p'_1}, \sigma_{1..q_1}) \leqslant \delta$ and $d_{\mathrm{dF}}(\pi_{p_2..p'_2}, \sigma_{1..q_1}) \leqslant \delta$, where $[p_2..p'_2] \subseteq [p_1..p'_1]$. Hence by Lemma 8.27, $d_{\mathrm{dF}}(\pi_{p_1..p'_2}, \sigma_{1..q_1}) \leqslant \delta$ and thus $p'_1 \leqslant p'_2$, which is a contradiction to the assumption.

For the second statement, let p be maximal such that $\pi_p > \sigma_{q_2} + \delta$. If p does not exist or $p < p_1$, we have that $Q'(p_1) = Q'(p_2)$ and hence by Lemma 8.35, $q_1 = q_2$. Note that additionally $p < p_2$, since otherwise $\sigma_{q_2} < \pi_p - \delta$ with $p \geqslant p_2$ shows that $q_2 \neq Q'(p_2)$ contradicting Lemma 8.35. Thus, in what follows, we can assume that $p_1 < p < p_2$.

Assume for contradiction that $q_1 > q_2$ and $p'_1 \geqslant p_2$. Then a feasible traversal ϕ of $\pi_{p_1..p'_1}$ and $\sigma_{1..q_1}$ visits (π_p, σ_q) for some $1 \leqslant q \leqslant q_1$. It even holds that $q < q_1$, since otherwise there is a feasible traversal of $\sigma_{1..q_1}$ and $\pi_{p_1..p}$ with $p < p'_1$, contradicting the choice of p'_1. Clearly, $\sigma_q > \sigma_{q_2}$, since π_p sees σ_q, while it does not see σ_{q_2}. Since by choice of p, σ_{q_2} sees all of $\pi_{p+1..n}$ and σ_q sees only more (including π_p), we conclude that we can traverse all points of $\pi_{p..n}$ while staying in σ_q. Concatenating

this traversal to the feasible traversal ϕ yields $d_{\mathrm{dF}}(\pi_{p_1..n}, \sigma_{1..q}) \leqslant \delta$ and thus $Q'(p_1) \leqslant q < q_1$, which is a contradiction to Lemma 8.35. This proves that $q_1 > q_2$ implies $p'_1 \leqslant p_2$. □

Algorithm 12 Given entry points E on π, compute all exits on σ.

```
1:  function σ-EXITS-FROM-π(π, σ, E)
2:      F ← ∅, q̄ ← m
3:      repeat
4:          p̂ ← pop minimal index from E
5:          p ← p̂, q ← 1
6:          Q' ← Q'(p)
7:          repeat
8:              if q' ← MaxGreedyStep_σ(π_{p..n}, σ_{q..Q'}) then
9:                  q ← q'
10:             if q ≠ Q' and p' ← MinGreedyStep_π(π_{p..n}, σ_{q..Q'}) then
11:                 p ← p'
12:         until q = Q' or no greedy step was found in the last iteration
13:         if q = Q' then
14:             F ← F ∪ Find-σ-Exits(π_{p..n}, σ_{Q'..q̄})
15:             q̄ ← Q' − 1
16:         E ← E ∩ [p + 1, n]
17:     until E = ∅
18:     return F
```

Lemma 8.37. *Algorithm 12 fulfills the following properties.*

1. *Let (p, q) with $q < Q'(\hat{p})$ be a greedy point pair of $\pi_{\hat{p}..n}$ and $\sigma_{1..Q'(\hat{p})}$ for which no greedy step exists. For all $e \in [\hat{p}, p]$, we have $Q(e) = \infty$.*

2. *For each considered $\hat{p}$, if we have $Q(\hat{p}) < \infty$ then the algorithm starts a recursive call* Find-σ-Exits$(\pi_{P(\hat{p})..n}, \sigma_{Q(\hat{p})..\bar{q}})$. *In this case, the point $(P(\hat{p}), Q(\hat{p}))$ is a greedy pair of $\pi_{\hat{p}..n}$ and σ.*

Proof. For the first statement, assume for contradiction $Q(e) < \infty$. By Lemma 8.35, $Q(e) = Q'(e)$, which implies that for all $q' < Q'(e) \leqslant Q'(\hat{p})$, we have $\sigma_{q'} < \sigma_{Q'(e)}$ and hence $\mathrm{vis}_\pi(p, q') \subseteq \mathrm{vis}_\pi(p, Q'(e))$. Hence, $\mathrm{STOP}_\sigma(\pi_{p..n}, \sigma_{q..Q'(\hat{p})}) \leqslant Q'(e)$, since otherwise $Q'(e) \leftarrow \mathrm{GreedyStep}_\sigma(\pi_{p..n}, \sigma_{q..Q'(\hat{p})})$. By Lemma 8.22, this proves that $d_{\mathrm{dF}}(\pi_{\hat{p}..n}, \sigma_{1..Q'(e)}) > \delta$. Since $d_{\mathrm{dF}}(\pi_{\hat{p}..e}, \sigma_{1..q'}) \leqslant \delta$ for some $q' < Q'(e)$, Lemma 8.28 yields $d_{\mathrm{dF}}(\pi_{e..n}, \sigma_{1..Q'(e)}) > \delta$. This is a contradiction to $Q(e) = Q'(e)$.

For the second statement, note that if $Q(\hat{p}) < \infty$, then by Lemma 8.35, $Q(\hat{p}) = Q'(\hat{p})$. Hence Lemma 8.22 yields that the algorithm finds a feasible traversal of $\pi_{\hat{p}..p}$ and $\sigma_{1..Q'(\hat{p})}$ for some $\hat{p} \leqslant p \leqslant n$. This shows that $P(\hat{p}) \leqslant p < \infty$. Let $\sigma' := \sigma_{1..Q(\hat{p})}$ and assume that there is a $p' < p$ with $d_{\mathrm{dF}}(\pi_{\hat{p}..p'}, \sigma') \leqslant \delta$ and let $(\tilde{p}, \tilde{q})$ be the greedy point of $\pi_{\hat{p}..n}$ and σ' right before the algorithm made a greedy step on π to some index in $(p', p]$. By maximality of the greedy steps on σ, there exists $\tilde{q} < q_{\min} <$

$Q(\hat{p})$ such that $\pi_{\tilde{p}}$ does not see $\sigma_{q_{\min}}$, since otherwise $Q(\hat{p}) \in \text{reach}_{\sigma'}(\tilde{p}, \tilde{\sigma})$ with $\text{vis}_\pi(\tilde{p}, \tilde{q}) \subsetneq \text{vis}_\pi(\tilde{p}, Q(\hat{p}))$, i.e., $Q(\hat{p})$ would be a greedy step on σ'. By minimality of greedy steps on π, $\text{vis}_{\sigma'}(\tilde{p}, \tilde{q}) \supsetneq \text{vis}_{\sigma'}(i, \tilde{q})$ for all $\tilde{p} \leqslant i \leqslant p'$. Hence, no vertex on $\pi_{\tilde{p}..p'}$ sees $\sigma_{q_{\min}}$, which proves $d_{\text{dF}}(\pi_{\tilde{p}..p'}, \sigma') > \delta$. Since $(\tilde{p}, \tilde{\sigma})$ is a greedy pair of $\pi_{\hat{p}..p'}$ and σ', this yields that $d_{\text{dF}}(\pi_{\hat{p}..p'}, \sigma') > \delta$ by Lemma 8.22, which is a contradiction to the assumption. Hence, the algorithm calls FIND-σ-EXITS$(\pi_{p..n}, \sigma_{Q'(\hat{p})..\bar{q}})$, where $p = P(\hat{p})$ and $Q'(\hat{p}) = Q(\hat{p})$.

It remains to show that $(P(\hat{p}), Q(\hat{p}))$ is also a greedy pair of $\pi_{\hat{p}..n}$ and the *complete* curve σ. By Lemma 8.35, every $\hat{p} \leqslant p < P(\hat{p})$ satisfies $\pi_p > \pi_{P(\hat{p})}$ and hence $\text{vis}_\sigma(p, q) \subseteq \text{vis}_\sigma(P(\hat{p}), q)$ for all $1 \leqslant q \leqslant m$. Hence, if at some greedy pair (p, q), $q \leqslant Q(\hat{p})$, a greedy step $p' \leftarrow \text{GREEDYSTEP}_\pi(\pi_{p..n}, \sigma)$ with $p' \geqslant P(\hat{p})$ exists, then also $P(\hat{p}) \leftarrow \text{GREEDYSTEP}_\pi(\pi_{p..n}, \sigma)$, which shows that $(P(\hat{p}), q)$ is a greedy point of $\pi_{\hat{p}..n}$ and σ. If $q = Q(\hat{p})$, then $(P(\hat{p}), Q(\hat{p}))$ is a greedy point pair. Otherwise, by Lemma 8.35, $P(\hat{p})$ sees all of $\sigma_{q..Q(\hat{p})}$ and $\sigma_q < \sigma_{Q(\hat{p})}$, hence $Q(\hat{p}) \in \text{GREEDYSTEP}_\sigma(\pi_{P(p)..n}, \sigma)$ and $(P(\hat{p}), Q(\hat{p}))$ is a greedy step of $\pi_{\hat{p}..n}$ and σ.

It is left to consider the case that for all greedy pairs (p, q), $q \leqslant Q(\hat{p})$, of $\pi_{\hat{p}..n}$ and σ, no greedy step to some $p' \geqslant P(\hat{p})$ exists. Then there is some (p, q) with $p < P(\hat{p})$ and $q \leqslant Q(\hat{p})$ for which no greedy step exists at all. We have $p_{\text{stop}} := \text{STOP}_\pi(\pi_{p..n}, \sigma_{q..m}) \leqslant P(\hat{p})$, since otherwise $P(\hat{p})$ would be a greedy step. Since Lemma 8.22 shows that $d_{\text{dF}}(\pi_{\hat{p}..p_{\text{stop}}}, \sigma_{1..q}) > \delta$, this contradicts $d_{\text{dF}}(\pi_{\hat{p}..P(\hat{p})}, \sigma_{1..Q(\hat{p})})$ being at most δ. □

Lemma 8.38. *Algorithm 12 correctly computes F^σ.*

Proof. Clearly, any exit found is contained in F^σ, since the methods σ-EXITS-FROM-π and FIND-σ-EXITS only use feasible steps. For the converse, let $e \in E$ be an arbitrary entry and consider the set $F_e^\sigma = \{q \mid d_{\text{dF}}(\pi_{e..n}, \sigma_{1..q}) \leqslant \delta\}$ of σ-exits corresponding to the entry e.

We first show that if $F_e^\sigma \neq \emptyset$ and hence $Q(e) < \infty$, we have

$$F_e^\sigma = \text{FIND-}\sigma\text{-EXITS}(\pi_{P(e)..n}, \sigma_{Q(e)..m}).$$

Let $\bar{e} \in F_e^\sigma$. By Lemma 8.37, $(P(e), Q(e))$ is a greedy pair of $\pi_{e..n}$ and σ and hence also of $\pi_{e..n}$ and $\sigma_{1..\bar{e}}$. Lemma 8.22 thus implies $d_{\text{dF}}(\pi_{P(e)..n}, \sigma_{Q(e)..\bar{e}}) \leqslant \delta$ and consequently $\bar{e} \in \text{FIND-}\sigma\text{-EXITS}(\pi_{P(e)..n}, \sigma_{Q(e)..m})$. The converse clearly holds as well.

Note that e is not considered as $\hat{p}$ in any iteration of the algorithm if and only if the algorithm considers some $\hat{p}$ with $e \in [\hat{p}{+}1..p]$, where either (i) the algorithm finds a greedy pair (p, q) of $\pi_{\hat{p}..n}$ and $\sigma_{1..Q'(\hat{p})}$ that allows no further greedy steps, or (ii) the algorithm calls FIND-σ-EXITS$(\pi_{p..n}, \sigma_{Q'(\hat{p})..\bar{q}})$, where $p = P(\hat{p})$ by Lemma 8.37. In the first case, $F_e^\sigma = \emptyset$ since Lemma 8.37 proves $Q(e) = \infty$. In the second case, if $F_e^\sigma \neq \emptyset$, we have $Q(e) < \infty$, and hence by Lemma 8.36, $Q(e) = Q(\hat{p})$ and $P(\hat{p}) \leqslant P(e)$. Since $\sigma_{Q(\hat{p})}$ sees all of $\pi_{P(\hat{p})..n}$, any exit reachable from $(P(e), Q(e))$ is reachable from $(P(\hat{p}), Q(\hat{p}))$ as well. Hence $F_e^\sigma \subseteq F_{\hat{p}}^\sigma$.

Let $\hat{p}_1 \leqslant .. \leqslant \hat{p}_k$ be the entries considered as $\hat{p}$ by the algorithm. It remains to show that the algorithm finds all exits $\bigcup_{i=1}^k F_{\hat{p}_i}^\sigma$. We inductively show that the algorithm computes $F_{\hat{p}_i}^\sigma \setminus \bigcup_{j<i} F_{\hat{p}_j}^\sigma$ in the loop corresponding to $\hat{p} = \hat{p}_i$. The base

case $i = 1$ follows immediately. Note that for every $i \geqslant 2$, the corresponding loop computes $\textsc{Find-}\sigma\textsc{-Exits}(\pi_{P(\hat{p}_i)..n}, \sigma_{Q(\hat{p}_i)..Q(\hat{p}_{i-1})-1}) = F^\sigma_{\hat{p}_i} \cap [Q(\hat{p}_i)..Q(\hat{p}_{i-1})-1]$. The claim follows if we can show $F^\sigma_{\hat{p}_i} \cap [Q(\hat{p}_{i-1})..m] \subseteq F_{\hat{p}_{i-1}}$. Let $\overline{e} \in F^\sigma_{\hat{p}_i}$ with $\overline{e} \geqslant Q(\hat{p}_{i-1})$. Then $d_{\mathrm{dF}}(\pi_{\hat{p}_i..n}, \sigma_{1..\overline{e}}) \leqslant \delta$. Together with $d_{\mathrm{dF}}(\pi_{\hat{p}_{i-1}..n}, \sigma_{1..Q(\hat{p}_{i-1})}) \leqslant \delta$, Lemma 8.27 shows that $d_{\mathrm{dF}}(\pi_{\hat{p}_{i-1}..n}, \sigma_{1..\overline{e}}) \leqslant \delta$ and hence $e \in F^\sigma_{\hat{p}_{i-1}}$. □

Lemma 8.39. *Algorithm 12 runs in time $\mathcal{O}((n+m)\log 1/\varepsilon)$.*

Proof. Consider the total cost of the calls $\textsc{Find-}\sigma\textsc{-Exits}(\pi_{I_i}, \sigma_{J_i})$. Since all J_i are disjoint and $\bigcup_i J_i \subseteq [1..m]$, Lemma 8.31 bounds the total cost of such calls by $\mathcal{O}(\sum_i |J_i| \log(1/\varepsilon)) = \mathcal{O}(m \log(1/\varepsilon))$. Let $p_1, \ldots, p_k$ denote the distinct indices considered as p during the execution of the algorithm. Between changing p_i to p_{i+1}, we will make at most one call to $\textsc{MaxGreedyStep}_\sigma$ (by maximality) and at most once call to $\textsc{MinGreedyStep}_\pi$. Hence $k \leqslant n$ bounds the number of calls to greedy steps by $\mathcal{O}(n \log(1/\varepsilon))$. □

Bibliography

[1] K. Bringmann and M. Künnemann. "Improved approximation for Fréchet distance on c-packed curves matching conditional lower bounds." Submitted. 2014. arXiv: 1408.1340 [cs.CG].

[2] K. Bringmann. "Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails." In: *Proc. 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS'14)*. To appear. 2014.

[3] K. Bringmann, T. Friedrich, and A. Krohmer. "De-anonymization of Heterogeneous Random Graphs in Quasilinear Time." In: *Proc. 22th Annual European Symposium on Algorithms (ESA'14)*. To appear. 2014.

[4] K. Bringmann, T. Friedrich, and P. Klitzke. "Generic Postprocessing via Subset Selection for Hypervolume and Epsilon-Indicator." In: *Proc. 13th International Conference on Parallel Problem Solving from Nature (PPSN XIII)*. To appear. 2014.

[5] K. Bringmann, F. Kuhn, K. Panagiotou, U. Peter, and H. Thomas. "Internal DLA: Efficient Simulation of a Physical Growth Model." In: *Proc. 41th International Colloquium on Automata, Languages, and Programming (ICALP'14)*. Vol. 8572. LNCS. 2014, 247–258.

[6] K. Bringmann, T. Friedrich, and P. Klitzke. "Two-dimensional Subset Selection for Hypervolume and Epsilon-Indicator." In: *Proc. 16th Genetic and Evolutionary Computation Conference (GECCO'14)*. 2014, 589–596.

[7] K. Bringmann, T. Sauerwald, A. Stauffer, and H. Sun. "Balls into bins via local search: cover time and maximum load." In: *Proc. 31th Symposium on Theoretical Aspects of Computer Science (STACS'14)*. Vol. 25. LIPIcs. 2014, 187–198.

[8] K. Bringmann, C. Engels, B. Manthey, and B. Raghavendra Rao. "Random Shortest Paths: Non-Euclidean Instances for Metric Optimization Problems." In: *Algorithmica* (2014), 1–21.

[9] K. Bringmann, D. Hermelin, M. Mnich, and E. J. van Leeuwen. "Parameterized Complexity Dichotomy for Steiner Multicut." Submitted. 2013. arXiv: 1404.7006 [cs.DS].

[10] V. Alvarez, K. Bringmann, and S. Ray. "A Simple Sweep Line Algorithm for Counting Triangulations and Pseudo-triangulations." Submitted. 2013. arXiv: 1312.3188 [cs.CG].

[11] V. Alvarez, K. Bringmann, S. Ray, and R. Seidel. "Counting Triangulations Approximately." In: *Proc. 25th Canadian Conference on Computational Geometry (CCCG'13)*. 2013.

[12] K. Bringmann. "Bringing Order to Special Cases of Klee's Measure Problem." In: *Proc. 38th International Symposium on Mathematical Foundations of Computer Science (MFCS'13)*. Vol. 8087. LNCS. 2013, 207–218.

[13] K. Bringmann, C. Engels, B. Manthey, and B. Raghavendra Rao. "Random Shortest Paths: Non-Euclidean Instances for Metric Optimization Problems." In: *Proc. 38th International Symposium on Mathematical Foundations of Computer Science (MFCS'13)*. Vol. 8087. LNCS. 2013, 219–230.

[14] K. Bringmann and T. Friedrich. "Exact and efficient generation of geometric random variates and random graphs." In: *Proc. 40th International Colloquium on Automata, Languages, and Programming (ICALP'13)*. Vol. 7965. LNCS. 2013, 267–278.

[15] K. Bringmann, B. Doerr, A. Neumann, and J. Sliacan. "Online Checkpointing with Improved Worst-Case Guarantees." In: *Proc. 40th International Colloquium on Automata, Languages, and Programming (ICALP'13)*. Vol. 7965. LNCS. 2013, 255–266.

[16] S. Anand, K. Bringmann, T. Friedrich, N. Garg, and A. Kumar. "Minimizing Maximum (Weighted) Flow-time on Related and Unrelated Machines." In: *Proc. 40th International Colloquium on Automata, Languages, and Programming (ICALP'13)*. Vol. 7965. LNCS. 2013, 13–24.

[17] K. Bringmann and T. Friedrich. "Parameterized Average-Case Complexity of the Hypervolume Indicator." In: *15th Genetic and Evolutionary Computation Conference (GECCO'13)*. 2013, 575–582.

[18] K. Bringmann and K. G. Larsen. "Succinct Sampling from Discrete Distributions." In: *Proc. 45th Annual ACM Symposium on Symposium on Theory of Computing (STOC'13)*. 2013, 775–782.

[19] K. Bringmann, T. Friedrich, C. Igel, and T. Voß. "Speeding Up Many-Objective Optimization by Monte Carlo Approximations." In: *Artificial Intelligence* 204 (2013), 22–29.

[20] K. Bringmann and T. Friedrich. "Approximation quality of the hypervolume indicator." In: *Artificial Intelligence* 195 (2013), 265–290.

[21] K. Bringmann and K. Panagiotou. "Efficient Sampling Methods for Discrete Distributions." In: *Proc. 39th International Colloquium on Automata, Languages, and Programming (ICALP'12)*. Vol. 7391. LNCS. 2012, 133–144.

[22] K. Bringmann and T. Friedrich. "Convergence of Hypervolume-Based Archiving Algorithms II: Competitiveness." In: *14th Genetic and Evolutionary Computation Conference (GECCO'12)*. 2012, 457–464.

[23] V. Alvarez, K. Bringmann, R. Curticapean, and S. Ray. "Counting Crossing-free Structures." In: *Proc. 28th Annual Symposium on Computational Geometry (SoCG'12)*. 2012, 61–68.

[24] K. Bringmann. "An improved algorithm for Klee's measure problem on fat boxes." In: *Computational Geometry: Theory and Applications* 45.5-6 (2012), 225–233.

[25] K. Bringmann and T. Friedrich. “Approximating the least hypervolume contributor: NP-hard in general, but fast in practice.” In: *Theoretical Computer Science* 425 (2012), 104–116.

[26] K. Bringmann, K. Mehlhorn, and A. Neumann. “Remarks on Category-Based Routing in Social Networks.” 2012. arXiv: `1202.2293 [cs.SI]`.

[27] K. Bringmann, T. Friedrich, F. Neumann, and M. Wagner. “Approximation-Guided Evolutionary Multi-Objective Optimization.” In: *Proc. 21st International Joint Conference on Artificial Intelligence (IJCAI'11)*. 2011, 1198–1203.

[28] K. Bringmann and T. Friedrich. “Convergence of Hypervolume-Based Archiving Algorithms I: Effectiveness.” In: *Proc. 13th Genetic and Evolutionary Computation Conference (GECCO'11)*. 2011, 745–752.

[29] T. Friedrich, K. Bringmann, T. Voß, and C. Igel. “The Logarithmic Hypervolume Indicator.” In: *Proc. 11th International Workshop on Foundations of Genetic Algorithms (FOGA'11)*. 2011, 81–92.

[30] K. Bringmann. “Klee's measure problem on fat boxes in time $O(n^{(d+2)/3})$.” In: *Proc. 26th Annual Symposium on Computational Geometry (SoCG'10)*. 2010, 222–229.

[31] K. Bringmann and T. Friedrich. “Tight Bounds for the Approximation Ratio of the Hypervolume Indicator.” In: *Proc. 11th International Conference on Parallel Problem Solving from Nature (PPSN XI)*. Vol. 6238. LNCS. 2010, 607–616.

[32] K. Bringmann and T. Friedrich. “The Maximum Hypervolume Set Yields Near-optimal Approximation.” In: *Proc. 12th Genetic and Evolutionary Computation Conference (GECCO'10)*. 2010, 511–518.

[33] K. Bringmann and T. Friedrich. “An Efficient Algorithm for Computing Hypervolume Contributions.” In: *Evolutionary Computation* 18.3 (2010), 383–402.

[34] K. Bringmann and T. Friedrich. “Approximating the volume of unions and intersections of high-dimensional geometric objects.” In: *Computational Geometry: Theory and Applications* 43.6-7 (2010), 601–610.

[35] K. Bringmann and T. Friedrich. “Approximating the least hypervolume contributor: NP-hard in general, but fast in practice.” In: *Proc. 5th International Conference on Evolutionary Multi-Criterion Optimization (EMO'09)*. 2009, 6–20.

[36] K. Bringmann and T. Friedrich. “Don't be greedy when calculating hypervolume contributions.” In: *Proc. 10th International Workshop on Foundations of Genetic Algorithms (FOGA'09)*. 2009, 103–112.

[37] K. Bringmann and T. Friedrich. “Approximating the volume of unions and intersections of high-dimensional geometric objects.” In: *Proc. 19th International Symposium on Algorithms and Computation (ISAAC'08)*. Vol. 5369. LNCS. 2008, 436–447.

[38] A. J. Walker. “New Fast Method for Generating Discrete Random Numbers with Arbitrary Distributions.” In: *Electronic Letters* 10.8 (1974), 127–128.

[39] S. A. Cook and R. A. Reckhow. “Time Bounded Random Access Machines.” In: *Journal of Computer and System Sciences* 7.4 (1973), 354–375.

[40] F. P. Preparata and M. I. Shamos. *Computational Geometry*. Texts and Monographs in Computer Science. 1985.

[41] A. Borodin and I. Munro. *The computational complexity of algebraic and numeric problems*. 1975.

[42] T. Hagerup. “Sorting and Searching on the Word RAM.” In: *Proc. 15th Symposium on Theoretical Aspects of Computer Science (STACS'98)*. Vol. 1373. LNCS. 1998, 366–398.

[43] R. A. Kronmal and A. V. Peterson. “On the Alias Method for Generating Random Variables from a Discrete Distribution.” In: *The American Statistician* 33.4 (1979), 214–218.

[44] M. Pătraşcu. “WebDiarios de Motocicleta Sampling a discrete distribution.” 2011. URL: `infoweekly.blogspot.com/2011/09/sampling-discrete-distribution.html`.

[45] L. Devroye. *Nonuniform random variate generation*. 1986.

[46] D. E. Knuth and A. C.-C. Yao. “The complexity of nonuniform random number generation.” In: *Symposium on Algorithms and Complexity: New Directions and Recent Results*. Ed. by J. F. Traub. 1976, 357–428.

[47] A. C. Yao. “Context-free grammars and random number generation.” In: *Combinatorial algorithms on words*. Vol. 12. 1985, 357–361.

[48] P. Flajolet and N. Saheb. “The complexity of generating an exponentially distributed variate.” In: *Journal of Algorithms* 7.4 (1986), 463–488.

[49] P. Flajolet, M. Pelletier, and M. Soria. “On Buffon machines and numbers.” In: *Proc. 22nd ACM-SIAM Symposium on Discrete Algorithms (SODA'11)*. 2011, 172–183.

[50] T. Hagerup, K. Mehlhorn, and J. I. Munro. “Maintaining Discrete Probability Distributions Optimally.” In: *Proc. 20th International Colloquium on Automata, Languages, and Programming (ICALP'93)*. 1993, 253–264.

[51] Y. Matias, J. S. Vitter, and W.-C. Ni. “Dynamic Generation of Discrete Random Variates.” In: *Theory of Computing Systems* 36.4 (2003), 329–358.

[52] D. E. Knuth. *The Art of Computer Programming. Vol. 2: Seminumerical Algorithms*. Third edition. 2009.

[53] J. Fischer and V. Heun. “A New Succinct Representation of RMQ-Information and Improvements in the Enhanced Suffix Array.” In: *Proc. 1st International Symposium on Combinatorics, Algorithms, Probabilistic and Experimental Methodologies (ESCAPE'07)*. Vol. 4614. LNCS. 2007, 459–470.

[54] A. Golynski. “Optimal lower bounds for rank and select indexes.” In: *Theoretical Computer Science* 387.3 (Nov. 2007), 348–359.

[55] M. Pătraşcu. "Succincter." In: *Proc. 49th IEEE Symposium on Foundations of Computer Science (FOCS'08)*. 2008, 305–313.

[56] R. Raman, V. Raman, and S. R. Satti. "Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets." In: *ACM Transactions on Algorithms* 3.4 (2007).

[57] G. S. Brodal, P. Davoodi, and S. S. Rao. "On Space Efficient Two Dimensional Range Minimum Data Structures." In: *Proc. 18th Annual European Symposium on Algorithms (ESA'10)*. Vol. 6347. LNCS. 2010, 171–182.

[58] A. Golynski, R. Raman, and S. S. Rao. "On the Redundancy of Succinct Data Structures." In: *Proc. 11th Scandinavian Workshop on Algorithm Theory (SWAT '08)*. 2008, 148–159.

[59] M. Pătraşcu and E. Viola. "Cell-Probe Lower Bounds for Succinct Partial Sums." In: *Proc. 21st ACM-SIAM Symposium on Discrete Algorithms (SODA'10)*. 2010, 117–122.

[60] Y. Dodis, M. Pătraşcu, and M. Thorup. "Changing Base without Losing Space." In: *Proc. 42nd Annual ACM Symposium on Theory of Computing (STOC'10)*. 2010, 593–602.

[61] J. S. Vitter. "Random sampling with a reservoir." In: *ACM Transactions on Mathematical Software* 11.1 (Mar. 1985), 37–57.

[62] M.-T. Tsai, D.-W. Wang, C.-J. Liau, and T.-S. Hsu. "Heterogeneous Subset Sampling." In: *Proc. 16th Annual International Computing and Combinatorics Conference (COCOON'10)*. 2010, 500–509.

[63] A. Ghosh, T. Roughgarden, and M. Sundararajan. "Universally utility-maximizing privacy mechanisms." In: *SIAM Journal on Computing* 41.6 (2012), 1673–1693.

[64] C. Karney. "Random number library, Version 1.9." June 2014. URL: `www.sourceforge.net/projects/randomlib`.

[65] M. J. Keeling and K. T. Eames. "Networks and epidemic models." In: *Journal of The Royal Society Interface* 2.4 (2005), 295–307.

[66] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. "Network Motifs: Simple Building Blocks of Complex Networks." In: *Science* 298.5594 (2002), 824–827.

[67] J. H. Kim and V. H. Vu. "Generating Random Regular Graphs." In: *Proc. 35th Annual ACM Symposium on Theory of Computing (STOC'03)*. 2003, 213–222.

[68] J. K. Blitzstein and P. Diaconis. "A Sequential Importance Sampling Algorithm for Generating Random Graphs with Prescribed Degrees." In: *Internet Mathematics* 6.4 (2011), 489–522.

[69] J. Ray, A. Pinar, and C. Seshadhri. "Are we there yet? When to stop a Markov chain while generating random graphs." In: *Proc. 9th International Conference on Algorithms and Models for the Web Graph (WAW'12)*. Vol. 7323. LNCS. 2012, 153–164.

[70] P. Erdős and A. Rényi. “On random graphs.” In: *Publ. Math. Debrecen* 6 (1959), 290–297.

[71] V. Batagelj and U. Brandes. “Efficient generation of large random networks.” In: *Physical Review E* 71.3 (2005), 036113.

[72] J. C. Miller and A. A. Hagberg. “Efficient Generation of Networks with Given Expected Degrees.” In: *Proc. 8th International Conference on Algorithms and Models for the Web Graph (WAW'11)*. Vol. 6732. LNCS. 2011, 115–126.

[73] A. A. Hagberg, D. A. Schult, and P. J. Swart. “Exploring network structure, dynamics, and function using NetworkX.” In: *Proc. 7th Python in Science Conference (SciPy'08)*. 2008, 11–15.

[74] K. Mehlhorn and S. Näher. *LEDA: A Platform for Combinatorial and Geometric Computing*. 1999.

[75] A. Blanca and M. Mihail. “Efficient Generation ε-close to $G(n,p)$ and Generalizations.” 2012. arXiv: `1204.5834 [cs.DS]`.

[76] F. Chung and L. Lu. “Connected Components in Random Graphs with Given Expected Degree Sequences.” In: *Annals of Combinatorics* 6.2 (2002), 125–145.

[77] M. Farach-Colton and M. Tsai. “Exact Sublinear Binomial Sampling.” In: *Proc. 24th International Symposium on Algorithms and Computation (ISAAC '13)*. Vol. 8283. LNCS. 2013, 240–250.

[78] P. Meakin and J. M. Deutch. “The formation of surfaces by diffusion limited annihilation.” In: *The Journal of Chemical Physics* 85 (1986), 2320.

[79] P. Diaconis and W. Fulton. “A growth model, a game, an algebra, Lagrange inversion, and characteristic classes.” In: *Rend. Sem. Mat. Univ. Politec. Torino* 49.1 (1991), 95–119.

[80] G. F. Lawler, M. Bramson, and D. Griffeath. “Internal diffusion limited aggregation.” In: *The Annals of Probability* 20.4 (1992), 2117–2140.

[81] A. Asselah and A. Gaudilliere. “Lower bounds on fluctuations for internal DLA.” In: *Probability Theory and Related Fields* 158.1-2 (2014), 39–53.

[82] A. Asselah and A. Gaudilliere. “From logarithmic to subdiffusive polynomial fluctuations for internal DLA and related growth models.” In: *Annals of Probability* 41.3A (2013), 1115–1159.

[83] D. Jerison, L. Levine, and S. Sheffield. “Logarithmic fluctuations for internal DLA.” In: *Journal of the American Mathematical Society* 25 (2012), 271–301.

[84] C. Moore and J. Machta. “Internal Diffusion-Limited Aggregation: Parallel Algorithms and Complexity.” In: *Journal of Statistical Physics* 99 (2000), 661–690.

[85] T. Friedrich and L. Levine. “Fast simulation of large-scale growth models.” In: *Random Structures & Algorithms* 42.2 (2013), 185–213.

[86] G. Jacobson. “Space-efficient static trees and graphs.” In: *Proc. 30th Annual Symposium on Foundations of Computer Science (FOCS'89)*. 1989, 549–554.

[87] Ş. Nacu and Y. Peres. “Fast Simulation of New Coins from Old.” In: *The Annals of Applied Probability* 15.1A (2005), 93–115.

[88] R. v. d. Hofstad. “Random graphs and complex networks.” 2009. URL: `www.win.tue.nl/~rhofstad/NotesRGCN.pdf`.

[89] I. Norros and H. Reittu. “On a conditionally Poissonian graph process.” In: *Advances in Applied Probability* 38.1 (2006), 59–75.

[90] R. Brent and P. Zimmermann. *Modern Computer Arithmetic*. 2010.

[91] R. Solovay and V. Strassen. “A Fast Monte-Carlo Test for Primality.” In: *SIAM Journal on Computing* 6.1 (1977), 84–85.

[92] M. Fürer. “Faster Integer Multiplication.” In: *SIAM Journal on Computing* 39.3 (2009), 979–1005.

[93] A. De, P. P. Kurur, C. Saha, and R. Saptharishi. “Fast integer multiplication using modular arithmetic.” In: *SIAM Journal on Computing* 42.2 (2013), 685–699.

[94] C. Lanczos. “A Precision Approximation of the Gamma Function.” In: *SIAM Journal on Numerical Analysis* 1.1 (1964), 86–96.

[95] J. Spouge. “Computation of the Gamma, Digamma, and Trigamma Functions.” In: *SIAM Journal on Numerical Analysis* 31.3 (1994), 931–944.

[96] J. He and X. Yao. “A study of drift analysis for estimating computation time of evolutionary algorithms.” In: *Natural Computing* 3.1 (2004), 21–35.

[97] G. F. Lawler and V. Limic. *Random Walk: A Modern Introduction*. Cambridge Studies in Advanced Mathematics. 2010.

[98] D. Belazzougui, F. C. Botelho, and M. Dietzfelbinger. “Hash, Displace, and Compress.” In: *Proc. 17th Annual European Symposium on Algorithms (ESA ’09)*. Vol. 5757. LNCS. 2009, 682–693.

[99] D. P. Dubhash and A. Panconesi. *Concentration of measure for the analysis of randomized algorithms*. 2009.

[100] H. Alt. “The computational geometry of comparing shapes.” In: *Efficient Algorithms*. Vol. 5760. LNCS. 2009, 235–248.

[101] H. Alt and M. Godau. “Computing the Fréchet distance between two polygonal curves.” In: *International Journal of Computational Geometry & Applications* 5.1-2 (1995), 78–99.

[102] M. Godau. “A natural metric for curves - computing the distance for polygonal chains and approximation algorithms.” In: *Proc. 8th Symposium on Theoretical Aspects of Computer Science (STACS’91)*. Vol. 480. LNCS. 1991, 127–136.

[103] H. Alt and M. Buchin. “Can we compute the similarity between surfaces?” In: *Discrete & Computational Geometry* 43.1 (2010), 78–99.

[104] B. Aronov, S. Har-Peled, C. Knauer, Y. Wang, and C. Wenk. “Fréchet distance for curves, revisited.” In: *Proc. 14th Annual European Symposium on Algorithms (ESA’06)*. Vol. 4168. LNCS. 2006, 52–63.

[105] H. Alt, C. Knauer, and C. Wenk. “Comparison of distance measures for planar curves.” In: *Algorithmica* 38.1 (2004), 45–58.

[106] A. Driemel, S. Har-Peled, and C. Wenk. “Approximating the Fréchet distance for realistic curves in near linear time.” In: *Discrete & Computational Geometry* 48.1 (2012), 94–127.

[107] E. W. Chambers, É. Colin de Verdière, J. Erickson, S. Lazard, F. Lazarus, and S. Thite. “Homotopic Fréchet distance between curves or, walking your dog in the woods in polynomial time.” In: *Computational Geometry* 43.3 (2010), 295–311.

[108] A. F. Cook and C. Wenk. “Geodesic Fréchet distance inside a simple polygon.” In: *ACM Transactions on Algorithms* 7.1 (2010), 193–204.

[109] K. Buchin, M. Buchin, and Y. Wang. “Exact algorithms for partial curve matching via the Fréchet distance.” In: *Proc. 20th ACM-SIAM Symposium on Discrete Algorithms (SODA’09)*. 2009, 645–654.

[110] A. Driemel and S. Har-Peled. “Jaywalking your dog: computing the Fréchet distance with shortcuts.” In: *SIAM Journal on Computing* 42.5 (2013), 1830–1866.

[111] A. Maheshwari, J.-R. Sack, K. Shahbaz, and H. Zarrabi-Zadeh. “Fréchet distance with speed limits.” In: *Computational Geometry* 44.2 (2011), 110–120.

[112] P. Indyk. “Approximate nearest neighbor algorithms for Fréchet distance via product metrics.” In: *Proc. 18th Annual Symposium on Computational Geometry (SoCG’02)*. 2002, 102–106.

[113] M. E. Munich and P. Perona. “Continuous dynamic time warping for translation-invariant curve alignment with applications to signature verification.” In: *Proc. 7th IEEE International Conference on Computer Vision*. Vol. 1. 1999, 108–115.

[114] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk. “On map-matching vehicle tracking data.” In: *Proc. 31st International Conference on Very Large Data Bases (VLDB’05)*. 2005, 853–864.

[115] K. Buchin, M. Buchin, J. Gudmundsson, M. Löffler, and J. Luo. “Detecting commuting patterns by clustering subtrajectories.” In: *International Journal of Computational Geometry & Applications* 21.3 (2011), 253–282.

[116] T. Eiter and H. Mannila. *Computing discrete Fréchet distance*. Tech. rep. CD-TR 94/64. Christian Doppler Laboratory for Expert Systems, TU Vienna, Austria, 1994.

[117] P. Agarwal, R. B. Avraham, H. Kaplan, and M. Sharir. “Computing the Discrete Fréchet Distance in Subquadratic Time.” In: *Proc. 24th ACM-SIAM Symposium on Discrete Algorithms (SODA’13)*. 2013, 156–167.

[118] K. Buchin, M. Buchin, W. Meulemans, and W. Mulzer. “Four Soviets Walk the Dog - with an Application to Alt’s Conjecture.” In: *Proc. 25th ACM-SIAM Symposium on Discrete Algorithms (SODA’14)*. 2014, 1399–1413.

[119] R. B. Avraham, O. Filtser, H. Kaplan, M. J. Katz, and M. Sharir. "The Discrete Fréchet Distance with Shortcuts via Approximate Distance Counting and Selection Techniques." 2013. arXiv: 1310.5245 [cs.CG].

[120] K. Buchin, M. Buchin, C. Knauer, G. Rote, and C. Wenk. "How difficult is it to walk the dog?" In: *Proc. 23rd European Workshop on Computational Geometry (EWCG'07)*. 2007, 170–173.

[121] A. Gajentaan and M. H. Overmars. "On a class of $O(n^2)$ problems in computational geometry." In: *Computational Geometry: Theory and Applications* 5.3 (1995), 165–185.

[122] A. Grønlund and S. Pettie. "Threesomes, Degenerates, and Love Triangles." In: *Proc. 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS'14)*. To appear. 2014.

[123] D. Chen, A. Driemel, L. J. Guibas, A. Nguyen, and C. Wenk. "Approximate Map Matching with respect to the Fréchet Distance." In: *Proc. 13th Workshop on Algorithm Engineering and Experiments (ALENEX'11)*. 2011, 75–83.

[124] S. Har-Peled and B. Raichel. "The Fréchet Distance Revisited and Extended." In: *Proc. 27th Annual Symposium on Computational Geometry (SoCG'11)*. 2011, 448–457.

[125] J. Gudmundsson and M. Smid. "Fréchet Queries in Geometric Trees." In: *Proc. 21st Annual European Symposium on Algorithms (ESA'13)*. Vol. 8125. LNCS. 2013, 565–576.

[126] R. Impagliazzo, R. Paturi, and F. Zane. "Which Problems Have Strongly Exponential Complexity?" In: *Journal of Computer and System Sciences* 63.4 (2001), 512–530.

[127] R. Impagliazzo and R. Paturi. "On the Complexity of k-SAT." In: *Journal of Computer and System Sciences* 62.2 (2001), 367–375.

[128] D. Lokshtanov, D. Marx, and S. Saurabh. "Lower bounds based on the Exponential Time Hypothesis." In: *Bulletin of the EATCS* 105 (2011), 41–72.

[129] M. Pătraşcu and R. Williams. "On the possibility of faster SAT algorithms." In: *Proc. 21nd ACM-SIAM Symposium on Discrete Algorithms (SODA'10)*. 2010, 1065–1075.

[130] L. Roditty and V. Vassilevska Williams. "Fast approximation algorithms for the diameter and radius of sparse graphs." In: *Proc. 45th Annual ACM Symposium on Symposium on Theory of Computing (STOC'13)*. 2013, 515–524.

[131] E. Dantsin and E. A. Hirsch. "Worst-Case Upper Bounds." In: *Handbook of Satisfiability* 185 (2009), 403–424.

[132] C. Calabro, R. Impagliazzo, and R. Paturi. "A duality between clause width and clause density for SAT." In: *Proc. 21st IEEE Conference on Computational Complexity (CCC'06)*. 2006, 252–260.

[133] A. Abboud and V. Vassilevska Williams. “Popular conjectures imply strong lower bounds for dynamic problems.” In: *Proc. 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS'14)*. To appear. 2014.

[134] M. Cygan, H. Dell, D. Lokshtanov, D. Marx, J. Nederlof, Y. Okamoto, R. Paturi, S. Saurabh, and M. Wahlström. “On Problems as Hard as CNF-SAT.” In: *Proc. 27th IEEE Conference on Computational Complexity (CCC'12)*. 2012, 74–84.

[135] R. Williams. “A new algorithm for optimal constraint satisfaction and its implications.” In: *Proc. 31th International Colloquium on Automata, Languages, and Programming (ICALP'04)*. Vol. 3142. LNCS. 2004, 1227–1237.

[136] R. Williams and H. Yu. “Finding orthogonal vectors in discrete structures.” In: *Proc. 25th ACM-SIAM Symposium on Discrete Algorithms (SODA'14)*. 2014, 1867–1877.

[137] G. S. Lueker. “A Data Structure for Orthogonal Range Queries.” In: *Proc. 19th Annual Symposium on Foundations of Computer Science (SFCS'78)*. 1978, 28–34.

[138] B. Chazelle and L. J. Guibas. “Fractional Cascading: I. A Data Structuring Technique.” In: *Algorithmica* 1.2 (1986), 133–162.

[139] D. E. Willard. “Predicate-oriented database search algorithms.” Tech. rep. TR-20-78. PhD thesis. Aiken Comput. Lab, Harvard Univ., 1978.