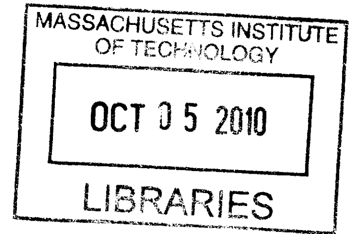
New Sublinear Methods in the Struggle Against Classical Problems

by

Krzysztof Onak

Magister, Uniwersytet Warszawski (2005)



Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science and Engineering

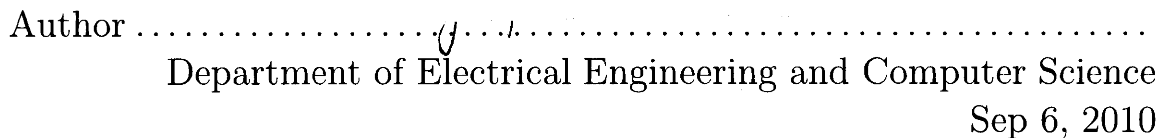
at the

ARCHIVES

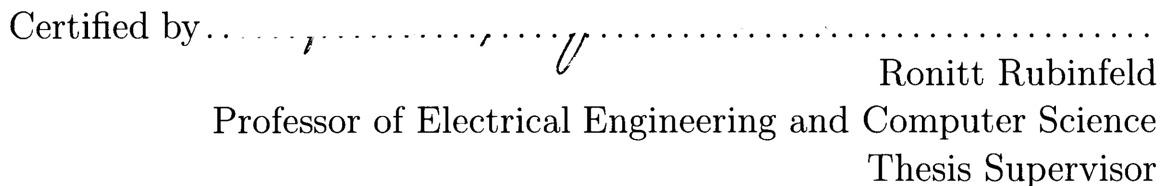
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2010

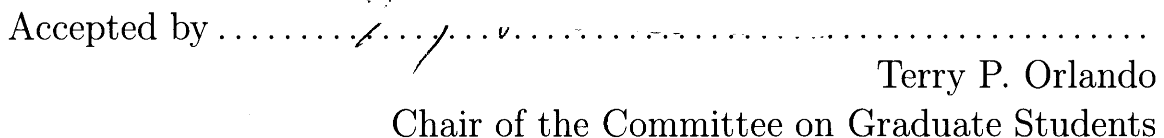
© Massachusetts Institute of Technology 2010. All rights reserved.



Author ....
Department of Electrical Engineering and Computer Science
Sep 6, 2010



Certified by..
Ronitt Rubinfeld
Professor of Electrical Engineering and Computer Science
Thesis Supervisor



Accepted by..
Terry P. Orlando
Chair of the Committee on Graduate Students

New Sublinear Methods in the Struggle Against Classical Problems

by

Krzysztof Onak

Submitted to the Department of Electrical Engineering and Computer Science
on Sep 6, 2010, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science and Engineering

Abstract

We study the time and query complexity of approximation algorithms that access only a minuscule fraction of the input, focusing on two classical sources of problems: combinatorial graph optimization and manipulation of strings. The tools we develop find applications outside of the area of sublinear algorithms. For instance, we obtain a more efficient approximation algorithm for edit distance and distributed algorithms for combinatorial problems on graphs that run in a constant number of communication rounds.

Combinatorial Graph Optimization Problems: The graph optimization problems considered by us include vertex cover, maximum matching, and dominating set. A graph algorithm is traditionally called a *constant-time* algorithm if it runs in time that is a function of only the maximum vertex degree, and in particular, does not depend on the number of vertices in the graph.

We show a general *local computation framework* that allows for transforming many classical greedy approximation algorithms into constant-time approximation algorithms for the optimal solution size. By applying the framework, we obtain the first constant-time algorithm that approximates the maximum matching size up to an additive εn, where ε is an arbitrary positive constant, and n is the number of vertices in the graph.

It is known that a purely additive εn approximation is not computable in constant time for vertex cover and dominating set. We show that nevertheless, such an approximation is possible for a wide class of graphs, which includes planar graphs (and other minor-free families of graphs) and graphs of subexponential growth (a common property of networks). This result is obtained via locally computing a good partition of the input graph in our local computation framework.

The tools and algorithms developed for these problems find several other applications:

- Our methods can be used to construct local distributed approximation algorithms for some combinatorial optimization problems.

- Our matching algorithm yields the first constant-time testing algorithm for distinguishing bounded-degree graphs that have a perfect matching from those far from having this property.

- We give a simple proof that there is a constant-time algorithm distinguishing bounded-degree graphs that are planar (or in general, have a minor-closed property) from those that are far from planarity (or the given minor-closed property, respectively). Our tester is also much more efficient than the original tester of Benjamini, Schramm, and Shapira (STOC 2008).

Edit Distance. We study a new *asymmetric* query model for edit distance. In this model, the input consists of two strings x and y, and an algorithm can access y in an unrestricted manner (without charge), while being charged for querying every symbol of x.

We design an algorithm in the asymmetric query model that makes a small number of queries to distinguish the case when the edit distance between x and y is small from the case when it is large. Our result in the asymmetric query model gives rise to a near-linear time algorithm that approximates the edit distance between two strings to within a polylogarithmic factor. For strings of length n and every fixed $\varepsilon > 0$, the algorithm computes a $(\log n)^{O(1/\varepsilon)}$ approximation in $n^{1+\varepsilon}$ time. This is an exponential improvement over the previously known near-linear time approximation factor $2^{\tilde{O}(\sqrt{\log n})}$ (Andoni and Onak, STOC 2009; building on Ostrovsky and Rabani, J. ACM 2007). The algorithm of Andoni and Onak was the first to run in $O(n^{2-\delta})$ time, for any fixed constant $\delta > 0$, and obtain a subpolynomial, $n^{o(1)}$, approximation factor, despite a sequence of papers.

We provide a nearly-matching lower bound on the number of queries. Our lower bound is the first to expose hardness of edit distance stemming from the input strings being "repetitive", which means that many of their substrings are approximately identical. Consequently, our lower bound provides the first rigorous separation on the complexity of approximation between edit distance and Ulam distance.

Thesis Supervisor: Ronitt Rubinfeld
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

This thesis would not have happened without the support and help of many people. I am grateful to my advisor Ronitt Rubinfeld for her guidance and all the trust she put in me. She is probably the most encouraging person I know, and she has always been an inexhaustible source of new research ideas. Ronitt has never put any constraints on me, but she has always let me freely explore topics I found interesting at a given moment, and she has never complained about the time I spent collaborating with other people.

I would like to thank Piotr Indyk for all the feedback and support I received from him. Piotr largely participated in creating the positive and friendly atmosphere in the MIT theory group. I am grateful to Piotr Indyk and Silvio Micali for serving on my thesis committee.

I owe a lot to my collaborators: Alex Andoni, Artur Czumaj, Ilias Diakonikolas, Gereon Frahling, Nick Harvey, Avinatan Hassidim, Piotr Indyk, David Karger, Jon Kelner, Phil Klein, Robi Krauthgamer, Homin Lee, Kevin Matulef, Andrew McGregor, Morteza Monemizadeh, Shay Mozes, Jelani Nelson, Huy Nguyen, Rina Panigrahy, Pawel Parys, Ronitt Rubinfeld, Rocco Servedio, Tasos Sidiropoulos, Christian Sohler, Andrew Wan, Oren Weimann, and Ning Xie. I learned a lot from them, and needless to say, many of the ideas in this thesis come from them.

I can hardly imagine a better environment for a PhD program than the MIT theory group. It was very vibrant and friendly. In particular, I am greatly indebted to Alex and David. We started the PhD program in the same year and shared many of its enjoyments and hardships. It is next to impossible to list all people who made my stay at MIT enjoyable. Let me personally express my gratefulness to Agata, Anh and Huy, Ankur, Anton, Arnab, Avinatan, Ben, Chih-yu, Jacob, Jelani, Kamila and Olek, Kevin, Mieszko, MinJi, Mira and Mihai, Ning, Rotem, Szymon, Tasos, and Violeta. If your name is missing from here, I owe you at least a dinner. I am also happy to say that I managed to keep in touch with a few friends from college, occasionally meeting them in the real world. Let me just mention Andrzej, Artur,

Basia, Marcin, Piotr, Szymek, Tomek (×2), Tunia, and Zosia. I always felt I could count on you, and you were often the first witnesses to how studying abroad affected my command of Polish.

I would not be here if several people from my home country had not influenced my life. I am greatly indebted to (as I later found out, relatively poorly paid) translators of foreign computer science books into Polish as well as authors of Polish books on the topic. Those books helped me establish my interests in (theoretical) computer science. I am grateful to the organizers of the Polish Olympiad in Informatics. The Olympiad helped me further develop my interests and was a great platform for meeting people with similar interests in high school. I would like to thank Krzysztof Diks, my undergraduate advisor at Warsaw University. His enthusiasm about computer science and optimistic approach to life have always been contagious, and he was one of the main reasons why I decided to study at Warsaw University.

Finally, I would like to thank all my family for their love and support. My parents showed a lot of understanding when I told them that I was considering going to graduate school in the US, and they have constantly been making sure that I was not lacking anything here. They also had a huge influence on my professional life by providing my first computational environment at home and teaching me how to program in LOGO on the ZX Spectrum.

Contents

1 Introduction **9**

1.1 Graph Optimization Problems 10

1.1.1 General Graphs 11

1.1.2 Better Algorithms for Important Subclasses 15

1.1.3 Follow-Up Work 21

1.1.4 A Note on Bounded Average-Degree Instances 21

1.2 Edit Distance 22

1.2.1 Historical Background 22

1.2.2 Results 24

1.2.3 Connections of the Asymmetric Query Model to Other Models 27

2 Combinatorial Problems on Graphs **29**

2.1 Definitions and the Model 29

2.2 Simple Example: Vertex Cover via Maximal Matching 30

2.3 A Local Computation Method 32

2.3.1 The Method 32

2.3.2 Non-Adaptive Performance 33

2.3.3 Adaptive Performance 35

2.3.4 Recent Improvements 38

2.4 General Transformation for Greedy Algorithms 39

2.4.1 Technique High-Level Description 39

2.4.2 Maximum Matching 40

2.4.3 Maximum Weight Matching 45

2.4.4 Set Cover 52
2.5 Better Algorithms for Hyperfinite Graphs 54
2.5.1 A Generic Partitioning Oracle 59
2.5.2 An Efficient Partitioning Oracle for Minor-Free Graphs 63
2.6 Other Applications of Our Methods 67
2.6.1 Local Distributed Algorithms 67
2.6.2 Testing the Property of Having a Perfect Matching 68
2.6.3 Testing Minor-Closed Properties 68
2.6.4 Approximating Distance to Hereditary Properties For Hyperfinite Graphs 71

3 Edit Distance **77**
3.1 Outline of Our Results 77
3.1.1 Outline of the Upper Bound 77
3.1.2 Outline of the Lower Bound 82
3.2 Fast Algorithms via Asymmetric Query Complexity 89
3.2.1 Edit Distance Characterization: the $\mathcal{E}$-distance 90
3.2.2 Sampling Algorithm 95
3.2.3 Near-Linear Time Algorithm 105
3.3 Query Complexity Lower Bound 106
3.3.1 Preliminaries 106
3.3.2 Tools for Analyzing Indistinguishability 108
3.3.3 Tools for Analyzing Edit Distance 116
3.3.4 The Lower Bound 122

Chapter 1

Introduction

Over the last two decades, computation has spread across all areas of our lives. The decreasing cost of information storage enables collecting more and more data by various sensors and monitoring tools. For instance, the Large Hadron Collider was predicted to produce 15 petabytes[1] of data annually [57]. How can one store this amount of data to enable fast processing later? How can one efficiently access and process data that has already been collected?

Traditionally, theoretical computer science identified efficiency with polynomial-time computability in the model where the algorithm freely accesses input. Computational challenges emerging nowadays are not always captured by this model due to either limitations on how data can be accessed, or simply due to the amount of data, which cannot be efficiently processed. Multiple frameworks and models have been developed to address this issue, and one of them is sublinear-time algorithms.

Sublinear-time algorithms and more generally, *sublinear-query algorithms* attempt to infer information from a dataset by accessing only a miniscule fraction of information in it. A classical example of a sublinear-time algorithm is binary search, in which one checks if an element appears in a sorted array in time logarithmic in the array length. Another example is uniform sampling from a set of items to estimate the fraction of those with a given property. For the latter problem, the Hoeffding bound implies that the fraction of elements with the property in a sample of size $O(1/\varepsilon^2)$

[1] 1 petabyte equals 10^{15} bytes.

differs from the fraction of elements in the entire set by at most an additive ε with probability 99/100.

The modern study of sublinear algorithms was initiated by the paper of Blum, Luby, and Rubinfeld [20]. They show an algorithm for approximately verifying that a function is a linear function. This algorithm played an important role in the original proof of the PCP theorem [11, 12]. The Blum-Luby-Rubinfeld paper together with papers of Rubinfeld and Sudan [75], and Goldreich, Goldwasser, and Ron [39] gave a rise to the field of *property testing*. In property testing, the goal is to distinguish inputs that have a specific property from those that are significantly different from any input that has the property. More formally, one says that an input is ε-*far* from a given property, if at least an ε-fraction of the input bits (or other units constituting the input, say, words, integers, etc.) need to be modified to obtain the property. A *tester* for the property distinguishes inputs with the property from those that are ε-far from having the property with probability at least 2/3.

The property testing field can be seen as a study of relaxed versions of decision problems. In this thesis, we mostly focus on approximation problems (studied before in the sublinear-time framework by, for instance, [39, 44, 22, 29, 13, 70, 62]), where we want to output an approximate value of a parameter of the input as opposed to making a single binary YES/NO decision. Nevertheless, some of our techniques find applications in constructing testers for natural properties.

This thesis focuses on two kinds of problems in the sublinear-query model: combinatorial graph optimization problems and edit distance. We describe our results with motivation in the next two sections.

1.1 Graph Optimization Problems

There has been an enormous amount of work on maximum matching, vertex cover, and set cover in the classical computation model, where the whole input is read. It is obviously not possible to compute a solution to these problems in time sublinear in the input size, since an optimal solution may itself have linear size. The question

we ask is whether it is possible to approximate just the optimal solution size in time sublinear in the input size. This question has been asked before for various graph problems (see for instance [39, 22, 13, 70, 62]).

Our algorithms work best for sparse graphs. Some examples of graphs for which our algorithms would work well are social networks and the World Wide Web. The following facts show that these graphs have a huge number of vertices:

- In July 2008, Google employees announced that the Google search engine had discovered one trillion unique URLs [6].
- The number of webpages indexed by the most popular search engines was estimated at the beginning of August 2010 to be more than 55 billion [1].
- In July 2010, it was announced that the social networking website Facebook had more than 500 million active users[2] [80].

However, despite the size of the above graphs, most vertices in them have relatively low degree. If only a small fraction of vertices have high degree, one can modify the graph in most cases so that the problem has to be solved only for a subgraph of small degree without modifying the solution size significantly. We discuss this issue in more detail in Section 1.1.4.

1.1.1 General Graphs

We say that a graph algorithm runs in *constant time* if its running time is bounded by a function of the maximum degree. We show a general technique that can be used to transform multiple classical greedy approximation algorithms into constant-time approximation algorithms for sparse graphs.

Our results belong to a line of research initiated by Parnas and Ron [70]. They show a randomized constant-time algorithm that with probability close to 1, outputs a value x such that $\mathrm{VC}(G) \leq x \leq 2 \cdot \mathrm{VC}(G) + \varepsilon n$, for any fixed $\varepsilon > 0$, where n is the number of vertices in the graph. Their algorithm stems from the observation

[2]A user is considered *active* by Facebook if he or she has visited the website in the last 30 days.

that if there is a distributed algorithm that in a constant number of rounds computes a good solution to a problem, then it is possible to approximate the size of such a solution efficiently. It suffices to compute the output of the algorithm on a few randomly selected vertices (or, for some problems, edges) to estimate the fraction of vertices (or edges) included in the solution. Since the distributed algorithm runs in a constant number of rounds the simulation can be done with a number of queries (and usually also in time) that is only a function of the maximum degree. It is crucial that the number of communication rounds the distributed algorithm uses be small. For instance, if there are more than $\log n$ communication rounds, then in an expander, the output for a given vertex may depend on the entire graph and it may not be possible to compute the distributed algorithm's output even for a single vertex.

As opposed to the approach of Parnas and Ron, ours does not rely on distributed algorithms. Moreover, for maximum matching, our method overcomes limitations of the previously known distributed algorithms.

Definitions and the Model

Before we formally state our results, let us introduce the approximation notion that we use.

Definition 1.1.1. *We say that a value $\hat{y}$ is an* (α,β)-approximation *to y, if*

$$y \leq \hat{y} \leq \alpha \cdot y + \beta.$$

We say that an algorithm A is an (α,β)-approximation algorithm *for a value $V(x)$ if it computes an (α,β)-approximation to $V(x)$ with probability at least $2/3$ for any proper input x.*

For simplicity, whenever we talk about an *(α,β)-approximation algorithm for problem $\mathcal{X}$* in this part of the thesis, we, in fact, mean an *(α,β)-approximation algorithm for the optimal solution size to problem $\mathcal{X}$.*

With a small exception, the input to problems considered in this part of the thesis is a graph. We make the following assumptions about how an algorithm can access

the input.

- The algorithm can select a vertex in the graph uniformly at random. The operation takes constant time.

- The algorithm can query the adjacency list of each vertex. It can make two types of queries, both answered in constant time. In the *degree query*, the algorithm finds out the degree of a vertex it specifies. By making a *neighbor query*, the algorithm learns the identity of the i-th neighbor of v, where both i and v are specified by the algorithm. Additionally, if the graph is weighted, the algorithm learns the weight of the edge connecting v with its i-th neighbor.

We also consider the set cover problem. Let $\mathcal{U}$ be the set of all elements and let S_i be the sets with which we cover $\mathcal{U}$. In this case we assume that the algorithm can uniformly select a random set S_i. Furthermore, for every element $x \in \mathcal{U}$, the algorithm can query the list of the sets S_i that contain x, and for every set S_i, the algorithm can query the list of the elements of S_i.

Overview of Graph Problems

We now list our results, and compare them with previous work in related areas. We express the complexity of our algorithms in terms of the number of queries they make, but the corresponding running time is independent of the size of input and is greater by at most an additional logarithmic factor, compared to the query complexity.

Maximum Matching. The only results on approximation of the maximum matching size in sublinear time that have been known before are the algorithms of Parnas and Ron [70], and Marko and Ron and [62]. Since their algorithms give a constant factor approximation to the minimum vertex cover size, they also give a constant factor approximation to the maximum matching size. The main obstacle to applying the general reduction of Parnas and Ron from distributed algorithms is that the known distributed algorithms [30, 32] for maximum matching run in a number of rounds that is polylogarithmic in the graph size, not constant.

Nevertheless, we show that there exists a constant-time $(1, \varepsilon n)$-approximation algorithm for maximum matching for graphs of maximum degree bounded by $d \geq 2$ with query complexity $2^{d^{O(1/\varepsilon)}}$.

Maximum Weight Matching. For bounded-degree weighted graphs of all weights in $[0, 1]$, we show a constant-time algorithm that computes a $(1, \varepsilon n)$-approximation to the maximum weight matching. This can be achieved by combining our techniques with a greedy algorithm based on techniques of Pettie and Sanders [72].

Vertex Cover. We show that there exists a $(2, \varepsilon n)$-approximation algorithm of query complexity $2^{O(d)}/\varepsilon^2$, for graphs of maximum degree bounded by d. Combining the results of Parnas and Ron [70] and Marko and Ron [62] yields a $(2, \varepsilon n)$-approximation algorithm for the minimum vertex cover size of running time and query complexity $d^{O(\log(d/\varepsilon))}$. Our algorithm has better dependency on ε, but worse on d. Furthermore, Trevisan showed that for any constant $c \in [1, 2)$, a $(c, \varepsilon n)$-approximation algorithm must use at least $\Omega(\sqrt{n})$ queries (the result appeared in [70]). Parnas and Ron [70] also showed that any $(O(1), \varepsilon n)$-approximation algorithm must make $\Omega(d)$ queries as long as n/d is greater than some other constant (unless d is sufficiently close to n).

Set Cover. We also show an approximation algorithm for sparse instances of set cover. Let $H(i)$ be the i-th harmonic number $\sum_{1 \leq j \leq i} 1/j$. Recall that $H(i) \leq 1 + \ln i$. We show that there is an $(H(s), \varepsilon n)$-approximation algorithm of query complexity $\left(\frac{2^{(st)^4}}{\varepsilon}\right)^{O(2^s)}$ for the minimum set cover size, for instances with n sets S_i, each of size at most s, and with each element in at most t different sets. As a special case of the set cover problem, we get an $(H(d+1), \varepsilon n)$-approximation algorithm of query complexity $\left(\frac{2^{d^8}}{\varepsilon}\right)^{O(2^d)}$ for the minimum dominating set size for graphs of degree bounded by d.

Previously by combining the results of Parnas and Ron [70] and Kuhn, Moscibroda and Wattenhofer [52], one could obtain an $(O(\log d), \varepsilon n)$-approximation algorithm for minimum dominating set of query complexity $d^{O(\log d)}/\varepsilon^2$.

Other Problems. An example of a problem in the same framework (i.e., for bounded degree graphs) is the approximation of the minimum spanning tree weight for graphs of degree bounded by d. Chazelle, Rubinfeld and Trevisan [22] show that if all edge weights are in $\{1, 2, \ldots, w\}$, then there is an algorithm that computes a multiplicative $(1 + \varepsilon)$-approximation to the minimum weight spanning tree problem in time $O(\frac{dw}{\varepsilon^2} \log \frac{w}{\varepsilon})$. Due to the nature of this problem, it does not fit into our framework, and a constant-time approximation algorithm cannot be obtained via our general reduction.

An External Application to Property Testing

Our constant-time algorithm for approximating the maximum matching size has an application to property testing in the sparse graph model [40]. For graphs with an even number of vertices, it can be used for distinguishing graphs that have a perfect matching from those that need to have at least εdn edges added to achieve this property (these are the graphs that are ε-far from having a perfect matching). Clearly all graphs that have a perfect matching have a maximum matching of size $n/2$, while the graphs that are ε-far have no matching of size greater than $(1/2 - \varepsilon d)n$. It therefore suffices to approximate the maximum matching size up to an additive $\varepsilon dn/3$ to solve the promise property testing problem, which can be done in constant time.

1.1.2 Better Algorithms for Important Subclasses

For many graph problems, we encounter a multiplicative barrier $\alpha > 1$ such that we do not know how to compute an $(\alpha, \varepsilon n)$-approximation in constant time. The results of Alon [3] and Trevisan [70] show that in fact, there are no constant-time $(1, \varepsilon n)$-approximation algorithm for some for maximum independent set, dominating set, and vertex cover. We therefore turn to a large class of natural graph families, for which we show constant-time $(1, \varepsilon n)$-approximation algorithms for the problems mentioned above.

As an example, consider planar graphs. Lipton and Tarjan [58] discovered the separator theorem, which implies the following. Any planar graph with maximum degree bounded by d can be partitioned into small components of size at most $\text{poly}(d, 1/\varepsilon)$ by removing only an ε-fraction of edges for any $\varepsilon > 0$. Alon, Seymour, and Thomas [4] show a generalization of the separator theorem to any excluded minor.

We develop techniques for computing such a partition for minor-free families of graphs and other families of graphs with similar properties by looking only at a constant radius neighborhood of each vertex before deciding which component it is assigned to. We construct a *partitioning oracle* that given query access to a graph from a specific family of graphs, provides query access to a fixed partition, and queries a fraction of the graph independent of the graph size.

Just as knowing the entire partition is useful for finding a good approximate solution [59], our local version is useful for approximating the size of the optimal solution in time independent of the actual graph size. Our partitioning oracles also find applications to other approximating and testing problems that we describe in more detail later in this section.

Graph families

We construct partitioning oracles for hyperfinite families of graphs with bounded degree. Informally, hyperfinite graphs are those that can be partitioned into constant-size components by removing a small fraction of edges. A formal definitions follows.

Definition 1.1.2.

- *Let $G = (V, E)$ be a graph. G is* (ε, k)-hyperfinite *if it is possible to remove $\varepsilon|V|$ edges of the graph such that the remaining graph has connected components of size at most k.*

- *Let ρ be a function from $\mathbb{R}_+$ to $\mathbb{R}_+$. A graph G is* ρ-hyperfinite *if for every $\varepsilon > 0$, G is $(\varepsilon, \rho(\varepsilon))$-hyperfinite.*

- *Let $\mathcal{C}$ be a family of graphs. $\mathcal{C}$ is* ρ-hyperfinite *if every graph in $\mathcal{C}$ is ρ-hyperfinite.*

Examples of bounded-degree hyperfinite families of graphs include bounded-degree graphs with an excluded minor [4] (for instance, bounded-degree planar graphs, bounded-degree graphs with constant tree-width), bounded-degree graphs of subexponential growth[3] [34], and the family of non-expanding bounded-degree graphs considered by Czumaj, Shapira, and Sohler [28].

Previous Results

Recall that Trevisan [70] shows that there is no constant-time $(2-\delta, \varepsilon n)$-approximation algorithm for vertex cover, for any $\delta > 0$, for sufficiently high constant degree d. Alon [3] shows similar lower bounds for maximum independent set (no constant-time $(o(d/\log d), \varepsilon n)$-approximation algorithm) and dominating set (no constant-time $(o(\log d), \varepsilon n)$-approximation algorithm). We show that these lower bounds can be overcome for any bounded-degree hyperfinite family of graphs.

Czygrinow, Hańćkowiak, and Wawrzyniak [33] show that one can construct constant-time distributed $(1+\varepsilon)$-approximation algorithms for planar graphs for the above problems. This implies the existence of constant-time $(1, \varepsilon n)$-approximation algorithms for planar graphs via the connection observed by Parnas and Ron [70]. One can show that their construction implies algorithms that run in $2^{\text{quasipoly}(1/\varepsilon)}$ time for most standard problems.

Elek [35] proves the existence of constant-time $(1, \varepsilon n)$-approximation algorithms for minimum vertex cover, minimum dominating set, and maximum independent set for bounded-degree graphs of subexponential growth. His paper does not provide any explicit bounds on the running time.

Our Results

We show that the lower bounds of Alon and Trevisan can be overcome for any bounded-degree hyperfinite family of graphs. In fact, this is true for a slightly larger family of graph families with bounded *average* degree, which includes any family of

[3]The *growth* of a graph or a family of graphs is a function $g : \mathbb{Z}_+ \to \mathbb{Z}_+$ such that $g(d)$ equals the maximum number of vertices at distance at most d from any vertex d.

(unbounded degree) graphs with an excluded minor. More precisely, for any such family of graphs, there are constant-time $(1, \varepsilon n)$-approximation algorithms for minimum vertex cover, minimum dominating set, and maximum independent set. For any family of graphs with an excluded minor, the running time of our algorithms is $2^{\mathrm{poly}(1/\varepsilon)}$. Note that finding algorithms of running time $2^{(1/\varepsilon)^{o(1)}}$ is unlikely, since by setting $\varepsilon = 1/(3n)$, this would yield subexponential randomized algorithms for NP-hard problems. The above three problems are NP-hard for planar graphs, even with degree bounded by 3 [37, 38].

External Applications to Property Testing

Testing minor-closed properties. Another application of our techniques is to property testing in the bounded-degree model [40]. We say that a graph property[4] is *minor closed* if it is closed under removal of edges, removal of vertices, and edge contraction. Examples of minor-closed families of graphs include planar graphs, outerplanar graphs, graphs of genus bounded by a constant, graphs of tree-width bounded by a constant, and series-parallel graphs.

In the case being considered, the goal of an ε-tester for a given minor-closed property $\mathcal{P}$ is to distinguish, with probability $2/3$, graphs that satisfy $\mathcal{P}$ from those that need to have at least εn edges deleted to satisfy $\mathcal{P}$, where $\varepsilon > 0$. Goldreich and Ron [40] show an $O(1/\varepsilon^3)$ tester for the property that the input graph is a forest, i.e., does not have K_3 as a minor. For a long time this was the only minor-closed property known to be testable in constant-time. The breakthrough result of Benjamini, Schramm, and Shapira [18] shows that any minor-closed property can be tested in constant time, this was the only minor-closed property that was known to be testable in constant time. However, the running time of the tester of Benjamini, Schramm, and Shapira is $2^{2^{2^{\mathrm{poly}(1/\varepsilon)}}}$, and the analysis is quite involved. We give a simple proof of their result, and present a tester that runs in $2^{\mathrm{poly}(1/\varepsilon)}$ time.

[4]In this thesis, all graph properties are defined for graphs with no labels and are therefore closed under permutation of vertices.

Approximation of distance to hereditary properties. A graph property is *hereditary* if it is closed under removal of vertices. Many natural graph families are hereditary, including all minor-closed graph families, perfect graphs, bipartite graphs, k-colorable graphs, graphs with an excluded induced subgraph (see the appendix of [28] for a longer list of hereditary properties). All those properties are known be testable in constant time for dense graphs in the adjacency matrix model even with *one-sided error* [5], i.e., a graph having a given property cannot be rejected. This is not the case in the bounded-degree model. For instance, testing bipartiteness requires $\Omega(\sqrt{n})$ queries [40], and testing three-colorability requires $\Omega(n)$ queries [21]. Motivated by these lower bounds, Czumaj, Shapira, and Sohler [28] turned to testing properties of specific bounded-degree non-expanding families of graphs, which include minor-closed families of graphs. For those graphs, they show that all hereditary properties are testable in constant-time (with one-sided error). Their proof holds true for any hyperfinite family of bounded-degree graphs.

We say that a hereditary property $\mathcal{P}$ is *degenerate* if there is an empty graph on some number of vertices that does not have $\mathcal{P}$. For every non-degenerate hereditary $\mathcal{P}$, every hyperfinite family $\mathcal{C}$ of bounded-degree graphs, and every $\varepsilon > 0$, one can additively approximate the number of edges that must be modified (i.e., inserted or removed) up to εn to achieve $\mathcal{P}$ in time independent of the graph size for graphs in $\mathcal{C}$. It is impossible to specify the running time even for a fixed family of graphs, since $\mathcal{P}$ need not even be computable. Nevertheless, if a non-degenerate $\mathcal{P}$ can be specified via a (potentially infinite) set of forbidden *connected* induced graphs, and there is an $T(n)$-time algorithm that checks if a graph on n vertices has $\mathcal{P}$, we show that the distance to $\mathcal{P}$ to can be approximated in any fixed bounded-degree family of graphs with an excluded minor in $2^{\text{poly}(1/\varepsilon)} \cdot T(\text{poly}(1/\varepsilon))$ time.

The reason behind excluding degenerate hereditary properties is the following. Every degenerate $\mathcal{P}$ excludes an empty graph on k vertices, for some constant k, which implies that if a graph G has $\mathcal{P}$, then it does not have an independent set of size k, and therefore, has $\Omega(n^2)$ edges. Since the input graph has $O(n)$ edges, a large number of edges must be inserted. On the contrary, for every non-degenerate

hereditary property, the distance to the property is always of order $O(n)$, since it suffices to remove all edges to achieve the property.

A sample application of our result is a $(1, \varepsilon n)$-approximation algorithm for the number of edges that must be removed from the input bounded-degree planar graph to make it 3-colorable. The running time of the algorithm can be made $2^{\text{poly}(1/\varepsilon)}$. The result of Czumaj *et al.* [28] only guarantees the existence of a constant-time algorithm that for planar bounded-degree graphs, can tell 3-colorable graphs from those that need to have at least εn edges removed, for every $\varepsilon > 0$.

Independently, Elek [35] proves the existence of constant-time approximation algorithms for distance approximation to union-closed monotone properties in bounded-degree graphs of subexponential growth. Even though a union-closed monotone property need not be hereditary, all natural union-closed monotone properties are hereditary[5]. On the other hand, the perfectness property of graphs is hereditary, but is not monotone.

For general bounded degree graphs, Marko and Ron [62] give a constant-time $(O(1), \varepsilon)$-approximation algorithm for the distance to H-freeness, where H is an arbitrary fixed graph. They also show a constant-time $(1, \varepsilon)$-approximation algorithm for the distance to cycle-freeness.

Local distributed approximation algorithms. A distributed algorithm is *local* if it runs in a number of rounds that is independent of the size of the underlying graph.

Our partitioning oracles, which provide query access to a partition of vertices, can also be simulated locally by a distributed algorithm. In the distributed algorithm, every vertex collects a constant-radius neighborhood and their random bits. Then each vertex simulates the oracle's computation and partition. Given the partition, one can compute a good approximate solution to many combinatorial problems.

The paper of Czygrinow, Hańćkowiak, and Wawrzyniak [33] shows that local randomized $(1 + \varepsilon)$-approximation distributed algorithms can be constructed for planar

[5]If a union-closed monotone property is closed under removing an isolated vertex, then it is hereditary. All union-closed monotone properties listed by Elek [35] are hereditary and non-degenerate.

graphs, for every $\varepsilon > 0$, for all problems considered by us (i.e., minimum vertex cover, maximum independent set, and dominating set). Their techniques work for any family of minor-free graphs. Earlier research on efficient distributed algorithms for minor-free graphs includes papers of Czygrinow and Hańćkowiak [31], and Lenzen, Oswald, and Wattenhofer [55]

1.1.3 Follow-Up Work

Our paper [66] proposes a pruning heuristic that can be applied to many of our algorithms. However, despite experimental evidence suggesting a much better performance, we were unable to show a better theoretical bound. This issue is addressed by Yoshida, Yamamoto, and Ito [79], who show a better theoretical bound for the method with the heuristic. In particular, they show that there is a $(2, \varepsilon n)$-approximation algorithm for vertex cover running in $\tilde{O}(d^4/\varepsilon^2)$ time. This is the first algorithm for vertex cover that runs in time polynomial in both d and $1/\varepsilon$. For maximum matching, their improved analysis yields a $(1, \varepsilon n)$-approximation algorithm that runs in $d^{O(1/\varepsilon^2)}$ time.

In a recent unpublished manuscript [67], we further improve the algorithm for vertex cover. We reduce the number of neighbors read from each neighborhood list by sampling to only access a small fraction of them. The running time of our $(2, \varepsilon n)$-approximation algorithm is $\tilde{O}(d^2/\varepsilon^2)$.

Inspired by our research, Alon [3] shows a constant-time $(O(d \log\log d/\log d), \varepsilon n)$-approximation algorithm for maximum independent set, and showed that there is no constant-time $(o(d/\log d), \varepsilon n)$-approximation algorithm for this problem. For dominating set, he shows that there is no constant-time $(\ln d, \varepsilon n)$-approximation algorithm.

1.1.4 A Note on Bounded Average-Degree Instances

Parnas and Ron [70] observe that one can easily turn some algorithms for bounded maximum-degree graphs into algorithms for bounded average-degree graphs. Consider the maximum matching problem. If the average degree is at most $\hat{d}$, then there are at most εn vertices of degree higher than $\hat{d}/\varepsilon$. Removing them from the graph

changes the maximum matching size by at most εn. Therefore, running the algorithm for bounded maximum degree and ignoring all vertices of degree greater than $\hat{d}/\varepsilon$ gives a $(1, 2\varepsilon)$-approximate solution.

For vertex cover, it is also safe to remove the high degree vertices. This changes the vertex cover size by at most εn as well. For dominating set, one can assume that the vertices of high degree belong to the dominating set. Again, this changes the solution size by at most εn, but this time, we cannot simply remove them from the graph. Instead whenever we see one of them from one their neighbors, we know that the given neighbor is already covered in the execution of the constant-time set cover algorithm, which we use for approximating the dominating set size.

Additionally, the above reductions can be performed in a similar way for dense graphs with few vertices of high degree. It suffices to find a bound d_ε such that all but an ε fraction of vertex degrees are bounded by d_ε. Then all vertices of degree higher than d_ε are considered to have high degree.

1.2 Edit Distance

In the second part of the thesis, we study an *asymmetric query* model for edit distance. In this model, the input consists of two strings x and y, and an algorithm can access y in an unrestricted manner, while being charged for querying every symbol of x. We both design an algorithm that makes a small number of queries in this model, and provide a nearly matching lower bound on the number of queries.

By using our sublinear-query algorithm, which also has relatively low running time, we obtain the first algorithm that approximates edit distance up to a polylogarithmic factor in $O(n^c)$ time, where c is a constant less than 2.

1.2.1 Historical Background

Manipulation of strings has long been central to computer science, arising from the high demand to process texts and other sequences efficiently. For example, for the simple task of *comparing* two strings (sequences), one of the first methods emerged

to be the *edit distance* (aka the Levenshtein distance) [56], defined as the minimum number of character insertions, deletions, and substitutions needed to transform one string into the other. This basic distance measure, together with its more elaborate versions, is widely used in a variety of areas such as computational biology, speech recognition, and information retrieval. Consequently, improvements in edit distance algorithms have the potential of major impact. As a result, computational problems involving edit distance have been studied extensively (see [65, 41] and references therein).

The most basic problem is that of computing the edit distance between two strings of length n over some alphabet. It can be solved in $O(n^2)$ time by a classical algorithm [78]; in fact this is a prototypical dynamic programming algorithm, see, e.g., the textbook [24] and references therein. Despite significant research over more than three decades, this running time has so far been improved only slightly to $O(n^2/\log^2 n)$ [63], which remains the fastest algorithm known to date.[6]

Still, a near-quadratic runtime is often unacceptable in modern applications that must deal with massive datasets, such as the genomic data. Hence practitioners tend to rely on faster heuristics [41, 65]. This has motivated the quest for faster algorithms at the expense of approximation, see, e.g., [45, Section 6] and [46, Section 8.3.2]. Indeed, the past decade has seen a serious effort in this direction.[7] One general approach is to design linear time algorithms that approximate the edit distance. A linear-time $\sqrt{n}$-approximation algorithm immediately follows from the exact algorithm of [54], which runs in time $O(n + d^2)$, where d is the edit distance between the input strings. Subsequent research improved the approximation factor, first to $n^{3/7}$ [15], then to $n^{1/3+o(1)}$ [17], and finally to $2^{\tilde{O}(\sqrt{\log n})}$ [10] (building on [68]). Predating some of this work was the *sublinear-time* algorithm of [16] achieving n^{ε} approximation, but only when the edit distance d is rather large.

[6]The result of [63] applies to constant-size alphabets. It was recently extended to arbitrarily large alphabets, albeit with an $O(\log \log n)^2$ factor loss in runtime [19].

[7]We shall not attempt to present a complete list of results for restricted settings (e.g., average-case/smoothed analysis, weakly-repetitive strings, and bounded distance-regime), for variants of the distance function (e.g., allowing more edit operations), or for related computational problems (such as pattern matching, nearest neighbor search, and sketching). See also the surveys of [65] and [76].

Better progress has been obtained on *variants* of edit distance, where one either restricts the input strings, or allows additional edit operations. An example from the first category is the edit distance on non-repetitive strings (e.g., permutations of $[n]$), termed *the Ulam distance* in the literature. The classical Patience Sorting algorithm computes the exact Ulam distance between two strings in $O(n \log n)$ time. An example in the second category is the case of two variants of the edit distance where certain block operations are allowed. Both of these variants admit an $\tilde{O}(\log n)$ approximation in near-linear time [27, 64, 26, 25].

Despite the efforts, achieving a polylogarithmic approximation factor for the classical edit distance has eluded researchers for a long time. In fact, this is has been the case not only in the context of linear-time algorithms, but also in the related tasks, such as nearest neighbor search, ℓ_1-embedding, or sketching. From a lower bounds perspective, only a *sublogarithmic* approximation has been ruled out for the latter two tasks [50, 51, 8], thus giving evidence that a sublogarithmic approximation for the distance computation might be much harder or even impossible to attain.

1.2.2 Results

Our first and main result is an algorithm that runs in near-linear time and approximates edit distance within a *polylogarithmic factor*. Note that this is *exponentially better* than the previously known factor $2^{\tilde{O}(\sqrt{\log n})}$ (in comparable running time), due to [68, 10].

Theorem 1.2.1 (Main). *For every fixed $\varepsilon > 0$, there is an algorithm that approximates the edit distance between two input strings $x, y \in \Sigma^n$ within a factor of $(\log n)^{O(1/\varepsilon)}$, and runs in $n^{1+\varepsilon}$ time.*

This development stems from a principled study of edit distance in a computational model that we call the *asymmetric query* model, and which we shall define shortly. Specifically, we design a query-efficient procedure in the said model, and then show how this procedure yields a near-linear time algorithm.

We also provide a query complexity lower bound for this model, which matches or

nearly-matches the performance of our procedure. A conceptual contribution of our query complexity lower bound is that it is the first one to expose hardness stemming from "repetitive substrings", which means that many small substrings of a string may be approximately equal. Empirically, it is well-recognized that such repetitiveness is a major obstacle for designing efficient algorithms. All previous lower bounds (in any computational model) failed to exploit it, while in our proof the strings' repetitive structure is readily apparent. More formally, our lower bound provides the first rigorous separation of edit distance from Ulam distance (edit distance on non-repetitive strings). Such a separation was not previously known in any studied model of computation, and in fact all the lower bounds known for the edit distance hold to (almost) the same degree for the Ulam distance. These models include: non-embeddability into normed spaces [50, 51, 8], lower bounds on sketching complexity [8, 7], and (symmetric) query complexity [16, 9].

Asymmetric Query Complexity. Before stating the results formally, we define the problem and the model precisely. Consider two strings $x, y \in \Sigma^n$ for some alphabet Σ, and let $\mathrm{ed}(x, y)$ denote the edit distance between these two strings. The computational problem is the promise problem known as the Distance Threshold Estimation Problem (DTEP) [77]: distinguish whether $\mathrm{ed}(x, y) > R$ or $\mathrm{ed}(x, y) \leq R/\alpha$, where $R > 0$ is a parameter (known to the algorithm) and $\alpha \geq 1$ is the *approximation factor*. We use DTEP_β to denote the case of $R = n/\beta$, where $\beta \geq 1$ may be a function of n.

In the *asymmetric query model*, the algorithm knows in advance (has unrestricted access to) one of the strings, say y, and has only *query access* to the other string, x. The *asymmetric query complexity* of an algorithm is the number of coordinates in x that the algorithm has to probe in order to solve DTEP with success probability at least $2/3$.

We now give complete statements of our upper and lower bound results. Both exhibit a smooth *tradeoff* between approximation factor and query complexity. For simplicity, we state the bounds in two extreme regimes of approximation ($\alpha = \mathrm{polylog}(n)$ and $\alpha = \mathrm{poly}(n)$). See Theorem 3.2.1 for the full statement of the upper bound, and Theorems 3.3.15 and 3.3.16 for the full statement of the lower bound.

Theorem 1.2.2 (Query complexity upper bound). *For every $\beta = \beta(n) \geq 2$ and fixed $0 < \varepsilon < 1$ there is an algorithm that solves* DTEP_β *with approximation $\alpha = (\log n)^{O(1/\varepsilon)}$, and makes βn^ε asymmetric queries. This algorithm runs in time $O(n^{1+\varepsilon})$.*

For every $\beta = O(1)$ and fixed integer $t \geq 2$ there is an algorithm for DTEP_β *achieving approximation $\alpha = O(n^{1/t})$, with $O(\log^{t-1} n)$ queries into x.*

It is an easy observation that our general edit distance algorithm in Theorem 1.2.1 follows immediately from the above query complexity upper bound theorem, by running the latter for all β that are a power of 2.

Theorem 1.2.3 (Query complexity lower bound). *For a sufficiently large constant $\beta > 1$, every algorithm that solves* DTEP_β *with approximation $\alpha = \alpha(n) > 2$ has asymmetric query complexity $2^{\Omega\left(\frac{\log n}{\log \alpha + \log\log n}\right)}$. Moreover, for every fixed non-integer $t > 1$, every algorithm that solves* DTEP_β *with approximation $\alpha = n^{1/t}$ has asymmetric query complexity $\Omega(\log^{\lfloor t \rfloor} n)$.*

We summarize in Table 1.1 our results and previous bounds for DTEP_β under edit distance and Ulam distance. For completeness, we also present known results for a common query model where the algorithm has query access to both strings (henceforth referred to as the *symmetric query* model). We point out two implications of our bounds on the asymmetric query complexity:

- There is a strong separation between edit distance and Ulam distances. In the Ulam metric, a *constant* approximation is achievable with only $O(\log n)$ asymmetric queries (see [2], which builds on [36]). In contrast, for edit distance, we show an exponentially higher complexity lower bound, of $2^{\Omega(\log n/\log\log n)}$, even for a larger (polylogarithmic) approximation.

- Our query complexity upper and lower bounds are nearly-matching, at least for a range of parameters. At one extreme, approximation $O(n^{1/2})$ can be achieved with $O(\log n)$ queries, whereas approximation $n^{1/2-\varepsilon}$ already requires $\Omega(\log^2 n)$ queries. At the other extreme, approximation $\alpha = (\log n)^{1/\varepsilon}$ can be achieved using $n^{O(\varepsilon)}$ queries, and requires $n^{\Omega(\varepsilon/\log\log n)}$ queries.

Model	Metric	Approx.	Complexity	Remarks
Near-linear time	Edit	$(\log n)^{O(1/\varepsilon)}$	$n^{1+\varepsilon}$	Theorem 1.2.1
	Edit	$2^{\tilde{O}(\sqrt{\log n})}$	$n^{1+o(1)}$	[10]
Symmetric query complexity	Edit	n^{ε}	$\tilde{O}(n^{\max\{1-2\varepsilon,(1-\varepsilon)/2\}})$	[16] (fixed $\beta > 1$)
	Ulam	$O(1)$	$\tilde{O}(\beta + \sqrt{n})$	[9]
	Ulam+edit	$O(1)$	$\tilde{\Omega}(\beta + \sqrt{n})$	[9]
Asymmetric query complexity	Edit	$n^{1/t}$	$O(\log^{t-1} n)$	Theorem 1.2.2 (fixed $t \in \mathbb{N}, \beta > 1$)
	Edit	$n^{1/t}$	$\Omega(\log^{\lfloor t \rfloor} n)$	Theorem 1.2.3 (fixed $t \notin \mathbb{N}, \beta > 1$)
	Edit	$(\log n)^{1/\varepsilon}$	$\beta n^{O(\varepsilon)}$	Theorem 1.2.2
	Edit	$(\log n)^{1/\varepsilon}$	$n^{\Omega(\varepsilon/\log\log n)}$	Theorem 1.2.3 (fixed $\beta > 1$)
	Ulam	$2 + \varepsilon$	$O_{\varepsilon}(\beta \log\log\beta \cdot \log n)$	[2]

Table 1.1: Known results for DTEP_{β} and arbitrary $0 < \varepsilon < 1$.

1.2.3 Connections of the Asymmetric Query Model to Other Models

The asymmetric query model is related and has implications for two previously studied models, namely the communication complexity model and the symmetric query model (where the algorithm has query access to both strings). Specifically, the former is less restrictive than our model (i.e., easier for algorithms) while the latter is more restrictive (i.e., harder for algorithms). Our upper bound gives an $O(\beta n^{\varepsilon})$ one-way communication complexity protocol for DTEP_{β} for polylogarithmic approximation.

Communication Complexity. In this setting, Alice and Bob each have a string, and they need to solve the DTEP_{β} problem by way of exchanging messages. The measure of complexity is the number of bits exchanged in order to solve DTEP_{β} with probability at least $2/3$.

The best non-trivial upper bound known is $2^{\tilde{O}(\sqrt{\log n})}$ approximation with constant communication via [68, 53]. The only known lower bound says that approximation α requires $\Omega(\frac{\log n \ / \ \log\log n}{\alpha})$ communication [8, 7].

The asymmetric model is "harder", in the sense that the query complexity is at least the communication complexity, up to a factor of $\log|\Sigma|$ in the complexity, since Alice and Bob can simulate the asymmetric query algorithm. In fact, our upper bound implies a communication protocol for the same DTEP_{β} problem with the same complexity, and it is a one-way communication protocol. Specifically, Alice can

just send the $O(\beta n^{\varepsilon})$ characters queried by the query algorithm in the asymmetric query model. This is the first communication protocol achieving polylogarithmic approximation for DTEP_β under edit distance with $o(n)$ communication.

Symmetric Query Complexity. In another related model, the measure of complexity is the number of characters the algorithm has to query in *both* strings (rather than only in one of the strings). Naturally, the query complexity in this model is at least as high as the query complexity in the asymmetric model. This model has been introduced (for the edit distance) in [16], and its main advantage is that it leads to *sublinear-time* algorithms for DTEP_β. The algorithm of [16] makes $\tilde{O}(n^{1-2\varepsilon}+n^{(1-\varepsilon)/2})$ queries (and runs in the same time), and achieves n^{ε} approximation. However, it only works for $\beta = O(1)$.

In the symmetric query model, the best query lower bound is of $\Omega(\sqrt{n/\alpha})$ for any approximation factor $\alpha > 1$ for both edit and Ulam distance [16, 9]. The lower bound essentially arises from the birthday paradox. Hence, in terms of separating edit distance from the Ulam metric, this symmetric model can give at most a quadratic separation in the query complexity (since there exists a trivial algorithm with $2n$ queries). In contrast, in our asymmetric model, there is no lower bound based on the birthday paradox, and, in fact, the Ulam metric admits a constant approximation with $O(\log n)$ queries [36, 2]. Our lower bound for edit distance is exponentially bigger.

Chapter 2

Combinatorial Problems on Graphs

In this chapter, we prove the graph approximation results described in the introduction (Section 1.1). For instance, we show a general transformation that turns many classical greedy algorithms into constant-time algorithms for the optimal solution size.

2.1 Definitions and the Model

We first restate relevant definitions from the introduction and recall the query model that we assume.

Constant-Time Algorithms. We say that a graph algorithm runs in *constant time* if its running time is bounded by a function of the maximum degree.

The Approximation Notion. We say that a value $\hat{y}$ is an *(α, β)-approximation* to y if

$$y \leq \hat{y} \leq \alpha \cdot y + \beta.$$

We say that an algorithm A is an *(α, β)-approximation algorithm* for a value $V(x)$ if it computes an (α, β)-approximation to $V(x)$ with probability at least 2/3 for any proper input x.

For simplicity, whenever we talk about an *(α, β)-approximation algorithm for problem $\mathcal{X}$* in this part of the thesis, we, in fact, mean an *(α, β)-approximation algorithm*

for the optimal solution size to problem $\mathcal{X}$.

The Query Model. With a small exception, the input to problems considered in this part of the thesis is a graph. We make the following assumptions about how an algorithm can access the input.

- The algorithm can select a vertex in the graph uniformly at random. The operation takes constant time.

- The algorithm can query the adjacency list of each vertex. It can make two types of queries, both answered in constant time. In the *degree query*, the algorithm finds out the degree of a vertex it specifies. By making a *neighbor query*, the algorithm learns the identity of the i-th neighbor of a vertex v, where both i and v are specified by the algorithm. Additionally, if the graph is weighted, the algorithm learns the weight of the edge connecting v with its i-th neighbor.

We also consider the set cover problem. Let $\mathcal{U}$ be the set of all elements and let S_i be the sets with which we conver $\mathcal{U}$. In this case we assume that the algorithm can uniformly select a random set S_i. Furthermore, for every element $x \in \mathcal{U}$, the algorithm can query the list of the sets S_i that contain x, and for every set S_i, the algorithm can query the list of the elements of S_i.

2.2 Simple Example: Vertex Cover via Maximal Matching

We start with an example illustrating a few of the main ideas behind our algorithms. We show how to compute a $(2, \varepsilon n)$-approximation to the minimum vertex cover size by locally computing a maximal independent set.

Algorithm. Our graph approximation algorithms follow a general framework of Parnas and Ron [70]. We construct an oracle that provides query access to a good

solution for the problem, and then make a few random queries to the oracle to estimate the size of the solution it provides.

To compute a $(2, \varepsilon n)$-approximation to vertex cover, we first design an oracle $\mathcal{O}$ that provides query access to a fixed *maximal* matching M. More precisely, the oracle answers queries of the form "Does (u, v) belong to M?" for every edge (u, v) of the graph. Using the connection discovered by Gavril [38] and Yannakakis [69], the set of vertices matched by any maximal matching is a vertex cover of size at most twice the optimum. Let us denote this set by S. We can construct an oracle $\mathcal{O}'$ that provides query access to S. For every query of the form "Does v belong to S?", $\mathcal{O}'$ queries $\mathcal{O}$ to find out if any of the edges incident to v belongs to M.

Given query access to $\mathcal{O}'$, we can now approximate the size of of S, and indirectly, the size of the optimal vertex cover. We select uniformly and independently at random $O(1/\varepsilon^2)$ vertices V'. We query $\mathcal{O}'$ to compute the fraction of vertices in V' that belong to S. Via the Hoeffding bound, with large constant probability (say, 99/100), this value is also an additive $\varepsilon/2$ approximation to the fraction of all vertices that belong to the vertex cover. By first adding to this value $\varepsilon/2$, and then multiplying it by n, we obtain the desired $(2, \varepsilon n)$-approximation to the minimum vertex cover size.

Our main contribution is a new way of implementing oracles such as the above for several problems. We now present our implementation of $\mathcal{O}$, the oracle for Maximal Matching. A random number $r(e) \in [0, 1]$ is assigned to each edge e of the graph[1]. In order to decide if an edge q is in the matching, the oracle first determines the set of edges adjacent to q of numbers $r(e)$ smaller than that of q, and recursively checks if any of them is in the matching. If at least one of the adjacent edges is in the matching, q is not in the matching; otherwise, it is.

Why does this work? Consider first the following trivial greedy algorithm for finding a maximal matching. The algorithm starts with an empty matching M. For

[1] In an implementation, we do not assign all numbers $r(e)$ at the beginning. We can postpone the assignment of the random number $r(e)$ to an edge e until we need it. After learning $r(e)$, we store it in case we need it later again. Moreover, arbitrary random real numbers in $[0, 1]$ cannot be generated in practice. Nevertheless, it suffices to discretize the range $[0, 1]$ so that two edges are assigned the same number with negligibly small probability. For instance, it suffices to assign to each edge a random integer between 0 and $100 \cdot n^4$.

each edge e, it checks if there is already an adjacent edge in M. If there is no such edge, it adds e to M. The final M is clearly a maximal matching, since every edge not in M is adjacent to at least one of the edges in M. Our oracle simulates this algorithm, considering edges in random order (which is generated by the random numbers $r(e)$).

Query Complexity. It remains to bound the query complexity of the algorithm. We analyze the above method in the next section, and show that the above algorithm runs in $2^{O(d)}/\varepsilon^2$ time and outputs a $(2, \varepsilon n)$-approximation to the minimum vertex cover size with probability at least $2/3$.

2.3 A Local Computation Method

We now formalize and generalize the local computation method that we saw in the previous section. We used it to determine locally whether a given edge was in a fixed maximal matching that was not a function of queries.

2.3.1 The Method

Now instead of focusing on edges as in the previous section, we want to run the recursive method on vertices to compute a maximal independent set. The maximal matching is simply a maximal independent set in the line graph[2] of the input graph, and the input graph can easily be remapped into its line graph locally. That is, every query to the line graph can be simulated with $O(1)$ queries to the input graph. We present this procedure for locally computing a maximal independent set as `Local_MIS` (see Algorithm 1).

The procedure for computing a maximal independent set has a generalization, which we need for some applications. As before, we independently and uniformly assign a random number $r(v) \in [0, 1]$ (after a proper discretization) to each vertex v.

[2]The line graph G_L of a graph $G = (V, E)$ is the graph with every vertex corresponding to an edge in E, and two vertices connected if the corresponding edges in E share an endpoint.

Algorithm 1: The recursive procedure `Local_MIS`(v) for locally computing a maximal independent set. The set is a function of random number $r(w)$ assigned to each vertex w. The procedure outputs **TRUE** on a vertex v if and only if v is in the maximal independent set.

```
1 lower_neighbors := the set of neighbors w of v such that r(w) < r(v)
2 in_the_set := TRUE
3 for w ∈ lower_neighbors do
4 |  if Local_MIS(w) then
5 |  |  in_the_set := FALSE
6 return in_the_set
```

Algorithm 2: The generic version `Generic_Local`(v) of the recursive local computation procedure. The procedure outputs a single value for each vertex. It uses a function f to compute the value for v out of the values for neighbors.

```
1 lower_neighbors := the set of neighbors w of v such that r(w) < r(v)
2 neighbor_value_set := ∅
3 for w ∈ lower_neighbors do
4 |  neighbor_value_set := neighbor_value_set ∪ {w, Generic_Local(w)};
5 return f(v, neighbor_value_set)
```

These numbers provide a random ordering of all vertices. We compute a value $x(v)$ for each vertex v. The value for a given vertex is a function of the vertex and the values computed for the neighbors w of v with numbers $r(w) < r(v)$. We present the modified procedure `Generic_Local` as Algorithm 2.

This procedure can be used to simulate locally a generic greedy algorithm that consider vertices one by one. We employ it later in the chapter. A notable use of this version of the procedure is described in Section 2.5.1, where it is used for locally computing a partition of the input graph.

2.3.2 Non-Adaptive Performance

We now bound the expected number of recursive calls that the local computation method needs for a fixed vertex.

Lemma 2.3.1. *Let G be a graph of maximum degree at most d. For any fixed vertex, the expected number of recursive calls of the local computation method is bounded by*

e^d/d.

Proof. The probability that the local computation method follows a specific path of length k is exactly $1/(k+1)!$, which is the probability that random numbers assigned to vertices on the path are in decreasing order. The number of vertices reachable via paths of length k is bounded by d^k. Therefore, the expected number of visited vertices at distance k is bounded by $d^k/(k+1)!$ via the linearity of expectation. By summing over all non-negative integers, which correspond to all possible distances, we obtain a bound on the expected number of recursive calls of the local computation method

$$\sum_{k=0}^{\infty} \frac{d^k}{(k+1)!} = \frac{1}{d} \sum_{k=0}^{\infty} \frac{d^{k+1}}{(k+1)!} \leq \frac{e^d}{d}.$$

□

Using Lemma 2.3.1, we immediately obtain a bound on the number of queries that the vertex cover algorithm from Section 2.2 uses in a graph with maximum degree bounded by d. The algorithm selects $O(1/\varepsilon^2)$ vertices, and then, for each edge incident to these vertices, it runs the recursive local computation method. The number of edges incident to the selected vertices is bounded by $O(d/\varepsilon^2)$, and the degree of vertices in the line graph is bounded by $d' = 2(d-1)$. Therefore, each execution of the local computation method makes $e^{d'}/d'$ recursive calls in expectation. By the linearity of expectation, the total expected number of recursive calls is bounded by $O\left(e^{d'}/\varepsilon^2\right)$. The query complexity of each recursive call is bounded by $O(d)$. The expected query complexity is therefore $O\left(d \cdot e^{2d-2}/\varepsilon^2\right)$. By Markov's inequality the query complexity is bounded by $O\left(d \cdot e^{2d-2}/\varepsilon^2\right)$ with probability 99/100. Assuming that the number of sampled vertices is large enough that the algorithm errs with probability at most 99/100 (due to the Hoeffding inequality), the algorithm outputs a $(2, \varepsilon n)$-approximation using $O\left(d \cdot e^{2d-2}/\varepsilon^2\right)$ queries with probability 49/50. Additionally, storing and accessing the random numbers $r(v)$ that have been already generated requires keeping a standard dictionary. This may result in an additional logarithmic factor in the running time.

Corollary 2.3.2. *There is a $(2, \varepsilon n)$-approximation algorithm for the minimum vertex cover size that makes $Q = 2^{O(d)}/\varepsilon^2$ queries and runs in $O(Q \log Q)$ time on graphs with maximum degree bounded by d.*

2.3.3 Adaptive Performance

In some applications, the set of vertices for which we run the local computation algorithm is not fixed in advance. The following lemma shows that any sequence of even adaptive queries is unlikely to result in a large number of recursive calls.

Lemma 2.3.3 (Locality Lemma). *Let G be a graph of maximum degree at most d. The local computation method is run from q vertices, where the next vertex can be a function of the previous start vertices and the output of the local computation method for them. The total number of recursive calls of the local computation method is bounded by $\frac{q^2}{\delta} \cdot C^{d^4}$ with probability at least $1 - \delta$, for any $\delta > 0$, where C is an absolute constant.*

Proof. Let us start with a few auxiliary definitions. We say that a node v can be *reached* or is *reachable* from a node w if there is a path $u_0 = w, u_1, \ldots, u_k = v$ such that $r(u_{i-1}) > r(u_i)$, for all $1 \leq i \leq k$. In other words, in order to compute the value $x(w)$, the local recursive computation method has to compute $x(v)$. The *reachability radius* $\mathrm{rr}(v)$ of a node v is the maximum length of a decreasing path of $r(u)$'s that starts at v.

Consider any algorithm $\mathcal{A}$ adaptively selecting q vertices that are starting points of the local recursive computation method. Let t be a non-negative integer. We give a bound on the probability that one of the selected vertices has reachability radius greater than t. We define the following events for each $i \in \{0, \ldots, q\}$:

1. A_i: each of the first i selected vertices has reachability radius at most t.

2. B_i: each vertex at distance at most $2(t+1)$ of the first i selected vertices has reachability radius at most t.

It is clear that each B_i is a more specific version of the corresponding A_i. Therefore, $\Pr[A_i] \geq \Pr[B_i]$. Moreover, let V_i, $1 \leq i \leq q$, be (a random variable representing) the set of vertices that are at distance greater than $2(t+1)$ from the first $i-1$ vertices selected by $\mathcal{A}$, and are at distance at most $2(t+1)$ from the i-th selected vertex.

For $i \in \{0, \dots, q-1\}$, we have

$$\begin{aligned}
\Pr[\neg B_{i+1}] &= \Pr[\neg B_i] + \Pr[B_i \wedge \neg B_{i+1}] \\
&= \Pr[\neg B_i] + \Pr[B_i \wedge \exists v \in V_{i+1}.\, \mathrm{rr}(v) \geq t] \\
&\leq \Pr[\neg B_i] + \Pr[A_i \wedge \exists v \in V_{i+1}.\, \mathrm{rr}(v) \geq t] \\
&\leq \Pr[\neg B_i] + \Pr[A_i] \cdot \Pr[\exists v \in V_{i+1}.\, \mathrm{rr}(v) \geq t | A_i] \\
&\leq \Pr[\neg B_i] + \Pr[\exists v \in V_{i+1}.\, \mathrm{rr}(v) \geq t | A_i].
\end{aligned}$$

We now bound $\Pr[\exists v \in V_{i+1}.\, \mathrm{rr}(v) \geq t | A_i]$. Note that if the event A_i is the case, then the algorithm does not know any random value $r(u)$ for vertices u at distance more than $t+1$ away from the vertices it selected earlier. For every vertex v in V_{i+1}, the event that the reachability radius of v is at most t is only conditioned on the values $r(u)$ for u at distance at most $t+1$. Therefore, if the event A_i holds, whatever the next vertex $\mathcal{A}$ selects, for every vertex v in V_{i+1}, the event that the reachability radius of each vertex in V_{i+1} is bounded by t is independent of the algorithm's knowledge.

What is the probability that for a given vertex v, the reachability radius is greater than t? The probability can be bounded by the probability that there is a path of length $t+1$ that starts at v, and the values r are strictly decreasing along the path. There are at most $d \cdot (d-1)^t$ such paths, and by symmetry, the probability that the values r decrease along a given path is $1/(t+2)!$ Hence the probability of the event is at most $\frac{d(d-1)^t}{(t+2)!}$ by the union bound.

The size of V_{i+1} is bounded by the number of vertices at distance at most $2(t+1)$ from the vertex that $\mathcal{A}$ selects next. It is at most

$$1 + d\sum_{i=0}^{2t+1}(d-1)^i \leq 1 + d\sum_{i=0}^{2t+1} d^i \leq d^{2t+3}.$$

Applying the union bound again, we obtain

$$\Pr[\exists v \in V_{i+1}.\, \mathrm{rr}(v) \geq t | A_i] \leq d^{2t+3} \cdot \frac{d(d-1)^t}{(t+2)!} \leq \frac{d^{3t+4}}{(t+2)!} \leq \left(\frac{3d^3}{t+2}\right)^{t+2}.$$

Finally, we have

$$\Pr[\neg A_i] \leq \Pr[\neg B_i] \leq i \cdot \left(\frac{3d^3}{t+2}\right)^{t+2}.$$

Let E_i be the event that the maximum reachability radius for all selected vertices is exactly i, and $E_{>i}$ the event that it is greater than i. We have, $\Pr[E_{>i}] \leq q \cdot \left(\frac{3d^3}{i+2}\right)^{i+2}$.

What is the expected total number T of vertices for which the local computation method is run? It is

$$\begin{aligned} T &\leq \sum_{i \geq 0} \Pr[E_i] \cdot q(1 + d \cdot \sum_{0 \leq j \leq i-1} (d-1)^j) \\ &\leq \sum_{i \geq 0} \Pr[E_i] \cdot q d^{i+1} \leq \sum_{i \geq 0} \Pr[E_{>i-1}] \cdot q d^{i+1} \\ &\leq \sum_{i \geq 0} q \left(\frac{3d^3}{i+1}\right)^{i+1} \cdot q d^{i+1} \leq q^2 \sum_{i \geq 0} \left(\frac{3d^4}{i+1}\right)^{i+1}. \end{aligned}$$

For $i \geq 6d^4 - 1$, we have

$$\sum_{i \geq 6d^4-1} \left(\frac{3d^4}{i+1}\right)^{i+1} \leq \sum_{i \geq 6d^4-1} 2^{-(i+1)} \leq 1.$$

Using calculus, one can show that the term $\left(\frac{3d^4}{i+1}\right)^{i+1}$ is maximized for $i+1 = \frac{3d^4}{e}$, and hence,

$$\sum_{i < 6d^4-1} \left(\frac{3d^4}{i+1}\right)^{i+1} \leq (6d^4 - 1) \cdot e^{\frac{3d^4}{e}}.$$

Algorithm 3: The recursive procedure `Local_MIS_2`(v) for locally computing a maximal independent set. The set is a function of random numbers $r(w)$ assigned to each vertex w. This a modified version of `Local_MIS`(v) (see Algorithm 1) that uses pruning to reduce the query complexity. The procedure outputs **TRUE** on a vertex v if and only if v is in the maximal independent set.

```
1 lower_neighbors := the set of neighbors w of v such that r(w) < r(v)
2 sorted := vertices w in lower_neighbors in ascending order of their r(w)
3 for i = 1,...,length(sorted) do
4     if Local_MIS_2(sorted[i]) then
5         return FALSE
6 return TRUE
```

We get

$$\begin{aligned} T &\leq q^2 \left(1 + (6d^4 - 1) \cdot e^{\frac{3d^4}{\epsilon}}\right) \\ &\leq q^2 \cdot 6d^4 \cdot e^{\frac{3d^4}{\epsilon}} \leq q^2 C^{d^4}, \end{aligned}$$

for some constant C. By Markov's inequality, the probability that the number of queries is greater than T/δ is at most δ. Hence with probability at least $1 - \delta$, the number of queries is bounded by $q^2 C^{d^4}/\delta$. □

2.3.4 Recent Improvements

In our paper [66], we suggested a heuristic that prunes the search tree of the local computation method for the problem of locally determining a maximal independent set. For a given vertex v, we consider the neighbors w with $r(w) < r(v)$ in order of their random numbers $r(v)$. As soon as we find a neighbor w in the maximal independent set, we interrupt the exploration of neighbors, since we already know that v is not in the maximal independent set. We give pseudocode for the modified procedure as Algorithm 3.

Yoshida, Yamamoto, and Ito [79] prove the following fact about the behavior of our heuristic for a random start vertex.

Theorem 2.3.4 (Theorem 2.1 in [79]). *The expected number of recursive calls that*

Algorithm 3 makes for a random *start vertex v is at most $1 + m/n$, where m and n are the number of edges and vertices in the graph, respectively.*

Using this theorem, Yoshida *et al.* show that there is a $(2, \varepsilon n)$-approximation algorithm for vertex cover that runs in $\tilde{O}(d^4/\varepsilon^2)$ time. In a recent unpublished manuscript [67], we improve the algorithm further. Our algorithm for approximating the vertex cover size runs in $\tilde{O}(d^2/\varepsilon^2)$ time.

2.4 General Transformation for Greedy Algorithms

2.4.1 Technique High-Level Description

We now give a high-level overview of our technique, which we applied in a specific setting in Section 2.2. We also briefly describe conditions that must be met in order to make our technique applicable.

Our technique transforms an algorithm $\mathcal{A}$ that computes an approximate solution to a problem into a constant-time algorithm that approximates the size of an optimal solution, provided $\mathcal{A}$ meets certain criteria. We require that $\mathcal{A}$ compute the approximate solution in a constant number of phases such that each phase is an application of any maximal set of disjoint local improvements. (The local improvements considered in each phase may be different.) Moreover, to achieve a constant running time, we require that each local improvement considered in a given phase intersect with at most a constant number of other considered local improvements. For example, the maximal matching algorithm of Section 2.2 constructs a maximal matching in just one phase, by taking a maximal set of non-adjacent edges.

The general idea behind the new constant-time algorithm that we construct is the following. Let k be the number of phases in the algorithm. For the i-th phase, where $1 \leq i \leq k$, we construct an oracle $\mathcal{O}_i$ that implements query access to the intermediate solution constructed by the i-th phase of the algorithm. ($\mathcal{O}_0$ gives the initial solution that the algorithm starts with.) $\mathcal{O}_i$ is itself given query access to $\mathcal{O}_{i-1}$, and simulates the i-th phase of the algorithm on the output of the $(i-1)$-st phase.

Finally, $\mathcal{O}_k$ provides query access to a solution that $\mathcal{A}$ could output. By using random queries to $\mathcal{O}_k$, we approximate the size of the solution computed by $\mathcal{A}$.

2.4.2 Maximum Matching

We start with an approximation algorithm for maximum matching.

Definitions and Notation

Let M be a *matching* in a graph $G = (V, E)$, that is, a subset of nonadjacent edges of G. A node v is *M-free* if v is not an endpoint of an edge in M. A path P is an *M-alternating* path if it consists of edges drawn alternately from M and from $E \setminus M$. A path P is an *M-augmenting* path if P is M-alternating and both endpoints of P are M-free nodes (i.e., $|P \cap M| = |P \cap (E \setminus M)| + 1$).

Properties of Matchings

Let $\oplus$ denote the symmetric difference of sets. If M is a matching and P is an M-augmenting path, then $M \oplus P$ is a matching such that $|M \oplus P| = |M| + 1$. Many matching algorithms search for augmenting paths until they construct a maximum matching, and one can show that in a non-maximum matching there is an augmenting path.

The correctness of our algorithm relies on the properties of matchings proven by Hopcroft and Karp [43]. The part of their contribution that is important to us is summarized below.

Fact 2.4.1 (Hopcroft and Karp [43]). *Let M be a matching with no augmenting paths of length smaller than t. Let $P^\star$ be a maximal set of vertex-disjoint M-augmenting paths of length t. Let A be the set of all edges in the paths in $P^\star$. There does not exist an $(M \oplus A)$-augmenting path of length smaller than or equal to t.*

We now prove an auxiliary lemma that connects the minimum length of an augmenting path and the quality of the matching.

Lemma 2.4.2. *Let M be a matching that has no augmenting paths of length smaller than $2t+1$. Let $M^\star$ be a maximum matching in the same graph. It holds $|M| \geq \frac{t}{t+1}|M^\star|$.*

Proof. Consider the set of edges $\Delta = M \oplus M^\star$. There are exactly $|M^\star| - |M|$ more edges from $M^\star$ then from M in Δ. Since M and $M^\star$ are matchings, each vertex is incident to at most two edges in Δ. Hence Δ can be decomposed into paths and cycles. Each path of even length and each cycle contain the same number of edges from M and $M^\star$. Each path P of odd length contains one more edge from $M^\star$ than from M. It if contained one more edge from M, it would be an $M^\star$-augmenting path; an impossibility. P is then an M-augmenting path. Summarizing, we have exactly $|M^\star| - |M|$ odd-length vertex-disjoint paths in Δ, and each of them is an M-augmenting path.

Since each M-augmenting path has length at least $2t-1$, this implies that $|M| \geq t(|M^\star| - |M|)$. Hence, $|M| \geq \frac{t}{t+1}|M^\star|$. □

The Algorithm

Consider the maximum matching problem in an unweighted graph of bounded degree d. It is well known that the size of any maximal matching is at least half of the maximum matching size. Because of that, we obtained a $(2, \varepsilon n)$-approximation algorithm for the maximum matching size in Corollary 2.3.2. We now show that our technique can be used to achieve better approximations in constant time.

A Sequential Algorithm. We simulate the following sequential algorithm. The algorithm starts with an empty matching M_0. In the i-th phase, it constructs a matching M_i from M_{i-1} as follows. Let $P^\star_{i-1}$ be a maximal set of vertex-disjoint M_{i-1}-augmenting paths of length $2i-1$. Let A_{i-1} be the set of all edges in the augmenting paths in $P^\star_{i-1}$. We set $M_i = M_{i-1} \oplus A_{i-1}$. If M_{i-1} is a matching, so is M_i. By induction, all M_i are matchings. The algorithm stops for some k, and returns M_k.

We now show that M_i has no augmenting path of length smaller than $2i+1$. M_1 is a maximal matching, so it has no augmenting path of length smaller than 3. Now, for the inductive step, assume that M_{i-1}, $i \geq 1$, has no augmenting path shorter than $2i-1$. $P^\star_{i-1}$ is a maximal set of vertex-disjoint M_{i-1}-augmenting paths of length $2i-1$. Therefore, it follows by Fact 2.4.1 that M_i does not have any augmenting path shorter than $2i+1$.

Set $k = \lceil 1/\delta \rceil$, and let $M^\star$ be a maximum matching. By Lemma 2.4.2, $\frac{k}{k+1}|M^\star| \leq |M_k| \leq |M^\star|$, which yields $|M^\star| \leq \frac{k+1}{k}|M_k| \leq (1+\delta)|M^\star|$. If we had an estimate α such that $2|M_k| \leq \alpha \leq 2|M_k| + \varepsilon n/2$, we could get a $(1+\delta, \varepsilon n)$-approximation to $|M^\star|$ by multiplying α by $\frac{k+1}{2k}$, which is at most 1.

The Constant-Time Algorithm. We construct a sequence of oracles $\mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_k$. A query to $\mathcal{O}_i$ is an edge $e \in E$. The oracle's reply indicates whether e is in M_i. To compute the required α, it suffices to estimate the fraction of vertices that are matched in $|M_k|$. In order to do so, one can sample $O(1/\varepsilon^2)$ vertices, and for each of them, check if any incident edge is in M_k or not. The correctness of the estimate with probability 5/6 follows from the Hoeffding bound.

The oracles $\mathcal{O}_i$ are constructed by using our technique for transforming algorithms into constant-time algorithms. $\mathcal{O}_i$ has access to $\mathcal{O}_{i-1}$, and simulates the i-th phase of the above algorithm. To compute $P^\star_{i-1}$ and also M_i, the oracle uses the local computation method from Section 2.3. We assume that each M_{i-1}-augmenting path P of length $2i-1$ is assigned a random number $r(P)$, which is uniformly and independently chosen from $[0,1]$. These random numbers give a random ordering of all the M_{i-1}-augmenting paths. $P^\star_{i-1}$ is the greedily constructed maximal set of vertex-disjoint M_{i-1}-augmenting paths P considered in order of their $r(P)$. To handle a query about an edge e, the oracle first finds out if $e \in M_{i-1}$, and then, checks if there is an M_{i-1}-augmenting path in $P^\star_{i-1}$ that contains e. If there is such a path, the answer of $\mathcal{O}_i$ to the query about e is the opposite of the answer of $\mathcal{O}_{i-1}$. Otherwise, it remains the same.

The oracle can easily learn all length-$(2i-1)$ M_{i-1}-augmenting paths that contain

e by querying G and O_{i-1}. To find out which augmenting paths are in $P^\star_{i-1}$, the oracle considers the following graph H_i. All the M_{i-1}-augmenting paths of length $2i-1$ are nodes of H_i. Two nodes P_1 and P_2 are connected in H_i if P_1 and P_2 share a vertex. To check if P is in $P^\star_{i-1}$, it suffices to check if any of the paths R corresponding to the vertices adjacent to P in H_i is in $P^\star_{i-1}$, for $r(R) < r(P)$. If none, $P \in P^\star_{i-1}$. Otherwise, P is not in $P^\star_{i-1}$. This procedure can be run recursively. This finishes the description of the algorithm.

Query Complexity. It remains to bound the number of queries of the entire algorithm to the graph. This is accomplished in the following lemma.

Lemma 2.4.3. *The number of queries of the algorithm is with probability* $5/6$ *of order* $\frac{2^{O(d^{9k})}}{\varepsilon^{2^{k+1}}}$*, where* $k = \lceil 1/\delta \rceil$*, and* $d \geq 2$ *is a bound on the maximum degree of the input graph.*

Proof. Our main algorithm queries $\mathcal{O}_k$ about edges adjacent to C'/ε^2 random vertices, where C' is a constant. Let $Q_{k+1} = C' \cdot d/\varepsilon^2$ be the number of the direct queries of the main algorithm to G. These queries are necessary to learn the edges that $\mathcal{O}_k$ is queried with. Let Q_{i+1} be an upper bound on the number of queries of the algorithm to $\mathcal{O}_i$. We now show an upper bound Q_i on the number of queries to G performed by $\mathcal{O}_i$. The upper bound holds with probability at least $1 - \frac{1}{6k}$. Q_i also bounds the number of queries to O_{i-1}, since $\mathcal{O}_i$ does not query any edge it has not learnt about from G. For each received query about an edge e, $\mathcal{O}_i$ first learns all edges within the distance of $2i-1$ from e, and checks which of them are in M_{i-1}. For a single e, this can be done with at most $d \cdot 2\sum_{j=0}^{2i-2}(d-1)^j \leq 2d^{2i}$ queries to both G and $\mathcal{O}_{i-1}$, and suffices to find all length-$(2i-1)$ M_{i-1}-augmenting paths that contain e.

There are at most id^{2i-1} length-$(2i-1)$ paths in G that contain a fixed vertex v. Each such path can be broken into two paths that start at v. The length of the shorter is between 0 and $i-1$, and there are at most d^t paths of length t that start at t.

The number of length-$(2i-1)$ M_{i-1}-augmenting paths that contain e is therefore at most id^{2i-1}. Moreover, the maximum degree of H_i can be bounded by the number

of length-$(2i-1)$ paths that intersect a given length-$(2i-1)$ augmenting path. Hence, the degree of H_i is at most $2i \cdot id^{2i-1} = 2i^2d^{2i-1}$. Finally, to find augmenting paths adjacent in H_i to a given augmenting path P, it suffices to learn whether e' is in M_{i-1}, for each edge e' within the radius of $2i$ from any of the vertices of P. This can be accomplished with at most $2i \cdot d\sum_{j=0}^{2i-1} d^j \leq 2id^{2i+1}$ queries to both G and $\mathcal{O}_{i-1}$.

In total, to answer queries about all, at most Q_{i+1} edges e, $\mathcal{O}_i$ must check membership in $P^\star_{i-1}$ for at most $Q_{i+1} \cdot 2id^{2i-1}$ augmenting paths. By the Locality Lemma (Lemma 2.3.3), the number of augmenting paths for which we recursively check membership in $P^\star_{i-1}$ is with probability $1 - \frac{1}{6k}$ at most

$$(Q_{i+1} \cdot id^{2i-1})^2 \cdot C'^{\left(2i^2d^{2i-1}\right)^4} \cdot 6k \leq 2^{O\left(d^{8i}\right)} \cdot kQ_{i+1}^2.$$

For each of them we compute all adjacent paths in H_i. Therefore, with probability $1 - \frac{1}{6k}$, the total number of $\mathcal{O}_i$'s queries to both $\mathcal{O}_{i-1}$ and G is bounded by

$$Q_{i+1} \cdot 2d^{2i} + 2^{O\left(d^{8i}\right)} \cdot kQ_{i+1}^2 \cdot 2id^{2i+1}$$

$$\leq 2^{O\left(d^{8i}\right)} \cdot kQ_{i+1}^2 =: Q_i.$$

The total number of queries to G in the entire algorithm can be bounded by

$$\sum_{j=1}^{k+1} Q_j \leq 2Q_1 \leq \left(\frac{C' \cdot d \cdot k}{\varepsilon^2}\right)^{2^k} \cdot 2^{O\left(d^{8k}\right) \cdot 2^k}$$

$$\leq \frac{2^{O\left(d^{9k}\right)}}{\varepsilon^{2^{k+1}}}.$$

□

We summarize the whole algorithm in the following theorem.

Theorem 2.4.4. *There is a $(1 + \delta, \varepsilon n)$-approximation algorithm for the maximum matching size that uses $\frac{2^{O(d^{9k})}}{\varepsilon^{2^{k+1}}}$ queries, where $d \geq 2$ is a bound on the maximum degree, and $k = \lceil 1/\delta \rceil$.*

Proof. We run the algorithm described above. If the algorithm exceeds the num-

ber of queries guaranteed in Lemma 2.4.3, we terminate it, and return an arbitrary result. The algorithm returns a $(1+\delta, \varepsilon n)$-approximation with probability at least $2/3$, because the sampling can return a wrong estimate with probability at most $1/6$, and the algorithm can exceed the allowed number of queries with probability at most $1/6$. $\square$

Finally, we can easily remove the multiplicative factor.

Corollary 2.4.5. *There is a $(1, \varepsilon n)$-approximation algorithm of query complexity $2^{d^{O(1/\varepsilon)}}$ for the maximum matching size, where $d \geq 2$ is a bound on the maximum degree.*

Proof. Using the algorithm of Theorem 2.4.4, we can get a $(1+\varepsilon, \varepsilon n/2)$-approximation for the maximum matching size, using $2^{d^{O(1/\varepsilon)}}$ queries. Since the size of the maximum matching is at most $n/2$, this approximation is also a $(1, \varepsilon n)$-approximation for the maximum matching size. $\square$

2.4.3 Maximum Weight Matching

We now show a generalization of the previous result. If the input is a weighted graph with the weight of every edge in $[0, 1]$ and maximum degree at most $d = O(1)$, we can approximate the maximum matching weight in the graph in constant time.

Definitions and Notation

Let $G = (V, E, w)$ be a weighted graph with maximum degree bounded by d, where $w : E \to [0, 1]$ is a weight function for edges of G. We use the following definitions and notation:

- For any collection $S \subset E$ of edges, we write $w(S)$ to denote the total weight of edges in S. That is, $w(S) = \sum_{e \in S} w(e)$.
- We write $\mathrm{MWM}(G)$ to denote the maximum weight of a matching in G.
- Let $S \subset E$ be a subset of edges. The *gain $g_M(S)$ of S on a matching M* equals $w(S \cap (E \setminus M)) - w(S \cap M)$.

- Let $P = \{(v_0, v_1), (v_1, v_2), (v_2, v_3), \ldots, (v_{t-1}, v_t)\}$ be a path in G. We say that P is an *M-augmenting path* for a matching M if:
  - P is M-alternating,
  - $M \oplus P$ is a matching,
  - no vertex in P appears twice except that v_0 may equal v_t (in which case the path is a simple cycle),
  - $g_M(P) > 0$.

  We say that P is a *(k, M)-augmenting path* if it is an M-augmenting path and has at most k edges.

Let S be a collection of edges in G, $w(S)$ is the total weight of all edges in S, $g_M(S)$ is the *gain* of S on matching M, $g_M(S) = w(S \cap (E \setminus M)) - w(S \cap M)$. P is an M-augmenting path if P is M-alternating and $g_M(P) > 0$. P is $k^\star$-M-augmenting path if P has at most k edges.

The Existence of a Constant Factor Improvement

The correctness of our algorithm relies on the following lemma proven by Pettie and Sanders [72].

Lemma 2.4.6 (Pettie and Sanders [72]). *For any matching M and any constant k, there exists a collection of vertex-disjoint (k, M)-augmenting paths such that* $g_M(A) = w(M \oplus A) \geq w(M) + \frac{k+1}{2k+1}\left(\frac{k}{k+1}\text{MWM}(G) - w(M)\right)$.

Edge Rounding

Let ε and δ be two real numbers in $(0, 1)$ to be fixed later. We set the weight of all edges of weight less than $\varepsilon/2$ to 0, and round the weight of each other edge down to the closest integer power of $(1 - \delta/3)$. Let $G' = (V, E, w')$ be the resulting graph. It holds

$$\text{MWM}(G') \geq (1 - \delta/3)\text{MWM}(G) - \varepsilon n/2.$$

For simplicity, we write w and g from now on to denote the weight and gain functions for G', respectively. The weight function w can easily be turned into w' on the fly. Whenever we read a weight $w(e)$ for an edge e, we substitute it with the rounded weight $w'(e)$.

Now, clearly, there are at most $W = \lceil \log(\varepsilon/2)/\log(1-\delta/3) \rceil + 2$ distinct edge weights in G. It follows that for any k, there are at most W^k possible values for the gain of any (M,k)-augmenting path. Let $g_1, g_2, \ldots, g_{T_k}$ (where T_k is an integer bounded by W^k) be the sequence of these values in decreasing order.

A Sequential Greedy Algorithm

Our goal is to show that our framework can be used to obtain a $(1+\delta, \varepsilon n)$-approximation algorithm for MWM(G) in constant time. In order to do so, we will locally simulate Algorithm 4. The algorithm is a modified version of the deterministic algorithm for finding a heavy matching due to Pettie and Sanders [72].

Algorithm 4: Sequential algorithm

1. $M := \emptyset$
2. $k := \lceil 3/\delta \rceil$
3. $L := \log(\delta/3)/\log\left(1 - \frac{1}{2k+1}\right)$
4. **for** $i := 1$ *to* L **do**
5.   **for** $j := 1$ *to* T_k **do**
6. $P^\star_{i,j} :=$ { a maximal set of (k, M)-augmenting paths with gain g_j}
7. $M := M \oplus S$
8. **return** M

We show that the algorithm has the following property.

Theorem 2.4.7. *Algorithm 4 outputs a matching of weight W such that*

$$(1-\delta)\cdot \text{MWM}(G) - \varepsilon n/2 \le W \le \text{MWM}(G).$$

Proof. Let M_i be the matching at the end of the i-th iteration of the outer loop (Lines 4–7), and let $P^\star_i = \bigcup_{j=1}^{T} P^\star_{i,j}$ be the set of all (k, M_{i-1})-augmenting paths chosen in

that iteration (these paths need not be vertex disjoint). We prove the following inequality, which holds for all iterations:

$$w(M_i) \geq w(M_{i-1}) + \frac{1}{2k+1}\left(\frac{k}{k+1}\text{MWM}(G') - w(M_{i-1})\right).$$

By Lemma 2.4.6, there exists a collection of vertex-disjoint (k, M_{i-1})-augmenting paths A_i such that

$$g_{M_{i-1}}(A_i) \geq \frac{k+1}{2k+1}\left(\frac{k}{k+1}\text{MWM}(G') - w(M_{i-1})\right).$$

For each augmenting path P in A_i, P must intersect with at least one augmenting path P' in $P_i^\star$ such that the gain of P' when *it is chosen* is at least the gain $g_{M_{i-1}}(P)$ of P on M_{i-1}. In addition, since each (k, M_{i-1})-augmenting path in $P_i^\star$ intersects with at most $k+1$ paths in A_i, the total gain of all augmenting paths in $P_i^\star$ is at least $\frac{1}{k+1}g_{M_{i-1}}(A_i)$. Thus,

$$w(M_i) - w(M_{i-1}) \geq \frac{1}{k+1}g_{M_{i-1}}(A_i) \geq \frac{1}{2k+1}\left(\frac{k}{k+1}\text{MWM}(G') - w(M_{i-1})\right).$$

By induction, for any $i \in \{1, \ldots, L\}$,

$$\frac{k}{k+1}\text{MWM}(G') - w(M_i) < \left(1 - \frac{1}{2k+1}\right)^i \cdot \text{MWM}(G').$$

Therefore,

$$\begin{aligned}
w(M_L) &\geq \left(1 - \left(1 - \frac{1}{2k+1}\right)^L\right) \cdot \frac{k}{k+1} \cdot \text{MWM}(G') \\
&\geq (1-\delta/3) \cdot (1-\delta/3) \cdot \text{MWM}(G') \\
&\geq (1-\delta/3)^3 \cdot \text{MWM}(G) - \varepsilon n/2 \\
&\geq (1-\delta) \cdot \text{MWM}(G) - \varepsilon n/2.
\end{aligned}$$

□

The Constant-Time Algorithm.

We approximate the weight W of the output of Algorithm 4 for $\delta = \varepsilon/2$. Then

$$\mathrm{MWM}(G) - \frac{3}{4}\varepsilon n \leq W \leq \mathrm{MWM}(G).$$

Therefore, it suffices to approximate W up to an additive $\varepsilon n/8$, and add $\frac{7}{8}\varepsilon n$ to the approximation, we obtain a $(1, \varepsilon n)$-approximation for $\mathrm{MWM}(G)$.

As before, we construct an oracle that provides query access to a matching that could be an output of Algorithm 4. Once we have such an oracle, it suffices to sample $O(1/\varepsilon^2)$ vertices v and sum the weights of the edges in the matching incident to these vertices. By the Hoeffding bound, this gives an additive $\varepsilon n/4$ approximation to $2W$ with probability 99/100. (The factor of 2 comes from the fact that we count the weight of each edge in the matching twice, once for each endpoint. Clearly, it suffices to divide this estimate by 2 to obtain the desired approximation to W.)

We now describe the construction of the oracle. We consider each inner iteration in the sequential algorithm as a phase. Let $M_{i,j}$ be the matching M at the end of the (i,j)-th phase (j^{th} iteration of inner loop in the i-th iteration of the outer loop). We construct a sequence of oracles $\mathcal{O}_{1,1}$, $\mathcal{O}_{1,2}$, ..., $\mathcal{O}_{1,T_k}$, $\mathcal{O}_{2,1}$, $\mathcal{O}_{2,2}$, ..., $\mathcal{O}_{2,T_k}$, ..., $\mathcal{O}_{L,1}$, $\mathcal{O}_{L,2}$, ..., $\mathcal{O}_{L,T_k}$. Each oracle provides query access to a single phase. A query to $\mathcal{O}_{i,j}$ is an edge $e \in E$. The oracle's reply indicates whether e is in $M_{i,j}$. It is easy to observe that each oracle $\mathcal{O}_{i,j}$ essentially works in the same way as each oracle in the constant-time algorithm in Section 2.4.2. They both compute a maximal set of vertex-disjoint alternating paths of constant-length that have some additional properties and apply them to the matching returned by the previous oracle.

How does the oracle $\mathcal{O}_{i,j}$ work? Let $\mathcal{O}^\star$ be the previous oracle, that is, let $\mathcal{O}^\star = \mathcal{O}_{i,j-1}$ if $j > 1$, and let $\mathcal{O}^\star = \mathcal{O}_{i-1,T_k}$, otherwise. $\mathcal{O}^\star$ provides query access to the matching that is the result of the previous phase. The oracle $\mathcal{O}_{i,j}$ works as follows. To answer a query about e, the oracle first makes queries to $\mathcal{O}^\star$ and G' to learn the graph structure and the matching provided by $\mathcal{O}^\star$ in a constant-radius neighborhood of e. Then the local computation method from Section 2.3 is applied in order to find

out recursively whether e belongs to an augmenting path selected in this phase. If yes, e belongs to the matching provided by $\mathcal{O}_{i,j}$ if and only if it does not belong to a matching provided by $\mathcal{O}^{\star}$. Otherwise, the reply of $\mathcal{O}_{i,j}$ for e is the same as the reply of $\mathcal{O}^{\star}$.

Clearly, $\mathcal{O}_{L,T_k}$ provides query access to a potential output of Algorithm 4.

Query Complexity

The following lemma gives a bound on the number of queries made by the constant-time algorithm.

Lemma 2.4.8. *The number of queries of the algorithm is with probability* $5/6$ *of order* $\frac{W^{2k^2} 2^{O\left(d^{6k} 2^{W^k}\right)}}{\varepsilon^{2k+1}}$, *where* $k = \lceil 3/\delta \rceil$, $W = \lceil \log(\varepsilon)/\log(1+\delta/3) \rceil + 2$, *and* $d \geq 2$ *is an upper bound on the maximum degree of the input graph.*

Proof. Our main algorithm queries $\mathcal{O}_{L,T_k}$ about the edges adjacent to C'/ε^2 random vertices, where C' is constant. Let $Q_{L+1,1} = C'd/\varepsilon^2$ be the number of direct queries of the main algorithm to both G' and $\mathcal{O}_{L,W^k}$. We now show a sequence of upper bounds on the number of queries $Q_{i,j}$ generated for every i and j by the oracle $\mathcal{O}_{i,j}$ to both the previous oracle and directly to the input graph. The probability that any of the upper bounds fails is bounded by $1/6$.

For each received query about an edge e, $\mathcal{O}_{i,j}$ first learns all edges within the distance $k-1$ from e in G' and $M_{i,j-1}$ by making at most $d \cdot 2\sum_{i=0}^{k-1}(d-1)^i \leq 2d^{k+1}$ queries to both G' and the oracle for the previous phase of the algorithm. Let M' be the matching computed in the previous phase. For any edge e, the number of (k, M')-augmenting paths that contain e is at most $2d^k$. For every fixed path length, every path of this length can be described as a sequence of edges starting at one of the endpoints, a special symbol marking the end of the first part of the path, and then a sequence of edges starting at the other endpoint. There are always at most $d-1$ edges one can follow in the next step, and together with the special symbol marking the end of the first path, we never have more than d choices. Therefore, for paths of a given length t, the number of augmenting paths going through e is at most

d^t, and summing over all lengths, we obtain the bound $\sum_{t=1}^{k} d^t \le 2d^k$.

Let $H_{i,j}$ be the intersection graph of augmenting paths in the (i,j)-th phase (i.e., all gain-g_j (k, M')-augmenting paths are nodes in $H_{i,j}$ and two nodes P_1 and P_2 are connected if and only if the corresponding paths share at least a vertex). The degree of $H_{i,j}$ is at most $(k+1)\sum_{i=1}^{k}(i+1)d^i \le (k+1)^2 d^{k+1}$. Finding all augmenting paths adjacent to a given augmenting path in $H_{i,j}$ can be accomplished with at most $(k+1)\sum_{i=0}^{k} d^i \le (k+1)d^{k+1}$ queries to G' and $\mathcal{O}_{i,j-1}$.

Let $\tilde{Q}$ equal $Q_{i,j+1}$ if $i < T_k$, and $Q_{i+1,1}$ otherwise. To answer queries about all, at most $\tilde{Q}$ edges e, $\mathcal{O}_{i,j}$ must check membership in $P^{\star}_{i,j}$ for at most $Q_{i,j+1} \cdot 2d^k$ augmenting paths. By the Locality Lemma (Lemma 2.3.3), the number of augmenting path for which we have to check membership in $P^{\star}_{i,j}$ is with probability $1 - \frac{1}{6LW^k}$ at most

$$(Q_{i,j+1} \cdot 2d^k)^2 \cdot C^{((k+1)^2 d^{k+1})^4} \cdot 6LW^k \le 2^{O(d^{5k})} W^k Q_{i,j+1}^2.$$

For each of them we compute all adjacent paths in $H_{i,j}$. Therefore, with probability $1 - \frac{1}{6LW^k}$, the total number of $\mathcal{O}_{i,j}$'s queries to both $\mathcal{O}_{i,j-1}$ and G' is bounded by

$$Q_{i,j+1} \cdot kd^k + 2^{O(d^{5k})} W^k Q_{i,j+1}^2 \cdot (k+1)d^{k+1}$$
$$\le 2^{O(d^{5k})} W^k Q_{i,j+1}^2 =: Q_{i,j}$$

The total number of queries to G in the entire algorithm with probability $5/6$ can be bounded by

$$\sum_{i=1}^{L} \sum_{j=1}^{W^k} Q_{i,j} + Q_{L+1,1} \le 2Q_{1,1} \le \left(\frac{C' \cdot d \cdot W^k}{\varepsilon^2}\right)^{2^{LW^k}} \cdot 2^{O(d^{5k}) \cdot 2^{LW^k}} \le \frac{W^{2k^2} 2^{O(d^{6k} 2^{W^k})}}{\varepsilon^{2k+1}}.$$

□

We summarize the algorithm in the following corollary, which is true because the running time and the query complexity are polynomially related.

Corollary 2.4.9. *There is a constant-time $(1+\delta, \varepsilon n)$-approximation algorithm for the weight of the heaviest matching in graphs with maximum degree d and all edge*

weights in $[0,1]$. *The algorithm runs in* $\frac{W^{2k^2} 2^{O(d^{6k} 2^{W^k})}}{\varepsilon^{2k+1}}$ *time, where* $k = \lceil \frac{3}{\delta} \rceil$, $W = \lceil \log(\varepsilon)/\log(1+\delta/3) \rceil + 1$, *and* $d \geq 2$ *is a bound on the maximum degree.*

This directly implies the following theorem.

Theorem 2.4.10. *There is a constant-time* $(1, \varepsilon n)$*-approximation algorithm for the weight of the heaviest matching in graphs with maximum degree* d *and all edge weights in* $[0,1]$. *The algorithm runs in* $2^{d^{O(1/\varepsilon)} \cdot 2^{(1/\varepsilon)^{O(1/\varepsilon)}}}$ *time.*

2.4.4 Set Cover

In the minimum set cover problem, an input consists of subsets S_1 to S_n of $U = \{1, \ldots, m\}$. Each element of U belongs to at least one of the sets S_i. The goal is to cover U with the minimum number of sets S_i, that is, to find a minimum size set I of indices such that $\bigcup_{i \in I} S_i = U$. Here, we want to approximate the optimal size of I.

We assume that for each set S_i, we have query access to a list of elements of S_i, and that for each element $u \in U$, we have query access to a list of indices of sets S_i that contain u.

The Classical Greedy Algorithm

Theorem 2.4.11. *Let* $H(i)$ *be the* i*-th harmonic number. There is an* $(H(s), \varepsilon n)$*-approximation algorithm with query complexity and running time* $\left(\frac{2^{(st)^4}}{\varepsilon}\right)^{O(2^s)}$ *for the minimum set cover size for instances with all* n *sets* S_i *of size at most* s, *and each element in at most* t *different sets.*

Proof. We simulate the classical greedy algorithm [48, 60] for the set cover problem. The algorithm starts from an empty cover, and keeps adding the set S_i which covers most elements that have not yet been covered, until the whole set U is covered. The approximation factor of the algorithm is at most $H(s) \leq 1 + \ln s$.

As in the local computation method from Section 2.3, we first consider all sets in random order and add to the cover those that cover s new elements at the time they are considered. Let $\mathcal{C}_1$ be the set of sets that were already included into the cover.

We then consider the remaining sets, also in random order, and we add to the cover those that cover $s-1$ new elements. This way we get $\mathcal{C}_2$, the set of all sets already included in the cover. We keep doing this until we cover the whole set U, and $\mathcal{C}_s$ is the final cover.

We create a sequence of oracles $\mathcal{O}_1$, $\mathcal{O}_2$, ..., $\mathcal{O}_s$ that correspond to the process described above. A query to an oracle $\mathcal{O}_j$ is an index i of a set S_i. The oracle's reply indicates whether S_i is in $\mathcal{C}_j$.

How is $\mathcal{O}_j$ implemented? We use the local computation method from Section 2.3. We assume that each set S_i is assigned a random number r_{ji}, which is uniformly and independently chosen from $[0, 1]$. These random numbers give a random ordering of sets S_i. To handle a query about a set S_k, we first ask $\mathcal{O}_{j-1}$ if S_k was already included in $\mathcal{C}_{j-1}$. (For $j = 1$, we assume that $\mathcal{C}_{j-1} = \mathcal{C}_0 = \emptyset$, so no query is necessary in this case.) If S_k was already in $\mathcal{C}_{j-1}$, then it is also in $\mathcal{C}_j$. Otherwise, we learn first the elements of S_k (at most s queries) and what sets S_i they belong to (at most st further queries). Then, we check for each of these S_i if it was already in $\mathcal{C}_{j-1}$ (at most st queries to $\mathcal{O}_{j-1}$), and for all of the S_i's that have $r_{ji} < r_{jk}$, we recursively check if they are in $\mathcal{C}_j$. Finally, using this knowledge, we can verify what number of elements of S_k is not yet covered, when S_k is considered. If the number of these elements is $s - j + 1$, then S_k is in $\mathcal{C}_j$. Otherwise, the number of the elements is lower than $s - j + 1$, and S_k is not in $\mathcal{C}_j$.

It is obvious that the above procedure simulates the classical greedy algorithm. Our sublinear approximation algorithm queries $\mathcal{O}_s$ for C'/ε^2 sets chosen at random, where C' is a sufficiently large constant, to estimate the fraction of sets which are in the cover to within $\varepsilon n/2$ with probability 5/6. By adding $\varepsilon n/2$ to the estimate, we get the desired approximation. We want to bound the number of queries. We set Q_s to $(C'/\varepsilon^2)^2 \cdot 6s \cdot C^{(st)^4}$, and define Q_j, for $1 \le j \le s-1$, to be $Q_j \le (Q_{j+1} \cdot (st+1))^2 \cdot 6s \cdot C^{(st)^4}$. By Lemma 2.3.3, each Q_i bounds the number of sets for which we check if they are in $\mathcal{C}_i$ with probability $1 - s \cdot \frac{1}{6s} = 1 - \frac{1}{6} = \frac{5}{6}$. It can be shown by induction

that

$$Q_{s-i} = \left(\frac{6s \cdot (st+1) \cdot C^{(st)^4}}{\varepsilon}\right)^{O(2^i)} = \left(\frac{2^{(st)^4}}{\varepsilon}\right)^{O(2^i)}$$

with probability at least 5/6. So with probability 5/6, the total number of queries is at most

$$(s + st) \cdot \sum_{i=1}^{s} Q_s = \left(\frac{2^{(st)^4}}{\varepsilon}\right)^{O(2^s)}.$$

Summarizing, with probability 2/3, the algorithm uses the above number of queries, and returns the desired approximation. □

Application to Dominating Set

In the dominating set problem, one is given a graph, and chooses a minimum-cardinality subset S of its vertices such that each vertex v of the graph is either in S or is adjacent to a vertex in S.

Theorem 2.4.12. *There is an $(H(d+1), \varepsilon n)$-approximation algorithm with query complexity and running time $\left(\frac{2^{d^8}}{\varepsilon}\right)^{O(2^d)}$, for the minimum dominating set size for graphs with the maximum degree bounded by d.*

Proof. The problem can be trivially expressed as an instance of the set cover problem. For each vertex v, we have a set S_v of size at most $d+1$ that consists of v and all neighbors of v. We want to cover the set of vertices of the graph with the minimum number of sets S_v. To approximate the minimum set cover size, we use the algorithm of Theorem 2.4.11. □

2.5 Better Algorithms for Hyperfinite Graphs

In this section, we construct better approximation algorithms for graphs that can be partitioned into components of constant-size by removing an arbitrarily small fraction of edges. We call families of such graphs *hyperfinite*. For completeness, we restate the definitions from the introduction:

- Let $G = (V, E)$ be a graph. G is (ε, k)-*hyperfinite* if it is possible to remove $\varepsilon|V|$ edges of the graph such that the remaining graph has connected components of size at most k.

- Let ρ be a function from $\mathbb{R}_+$ to $\mathbb{R}_+$. A graph G is ρ-*hyperfinite* if for every $\varepsilon > 0$, G is $(\varepsilon, \rho(\varepsilon))$-hyperfinite.

- Let $\mathcal{C}$ be a family of graphs. $\mathcal{C}$ is ρ-*hyperfinite* if every graph in $\mathcal{C}$ is ρ-hyperfinite.

Recall that examples of hyperfinite families of graphs include bounded-degree graphs with an excluded minor, and graphs of subexponential growth (see Section 1.1.2 for more details).

We now define the main tool used in this part of the thesis. A partitioning oracle provides query access to a global partition of the graph into small components.

Definition 2.5.1. *We say that $\mathcal{O}$ is an (ε, k)-*partitioning oracle *for a family $\mathcal{C}$ of graphs if given query access to a graph $G = (V, E)$ in the adjacency-list model, it provides query access to a partition P of V. For a query about $v \in V$, $\mathcal{O}$ returns $P[v]$. The partition has the following properties:*

- *P is a function of the graph and random bits of the oracle. In particular, it does not depend on the order of queries to $\mathcal{O}$.*

- *For every $v \in V$, $|P[v]| \le k$ and $P[v]$ induces a connected graph in G.*

- *If G belongs to $\mathcal{C}$, then $|\{(v, w) \in E : P[v] \neq P[w]\}| \le \varepsilon|V|$ with probability $9/10$.*

The most important property of our oracles is that they compute answers in time independent of the graph size by using only local computation. We give two methods for constructing partitioning oracles for different classes of graphs. We briefly describe them below.

A Generic Partitioning Oracle for Hyperfinite Graphs (Section 2.5.1).

We give a generic oracle that works for *any* hyperfinite family of graphs.

Theorem 2.5.2. *Let G be an $(\varepsilon, \rho(\varepsilon))$-hyperfinite graph with degree bounded by $d \geq 2$. Suppose that the value $\rho(\varepsilon^3/3456000)$ is known, that is, it can either be efficiently computed, or is hard-wired for a given $\varepsilon > 0$ of interest. There is an $(\varepsilon d, \rho(\varepsilon^3/3456000))$-partitioning oracle with the following properties. The oracle answers every query, using $2^{d^{O(\rho(\varepsilon^3/3456000))}}/\varepsilon$ queries to the graph. If q is the total number of queries to the oracle, the total amount of the oracle's computation is bounded by $q \log q \cdot 2^{d^{O(\rho(\varepsilon^3/3456000))}}/\varepsilon$.*

The oracle's construction is based on locally simulating a greedy global partitioning procedure. The procedure repeatedly selects a random vertex v and tries to remove a small component containing it by cutting few edges. If there is no such component, only v is removed. This procedure can easily be simulated locally using a local computation paradigm of Nguyen and Onak [66].

An Efficient Partitioning Oracle for Minor-Free Graphs (Section 2.5.2).

It follows from the Separator Theorem (in particular from Proposition 4.1 in [4]) that every minor-free family of graphs with maximum degree bounded by d is $(\varepsilon, O(d^2/\varepsilon^2))$-hyperfinite. The direct application of Theorem 2.5.2 gives an oracle that makes $2^{d^{\mathrm{poly}(1/\varepsilon)}}$ queries to the graph, for every query to the oracle. We give a more efficient oracle that uses only $d^{\mathrm{poly}(1/\varepsilon)}$ queries.

Theorem 2.5.3. *Let H be a fixed minor. For every H-minor-free graph G with degree bounded by $d \geq 2$, there is an $(\varepsilon, C \cdot d^2/\varepsilon^2)$-partitioning oracle, where C is a constant that only depends on H. Let q be the number of non-adaptive queries to the oracle. The oracle makes $d^{\mathrm{poly}(1/\varepsilon)}$ queries to the input graph per query, and the total amount of the oracle's computation is $q \log q \cdot d^{\mathrm{poly}(1/\varepsilon)}$.*

The construction of the oracle is based on a deterministic clustering method of Czygrinow, Hańćkowiak, and Wawrzyniak [33, Section 2].

We also note that for graphs with polynomial growth, Jung and Shah [49] provide methods that can be used to construct a partitioning oracle that makes $\mathrm{poly}(1/\varepsilon)$ queries to the graph for each query to the oracle.

Constant-Time Approximation Algorithms

We first describe an application of partitioning oracles to approximating the size of an optimal solution for combinatorial problems on hypefinite families of graphs. As an example, consider the minimum vertex cover problem. We use a partitioning oracle to obtain access to a partition of the input graph. The union of optimal vertex covers over all connected components constitutes a set of size within an additive $O(\varepsilon n)$ of the minimum vertex cover size of the original graph. By sampling $O(1/\varepsilon^2)$ vertices and computing the fraction of those that belong to the optimal vertex cover in their component, we obtain a $(1, O(\varepsilon n))$-approximation to the minimum vertex cover size of the original graph.

A formal theorem and proof follow. To achieve a good approximation, a bound on the average degree is needed. Note that every family of graphs with an excluded minor has average degree bounded by a constant.

Theorem 2.5.4. *Let $\mathcal{C}$ be a family of graphs with average degree bounded by $\tilde{d}$. Let $\varepsilon > 0$. Let $\mathcal{O}$ be an $(\varepsilon/3, k)$-partitioning oracle for the family $\mathcal{C}|_{3\tilde{d}/\varepsilon}$. There is a $(1, \varepsilon n)$-approximation algorithm for the minimum vertex cover size in any graph $G = (V, E)$ in $\mathcal{C}$. The algorithm*

- *gives $\mathcal{O}$ query access to the graph $G|_{3\tilde{d}/\varepsilon}$,*
- *makes $O(1/\varepsilon^2)$ uniformly distributed queries to $\mathcal{O}$,*
- *uses $2^{O(k)}/\varepsilon^2 + O(\tilde{d}k/\varepsilon^3)$ time for computation.*

The same holds for the maximum independent set problem, and the minimum dominating set problem.

Proof. All edges from G missing in $G|_{3\tilde{d}/\varepsilon}$ can be covered by vertices of degree greater than $3\tilde{d}/\varepsilon$ in G. We write $G' = (V, E')$ to denote $G|_{3\tilde{d}/\varepsilon}$. Note that the number of such vertices is by Markov's inequality at most $\varepsilon n/3$. Therefore, we have

$$\mathrm{VC}(G) - \varepsilon n/3 \leq \mathrm{VC}(G') \leq \mathrm{VC}(G).$$

The adjacency list of every vertex v in G' can easily be computed in $O(3\tilde{d}/\varepsilon)$ time. If the degree of v is greater than $3\tilde{d}/\varepsilon$ in G, then v is an isolated vertex in G'. Otherwise, we go over the neighbors of v in G, and each neighbor w in G remains a neighbor in G' if and only if w has degree at most $3\tilde{d}/\varepsilon$ in G. We give $\mathcal{O}$ query access to G'. With probability $9/10$, $\mathcal{O}$ provides query access to a partition P such that the number of edges $(v, w) \in E'$ with $P[v] \neq P[w]$ is at most $\varepsilon n/3$. Let $G'' = (V, E'')$ be G' with those edges removed. Since they can be covered with $\varepsilon n/3$ vertices, we have

$$\mathrm{VC}(G') - \varepsilon n/3 \leq \mathrm{VC}(G'') \leq \mathrm{VC}(G'),$$

that is,

$$\mathrm{VC}(G) - 2\varepsilon n/3 \leq \mathrm{VC}(G'') \leq \mathrm{VC}(G).$$

To get a $(1, \varepsilon n)$-approximation to $\mathrm{VC}(G)$, it suffices to estimate $\mathrm{VC}(G'')$ up to $\pm\varepsilon n/6$. By the Chernoff bound, we achieve this with probability $9/10$ by sampling $O(1/\varepsilon^2)$ vertices and computing the fraction of them in a fixed minimum vertex cover of G''. Such a vertex cover can be obtained by computing a minimum vertex cover for each connected component of G'' independently. Therefore, for every vertex v in the sample, we obtain $P[v]$ from $\mathcal{O}$. We compute a minimum vertex cover for the component induced by $P[v]$ in such a way that the vertex cover does not depend on which vertex in $P[v]$ was the query point. Finally, we check if the query point v belongs to the computed vertex cover for the component. In total, our procedure takes at most $O\left(k \cdot \tilde{d}/\varepsilon^3\right) + 2^{O(k)}/\varepsilon^2$ time.

To prove the same statement for minimum dominating set, we assume that all the high degree nodes are in the dominating set, and we take this into account when we compute optimal solutions for each connected component in the partition. This can increase the solution size by $\varepsilon n/3$ at most. For maximum independent set, it suffices to recall that the sum of the size of the maximum independent set and the size of the minimum vertex cover equals n, so a $(1, \varepsilon n)$-approximation to one of the problems immediately implies a $(1, \varepsilon n)$-approximation to the other one. $\square$

We now use the fact that the average degree of a graph with an excluded minor is $O(1)$. We combine Theorem 2.5.3 and Theorem 2.5.4, and achieve the following corollary.

Corollary 2.5.5. *For every H-minor free family of graphs (with no restriction on the maximum degree), there are $(1, \varepsilon n)$-approximation algorithms for the optimal solution size for minimum vertex cover, maximum independent set, and minimum dominating set that run in $2^{\text{poly}(1/\varepsilon)}$ time.*

2.5.1 A Generic Partitioning Oracle

Local Computation

We start by presenting a partitioning oracle that works for any family of hyperfinite graphs. We later show more efficient oracles for specific families of hyperfinite graphs.

We reuse a general method for local computation that was introduced by Nguyen and Onak [66]. Consider a graph with random numbers in $[0, 1]$ independently assigned to each of its vertices. Suppose that to compute a specific function f of a vertex v, you first need to compute recursively the same function for neighbors of v that were assigned a smaller number than that of v. The following lemma gives a bound on the expected number of vertices for which f must be computed.

Lemma 2.5.6 ([66], proof of Lemma 12). *Let $G = (V, E)$ be a graph of degree bounded by $D \geq 2$, and let $g : V \times (V \times A)^\star \to A$ be a function. A random number $r(v) \in [0, 1]$ is independently and uniformly assigned to each vertex v of G. A function $f_r : V \to A$ is defined recursively, using g. For each vertex v, we have*

$$f_r(v) = g(v, \{(w, f_r(w)) : r(w) < r(v)\}).$$

Let $S \subseteq V$ be a set of vertices v selected independently of r, for which we want to learn $f_r(v)$. The expected number of vertices w for which we have to recursively compute $f_r(w)$ in order to compute $f_r(v)$ for $v \in S$ is at most $|S| \cdot 2^{O(D)}$.

Algorithm 5: The global partitioning algorithm with parameters k and δ

```
1  (π_1, ..., π_n) := random permutation of vertices
2  P := ∅
3  for i = 1, ..., n do
4      if π_i still in the graph then
5          if there exists a (k, δ)-isolated neighborhood of π_i in the remaining graph
           then
6              S := this neighborhood
7          else
8              S := {π_i}
9          P := P ∪ {S}
10         remove vertices in S from the graph
```

The Oracle

We introduce an auxiliary definition of a small subset S of vertices that contains a specific node, and has a small number of outgoing edges relatively to S.

Definition 2.5.7. Let $G = (V, E)$ be a graph. For any subset $S \subset V$, we write $e_G(S)$ to denote the number of edges in E that have exactly one endpoint in S.

We say that $S \subseteq V$ is a (k, δ)-*isolated neighborhood* of $v \in V$ if $v \in S$, the subgraph induced by S is connected, $|S| \le k$, and $e_G(S) \le \delta|S|$.

We now show that a random vertex has an isolated neighborhood of required properties with high probability.

Lemma 2.5.8. *Let $G = (V, E)$ be a $\rho(\varepsilon)$-hyperfinite graph with degree bounded by d, where $\rho(\varepsilon)$ is a function from $\mathbb{R}_+$ to $\mathbb{R}_+$. Let $G' = (V', E')$ be a subgraph of G that is induced by at least δn vertices. For any $\varepsilon \in (0, 1)$, the probability that a random vertex in G' does not have a $(\rho(\varepsilon^2\delta/28800), \varepsilon/120)$-isolated neighborhood in G' is at most $\varepsilon/120$.*

Proof. Any induced subgraph of G can still be partitioned into components of size at most $\rho(\varepsilon)$ by removing at most εn edges. Since G' has at least δn vertices, it is still $(\varepsilon/\delta, \rho(\varepsilon))$-hyperfinite for any $\varepsilon > 0$, or equivalently, it is $(\varepsilon, \rho(\varepsilon \cdot \delta))$-hyperfinite for any $\varepsilon > 0$.

Therefore, there is a set $S' \subseteq E'$ of at most $(\varepsilon^2/28800)|V'|$ edges such that if all the edges in S' are removed, the number of vertices in each connected component is at most $\rho(\varepsilon^2\delta/28800)$. Denote the achieved partition of vertices into connected components by P. We have

$$\mathbb{E}_{v \in V'}\left[\frac{e_G(P[v])}{|P[v]|}\right] = \sum_{S \in P} \frac{|S|}{|V'|} \cdot \frac{e_G(S)}{|S|} = \frac{2|S'|}{|V'|} \leq \frac{\varepsilon^2}{14400}.$$

By Markov's inequality, the probability that a random $v \in V'$ is such that $e(P[v])/|P[v]| > \frac{\varepsilon}{120}$ is at most $\varepsilon/120$. Otherwise, $P[v]$ is an $(\rho(\varepsilon^2\delta/28800), \varepsilon/120)$-isolated neighborhood of v. □

Finally, we now use the above lemma to construct a partitioning oracle.

Proof of Theorem 2.5.2. We set $k = \rho(\varepsilon^3/3456000)$ and $\delta = \varepsilon/120$. Consider the global Algorithm 5 with these parameters. The algorithm partitions the vertices of the input graph into sets of size at most k. We define a sequence of random variables X_i, $1 \leq i \leq n$, as follows. X_i corresponds to the i-th vertex removed by Algorithm 5 from the graph. Say, the remaining graph has $n - t$ vertices, and the algorithm is removing a set S of r vertices. Then we set $X_{t+1} = \ldots = X_{t+r} = e_{G'}(S)/r$, where G' is the graph before the removal of S. Note that $\sum_{i=1}^{n} X_i$ equals the number of edges between different parts in P. For every i, if X_i corresponds to a set S that was a (k, δ)-isolated neighborhood of the sampled vertex, then $X_i \leq \delta = \varepsilon/120$. Otherwise, we only know that $X_i \leq d$. However, by Lemma 2.5.8, if $i \leq n - \varepsilon n/120$, this does not happen with probability greater than $\varepsilon/120$. Therefore, for every $i \leq n - \varepsilon n/120$, we have

$$\mathbb{E}[X_i] \leq \varepsilon/120 + d \cdot \varepsilon/120 \leq 2\varepsilon d/120.$$

For $i > n - \varepsilon n/120$, we again use the bound $X_i \leq d$. Altogether, this gives that the expected number of edges connecting different parts of P is at most $2\varepsilon dn/120 + \varepsilon dn/120 < \varepsilon dn/40$. Markov's inequality implies that the number of such edges is at most $\varepsilon dn/2$ with probability $1 - 1/20$.

Algorithm 5 can be simulated locally as follows. For each vertex v, we want to

compute $P[v]$. Instead of a random permutation, we independently assign a random number $r(v)$ uniformly selected from the range $[0, 1]$. We only generate $r(v)$'s when they are necessary, and we store them as they may be needed later again. The numbers generate a random ordering of vertices. To compute $P[v]$, we first recursively compute $P[w]$ for each vertex w with $r(w) < r(v)$ and distance to v at most $2 \cdot k$. If $v \in P[w]$ for one of those w, then $P[v] = P[w]$. Otherwise, we search for a (k, δ)-isolated neighborhood of v, keeping in mind that all vertices in $P[w]$ that we have recursively computed are no longer in the graph. If we find such an neighborhood, we set $P[v]$ to it. Otherwise, we set $P[v] = \{v\}$.

We now analyze the above local simulation procedure, using Lemma 2.5.6. Our computation graph is $G^\star = (V, E^\star)$, where $E^\star$ connects all pairs of vertices that are at distance at most $2 \cdot k$ in the input graph. The degree of $G^\star$ is bounded by $D = d^{O(k)}$. The expected number of vertices w for which we have to compute $P[w]$ to obtain $P[v]$ for a fixed vertex v is at most $T = 2^{d^{O(k)}}$. Suppose that we run the procedure for every vertex in the graph, but we never recursively compute $P[w]$ for more than $T' = 40T/\varepsilon$ vertices w. The probability that we do not compute $P[v]$ for a given v is by Markov's inequality at most $\varepsilon/40$. The expected number of vertices that we fail for is bounded by $\varepsilon n/40$. Using Markov's inequality again, with probability $1 - 1/20$, the number of such vertices is bounded by $\varepsilon n/2$.

The oracle works as follows for a given query vertex v. It first checks whether there is at least one vertex u for which we compute $P[u]$ using at most T' recursive calls, and $v \in P[u]$. This can be done by running the recursive simulation procedure for every vertex u at distance less than k from v. If there is such vertex u, the oracle returns $P[u]$. Otherwise, it returns the singleton $\{v\}$. This procedure clearly provides query access to a partition P' of the vertex set with all components of size at most k.

Let us show now that the number of edges cut in P' is likely to be small. First, we know that P cuts at most $\varepsilon dn/2$ vertices with probability at least $1 - 1/20$. P' additionally partitions components of in P in which the recursive evaluation does not finish in T' recursive calls for any vertex. The number of such vertices, and also the number of vertices in such components, is at most $\varepsilon n/2$ with probability $1 - 1/20$.

This means that with probability $1 - 1/20$, at most $\varepsilon dn/2$ additional edges are cut in P'. Therefore, with probability at least $1 - 1/20 - 1/20 = 9/10$, the P' cuts at most εdn edges.

Let us now bound the number of graph queries that the oracle makes to answer a single query. It simulates the recursive procedure starting at $d^{O(k)}$ vertices with at most $T' = 2^{d^{O(k)}}/\varepsilon$ recursive calls each time. Each recursive call has query complexity $d^{O(k)}$ associated with reading the neighborhood of radius $O(k)$ at every vertex. In total, this gives a bound of $2^{d^{O(k)}}/\varepsilon$ queries. For q queries, the amount of computation the oracle uses is $q \log q \cdot 2^{d^{O(k)}}/\varepsilon$. The extra logarithmic factor comes from a dictionary that is necessary to store random numbers $r(v)$ assigned to vertices v of the graph, once they are generated. $\square$

2.5.2 An Efficient Partitioning Oracle for Minor-Free Graphs

In this section, we describe an efficient partitioning oracle for minor-free graphs. The construction is inspired by a superconstant-time partitioning algorithm of Czygrinow, Hańćkowiak, and Wawrzyniak [33, Section 2].

We describe a local distributed partitioning algorithm that builds the required partition. Recall that a distributed algorithm is local if it runs in a constant number of rounds. Such an algorithm can be used to construct an efficient partitioning oracle, since for every query point, one can compute the distributed algortihm's output by simulating it for nodes within a constant radius around the query point. See the paper of Parnas and Ron [70] for more details.

Preliminaries

We use the following facts. The second of them follows from the first fact and the Nash-Williams Theorem.

Fact 2.5.9. *Let H be an arbitrary fixed minor. There is a constant $c^\star_H > 1$ such that*

1. *In every H-minor-free graph $G = (V, E)$, $|E| \le c^\star_H \cdot |V|$.*

Algorithm 6: A single contraction step for weighted minor-free graphs $G' = G/(V_1, \ldots, V_k)$

1. Every vertex w_i, $1 \le i \le k$, in G' selects a random color among red, blue, and green.
2. Every vertex w_i, $1 \le i \le k$, in G' selects a neighbor w'_i such that (w_i, w'_i) is the heaviest edge w_i is incident with.
3. For every red vertex w_i, $1 \le i \le k$, construct the set $Q_i \subseteq [k]$ that consists of j such that w_j is blue, and $w'_j = w_i$.
4. For every such Q_i, merge V_i with all V_j, $j \in Q_i$.

2. *The edge set E of every H-minor-free graph $G = (V, E)$ can be partitioned into at most $c^\star_H$ forests.*

Throughout the section we use graph contractions. A graph contraction is a weighted graph that is a result of contracting some subsets of vertices. Graph contractions preserve weights of edges going between the contracted subsets of vertices.

Definition 2.5.10. *Let $V_1, \ldots, V_k$ be a partition of the set of vertices V of a weighted graph $G = (V, E, \omega)$. We define the* contraction $G/(V_1, \ldots, V_k)$ of G with respect to the partition $V_1, \ldots, V_k$ *as the following weighted graph $G' = (V', E', \omega')$. G' has k vertices w_i, $1 \le i \le k$. An edge (w_i, w_j) belongs to G' if and only if $i \ne j$, and there are $v_i \in V_i$, and $v_j \in V_j$ such that $(v_i, v_j) \in E$. The weight $\omega'((w_i, w_j))$ of such an edge equals $\sum_{x \in V_i} \sum_{y \in V_j} \omega((x, y))$.*

Note that the following holds by definition for minor-free graphs.

Fact 2.5.11. *Let $V_1, \ldots, V_k$ be a partition of the set of vertices V of an H-minor-free graph G. If every set V_i of vertices induces a connected component in G, then $G/(V_1, \ldots, V_k)$ is H-minor-free as well.*

Single Local Contraction

We now describe a single step of the partitioning algorithm. We assume we have a weighted planar graph with non-negative weights. Algorithm 6 finds a number of disjoint stars in the graph and contracts them. We prove the following fact.

Lemma 2.5.12. *Let H be a fixed minor, and let $c_H^\star$ be the constant from Fact 2.5.9. Let G be a weighted H-minor-free graph with non-negative weights.*

Let $V_1, \ldots, V_k$ be a partition of vertices of an H-minor-free graph $G = (V, E, \omega)$ such that each V_i, $1 \le i \le k$, induces a connected component in G. Algorithm 6 turns $G_1 = G/(V_1, \ldots, V_k)$ into a graph $G_2 = G/(V_1', \ldots, V_{k'}')$ such that with probability $1/(36c_H^\star - 1)$, the total weight of edges in G_2 is at most $(1 - 1/(36c_H^\star))$ of the weight of G_1.

Proof. Let w be the total weight of edges in G_1. By Fact 2.5.9, the edge set of G_1 can be decomposed into at most $c_H^\star$ forests. At least one of them has weight at least $w/c_H^\star$. Root every tree in this forest, and direct each edge towards the corresponding root. The outdegree of each vertex v is at most 1. For vertices v with outdegree 1, let a_v be the weight of the outgoing edge, and let $a_v = 0$, otherwise.

Let us move back to G_1. If every vertex v selects the heaviest incident edge as in Step 2 of Algorithm 6, then the weight of this edge is at least a_v. Since every edge can be selected at most twice (by its both endpoints), the total weight of selected edges is at least $w/(2c_H^\star)$. Each of these edges is then contracted with probability at least 1/9, if its endpoints are assigned the right configuration of colors. Therefore, the expected weight of contracted edges is at least $w/(18c_H^\star)$. This implies that the expected weight of edges that are not contracted is at most $w(1 - 1/(18c_H^\star))$. By Markov's inequality, the weight of edges that are not contracted is greater than $w(1 - 1/(36c_H^\star))$ with probability at most

$$\frac{1 - 1/(18c_H^\star)}{1 - 1/(36c_H^\star)} = \frac{36c_H^\star - 2}{36c_H^\star - 1}.$$

□

Additionally, note that the number of communication rounds Algorithm 6 needs is constant with respect to G'.

Full Distributed Algorithm

We now show a local distributed algorithm that computes a good partition by running Algorithm 6 multiple times.

Lemma 2.5.13. *Let H be a fixed minor, and let $\varepsilon \in (0,1)$. There is a distributed partitioning algorithm that runs in $\mathrm{poly}(1/\varepsilon)$ rounds on every H-minor-free graphs $G = (V, E)$, and determines a partition of the graph such that:*

- *The diameter of each connected component is bounded by $\mathrm{poly}(1/\varepsilon)$.*
- *With probability $9/10$, the number of cut edges is at most $\varepsilon|V|$.*

If the degree of the graph is bounded by $d \geq 2$, then the total amount of computation per node is bounded by $d^{\mathrm{poly}(1/\varepsilon)}$.

Proof. We assume that the weight of each edge of the input graph is 1. In the initial partition each vertex belongs to a separate set. The distributed algorithm simulates Algorithm 6 exactly $M = 7 \cdot (36c_H^\star - 1) \cdot \lceil \log_{(1-1/(36c_H^\star))}(\varepsilon/c_H^\star) \rceil = O(\log(1/\varepsilon))$ times. Each execution of Algorithm 6 is conducted on the latest partition of the graph and produces a new partition. The diameter of each connected component in the i-th partition is bounded by C_1^i, where C_1 is a constant. Therefore, simulating the i-th execution of the algorithm only requires C_2^i communication rounds, for some constant C_2. For any fixed constant C, $C^M = \mathrm{poly}(1/\varepsilon)$, so the diameter of each component is bounded by $\mathrm{poly}(1/\varepsilon)$, and the total number of communication rounds is $\mathrm{poly}(1/\varepsilon)$.

Recall now Lemma 2.5.12. With probability $1/(36c_H^\star - 1)$, a single execution of Algorithm 6 decreases the number of edges of the orginal graph that are cut by the current partition by a factor of $1 - 1/(36c_H^\star)$. The expected number of times this happens is at least $7 \cdot \lceil \log_{(1-1/(36c_H^\star))}(\varepsilon/c_H^\star) \rceil$. By the Chernoff bound, the probability that this happens fewer than $\lceil \log_{(1-1/(36c_H^\star))}(\varepsilon/c_H^\star) \rceil$ times is bounded by $\exp(-7 \cdot (6/7)^2/2) < 1/10$. Hence the final partition cuts $\varepsilon|V|$ edges with probability less than $9/10$.

If the maximum vertex degree of the input graph is bounded by d, then the size of each component and the degree of each vertex in the current graph contraction is always of order $d^{\mathrm{poly}(1/\varepsilon)}$. This enables computing the partition with at most $d^{\mathrm{poly}(1/\varepsilon)}$ computation per vertex. □

Partitioning Oracle

We now finish the construction of our oracle.

Proof of Theorem 2.5.3. For every query the oracle locally simulates the distributed algorithm from Lemma 2.5.13 with the algorithm's ϵ set to $\varepsilon/2$. See the paper of Parnas and Ron [70] for details how to conduct such simulation. The required number of queries to the graph is $d^{\text{poly}(1/\varepsilon)}$ for every query to the oracle. The oracle needs to store previous coin tosses for each seen vertex in the graph. The total computation time for q queries is at most $q \log q \cdot d^{\text{poly}(1/\varepsilon)}$, where the extra $\log q$ factor comes from the use of a dictionary.

This way the oracle determines the connected component $Q = (V', E')$ that contains the query point. The oracle further subpartitions the component by using the algorithm from Proposition 4.1 of [4] to determine at most $\varepsilon|V'|/(2d)$ vertices $V'' \subseteq V'$ such that removing them leaves connected components of size at most $O(d/\varepsilon^2)$. The running time of this algorithm is polynomial in the size of Q. It is important that V'' is computed in such a way that it does not depend on which of the vertices in V' is the query point. By cutting at most $\varepsilon|V'|/2$ edges incident to the vertices in V'', we finally achieve the required partition.

With probability 9/10, the distributed algorithm from Lemma 2.5.13 cuts at most $\varepsilon|V|/2$ edges, and then the algorithm from Proposition 4.1 of [4] for each component cuts in total at most another $\varepsilon|V|/2$ edges. □

2.6 Other Applications of Our Methods

2.6.1 Local Distributed Algorithms

Our technique can be used to construct distributed algorithms that with constant probability produce a good solution for graph problems that we have considered. For instance, for the maximum matching problem, there is an algorithm that in $c = c(\varepsilon, d)$ communication rounds collects information about all vertices and edges in the radius c, and random numbers assigned to each prospective augmenting path. For all but a

constant fraction of edges, the knowledge suffices to decide if they are in the matching or not. If the radius-c neighborhood does not suffice to decide if an edge is in the matching, we decide it is not. With high probability, only a small fraction of edges that should be in the matching is not included.

2.6.2 Testing the Property of Having a Perfect Matching

We consider the class $\mathcal{C}_d$ of graphs of the maximum degree bounded by d. In property testing of graphs in the bounded degree model [40], one wants to distinguish two subsets of $\mathcal{C}_d$: graphs that have a property P and those that need to have at least εdn edges added or removed to have P, where $\varepsilon > 0$ is a parameter.

Consider the property of having a perfect matching (we focus on graphs with an even number of nodes). Clearly, for a graph with a perfect matching, the maximum matching size is $n/2$. On the other hand, for any graph that must have at least εdn edges added or removed to have P, the maximum matching size is smaller than $n/2 - \Omega(\varepsilon dn)$. Our maximum matching algorithm can then be used to efficiently solve the testing problem in constant time.

2.6.3 Testing Minor-Closed Properties

We now describe how partitioning oracles can be used for testing if a bounded-degree graph has a minor-closed property. The constant-time testability of minor-closed properties was first established by Benjamini, Schramm, and Shapira [18].

We now recall the definition of *property testing* in the bounded degree model [40]. A graph G is *ε-far* from a property $\mathcal{P}$ if it must undergo at least εdn graph operations to satisfy $\mathcal{P}$, where a single graph operation is either an edge removal or an edge insertion. An *ε-tester* T for property $\mathcal{P}$ is a randomized algorithm that has query access to G in the sense defined in the preliminaries, and:

- if G satisfies $\mathcal{P}$, T accepts with probability at least $2/3$,
- if G is ε-far from $\mathcal{P}$, T rejects with probability at least $2/3$.

Algorithm 7: Tester for H-Minor Freeness (for sufficiently large graphs)

Input: query access to a partition P given by an $(\varepsilon d/4, k)$-partitioning oracle for H-minor free graphs with degree bounded by d for the input graph

1 $f := 0$
2 **for** $j = 1, \ldots, t_1$ *(where* $t_1 = O(1/\varepsilon^2)$*)* **do**
3 Pick a random $v \in V$ and a random $i \in [d]$
4 **if** *v has $\geq i$ neighbors, and the i-th neighbor of v not in $P[v]$* **then**
$f := f + 1$
5 **if** $f/t_1 \geq \frac{3}{8}\varepsilon$ **then** REJECT
6 Select independently at random a set S of $t_2 = O(1/\varepsilon)$ vertices of the graph
7 **if** *the graph induced by $\bigcup_{x \in S} P[x]$ is not H-minor free* **then** REJECT
8 **else** ACCEPT

Lemma 2.6.1. *Let H be a fixed graph. Let $\mathcal{O}$ be an $(\varepsilon d/4, k)$-partitioning oracle for the family of H-minor free graphs with degree bounded by d. There is an ε-tester for the property of being H-minor free in the bounded-degree model that provides $\mathcal{O}$ with query access to the input graph, makes $O(1/\varepsilon^2)$ uniform queries to $\mathcal{O}$, and uses $O((dk \log k)/\varepsilon + k^3/\varepsilon^6) = \mathrm{poly}(d, k, 1/\varepsilon)$ time for computation.*

Proof. For sufficiently large graphs, our tester is Algorithm 7. The value t_1 equals C_1/ε^2 for a sufficiently high constant C_1 such that by the Chernoff bound the number of edges cut by the partition P is approximated up to $\pm\varepsilon dn/8$ with probability $9/10$. Let $t_3 = C_2/\varepsilon$ be an upper bound on the expected time to hit a set of size $\varepsilon|X|$ by independently taking random samples from X, where C_2 is a sufficiently large constant. We set t_2 in the algorithm to $10 \cdot q \cdot t_3$, where q is the number of connected components in H. Finally, we set t_4 to $C_3 \cdot k \cdot t_2^2$ for a sufficiently high constant C_3 such that for graphs on more than t_4 nodes, the probability that two samples from S belong to the same component $P[v]$ is at most $1/10$.

If the number of vertices in the graph is at most $t_4 = O(k/\varepsilon^2)$, we read the entire graph, and check if the input is H-minor free in $O((k/\varepsilon^2)^3)$ time via the cubic algorithm of Robertson and Seymour [73]. For larger graphs, we run Algorithm 7. The loop in Lines 2–4 takes at most $O(d/\varepsilon^2)$ time. In Line 7, the induced graph can be read in $O((dk \log k)/\varepsilon)$ time, and then $O((k/\varepsilon)^3)$ time suffices to test whether it is H-minor free. Therefore, the amount of computation that the algorithm uses is

$O((dk \log k)/\varepsilon + (k/\varepsilon^2)^3)$.

If G is H-minor free, then the fraction of edges cut by P is with probability $1-1/10$ at most $\varepsilon dn/4$. If this is the case, the estimate on the number of cut edges edges is at most $3\varepsilon dn/8$ with probability $1 - 1/10$. Moreover, every induced subgraph of G is also H-minor free, so G cannot be rejected in the loop in Line 5 of the algorithm. Hence, G is accepted with probability at least $8/10 > 2/3$.

Consider now the case when G is ε-far. If the partition P cuts more than $\varepsilon dn/2$ edges, the graph is rejected with probability $1 - 1/10 > 2/3$. Suppose now that P cuts fewer than $\varepsilon dn/2$ edges and the tester does not reject in Line 5. Let G' be the new graph after the partition. G' remains $\varepsilon/2$-far from H-minor freeness, and there are at least $\varepsilon dn/2$ edges that must be removed to get an H-minor free graph. This implies that G' is $\varepsilon/2$-far from H_i-minor freeness also for every connected component H_i, $1 \leq i \leq q$, of H. For every i, at least an $\varepsilon dn/2$ edges belong to a component of G' that is not H_i-minor free. It follows that at least εn vertices are incident to such an edge. Therefore, it suffices to pick in expectation t_3 random nodes to find a component that is not H_i-minor free. For q connected components of H, it suffices to pick in expectation $q \cdot t_3$ random nodes to find each of them. By picking, $10 \cdot q \cdot t_3$ random nodes, we find the components with probability $1-1/10$. Furthermore, since the considered graph is large, i.e., has at least t_4 nodes, the components for each i are different with probability $1 - 1/10$, and the graph is rejected in Line 7. Therefore, the probability that a graph that is ε-far is accepted is at most $3/10 < 1/3$. $\square$

By combining Theorem 2.5.3 with Lemma 2.6.1, we obtain a $2^{\text{poly}(1/\varepsilon)}$-time tester for H-minor freeness for graphs of degree $O(1)$. Since every minor-closed property can be expressed via a finite set of excluded minors H [74], it suffices to test if the input is ε/s-far from being minor free for each of them, where s is their number. We arrive at the following theorem.

Theorem 2.6.2. *For every minor-closed property $\mathcal{P}$, there is a uniform ε-tester for $\mathcal{P}$ in the bounded-degree model that runs in $2^{\text{poly}(1/\varepsilon)}$ time.*

Algorithm 8: Approximating distance to not having a set of connected graphs as induced subgraphs

Input: set $\mathcal{H}$ of connected graphs (does not include the graph on one vertex)
Input: query access to a partition P given by an $(\varepsilon d/4, k)$-partitioning oracle for a family $\mathcal{C}$ of graphs

1 $f := 0$
2 **for** $j = 1, \ldots, t$ *(where* $t = O(1/\varepsilon^2)$*)* **do**
3 Pick a random $v \in V$
4 $q :=$ the minimum number of edge operations to make the graph induced by $P[v]$ have no graph in $\mathcal{H}$ as an induced subgraph
5 $f := f + \frac{q}{d \cdot |P[v]|}$
6 Return $f/t + \varepsilon/2$.

2.6.4 Approximating Distance to Hereditary Properties For Hyperfinite Graphs

Parnas, Ron, and Rubinfeld [71] studied generalizations of property testing: *tolerant testing* and *distance approximation*. For a given property $\mathcal{P}$, and an appropriately defined distance to $\mathcal{P}$, an $(\varepsilon_1, \varepsilon_2)$-*tolerant tester* for $\mathcal{P}$ distinguishes inputs at distance at most ε_1 from $\mathcal{P}$ and those at distance at least ε_2 from $\mathcal{P}$ with probability at least $2/3$, where $0 \leq \varepsilon_1 < \varepsilon_2$. An (α, β)-*distance approximation algorithm* for $\mathcal{P}$ computes an (α, β)-approximation to the distance of the input to $\mathcal{P}$ with probability $2/3$. In the following, we study constant-time $(1, \delta)$-*distance approximation algorithms* with δ being a parameter. Such algorithms immediately yield constant-time $(\varepsilon_1, \varepsilon_2)$-tolerant testers by setting δ to $(\varepsilon_2 - \varepsilon_1)/2$.

In the bounded-degree model, the *distance* to a given property $\mathcal{P}$ is $k/(dn)$, where k is the minimum number of graph operations (edge insertions and deletions) that are needed to make the graph achieve $\mathcal{P}$. All input graphs have the maximum degree bounded by d, but the closest graph with property $\mathcal{P}$ need not have the maximum degree bounded by d.

As a warmup, we construct an algorithm for the case when the hereditary property can be expressed as a finite set of forbidden induced subgraphs. The algorithm illustrates basic ideas behind distance approximation. We will later use it to construct an algorithm for general hereditary properties.

Lemma 2.6.3. *Let $\mathcal{H}$ be a fixed set of connected graphs that does not contain the one-vertex graph. Let $\mathcal{O}$ be an $(\varepsilon d/4, k)$-partitioning oracle for a family $\mathcal{C}$ of graphs with degree bounded by d, where k is a function of only ε. There is a $(1, \varepsilon)$-approximation algorithm for the distance to the property of not having any graph in $\mathcal{H}$ as an induced subgraph, for graphs in $\mathcal{C}$. The algorithm provides $\mathcal{O}$ with query access to the input graph, makes $O(1/\varepsilon^2)$ random uniformly distributed queries to $\mathcal{O}$, and uses $(O(dk) + 2^{O(k^2)})/\varepsilon^2$ time for computation.*

Proof. We use Algorithm 8. The partition P cuts at most $\varepsilon dn/4$ edges with probability $1 - 1/10$, which implies that the distance to the property changes by at most $\pm\varepsilon/4$. Consider the new graph G' with connected components corresponding to the partition of P. Every graph in $H \in \mathcal{H}$ is connected, so H can only appear as an induced subgraph of a connected component of G'. Therefore, it does not make sense to add edges connecting components of G'. This would not exclude any existing induced graph from $\mathcal{H}$. Hence, any shortest sequence of operations that removes from G' all induced copies of graphs in $\mathcal{H}$, does this for each connected component in G' separately.

The value $t = O(1/\varepsilon^2)$ in the algorithm is chosen such that we estimate the number of edge operations divided by dn up to $\pm\varepsilon/4$ with probability $1-1/10$ by the Chernoff bound. Therefore, the algorithm returns a correct estimate with probability at least $1 - 1/10 - 1/10 = 4/5$. The best set of edge operations can be found for a single component in $2^{O(k^2)}$ time by enumerating all possible modifications, and verifying that none of the graphs in $\mathcal{H}$ on at most k nodes are present as an induced subgraph. □

We now construct an approximation algorithm that works for any non-degenerate hereditary property. Recall that a property is degenerate if it prohibits an empty graph on some number of nodes. The proof reuses some ideas of Czumaj, Shapira, and Sohler [28], who showed a one-sided tester for hereditary properties of hyperfinite families of bounded-degree graphs.

Lemma 2.6.4. *Let $\mathcal{P}$ be a non-degenerate hereditary property. Let $\mathcal{O}$ be an $(\varepsilon d/16, k)$-partitioning oracle for a family $\mathcal{C}$ of graphs with degree bounded by d. There is a*

(non-uniform) $(1, \varepsilon)$*-approximation algorithm for the distance to* $\mathcal{P}$ *for graphs in* $\mathcal{C}$*. The algorithm provides* $\mathcal{O}$ *with query access to the input graph, and makes a constant number of uniformly distributed queries to the oracle. Its running time is independent of the graph size.*

Proof. Let $\mathcal{H}$ be the set of all graphs that do not have $\mathcal{P}$. Since $\mathcal{P}$ is hereditary, if a graph has any of the graphs in $\mathcal{H}$ as an induced subgraph, it does not have $\mathcal{P}$. Consider a subset $\mathcal{H}'$ of $\mathcal{H}$ that only consists of graphs $H \in \mathcal{H}$ that have all components of size at most k. There are at most $t = 2^{O(k^2)}$ different connected graphs $A_1, \ldots, A_t$ on at most k vertices. Every graph in $\mathcal{H}'$ can be represented as a vector $a \in \mathbb{N}^t$, where a_i is the number of times A_i appears as a connected component. For a graph $H \in \mathcal{H}'$, its *configuration* is the vector $c \in \{0,1\}^t$ such that for each i, $1 \leq i \leq t$, $c_i = 0$ if and only if $a_i = 0$. We say that a configuration $c \in \{0,1\}^t$ is *present* if there is a graph in $\mathcal{H}'$ with configuration c. We call the one-vertex graph *trivial*. Recall that $\mathcal{H}$ is non-degenerate. This implies that for each present configuration c, there is i such that $c_i = 1$, and A_i is non-trivial. A subset $\mathcal{X}$ of $\mathcal{A} = \{A_i : 1 \leq i \leq t\}$ is *hitting* if it does not contain the trivial graph, and for every present configuration c, there is j such that $c_j = 1$ and $A_j \in \mathcal{X}$. For non-degenerate $\mathcal{H}$, there always exists at least one hitting subset of $\mathcal{A}$.

Since there exists a $(\varepsilon d/16, k)$-partitioning oracle for the input graph G, G is $(\varepsilon d/16, k)$-hyperfinite, and there is a graph G' with components of size at most k that can be created by removing at most $\varepsilon dn/16$ edges from G. G' is at distance at most $\varepsilon/16$ from G. The distance of G' to $\mathcal{P}$ is bounded from above by the minimum distance from having no induced subgraph in a hitting set $\mathcal{X}$, where the minimum is taken over all hitting sets $\mathcal{X}$. If we exclude at least one connected component for every graph in $\mathcal{H}'$, we get a graph that satisfies $\mathcal{P}$. We write M to denote the above minimum distance to excluding a hitting set from G'. Note that the shortest sequence of operations that exclude a given hitting set $\mathcal{X}$ does not add edges between different connected components of G. These edges do not remove any existing copy of a graph in $\mathcal{X}$. Note that M is bounded by 1, since it suffices to remove all edges in G' to achieve $\mathcal{P}$.

We now claim that in fact, we have to exclude some hitting set almost entirely for sufficiently large graphs, or a sequence of operations turning the graph into a graph that has $\mathcal{P}$ is long. For every present configuration $c \in \{0,1\}^t$ (the number of them is finite), we fix an arbitrary graph $H_c \in \mathcal{H}'$ with this configuration. Consider any sequence of at most $(M - \varepsilon/4) \cdot dn$ operations that turns G' into a graph G''. We will show that for n greater than some constant (which depends on ε, d, k, and $\mathcal{P}$), G'' has an induced copy of one of the graphs H_c. Let $G_\star$ be G'' with only edges that connect vertices in the same connected component in G'. By the definition of M, $G_\star$ must be $\varepsilon/4$-far from having any of the hitting sets excluded. We claim that there is a present configuration c such that for every non-trivial A_i with $c_i = 1$, the distance of $G_\star$ to not having A_i as an induced subgraph is at least $\varepsilon/(8k2^t)$. Suppose for contradiction that for every present configuration c, there is i such that A_i is a non-trivial graph, $c_i = 1$, and the distance of $G_\star$ from not having A_i as an induced subgraph is less than $\varepsilon/(8k2^t)$. For every present configuration c, removing such an A_i from $G_\star$ requires a sequence of fewer than $\varepsilon dn/(8k2^t)$ graph operations. For every inserted or deleted edge (u,v) by such an sequence of operations, let us delete from $G_\star$ all edges incident to both u and v. This is fewer than $\varepsilon dn/(4 \cdot 2^t)$ graph deletions for every present configuration c, and this way we do not introduce any new connected induced subgraph. By going over all present configurations, we can entirely remove all induced copies of at least one graph in each configuration with fewer than $\varepsilon dn/4$ graph deletions. This implies that we can exclude a hitting set with fewer than $\varepsilon dn/4$ graph operations. This yields a contradiction.

We proved that there is a present configuration c such that for every i such that $c_i = 1$ and A_i is non-trivial, the distance of $G_\star$ to not having A_i as an induced subgraph is at least $\varepsilon/(8k2^t)$. Note that because each connected component in $G_\star$ has at most k vertices, the number of vertex disjoint copies of H_c is $\Omega_{\varepsilon,d,k}(n)$ in $G_\star$. Let q be the number of connected components in H_c. We can pick sets I_i, $1 \le i \le q$, of subgraphs of $G_\star$ such that each I_i, $1 \le i \le q$, is a set of induced copies of the i-th connected component of H_c, $|I_i| \ge \lfloor n/C \rfloor$ (where C only depends on ε, d, k, and the choice of graphs H_c), and the graphs in $\bigcup_i I_i$ are pairwise vertex disjoint. Note that

each induced subgraph of $G_\star$ that appears in I_i is also an induced graph in G''. There are at least $\lfloor n/C \rfloor^q$ ways of selecting one subgraph from each I_i. Consider one of such choices. If there were no additional edges between the selected subgraphs, this would give us an induced copy of H_c. The total number of edges in G'' is at most $2dn$, and each edge connects at most 2 subgraphs in $\bigcup I_i$. This means that each edge can make at most n^{q-2} choices of one subgraph from each I_i not give an induced copy of H_c. For sufficiently large n, we have $2dn \cdot n^{q-2} < \lfloor n/C \rfloor^q$, and there is an induced copy of H_c. Summarizing, for sufficiently large graphs, the distance of G' to $\mathcal{P}$ is at least $M - \varepsilon/4$.

Therefore, the distance of G to $\mathcal{P}$ is between $M - 5\varepsilon/16$ and $M + \varepsilon/16$. Moreover, M is approximated up to $\pm\varepsilon/16$ by M', which we define as the distance of G to entirely excluding one of the hitting sets. Therefore, to get a sufficiently good approximation to the distance of G to $\mathcal{P}$, it suffices to compute $(1, \varepsilon n/4)$-approximation to M' for sufficiently large graphs. This can be done by using the algorithm of Lemma 2.6.3 for all hitting sets, and amplifying the probability of its success in the standard way. For small graphs, we hard-wire the exact solution to the problem. □

Chapter 3

Edit Distance

In this section, we study the *asymmetric* query model for edit distance. Recall that the input in this model consists of two strings x and y. An algorithm can access y in an unrestricted manner (without charge), and it is charged only for querying every symbol of x.

We prove both upper and lower bounds on the number of queries necessary to distinguish pairs of close strings from pairs of strings far apart. In particular, we prove a more general version of the results (Theorem 1.2.2, Theorem 1.2.3) we stated in Section 1.2. Our upper bound leads to the first algorithm that computes a multiplicative $\log^{O(1/\varepsilon)} n$-approximation to edit distance between two strings of length n in $n^{1+\varepsilon}$ time, for any constant $\varepsilon > 0$.

3.1 Outline of Our Results

Before we give full proofs, we first sketch the main ideas behind our proofs.

3.1.1 Outline of the Upper Bound

In this section, we provide an overview of our algorithmic results, in particular of the proof of Theorem 1.2.2. Full statements and proofs of the results appear in Section 3.2.

Our proof has two major components. The first one is a characterization of edit distance by a different "distance", denoted $\mathcal{E}$, which approximates edit distance well. The second component is a sampling algorithm that approximates $\mathcal{E}$ up to a constant factor by making a small number of queries into x. We describe each of the components below. In the following, for a string x and integers $s, t \geq 1$, $x[s : t]$ denotes the substring of x comprising of $x[s], \ldots, x[t-1]$.

Edit Distance Characterization: the $\mathcal{E}$-distance

Our characterization of $\mathrm{ed}(x, y)$ may be viewed as computation on a tree, where the nodes correspond to substrings $x[s : s+l]$, for some start position $s \in [n]$ and length $l \in [n]$. The root is the entire string $x[1 : n+1]$. For a node $x[s : s+l]$, we obtain its children by partitioning $x[s : s+l]$ into b equal-length blocks, $x[s+j \cdot l/b : s+(j+1) \cdot l/b]$, where $j \in \{0, 1, \ldots b-1\}$. Hence $b \geq 2$ is the arity of the tree. The height of the tree is $h \stackrel{\text{def}}{=} \log_b n$. We also use the following notation: for level $i \in \{0, 1, \ldots h\}$, let $l_i \stackrel{\text{def}}{=} n/b^i$ be the length of strings at that level. Let $B_i \stackrel{\text{def}}{=} \{1, l_i+1, 2l_i+1, \ldots\}$ be the set of starting positions of blocks at level i.

The characterization is asymmetric in the two strings and is defined from a node of the tree to a position $u \in [n]$ of the string y. Specifically, if $i = h$, then the *$\mathcal{E}$-distance* of $x[s]$ to a position u is 0 only if $x[s] = y[u]$ and $u \in [n]$, and 1 otherwise. For $i \in \{0, 1, \ldots h-1\}$ and $s \in B_i$, we recursively define the $\mathcal{E}$-distance $\mathcal{E}(i, s, u)$ of $x[s : s+l_i]$ to a position u as follows. Partition $x[s : s+l_i]$ into b blocks of length $l_{i+1} = l_i/b$, starting at positions $s + t_j$, where $t_j \stackrel{\text{def}}{=} j \cdot l_{i+1}$, $j \in \{0, 1, \ldots b-1\}$. Intuitively, we would like to define the $\mathcal{E}$-distance $\mathcal{E}(i, s, u)$ as the summation of the $\mathcal{E}$-distances of each block $x[s+t_j : s+t_j+l_{i+1}]$ to the corresponding position in y, i.e., $u + t_j$. Additionally, we allow each block to be displaced by some shift r_j, incurring an additional charge of $|r_j|$ in the $\mathcal{E}$-distance. The shifts r_j are chosen such as to minimize the final distance. Formally,

$$\mathcal{E}(i, s, u) \stackrel{\text{def}}{=} \sum_{j=0}^{b-1} \min_{r_j \in \mathbb{Z}} \mathcal{E}(i+1, s+t_j, u+t_j+r_j) + |r_j| \,. \tag{3.1}$$

The $\mathcal{E}$-distance from x to y is just the $\mathcal{E}$-distance from $x[1:n+1]$ to position 1, i.e., $\mathcal{E}(0,1,1)$.

We illustrate the $\mathcal{E}$-distance for $b=4$ in Fig. 3-1. Notice that without the shifts (i.e., when all $r_j=0$), the $\mathcal{E}$-distance is exactly equal to the Hamming distance between the corresponding strings. Hence the shifts r_j are what differentiates the Hamming distance and $\mathcal{E}$-distance.

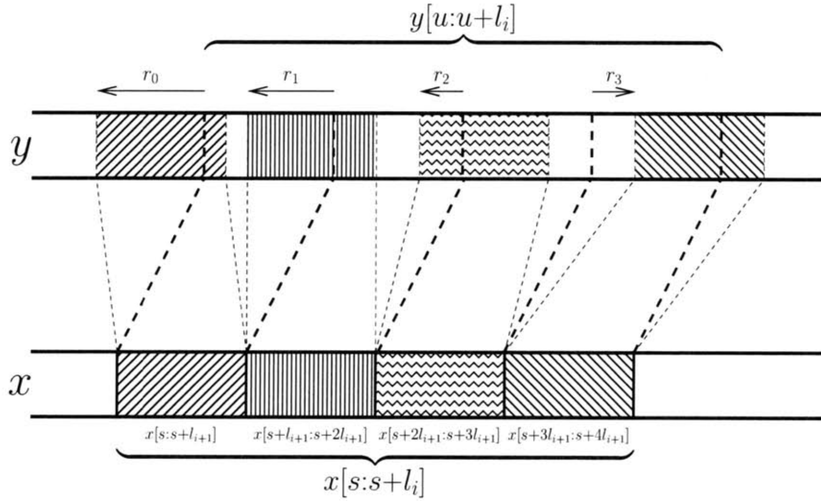


Figure 3-1: Illustration of the $\mathcal{E}$-distance $\mathcal{E}(i,s,u)$ for $b=4$. The pairs of blocks of the same shading are the blocks whose $\mathcal{E}$-distance is used for computing $\mathcal{E}(i,s,u)$.

We prove that the $\mathcal{E}$-distance is a $O(bh)=O(\frac{b}{\log b}\log n)$ approximation to $\mathrm{ed}(x,y)$ (see Theorem 3.2.3). For $b=2$, the $\mathcal{E}$-distance is a $O(\log n)$ approximation to $\mathrm{ed}(x,y)$, but unfortunately, we do not know how to compute it or approximate it well in better than quadratic time. It is also easy to observe that one can compute a $1+\varepsilon$ approximation to $\mathcal{E}$-distance in $\tilde{O}_\varepsilon(n^2)$ time via a dynamic programming that considers only r_j's which are powers of $1+\varepsilon$. Instead, we show that, using the query algorithm (described next), we can compute a $1+\varepsilon$ approximation to $\mathcal{E}$-distance for $b=(\log n)^{O(1/\varepsilon)}$ in $n^{1+\varepsilon}$ time.

Sampling Algorithm

We now describe the ideas behind our sampling algorithm. The sampling algorithm approximates the $\mathcal{E}$-distance between x and y up to a constant factor. The query

complexity is $Q \leq \beta \cdot (\log n)^{O(h)} = \beta \cdot (\log n)^{\log_b n}$ for distinguishing $\mathcal{E}(0,1,1) > n/\beta$ from $\mathcal{E}(0,1,1) \leq n/(2\beta)$. For the rest of this overview, it is instructive to think about the setting where $\beta = n^{0.1}$ and $b = n^{0.01}$, although our main result actually follows by setting $b = (\log n)^{O(1/\varepsilon)}$.

The idea of the algorithm is to prune the characterization tree, and in particular prune the children of each node. If we retain only polylog n children for each node, we would obtain the claimed $Q \leq (\log n)^{O(h)}$ leaves at the bottom, which correspond to the sampled positions in x. The main challenge is how to perform this pruning.

A natural idea is to uniformly subsample polylog n out of b children at each node, and use Chernoff-type concentration bounds to argue that Equation (3.1) may be approximated only from the $\mathcal{E}$-distance estimates of the subsampled children. Note that, since we use the minimum operator at each node, we have to aim, at each node, for an estimate that holds with high probability.

How much do we have to subsample at each node? The "rule of thumb" for a Chernoff-type bound to work well is as follows. Suppose we have quantities $a_1, \ldots a_m \in [0, \rho]$ respecting an upper bound $\rho > 0$, and let $\sigma = \sum_{j \in [m]} a_j$. Suppose we subsample several $j \in [m]$ to form a set J. Then, in order to estimate σ well (up to a small multiplicative factor) from a_j for $j \in J$, we need to subsample essentially a total of $|J| \approx \frac{\rho}{\sigma} \cdot m \log m$ positions $j \in [m]$. We call this Uniform Sampling Lemma (see Lemma 3.2.11 for complete statement).

With the above "sampling rule" in mind, we can readily see that, at the top of the tree, until a level i, where $l_i = n/\beta$, there is no pruning that may be done (with the notation from above, we have $\rho = l_i = n/\beta$ and $\sigma = n/\beta$). However, we hope to prune the tree at the subsequent levels.

It turns out that such pruning is not possible as described. Specifically, consider a node v at level i and its children v_j, for $j = 0, \ldots b-1$. Suppose each child contributes a distance a_j to the sum $\mathcal{E}$ at node v (in Equation (3.1), for fixed u). Then, because of the bound on length of the strings, we have that $a_j \leq l_{i+1} = (n/\beta)/b$. At the same time, for an average node v, we have $\sum_{j=0}^{b-1} a_j \approx l_i/\beta = n/\beta^2$. By the Uniform Sampling Lemma from above, we need to take a subsample of size $|J| \approx \frac{n/(\beta b)}{n/\beta^2} \cdot b \log b =$

$\beta \log b$. If β were constant, we would obtain $|J| \ll b$ and hence prune the tree (and, indeed, this approach works for $\beta \ll b$). However, once $\beta \gg b$, such pruning does not seem possible. In fact, one can give counter-examples where such pruning approach fails to approximate the $\mathcal{E}$-distance.

To address the above challenge, we develop a way to prune the tree *non-uniformly*. Specifically, for different nodes we will subsample its children at different, well-controlled rates. In fact, for each node we will assign a "precision" w with the requirement that a node v, at level i, with precision w, must estimate its $\mathcal{E}$-distances to positions u up to an *additive error* l_i/w. The pruning and assignment of precision will proceed top-bottom, starting with assigning a precision 4β to the root node. Intuitively, the higher the precision of a node v, the denser is the subsampling in the subtree rooted at v.

Technically, our main tool is a *Non-uniform Sampling Lemma*, which we use to assign the necessary precisions to nodes. It may be stated as follows (see Lemma 3.2.12 for a more complete statement). The lemma says that there exists some distribution $\mathcal{W}$ and a reconstruction algorithm R such that the following two conditions hold:

- Fix some $a_j \in [0, 1]$ for $j \in [m]$, with $\sigma = \sum_j a_j$. Also, pick w_j i.i.d. from the distribution $\mathcal{W}$ for each $j \in [m]$. Let $\hat{a}_j$ be estimators of a_j, up to an additive error of $1/w_j$, i.e., $|a_j - \hat{a}_j| \leq 1/w_j$. Then the algorithm R, given $\hat{a}_j$ and w_j for $j \in [m]$, outputs a value that is inside $[\sigma - 1, \sigma + 1]$, with high probability.

- $\mathbb{E}_{w \in \mathcal{W}}[w] = \operatorname{polylog} m$.

To internalize this statement, fix $\sigma = 10$, and consider two extreme cases. At one extreme, consider some set of 10 j's such that $a_j = 1$, and all the others are 0. In this case, the previous uniform subsampling rule does not yield any savings (to continue the parallel, uniform sampling can be seen as having $w_j = m$ for the sampled j's and $w_j = 1$ for the non-sampled j's). Instead, it would suffice to take all j's, but approximate them up to "weak" (cheap) precision (i.e., set $w_j \approx 100$ for all j's). At the other extreme is the case when $a_j = 10/m$ for all j. In this case, subsampling would work but then one requires a much "stronger" (expensive) precision, of the

order of $w_j \approx m$. These examples show that one cannot choose all w_j to be equal. If w_j's are too small, it is impossible to estimate σ. If w_j's are too big, the expectation of w cannot be bounded by $\operatorname{polylog} m$, and the subsampling is too expensive.

The above lemma is somewhat inspired by the sketching and streaming technique introduced by Indyk and Woodruff [47] (and used for the F_k moment estimation), where one partitions elements a_j by weight level, and then performs corresponding subsampling in each level. Although related, our approach to the above lemma differs: for example, we avoid any definition of the weight level (which was usually the source of some additional complexity of the use of the technique). For completeness, we mention that the distribution $\mathcal{W}$ is essentially the distribution with probability distribution function $f(x) = \nu/x^2$ for $x \in [1, m^3]$ and a normalization constant ν. The algorithm R essentially uses the samples that were (in retrospect) well-approximated, i.e., $\hat{a}_j \gg 1/w_j$, in order to approximate σ.

In our $\mathcal{E}$-distance estimation algorithm, we use both uniform and non-uniform subsampling lemmas at each node to both prune the tree and assign the precisions to the subsampled children. We note that the lemmas may be used to obtain a multiplicative $(1 + \varepsilon')$-approximation for arbitrary small $\varepsilon' > 0$ for each node. To obtain this, it is necessary to use $\varepsilon \approx \varepsilon'/\log n$, since over $h \approx \log n$ levels, we collect a multiplicative approximation factor of $(1 + \varepsilon)^h$, which remains constant only as long as $\varepsilon = O(1/h)$.

3.1.2 Outline of the Lower Bound

In this section we outline the proof of Theorem 1.2.3. The full proof appears in Section 3.3. Here, we focus on the main ideas, skipping or simplifying some of the technical issues.

As usual, the lower bound is based on constructing "hard distributions", i.e., distributions (over inputs) that cannot be distinguished using few queries, but are very different in terms of edit distance. We sketch the construction of these distributions in Section 3.1.2. The full construction appears in Section 3.3.4. In Section 3.1.2, we sketch the machinery that we developed to prove that distinguishing these distribu-

tions requires many queries; the details appear in Section 3.3.2. We then sketch in Section 3.1.2 the tools needed to prove that the distributions are indeed very different in terms of edit distance; the detailed version appears in Section 3.3.3.

The Hard Distributions

We shall construct two distributions $\mathcal{D}_0$ and $\mathcal{D}_1$ over strings of a given length n. The distributions satisfy the following properties. First, every two strings in the support of the same distribution $\mathcal{D}_i$, denoted $\text{supp}(\mathcal{D}_i)$, are close in edit distance. Second, every string in $\text{supp}(\mathcal{D}_0)$ is far in edit distance from every string in $\text{supp}(\mathcal{D}_1)$. Third, if an algorithm correctly distinguishes (with probability at least 2/3) whether its input string is drawn from $\mathcal{D}_0$ or from $\mathcal{D}_1$, it must make many queries to the input.

Given two such distributions, we let x be any string from $\text{supp}(\mathcal{D}_0)$. This string is fully known to the algorithm. The other string y, to which the algorithm only has query access, is drawn from either $\mathcal{D}_0$ or $\mathcal{D}_1$. Since distinguishing the distributions apart requires many queries to the string, so does approximating edit distance between x and y.

Randomly Shifted Random Strings. The starting point for constructing these distributions is the following idea. Choose at random two base strings $z_0, z_1 \in \{0,1\}^n$. These strings are likely to satisfy some "typical properties", e.g. be far apart in edit distance (at least $n/10$). Now let each $\mathcal{D}_i$ be the distribution generated by selecting a cyclic shift of z_i by r positions to the right, where r is a uniformly random integer between 1 and $n/1000$. Every two strings in the same $\text{supp}(\mathcal{D}_i)$ are at distance at most $n/500$, because a cyclic shift by r positions can be produced by r insertions and r deletions. At the same time, by the triangle inequality, every string in $\text{supp}(D_0)$ and every string in $\text{supp}(\mathcal{D}_1)$ must be at distance at least $n/10 - 2 \cdot n/500 \geq n/20$.

How many queries are necessary to learn whether an input string is drawn from $\mathcal{D}_0$ or from $\mathcal{D}_1$? If the number q of queries is small, then the algorithm's view is close to a uniform distribution on $\{0,1\}^q$ under both $\mathcal{D}_0$ and $\mathcal{D}_1$. Thus, the algorithm is unlikely to distinguish the two distributions with probability significantly higher than

1/2. This is the case because each base string z_i is chosen at random and because we consider many cyclic shifts of it. Intuitively, even if the algorithm knows z_0 and z_1, the random shift makes the algorithm's view a nearly-random pattern, because of the random design of z_0 and z_1. Below we introduce rigorous tools for such an analysis. They prove, for instance, that even an adaptive algorithm for this case, and in particular every algorithm that distinguishes edit distance $\leq n/500$ and $\geq n/20$, must make $\Omega(\log n)$ queries.

One could ask whether the $\Omega(\log n)$ lower bound for the number of queries in this construction can be improved. The answer is negative, because for a sufficiently large constant C, by querying any consecutive $C \log n$ symbols of z_1, one obtains a pattern that most likely does not occur in z_0, and therefore, can be used to distinguish between the distributions. This means that we need a different construction to show a superlogarithmic lower bound.

Substitution Product. We now introduce the *substitution product*, which plays an important role in our lower bound construction. Let $\mathcal{D}$ be a distribution on strings in Σ^m. For each $a \in \Sigma$, let $\mathcal{E}_a$ be a distribution on $(\Sigma')^{m'}$, and denote their entire collection by $\mathcal{E} \stackrel{\text{def}}{=} (\mathcal{E}_a)_{a \in \Sigma}$. Then the substitution product $\mathcal{D} \circledast \mathcal{E}$ is the distribution generated by drawing a string z from $\mathcal{D}$, and independently replacing every symbol z_i in z by a string B_i drawn from $\mathcal{E}_{z_i}$.

Strings generated by the substitution product consist of m blocks. Each block is independently drawn from one of the $\mathcal{E}_a$'s, and a string drawn from $\mathcal{D}$ decides which $\mathcal{E}_a$ each block is drawn from.

Recursive Construction. We build on the previous construction with two random strings shifted at random, and extend it by introducing recursion. For simplicity, we show how this idea works for two levels of recursion. We select two random strings z_0 and z_1 in $\{0,1\}^{\sqrt{n}}$. We use a sufficiently small positive constant c to construct two distributions $\mathcal{E}_0$ and $\mathcal{E}_1$. $\mathcal{E}_0$ and $\mathcal{E}_1$ are generated by taking a cyclic shift of z_0 and z_1, respectively, by r symbols to the right, where r is a random integer between 1 and

$c\sqrt{n}$. Let $\mathcal{E} \stackrel{\text{def}}{=} (\mathcal{E}_i)_{i\in\{0,1\}}$.

Our two hard distributions on $\{0,1\}^n$ are $\mathcal{D}_0 \stackrel{\text{def}}{=} \mathcal{E}_0 \circledast \mathcal{E}$, and $\mathcal{D}_1 \stackrel{\text{def}}{=} \mathcal{E}_1 \circledast \mathcal{E}$. As before, one can show that distinguishing a string drawn from $\mathcal{E}_0$ and a string drawn from $\mathcal{E}_1$ is likely to require $\Omega(\log n)$ queries. In other words, the algorithm has to *know* $\Omega(\log n)$ symbols from a string selected from one of $\mathcal{E}_0$ and $\mathcal{E}_1$. Given the recursive structure of $\mathcal{D}_0$ and $\mathcal{D}_1$, the hope is that distinguishing them requires at least $\Omega(\log^2 n)$ queries, because at least intuitively, the algorithm "must" know for at least $\Omega(\log n)$ blocks which $\mathcal{E}_i$ they come from, each of the blocks requiring $\Omega(\log n)$ queries. Below, we describe techniques that we use to formally prove such a lower bound. It is straightforward to show that every two strings drawn from the same $\mathcal{D}_i$ are at most $4cn$ apart. It is slightly harder to prove that strings drawn from $\mathcal{D}_0$ and $\mathcal{D}_1$ are far apart. The important ramification is that for some constants c_1 and c_2, distinguishing edit distance $< c_1 n$ and $> c_2 n$ requires $\Omega(\log^2 n)$ queries, where one can make c_1 much smaller than c_2. For comparison, under the Ulam metric, $O(\log n)$ queries suffice for such a task (deciding whether distance between a known string and an input string is $< c_1 n$ or $> c_2 n$, assuming $2c_1 < c_2$ [2]).

To prove even stronger lower bounds, we apply the substitution product several times, not just once. Pushing our approach to the limit, we prove that distinguishing edit distance $O(n/\operatorname{polylog} n)$ from $\Omega(n)$ requires $n^{\Omega(1/\log\log n)}$ queries. In this case, $\Theta(\log n/\log\log n)$ levels of recursion are used. One slight technical complication arises in this case. Namely, we need to work with a larger alphabet (rather than binary). Our result holds true for the binary alphabet nonetheless, since we show that one can effectively reduce the larger alphabet to the binary alphabet.

Bounding the Number of Queries

We start with definitions. Let $\mathcal{D}_0, \ldots, \mathcal{D}_k$ be distributions on the same finite set Ω with $p_1, \ldots, p_k : \Omega \to [0,1]$ as the corresponding probability mass functions. We say that the distributions are *α-similar*, where $\alpha \geq 0$, if for every $\omega \in \Omega$,

$$(1-\alpha)\cdot \max_{i=1,\ldots,k} p_i(\omega) \leq \min_{i=1,\ldots,k} p_i(\omega).$$

For a distribution $\mathcal{D}$ on Σ^n and $Q \subseteq [n]$, we write $\mathcal{D}|_Q$ to denote the distribution created by projecting every element of Σ^n to its coordinates in Q. Let this time $\mathcal{D}_1, \ldots, \mathcal{D}_k$ be probability distributions on Σ^n. We say that they are *uniformly α-similar* if for every subset Q of $[n]$, the distributions $\mathcal{D}_1|_Q, \ldots, \mathcal{D}_k|_Q$ are $\alpha|Q|$-similar. Intuitively, think of Q as a sequence of queries that the algorithm makes. If the distributions are uniformly α-similar for a very small α, and $|Q| \ll 1/\alpha$, then from the limited point of view of the algorithm (even an adaptive one), the difference between the distributions is very small.

In order to use the notion of uniform similarity for our construction, we prove the following three key lemmas.

Uniform Similarity Implies a Lower Bound on the Number of Queries (Lemma 3.3.4). This lemma formalizes the ramifications of uniform α-similarity for a pair of distributions. It shows that if an algorithm (even an adaptive one) distinguishes the two distributions with probability at least 2/3, then it has to make at least $1/(6\alpha)$ queries. The lemma implies that it suffices to bound the uniform similarity in order to prove a lower bound on the number of queries.

The proof is based on the fact that for every setting of the algorithm's random bits, the algorithm can be described as a decision tree of depth q, if it always makes at most q queries. Then, for every leaf, the probability of reaching it does not differ by more than a factor in $[1 - \alpha q, 1]$ between the two distributions. This is enough to bound the probability the algorithm outputs the correct answer for both the distributions.

Random Cyclic Shifts of Random Strings Imply Uniform Similarity (Lemma 3.3.7). This lemma constructs block-distributions that are uniformly similar using cyclic shifts of random base strings. It shows that if one takes n random base strings in Σ^n and creates n distributions by shifting each of the strings by a random number of indices in $[1, s]$, then with probability at least 2/3 (over the choice of the base strings) the created distributions are uniformly $O(1/\log_{|\Sigma|} \frac{s}{\log n})$-similar.

It is easy to prove this lemma for any set Q of size 1. In this case, every shift gives an independent random bit, and the bound directly follows from the Chernoff

bound. A slight obstacle is posed by the fact that for $|Q| \geq 2$, sequences of $|Q|$ symbols produced by different shifts are not necessarily independent, since they can share some of the symbols. To address this issue, we show that there is a partition of shifts into at most $|Q|^2$ large sets such that no two shifts of Q in the same set overlap. Then we can apply the Chernoff bound independently to each of the sets to prove the bound.

In particular, using this and the previous lemmas, one can show the result claimed earlier that shifts of two random strings in $\{0,1\}^n$ by an offset in $[1, cn]$ produce distributions that require $\Omega(\log n)$ queries to be distinguished. It follows from the lemma that the distributions are likely to be uniformly $O(1/\log n)$-similar.

Substitution Product Amplifies Uniform Similarity (Lemma 3.3.8). Perhaps the most surprising property of uniform similarity is that it nicely composes with the substitution product. Let $\mathcal{D}_1, \ldots, \mathcal{D}_k$ be uniformly α-similar distributions on Σ^n. Let $\mathcal{E} = (\mathcal{E}_a)_{a\in\Sigma}$, where $\mathcal{E}_a$, $a \in \Sigma$, are uniformly β-similar distributions on $(\Sigma')^{n'}$. The lemma states that $\mathcal{D}_1 \circledast \mathcal{E}, \ldots, \mathcal{D}_k \circledast \mathcal{E}$ are uniformly $\alpha\beta$-similar.

The main idea behind the proof of the lemma is the following. Querying q locations in a string that comes from $\mathcal{D}_i \circledast \mathcal{E}$, we can see a difference between distributions in at most βq blocks in expectation. Seeing the difference is necessary to discover which $\mathcal{E}_j$ each of the blocks comes from. Then only these blocks can reveal the identity of $\mathcal{D}_i \circledast \mathcal{E}$, and the difference in the distribution if q' blocks are revealed is bounded by $\alpha q'$.

The lemma can be used to prove the earlier claim that the two-level construction produces distributions that require $\Omega(\log^2 n)$ queries to be told apart.

Preserving Edit Distance

It now remains to describe our tools for analyzing the edit distance between strings generated by our distributions. All of these tools are collected in Section 3.3.3. In most cases we focus in our analysis on $\underline{\text{ed}}$, which is the version of edit distance that only allows for insertions and deletions. It clearly holds that $\text{ed}(x,y) \leq \underline{\text{ed}}(x,y) \leq 2 \cdot \text{ed}(x,y)$, and this connection is tight enough for our purposes. An additional

advantage of $\underline{\mathrm{ed}}$ is that for any strings x and y, $2\,\mathrm{LCS}(x,y) + \underline{\mathrm{ed}}(x,y) = |x| + |y|$.

We start by reproducing a well known bound on the longest common substring of randomly selected strings (Lemma 3.3.9). It gives a lower bound on $\mathrm{LCS}(x,y)$ for two randomly chosen strings. The lower bound then implies that the distance between two strings chosen at random is large, especially for a large alphabet.

Theorem 3.3.10 shows how the edit distance between two strings in Σ^n changes when we substitute every symbol with a longer string using a function $B : \Sigma \to (\Sigma')^{n'}$. The relative edit distance (that is, edit distance divided by the length of the strings) shrinks by an additive term that polynomially depends on the maximum relative length of the longest common string between $B(a)$ and $B(b)$ for different a and b. It is worth to highlight the following two issues:

- We do not need a special version of this theorem for distributions. It suffices to first bound edit distance for the recursive construction when instead of strings shifted at random, we use strings themselves. Then it suffices to bound by how much the strings can change as a result of shifts (at all levels of the recursion) to obtain desired bounds.

- The relative distance shrinks relatively fast as a result of substitutions. This implies that we have to use an alphabet of size polynomial in the number of recursion levels. The alphabet never has to be larger than polylogarithmic, because the number of recursion levels is always $o(\log n)$.

Finally, Theorem 3.3.12 and Lemma 3.3.14 effectively reduce the alphabet size, because they show that a lower bound for the binary alphabet follows immediately from the one for a large alphabet, with only a constant factor loss in the edit distance. It turns out that it suffices to map every element of the large alphabet Σ to a random string of length $\Theta(\log |\Sigma|)$ over the binary alphabet.

The main idea behind proofs of the above is that strings constructed using a substitution product are composed of rather rigid blocks, in the sense that every alignment between two such strings, say $x \circledast \mathcal{E}$ and $y \circledast \mathcal{E}$, must respect (to a large extent) the block structure, in which case one can extract from it an alignment between the two

initial strings x and y.

3.2 Fast Algorithms via Asymmetric Query Complexity

In this section we describe our near-linear time algorithm for estimating the edit distance between two strings. As we mentioned in the introduction, the algorithm is obtained from an efficient query algorithm.

The main result of this section is the following query complexity upper bound theorem, which is a full version of Theorem 1.2.2. It implies our near-linear time algorithm for polylogarithmic approximation (Theorem 1.2.1).

Theorem 3.2.1. *Let $n \geq 2$, $\beta = \beta(n) \geq 2$, and integer $b = b(n) \geq 2$ be such that $(\log_b n) \in \mathbb{N}$.*

There is an algorithm solving DTEP_β *with approximation $\alpha = O(b \log_b n)$ and $\beta \cdot (\log n)^{O(\log_b n)}$ queries into x. The algorithm runs in $n \cdot (\log n)^{O(\log_b n)}$ time.*

For every constant $\beta = O(1)$ and integer $t \geq 2$, there is an algorithm for solving DTEP_β *with $O(n^{1/t})$ approximation and $O(\log n)^{t-1}$ queries. The algorithm runs in $\tilde{O}(n)$ time.*

In particular, note that we obtain Theorem 1.2.1 by setting $b = (\log n)^{c/\varepsilon}$ for a suitably high constant $c > 1$.

The proof is partitioned in three stages. (The first stage corresponds to the first "major component" mentioned in Introduction, and Section 3.1.1, and the next two stages correspond to the second "major component".) In the first stage, we describe a characterization of edit distance by a different quantity, namely $\mathcal{E}$-distance, which approximates edit distance well. The characterization is parametrized by an integer parameter $b \geq 2$. A small b leads to a small approximation factor (in fact, as small as $O(\log n)$ for $b = 2$), whereas a large b leads to a faster algorithm. In the second stage, we show how one can design a sampling algorithm that approximates $\mathcal{E}$-distance for some setting of the parameter b, up to a constant factor, by making a small number of

queries into x. In the third stage, we show how to use the query algorithm to obtain a near-linear time algorithm for edit distance approximation.

The three stages are described in the following three sections, and all together give the proof of Theorem 3.2.1.

3.2.1 Edit Distance Characterization: the $\mathcal{E}$-distance

Our characterization may be viewed as computation on a tree, where the nodes correspond to substrings $x[s : s + l]$, for some start position $s \in [n]$ and length[1] $l \in [n]$. The root is the entire string $x[1 : n + 1]$. For a node $x[s : s + l]$, the children are blocks $x[s + j \cdot l/b : s + (j + 1) \cdot l/b]$, where $j \in \{0, 1, \dots b - 1\}$, and b is the arity of the tree. The $\mathcal{E}$-distance for the node $x[s : s + l]$ is defined recursively as a function of the distances of its children. Note that the characterization is asymmetric in the two strings.

Before giving the definition we establish further notation. We fix the arity $b \geq 2$ of the tree, and let $h \stackrel{\text{def}}{=} \log_b n \in \mathbb{N}$ be the height of the tree. Fix some tree level i for $0 \leq i \leq h$. Consider some substring $x[s : s + l_i]$ at level i, where $l_i \stackrel{\text{def}}{=} n/b^i$. Let $B_i \stackrel{\text{def}}{=} \{1, l_i + 1, 2l_i + 1, \dots\}$ be the set of starting positions of blocks at level i.

Definition 3.2.2 ($\mathcal{E}$-distance). *Consider two strings x, y of length $n \geq 2$. Fix $i \in \{0, 1, \dots h\}$, $s \in B_i$, and a position $u \in \mathbb{Z}$.*

If $i = h$, then the $\mathcal{E}$-distance *of $x[s : s + l_i]$ to the position u is 1 if $u \notin [n]$ or $x[s] \neq y[u]$, and 0 otherwise.*

For $i \in \{0, 1, \dots h - 1\}$, we recursively define the $\mathcal{E}$-distance *$\mathcal{E}_{x,y}(i, s, u)$ of $x[s : s + l_i]$ to the position u as follows. Partition $x[s : s + l_i]$ into b blocks of length $l_{i+1} = l_i/b$, starting at positions $s + jl_{i+1}$, where $j \in \{0, 1, \dots b - 1\}$. Then*

$$\mathcal{E}_{x,y}(i, s, u) \stackrel{\text{def}}{=} \sum_{j=0}^{b-1} \min_{r_j \in \mathbb{Z}} \mathcal{E}_{x,y}(i + 1, s + jl_{i+1}, u + jl_{i+1} + r_j) + |r_j| .$$

[1] We remind that the notation $x[s : s + l]$ corresponds to characters $x[s], x[s + 1], \dots x[s + l - 1]$. More generally, $[s : s + l]$ stands for the interval $\{s, s + 1, \dots, s + l - 1\}$. This convention simplifies subsequent formulas.

The $\mathcal{E}$-distance from x to y is just the $\mathcal{E}$-distance from $x[1:n+1]$ to position 1, i.e., $\mathcal{E}_{x,y}(0,1,1)$.

We illustate the $\mathcal{E}$-distance for $b=4$ in Figure 3-1. Since x and y will be clear from the context, we will just use the notation $\mathcal{E}(i,s,u)$ without indices x and y.

The main property of the $\mathcal{E}$-distance is that it gives a good approximation to the edit distance between x and y, as quantified in the following theorem, which we prove below.

Theorem 3.2.3 (Characterization). *For evry $b \geq 2$ and two strings $x, y \in \Sigma^n$, the $\mathcal{E}$-distance between x and y is a $6 \cdot \frac{b}{\log b} \cdot \log n$ approximation to the edit distance between x and y.*

We also give an alternative, equivalent definition of the $\mathcal{E}$-distance between x and y. It is motivated by considering the matching (alignment) induced by the $\mathcal{E}$-distance when computing $\mathcal{E}(0,1,1)$. In particular, when computing $\mathcal{E}(0,1,1)$ recursively, we can consider all the "matching positions" (positions $u + jl_{i+1} + r_j$ for r_j's achieving the minimum). We denote by Z a vector of integers $z_{i,s}$, indexed by $i \in \{0, 1, \ldots h\}$ and $s \in B_i$, where $z_{0,1} = 1$ by convention. The coordinate $z_{i,s}$ should be understood as the position to which we match the substring $x[s : s + l_i]$ in the calculation of $\mathcal{E}(0,1,1)$. Then we define the cost of Z as

$$\text{cost}(Z) \stackrel{\text{def}}{=} \sum_{i=0}^{h-1} \sum_{s \in B_i} \sum_{j=0}^{b-1} |z_{i,s} + jl_{i+1} - z_{i+1,s+jl_{i+1}}|.$$

The cost of Z can be seen as the sum of the displacements $|r_j|$ that appear in the calculation of the $\mathcal{E}$-distance from Definition 3.2.2. The following claim asserts an alternative definition of the $\mathcal{E}$-distance.

Claim 3.2.4 (Alternative definition of $\mathcal{E}$-distance). *The $\mathcal{E}$-distance between x and y is the minimum of*

$$\text{cost}(Z) + \sum_{s \in [n]} \text{H}(x[s], y[z_{h,s}]) \tag{3.2}$$

over all choices of the vector $Z = (z_{i,s})_{i\in\{0,1,\dots h\},s\in B_i}$ *with* $z_{0,1} = 1$, *where* $\mathrm{H}(\cdot,\cdot)$ *is the Hamming distance, namely* $H(x[s], y[z_{h,s}])$ *is 1 if* $z_{h,s} \notin [n]$ *or* $x[s] \neq y[z_{h,s}]$, *and 0 otherwise.*

Proof. The quantity (3.2) simply unravels the recursive formula from Definition 3.2.2. The equivalence between them follows from the fact that $|z_{i,s} + jl_{i+1} - z_{i+1,s+jl_{i+1}}|$ directly corresponds to quantities $|r_j|$ in the $\mathcal{E}_{x,y}(i, s, z_{i,s})$ definition, which appear in the computation on the tree, and the $\sum_{s\in[n]} \mathrm{H}(x[s], y[z_{h,s}])$ term corresponds to the summation of $\mathcal{E}_{x,y}(h, s, z_{h,s})$ over all $s \in [n]$. □

We are now ready to prove Theorem 3.2.3.

Proof of Theorem 3.2.3. Fix $n, b \geq 2$ and let $h \stackrel{\text{def}}{=} \log_b n$. We break the proof into two parts, an upper bound and a lower bound on the $\mathcal{E}$-distance (in terms of edit distance). They are captured by the following two lemmas, which we shall prove shortly.

Lemma 3.2.5. *The* $\mathcal{E}$*-distance between* x *and* y *is at most* $3hb \cdot \mathrm{ed}(x, y)$.

Lemma 3.2.6. *The edit distance* $\mathrm{ed}(x, y)$ *is at most twice the* $\mathcal{E}$*-distance between* x *and* y.

Combining these two lemmas gives $\frac{1}{2}\,\mathrm{ed}(x, y) \leq \mathcal{E}_{x,y}(0, 1, 1) \leq 5hb \cdot \mathrm{ed}(x, y)$, which proves Theorem 3.2.3. □

We proceed to prove these two lemmas.

Proof of Lemma 3.2.5. Let $A : [n] \to [n] \cup \{\perp\}$ be an optimal alignment from x to y. Namely A is such that:

- If $A(s) \neq \perp$, then $x[s] = y[A(s)]$.
- If $A(s_1) \neq \perp$, $A(s_2) \neq \perp$, and $s_1 < s_2$, then $A(s_1) < A(s_2)$.
- $L \stackrel{\text{def}}{=} |A^{-1}(\perp)|$ is minimized.

Note that $n - L$ is the length of the Longest Common Subsequence (LCS) of x and y. It clearly holds that $\frac{1}{2}\operatorname{ed}(x, y) \leq L \leq \operatorname{ed}(x, y)$.

To show an upper bound on the $\mathcal{E}$-distance, we use the alternative characterization from Claim 3.2.4. Specifically, we show how to construct a vector Z proving that the $\mathcal{E}$-distance is small.

At each level $i \in \{1, 2, \dots h\}$, for each block $x[s : s + l_i]$ where $s \in B_i$, we set $z_{i,s} \stackrel{\text{def}}{=} A(j)$, where j is the smallest integer $j \in [s : s + l_i]$ such that $A(j) \neq \perp$ (i.e., to match a block we use the first in it that is aligned under the alignment A). If no such j exists, then $z_{i,s} \stackrel{\text{def}}{=} z_{i-1,s'} + (s - s')$, where $s' \stackrel{\text{def}}{=} l_{i-1} \cdot \lfloor (s-1)/l_{i-1} \rfloor + 1$, that is, s' is such that $x[s' : s' + l_{i-1}]$ is the parent of $x[s : s + l_i]$ in the tree.

Note that it follows from the definition of $z_{h,s}$ and L that $\sum_{s \in [n]} \mathrm{H}(x[s], y[z_{h,s}]) = L$. It remains to bound the other term $\operatorname{cost}(Z)$ in the alternative definition of $\mathcal{E}$-distance.

To accomplish this, for every $i \in \{0, 1, 2, \dots, h-1\}$ and $s \in B_i$, we define $d_{i,s}$ as the maximum of $|z_{i,s} + j l_{i+1} - z_{i+1, s + j l_{i+1}}|$ over $j \in \{0, \dots, b-1\}$. Although we cannot bound each $d_{i,s}$ separately, we bound the sum of $d_{i,s}$ for each level i.

Claim 3.2.7. *For each $i \in \{0, 1, \dots h\}$, we have that $\sum_{s \in B_i} d_{i,s} \leq 2L$.*

Proof. We shall prove that each $d_{i,s}$ is bounded by $X_{i,s} + Y_{i,s}$, where $X_{i,s}$ and $Y_{i,s}$ are essentially the number of unmatched positions in x and in y, respectively, that contribute to $d_{i,s}$. We then argue that both $\sum_{s \in B_i} X_{i,s}$ and $\sum_{s \in B_i} Y_{i,s}$ are bounded by L, thus completing the proof of the claim.

Formally, let $X_{i,s}$ be the number of positions $j \in [s : s + l_i]$ such that $A(j) = \perp$. If $X_{i,s} = l_i$, then clearly $d_{i,s} = 0$. It is also easily verified that if $X_{i,s} = l_i - 1$, then $d_{i,s} \leq l_i - 1$. In both cases, $d_{i,s} \leq X_{i,s}$, and we also set $Y_{i,s} \stackrel{\text{def}}{=} 0$.

If $X_{i,s} \leq l_i - 2$, let j' be the largest integer $j' \in [s : s + l_i]$ for which $A(j') \neq \perp$ (note that j' exists and it is different from the smallest such possible integer, which was called j when we defined $z_{i,s}$, because $X_{i,s} \leq l_i - 2$). In this case, let $Y_{i,s}$ be $A(j') - z_{i,s} + 1 - (l_i - X_{i,s})$, which is the number of positions in y between $z_{i,s}$ and $A(j')$ (inclusive) that are not aligned under A. Let $\Delta_{i,s,j} \stackrel{\text{def}}{=} z_{i,s} + j l_{i+1} - z_{i+1, s + j l_{i+1}}$ for

$j \in \{0, \ldots, b-1\}$. By definition, it holds $d_{i,s} = \max_j |\Delta_{i,s,j}|$. Now fix j. If $\Delta_{i,s,j} \neq 0$, then there is an index $k \in [s + jl_{i+1} : s + (j+1)l_{i+1}]$ such that $A(k) = z_{i+1,s+jl_{i+1}}$. If $\Delta_{i,s,j} > 0$ (which corresponds to a shift to the left), then at least $\Delta_{i,s,j}$ indices $j' \in [s : k]$ are such that $A(j') = \perp$, and therefore, $|\Delta_{i,s,j}| \leq X_{i,s}$. If $\Delta_{i,s,j} < 0$ (which corresponds to a shift to the right), then at least $|\Delta_{i,s,j}|$ positions in y between $z_{i,s}$ and $z_{i+1,s+jl_{i+1}}$ are not aligned in A. Thus, $|\Delta_{i,s,j}| \leq Y_{i,s}$.

In conclusion, for every $s \in B_i$, $d_{i,s} \leq X_{i,s} + Y_{i,s}$. Observe that $\sum_{s \in B_i} X_{i,s} = L$ and $\sum_{s \in B_i} Y_{i,s} \leq L$ (because they correspond to distinct positions in x and in y that are not aligned by A). Hence, we obtain that $\sum_{s \in B_i} d_{i,s} \leq \sum_{s \in B_i} X_{i,s} + Y_{i,s} \leq 2L$. ☐

We now claim that $\text{cost}(Z) \leq 2hbL$. Indeed, consider a block $x[s : s + l_i]$ for some $i \in \{0, 1, \ldots h-1\}$ and $s \in B_i$, and one of its children $x[s + jl_{i+1} : s + (j+1)l_{i+1}]$ for $j \in \{0, 1, \ldots b-1\}$. The contribution of this child to the sum $\text{cost}(Z)$ is $|z_{i,s} + jl_{i+1} - z_{i+1,s+jl_{i+1}}| \leq d_{i,s}$ by definition. Hence, using Claim 3.2.7, we conclude that

$$\text{cost}(Z) \leq \sum_{i=0}^{h-1} \sum_{s \in B_i} \sum_{j=0}^{b-1} d_{i,s} \leq \sum_{i=0}^{h-1} \sum_{s \in B_i} d_{i,s} \cdot b \leq h \cdot 2L \cdot b.$$

Finally, by Claim 3.2.4, we have that the $\mathcal{E}$-distance between x and y is at most $2hbL + L \leq 2hb \cdot \text{ed}(x, y) + \text{ed}(x, y) \leq 3hb \cdot \text{ed}(x, y)$. ☐

Proof of Lemma 3.2.6. We again use the alternative characterization given by Claim 3.2.4. Let Z be the vector obtaining the minimum of Equation (3.2). Define, for $i \in \{0, 1, \ldots h\}$ and $s \in B_i$,

$$\delta_{i,s} \stackrel{\text{def}}{=} \sum_{s' \in [s:s+l_i]} \text{H}(x[s'], y[z_{h,s'}]) + \sum_{i': i \leq i' < h} \quad \sum_{s' \in B_{i'} \cap [s:s+l_i]} \quad \sum_{j=0}^{b-1} \left| z_{i',s'} + jl_{i'+1} - z_{i'+1,s'+jl_{i'+1}} \right|.$$

Note that $\delta_{0,1}$ equals the $\mathcal{E}$-distance by Claim 3.2.4. Also, we have the following inductive equality for $i \in \{0, 1, \ldots h-1\}$ and $s \in B_i$:

$$\delta_{i,s} = \sum_{j=0}^{b-1} \left(\delta_{i+1,s+jl_{i+1}} + \left| z_{i,s} + jl_{i+1} - z_{i+1,s+jl_{i+1}} \right| \right). \tag{3.3}$$

We now prove inductively for $i \in \{0, 1, 2 \dots h\}$ that for each $s \in B_i$, the length of the LCS of $x[s : s + l_i]$ and $y[z_{i,s} : z_{i,s} + l_i]$ is at least $l_i - \delta_{i,s}$.

For the base case, when $i = h$, the inductive hypothesis is trivially true. If $x[s] = y[z_{i,s}]$, then the LCS is of length 1 and $\delta_{h,s} = 0$. If $x[s] \neq y[z_{i,s}]$, then the LCS is of length 0 and $\delta_{h,s} = 1$.

Now we prove the inductive hypothesis for $i \in \{0, 1, \dots h - 1\}$, assuming it holds for $i + 1$. Fix a string $x[s : s + l_i]$, and let $s_j = s + jl_{i+1}$ for $j \in \{0, 1, \dots b - 1\}$. By the inductive hypothesis, for each $j \in \{0, 1, \dots b - 1\}$, the length of the LCS between $x[s_j : s_j + l_{i+1}]$ and $y[z_{i+1,s_j} : z_{i+1,s_j} + l_{i+1}]$ is at least $l_{i+1} - \delta_{i+1,s_j}$. In this case, the substring in y starting at $z_{i,s} + jl_{i+1}$, namely $y[z_{i,s} + jl_{i+1} : z_{i,s} + (j + 1)l_{i+1}]$, has an LCS with $x[s_j : s_j + l_{i+1}]$ of length at least $l_{i+1} - \delta_{i+1,s_j} - |z_{i,s} + jl_{i+1} - z_{i+1,s_j}|$. Thus, by Equation (3.3), the LCS of $x[s : s + l_i]$ and $y[z_{i,s} : z_{i,s} + l_i]$ is of length at least

$$\sum_{j=0}^{b-1} \left(l_{i+1} - \delta_{i+1,s_j} - |z_{i,s} + jl_{i+1} - z_{i+1,s_j}|\right) = l_i - \delta_{i,s},$$

which finishes the proof of the inductive step.

For $i = 0$, this implies that $\mathrm{ed}(x, y) \leq 2\delta_{0,1} = 2\mathcal{E}_{x,y}(0, 1, 1)$. □

3.2.2 Sampling Algorithm

We now describe the sampling and estimation algorithms that are used to obtain our query complexity upper bounds. In particular, our algorithm approximates the $\mathcal{E}$-distance defined in the previous section. The guarantee of our algorithms is that the output $\hat{\mathcal{E}}$ satisfies $(1 - o(1))\mathcal{E}(0, 1, 1) - n/\beta \leq \hat{\mathcal{E}} \leq (1 + o(1))\mathcal{E}(0, 1, 1) + n/\beta$. This is clearly sufficient to distinguish between $\mathcal{E}(0, 1, 1) \leq n/\beta$ and $\mathcal{E}(0, 1, 1) \geq 4n/\beta$. After presenting the algorithm, we prove its correctness and prove that it only samples $\beta \cdot n^{O(\varepsilon)}$ positions of x in order to make the decision.

Algorithm Description

We now present our sampling algorithm, as well as the estimation algorithm, which given y and the sample of x, decides DTEP_β.

Sampling algorithm. To subsample x, we start by partitioning x recursively into blocks as defined in Definition 3.2.2. In particular, we fix a tree of arity b, indexed by pairs (i, s) for $i \in \{0, 1, \ldots h\}$, and $s \in B_i$. At each level $i = 0, \ldots h$, we have a subsampled set $C_i \subseteq B_i$ of vertices at that level of the tree. The set C_i is obtained from the previous one by extending C_{i-1} (considering all the children), and a careful subsampling procedure. In fact, for each element in C_i, we also assign a number $w \geq 1$, representing a "precision" and describing how well we want to estimate the $\mathcal{E}$ distance at that node, and hence governing the subsampling of the subtree rooted at the node.

Our sampling algorithm works as follows. We use a (continuous) distribution $\mathcal{W}$ on $[1, n^3]$, which we define later, in Lemma 3.2.12.

Algorithm 9: Sampling Algorithm

---

1. Take C_0 to be the root vertex (indexed $(i, s) = (0, 1)$), with precision $w_{(0,1)} = \beta$.
2. **for** *each level* $i = 1, \ldots, h$, *we construct* C_i *as follows* **do**
3. Start with C_i being empty.
4. **for** *each node* $v = (i-1, s) \in C_{i-1}$ **do**
5. Let w_v be its precision, and set $p_v = \frac{w_v}{b} \cdot O(\log^3 n)$.
6. If $p_v \geq 1$, then set $J_v = \{(i, s + jl_i) \mid 0 \leq j < b\}$ to be the set of all the b children of v, and add them to C_i, each with precision p_v.
7. Otherwise, when $p_v < 1$, sample each of the b children of v with probability p_v, to form a set $J_v \subseteq \{i\} \times ([s : s + l_{i-1}] \cap B_i)$. For each $v' \in J_v$, draw $w_{v'}$ i.i.d. from $\mathcal{W}$, and add node v' to C_i with precision $w_{v'}$.
8. Query the characters $x[s]$ for all $(h, s) \in C_h$ — this is the output of the algorithm.

Estimation Algorithm. We compute a value $\tau(v, z)$, for each node $v \in \cup_i C_i$ and position $z \in [n]$, such that $\tau(v, z)$ is a good approximation ($1 + o(1)$ factor) to the $\mathcal{E}$-distance of the node v to position z.

We also use a "reconstruction algorithm" R, defined in Lemma 3.2.12. It takes as input at most b quantities, their precision, and outputs a positive number.

Algorithm 10: Estimation Algorithm

1. For each sampled leaf $v = (h, s) \in C_h$ and $z \in [n]$ we set $\tau(v, z) = \mathrm{H}(x[s], y[z])$.
2. **for** *each level* $i = h-1, j-2, \ldots, 0$, *position* $z \in [n]$, *and node* $v \in C_i$ *with precision* w_v **do**
3. We apply the following procedure $P(v, z)$ to obtain $\tau(v, z)$.
4. For each $v' \in J_v$, where $v' = (i+1, s + jl_{i+1})$ for some $0 \leq j < b$, let
$$\delta_{v'} \stackrel{\text{def}}{=} \min_{k:|k|\leq n} \tau(v', z + jl_{i+1} + k) + |k|.$$
5. If $p_v \geq 1$, then let $\tau(v, z) = \sum_{v' \in J_v} \delta_{v'}$.
6. If $p_v < 1$, set $\tau(v, z)$ to be the output of the algorithm R on the vector $(\frac{\delta_{v'}}{l_{i+1}})_{v' \in J_v}$ with precisions $(w_{v'})_{v' \in J_v}$, multiplied by l_{i+1}/p_v.
7. The output of the algorithm is $\tau(r, 1)$ where $r = (0, 1)$ is the root of the tree.

Analysis Preliminaries: Approximators and a Concentration Bound

We use the following approximation notion that captures both an additive and a multiplicative error. For convenience, we work with factors e^{ε} instead of usual $1 + \varepsilon$.

Definition 3.2.8. *Fix $\rho > 0$ and some $f \in [1, 2]$. For a quantity $\tau \geq 0$, we call its* (ρ, f)-approximator *any quantity $\hat{\tau}$ such that $\tau/f - \rho \leq \hat{\tau} \leq f\tau + \rho$.*

It is immediate to note the following *additive property*: if $\hat{\tau}_1, \hat{\tau}_2$ are (ρ, f)-approximators to τ_1, τ_2 respectively, then $\hat{\tau}_1 + \hat{\tau}_2$ is a $(2\rho, f)$-approximator for $\tau_1 + \tau_2$. Also, there's a composion property: if $\hat{\tau}'$ is an (ρ', f')-approximator to $\hat{\tau}$, which itself is a (ρ, f)-approximator to τ, then $\hat{\tau}'$ is a $(\rho' + f'\rho, ff')$-aproximator to τ.

The definition is motivated by the following concentration statement on the sum of random variables. The statement is an immediate application of the standard Chernoff/Hoeffding bounds.

Lemma 3.2.9 (Sum of random variables)**.** *Fix $n \in \mathbb{N}$, $\rho > 0$, and error probability δ. Let $Z_i \in [0, \rho]$ be independent random variables, and let $\zeta > 0$ be a sufficiently*

large absolute constant. Then for every $\varepsilon \in (0,1)$, the summation $\sum_{i\in[n]} Z_i$ is a $(\zeta\rho\frac{\log 1/\delta}{\varepsilon^2}, e^{\varepsilon})$-approximator to $\mathbb{E}\left[\sum_{i\in[n]} Z_i\right]$, with probability $\geq 1-\delta$.

Proof of Lemma 3.2.9. By rescaling, it is sufficient to prove the claim for $\rho = 1$. Let $\mu = \mathbb{E}\left[\sum_{i\in[n]} Z_i\right]$. If $\mu > \frac{\zeta}{4} \cdot \frac{\log 1/\delta}{\varepsilon^2}$, then, a standard application of the Chernoff implies that $\sum_i Z_i$ is a e^{ε} approximation to μ, with $\geq 1-\delta$ probability, for some sufficiently high $\zeta > 0$.

Now assume that $\mu \leq \frac{\zeta}{4} \cdot \frac{\log 1/\delta}{\varepsilon^2}$. We use the following variant of the Hoeffding inequality, which can be derived from [42].

Lemma 3.2.10 (Hoeffding bound). *Let Z_i be n independent random variables such that $Z_i \in [0,1]$, and $\mathbb{E}\left[\sum_{i\in[n]} Z_i\right] = \mu$. Then, for any $t > 0$, we have that* $\Pr\left[\sum_i Z_i \geq t\right] \leq e^{-(t-2\mu)}$.

We apply the above lemma for $t = \zeta \cdot \frac{\log 1/\delta}{\varepsilon^2}$. We obtain that $\Pr[\sum_i Z_i \geq t] \leq e^{-t/2} = e^{-\Omega(\log 1/\delta)} < \delta$, which completes the proof that $\sum_i Z_i$ is a $(\zeta\frac{\log 1/\delta}{\varepsilon^2}, e^{\varepsilon})$-approximator to μ (when $\rho = 1$). □

Main Analysis Tools: Uniform and Non-uniform Sampling Lemmas

We present our two main subsampling lemmas that are applied, recursively, at each node of the tree. The first lemma, on Uniform Sampling, is a simple Chernoff bound in a suitable regime.

The second lemma, called Non-uniform Sampling Lemma, is the heart of our sampling, and is inspired by a sketching/streaming technique introduced in [47] for optimal estimation of F_k moments in a stream. Although a relative of their method, our lemma is different both in intended context and actual technique.We shall use the constant $\zeta > 0$ coming from Lemma 3.2.9.

Lemma 3.2.11 (Uniform Sampling). *Fix $b \in \mathbb{N}$, $\varepsilon > 0$, and error probability $\delta > 0$. Consider some a_j, $j \in [b]$, such that $a_j \in [0, 1/b]$. For arbitrary $w \in [1, \infty)$, construct the set $J \subseteq [b]$ by subsampling each $j \in [b]$ with probability $p_w = \min\{1, \frac{w}{b} \cdot \zeta\frac{\log 1/\delta}{\varepsilon^2}\}$. Then, with probability at least $1-\delta$, the value $\frac{1}{p_w}\sum_{j\in J} a_j$ is a $(1/w, e^{\varepsilon})$-approximator to $\sum_{j\in[b]} a_j$, and $|J| \leq O(w \cdot \frac{\log 1/\delta}{\varepsilon^2})$.*

Proof. If $p_w = 1$, then $J = [b]$ and there is nothing to prove; so assume that $p_w = \frac{w}{b} \cdot \zeta \frac{\log 1/\delta}{\varepsilon^2} < 1$ for the rest.

The bound on $|J|$ follows from a standard application of the Chernoff bound: $\mathbb{E}[|J|] = p_w b \le O(w \cdot \frac{\log 1/\delta}{\varepsilon^2})$, hence the probability that $|J|$ exceeds twice the quantity is at most $e^{-\Omega(\log 1/\delta)} \le \delta/2$.

We are going to apply Lemma 3.2.9 to the variables $Z_j = a_j/p_w \cdot \chi[j \in J]$, where the indicator variable $\chi[j \in J]$ is 1 iff $j \in J$. Note that $0 \le Z_j \le \frac{\varepsilon^2}{w \cdot \zeta \log 1/\delta}$. We thus obtain that $\sum_{j\in[b]} Z_j$ is a $(\frac{\zeta\varepsilon^{-2}\log 1/\delta}{w \cdot \zeta\varepsilon^{-2}\log 1/\delta}, e^{\varepsilon})$-approximator, and hence $(1/w, e^{\varepsilon})$-approximator, to $\mathbb{E}\left[\sum_j Z_j\right] = \sum_{j\in[b]} p_w \cdot \frac{a_j}{p_w} = \sum_{j\in[b]} a_j$. □

We now present and prove the Non-uniform Sampling Lemma.

Lemma 3.2.12 (Non-uniform Sampling). *Fix integers $n \le N$, approximation $\varepsilon > 0$, factor $1 < f < 1.1$, error probability $\delta > 0$, and an "additive error bound" $\rho > 6n/\varepsilon/N^3$. There exists a distribution $\mathcal{W}$ on the real interval $[1, N^3]$ with $\mathbb{E}_{w\in\mathcal{W}}[w] \le O(\frac{1}{\rho} \cdot \frac{\log 1/\delta}{\varepsilon^3} \cdot \log N)$, as well as a "reconstruction algorithm" R, with the following property.*

Take arbitrary $a_i \in [0,1]$, for $i \in [n]$, and let $\sigma = \sum_{i\in[n]} a_i$. Suppose one draws w_i i.i.d. from $\mathcal{W}$, for each $i \in [n]$, and let $\hat{a}_i$ be a $(1/w_i, f)$-approximator of a_i. Then, given $\hat{a}_i$ and w_i for all $i \in [n]$, the algorithm R generates a $(\rho, f \cdot e^{\varepsilon})$-approximator to σ, with probability at least $1 - \delta$.

For concreteness, we mention that $\mathcal{W}$ is the maximum of $O(\frac{1}{\rho} \cdot \frac{\log 1/\delta}{\varepsilon^3})$ copies of the (truncated) distribution $1/x^2$ (essentially equivalent to a distribution of x where the logarithm of x is distributed geometrically).

Proof. We start by describing the distribution $\mathcal{W}$ and the algorithm R. Fix $k = \frac{2\zeta}{\rho} \cdot \frac{\log 1/\delta}{(\varepsilon/2)^3}$. We first describe a related distribution: let $\mathcal{W}_1$ be distribution on x such that the pdf function is $p_1(x) = \nu/x^2$ for $1 \le x \le N^3$ and $p_1(x) = 0$ otherwise, where $\nu = (\int_1^\infty p_1(x)\,\mathrm{d}x)^{-1} = (1 - 1/N^3)^{-1}$ is a normalization constant. Then $\mathcal{W}$ is the distribution of x where we choose k i.i.d. variables $x_1, \ldots x_k$ from $\mathcal{W}_1$ and then set $x = \max_{i\in[k]} x_i$. Note that the pdf of $\mathcal{W}$ is $p(x) = \nu^k \frac{k}{x^2}(1 - 1/x)^{k-1}$.

The algorithm R works as follows. For each $i \in [n]$, we define k "indicators" $s_{i,j} \in \{0, 1/k\}$ for $j \in [k]$. Specifically, we generate the set of random variables $w_{i,j} \in \mathcal{W}_1$, $j \in [k]$, conditioned on the fact that $\max_{j\in[k]} w_{i,j} = w_i$. Then, for each $i \in [n], j \in [k]$, we set $s_{i,j} = 1/k$ if $\hat{a}_i \geq t/w_i$ for $t = 3/\varepsilon$, and $s_{i,j} = 0$ otherwise. Finally, we set $s = \sum_{i\in[n],j\in[k]} s_{i,j}$ and the algorithm outputs $\hat{\sigma} = st/\nu$ (as an estimate for σ).

We note that the variables $w_{i,j}$ could be thought as being chosen i.i.d. from $\mathcal{W}_1$. For each, the value $\hat{a}_i$ is an $(1/w_{i,j}, f)$-approximator to a_i since $\hat{a}_i$ is a $(1/\max_j w_{i,j}, f)$-approximator to a_i.

It is now easy to bound $\mathbb{E}_{w\in\mathcal{W}}[w]$. Indeed, we have $\mathbb{E}_{w\in\mathcal{W}_1}[w] = \int_1^{N^3} x \cdot \nu/x^2 \,\mathrm{dx} \leq O(\log N)$. Hence $\mathbb{E}_{w\in\mathcal{W}}[w] \leq \sum_{j\in[k]} \mathbb{E}_{w\in\mathcal{W}_1}[w] \leq O(k \log N) = O(\frac{1}{\rho} \cdot \frac{\log 1/\delta}{\varepsilon^3} \cdot \log N)$.

We now need to prove that $\hat{\sigma}$ is an approximator to σ, with probability at least $1 - \delta$. We first compute the expectation of $s_{i,j}$, for each $i \in [n], j \in [k]$. This expectation depends on the approximator values $\hat{a}_i$, which itself may depend on w_i. Hence we can only give upper and lower bounds on the expectation $\mathbb{E}[s_{i,j}]$. Later, we want to apply a concentration bound on the sum of $s_{i,j}$. Since $s_{i,j}$ may be interdependent, we will apply the concentration bound on the upper/lower bounds of $s_{i,j}$ to give bounds on $s = \sum s_{i,j}$.

Formally, we define random variables $\overline{s}_{i,j}, \underline{s}_{i,j} \in \{0, 1/k\}$. We set $\overline{s}_{i,j} = 1/k$ iff $w_{i,j} \geq (t-1)/(fa_i)$, and 0 otherwise. Similarly, we set $\underline{s}_{i,j} = 1/k$ iff $w_{i,j} < f(t+1)/a_i$, and 0 otherwise. We now claim that

$$\underline{s}_{i,j} \leq s_{i,j} \leq \overline{s}_{i,j}. \tag{3.4}$$

Indeed, if $s_{i,j} = 1/k$, then $\hat{a}_i \geq t/w_{i,j}$, and hence, using the fact that $\hat{a}_i$ is a $(1/w_{i,j}, f)$-approximator to a_i, we have $w_{i,j} \geq (t-1)/(fa_i)$, or $\overline{s}_{i,j} = 1/k$. Similarly, if $s_{i,j} = 0$, then $\hat{a}_i < t/w_{i,j}$, and hence $w_{i,j} < f(t+1)/a_i$, or $\underline{s}_{i,j} = 0$. Note that each collection $\{\overline{s}_{i,j}\}$ and $\{\underline{s}_{i,j}\}$ is a collection of independent random variables.

We now bound $\mathbb{E}[\overline{s}_{i,j}]$ and $\mathbb{E}[\underline{s}_{i,j}]$. For the first quantity, we have:

$$\mathbb{E}[\overline{s}_{i,j}] = \int_{(t-1)/(fa_i)}^{N^3} \tfrac{1}{k} p_1(x)\,\mathrm{dx} \leq \tfrac{fa_i}{k(t-1)} \int_1^{\infty} \nu/x^2\,\mathrm{dx} = \nu/k \cdot \tfrac{fa_i}{t-1}.$$

For the second quantity, we have:

$$\mathbb{E}[\underline{s}_{i,j}] = \int_{f(t+1)/a_i}^{N^3} p_1(x)\,\mathrm{dx} = \nu/k \cdot (\tfrac{a_i/f}{t+1} - 1/N^3).$$

Finally, using Eqn. (3.4) and the fact that $\mathbb{E}[s] = \sum_{i,j} \mathbb{E}[s_{i,j}]$, we can bound $\mathbb{E}[\hat{\sigma}] = \mathbb{E}[st/\nu]$ as follows:

$$\tfrac{t}{f(t+1)} \sum_{i\in[n]} a_i - nt/N^3 \leq \tfrac{t}{\nu} \sum_{i,j} \mathbb{E}[\underline{s}_{i,j}] \leq \mathbb{E}[ts/\nu] \leq \tfrac{t}{\nu} \sum_{i,j} \mathbb{E}[\overline{s}_{i,j}] \leq f \sum_{i\in[n]} a_i \cdot \tfrac{t}{t-1}.$$

Since each $\overline{s}_{i,j}, \underline{s}_{i,j} \in [0, 1/k]$ for $k = O(\frac{t}{\rho} \cdot \frac{\log 1/\delta}{\varepsilon^2})$, we can apply Lemma 3.2.9 to obtain a high concentration bound. For the upper bound, we obtain, with probability at least $1 - \delta/2$:

$$ts/\nu \leq e^{\varepsilon/2} \cdot \mathbb{E}\left[t/\nu \cdot \sum_{i,j} s_{i,j}\right] + \rho \leq e^{\varepsilon/2} \cdot f \sum a_i \cdot \tfrac{t}{t-1} + \rho \leq e^{\varepsilon} \cdot f \cdot \sigma + \rho.$$

Similarly, for the lower bound, we obtain, with probability at least $1 - \delta/2$:

$$ts/\nu \geq e^{-\varepsilon/2} \cdot (\sum a_i \cdot \tfrac{t}{f(t+1)} - nt/N^3) - \rho/2 \geq e^{-\varepsilon}/f \cdot \sigma - \rho,$$

using that $\rho/2 \geq nt/N^3$. This completes the proof that $\hat{\sigma}$ is a $(\rho, f \cdot e^{\varepsilon})$-approximator to σ, with probability at least $1 - \delta$. □

Correctness and Sample Bound for the Main Algorithm

Now, we prove the correctness of the algorithms 9, 10 and bound its query complexity. We note that we use Lemmas 3.2.11 and 3.2.12 with $\delta = 1/n^3$, $\varepsilon = 1/\log n$, and $N = n$ (which in particular, completely determine the distribution $\mathcal{W}$ and algorithm R used

in the algorithms 9 and 10).

Lemma 3.2.13 (Correctness). *For $b = \omega(1)$, the output of the Algorithm 10 (Estimation), is a $(n/\beta, 1+o(1))$-approximator to the $\mathcal{E}$-distance from x to y, w.h.p.*

Proof. From a high level view, we prove inductively from $i = 0$ to $i = h$ that expanding/subsampling the current C_i gives a good approximator, namely a $e^{O((h-i)/\log n)}$ factor approximation, with probability at least $1 - i/n^{\Omega(1)}$. Specifically, at each step of the induction, we expand and subsample each node from the current C_i to form the set C_{i+1} and use Lemmas 3.2.11 and 3.2.12 to show that we don't loose on the approximation factor by more than $e^{O(1/\log n)}$.

In order to state our main inductive hypothesis, we define a hybrid distance, where the $\mathcal{E}$-distance of nodes at high levels (big i) is computed standardly (via Definition 3.2.2), and the $\mathcal{E}$-distance of the low-level nodes is estimated via sets C_i. Specifically, for fixed $f \in [1, 1.1]$, and $i \in \{0, 1, \ldots h\}$, we define the following $(C_0, C_1 \ldots C_i, f)$-*$\mathcal{E}$-distance.* For each vertex $v = (i, s)$ such that $v \in C_i$ has precision w_v, and $z \in [n]$, let $\tau_i(v, z)$ to be some $(l_i/w_v, f)$–approximator to the distance $\mathcal{E}(i, s, z)$. Then, iteratively for $i' = i-1, i-2, \ldots, 0$, for all $v \in C_{i'}$ and $z \in [d]$, we compute $\tau_i(v, z)$ by applying the procedure $P(v, z)$ (defined in the Algorithm 10), using τ_i instead of τ.

We prove the following inductive hypothesis, for some suitable constants $t = 2$ and $r = \Theta(1)$ (sufficiently high r suffices).

IH$_i$: For any $f \in [1, 1.1]$, the $(C_0, C_1, \ldots C_i, f)$–$\mathcal{E}$–distance is a $(n/\beta, f \cdot e^{i \cdot t/\log n})$-approximator to the $\mathcal{E}$–distance from x to y, with probability at least $1 - i \cdot e^{-r \log n}$.

Base case is $i = 0$, namely that (C_0, f)-$\mathcal{E}$-distance is a $(n/\beta, f)$-approximator to the $\mathcal{E}$-distance between x and y. This case follows immediately from the definition of the (C_0, f)-$\mathcal{E}$-distance and the initialization step of the Sampling Algorithm.

Now we prove the inductive hypothesis IH$_{i+1}$, assuming IH$_i$ holds for some given $i \in \{0, 1, \ldots h-1\}$. We remind that we defined the quantity $\tau_{i+1}(v, z)$, for all $v \in C_{i+1} \subseteq \overline{C}_i$, where $\overline{C}_i = \{(i+1, s + jl_{i+1}) \mid (i, s) \in C_i, j \in \{0, \ldots b-1\}\}$ and $z \in [n]$,

to be a $(l_{i+1}/w_v, f)$–approximator of the corresponding $\mathcal{E}$-distance, namely $\mathcal{E}(v,z)$. The plan is to prove that, for all $v \in C_i$ with precision w_v, the quantity $\tau_{i+1}(v,z)$ is a $(l_i/w_v, f \cdot e^{2/\log n})$–approximator to $\mathcal{E}(v,z)$ with good probability — which we do in the claim below. Then, by definition of τ_i and IH_i, this implies that $\tau_{i+1}((0,1),1)$ is equal to the $(C_0, \ldots C_i, f \cdot e^{2/\log n} \cdot e^{i \cdot t/\log n})$–$\mathcal{E}$–distance, and hence is a $(n/\beta, f \cdot e^{(2+it)/\log n})$–approximator to the $\mathcal{E}$–distance from x to y. This will complete the proof of IH_{i+1}. We now prove the main technical step of the above plan.

Claim 3.2.14. *Fix $v \in C_i$ with precision $w \stackrel{\text{def}}{=} w_v$, where $v = (i,s)$, and some $z \in [n]$. For $j \in \{0, \ldots b-1\}$, let v_j be the jth child of v; i.e., $v_j = (i+1, s+jl_{i+1})$. For $v_j \in C_{i+1}$ with precision $w_j \stackrel{\text{def}}{=} w_{v_j}$, and $z' \in [n]$, let $\tau_{i+1}(v_j, z')$ be a $(l_{i+1}/w_j, f)$–approximator to $\mathcal{E}(v_j, z')$.*

Apply procedure $P(v,z)$ using $\tau_{i+1}(v_j,z')$ estimates, and let δ be the output. Then δ is a $(l_i/w, fe^{2/\log n})$–approximator to $\mathcal{E}(v,z)$, with probability at least $1 - e^{-\Omega(\log n)}$.

Proof. For each $v_j \in J_v$, where J_v is as defined in Algorithm 9, we define the following quantities:

$$\delta_{v_j} \stackrel{\text{def}}{=} \min_{k:|k|\le n} \mathcal{E}(v_j, z+jl_{i+1}+k) + |k| \qquad \hat{\delta}_{v_j} \stackrel{\text{def}}{=} \min_{k:|k|\le n} \tau_{i+1}(v_j, z+jl_{i+1}+k) + |k|.$$

It is immediate to see that $\hat{\delta}_{v_j}$ is a $(l_{i+1}/w_j, f)$–approximator to δ_{v_j} by the definition of τ_{i+1}.

If $p_v \ge 1$, then we have that $w_j = \frac{w}{b} \cdot O(\log^3 n)$ for all $v_j \in J_v$. Then, by the additive property of $(l_{i+1}/w_j, f)$–approximators, $\delta = \sum_{v_j \in J_v} \hat{\delta}_{v_j}$ is a $(l_i/w, f)$–approximator to $\sum_{v_j \in J_v} \delta_{v_j} = \mathcal{E}(v,z)$.

Now suppose $p_v < 1$. Then, by Lemma 3.2.11, $\delta' = \frac{1}{p_v} \sum_{v_j \in J_v} \delta_{v_j}$ is a $(l_i/2w, e^{1/\log n})$–approximator to $\sum_{j=0}^{b-1} \delta_{v_j} = \mathcal{E}(v,z)$, with high probability. Furthermore, by Lemma 3.2.12 for $\rho = 1$, since $w_j \in \mathcal{W}$ are i.i.d. and $\frac{\hat{\delta}_{v_j}}{l_{i+1}}$ are each an $(1/w_j, f)$–approximator to $\frac{\delta_{v_j}}{l_{i+1}}$ respectively, then R outputs a value δ'' that is a $(1, f \cdot e^{1/\log n})$–approximator to $\sum_{v_j \in J_v} \frac{\delta_{v_j}}{l_{i+1}} = \frac{p_v}{l_{i+1}} \delta'$. In other words, $\delta = \frac{l_{i+1}}{p_v} \delta''$ is a $(l_{i+1}/p_v, f \cdot e^{1/\log n})$–approximator to δ'. Since $l_{i+1}/p_v \le l_i/(3w)$, combining the two approximator guarantees, we obtain that δ is a $(l_i/w, f \cdot e^{2/\log n})$–approximator to $\mathcal{E}(v,z)$, w.h.p. ☐

We now apply a union bound over all $v \in C_i$ and $z \in [n]$, and use the above Claim 3.2.14. We now apply IH_i to deduce that $\tau_{i+1}((0,1),1)$ is a $(n/\beta, f \cdot e^{ti/\log n} \cdot e^{2/\log n})$–approximator with probability at least

$$1 - ie^{-r\log n} - e^{-\Omega(\log n)} \geq 1 - (i+1)e^{-r\log n},$$

for some suitable $r = \Theta(1)$. This proves IH_{i+1}.

Finally we note that IH_h implies that $(C_0, \ldots C_h, f)$–$\mathcal{E}$–distance is a $(n/\beta, f \cdot e^{th/\log n})$–approximator to the $\mathcal{E}$–distance between x and y. We conclude the lemma with the observation that our Estimation Algorithm 10 outputs precisely the $(C_0, \ldots C_h, 1)$–$\mathcal{E}$–distance. □

It remains to bound the number of positions that Algorithm 10 queries into x.

Lemma 3.2.15 (Sample size). *The Sampling Algorithm queries $Q_b = \beta(\log n)^{O(\log_b n)}$ positions of x, with probability at least $1 - o(1)$. When $b = n^{1/t}$ for fixed constant $t \in \mathbb{N}$ and $\beta = O(1)$, we have $Q_b = (\log n)^{t-1}$ with probability at least 2/3.*

Proof. We prove by induction, from $i = 0$ to $i = h$, that $\mathbb{E}\left[|C_i|\right] \leq \beta \cdot (\log n)^{ic}$, and $\mathbb{E}\left[\sum_{v \in C_i} w_v\right] \leq \beta \cdot (\log n)^{ic+5}$ for a suitable $c = \Theta(1)$. The base case of $i = 0$ is immediate by the initialization of the Sampling Algorithm 9. Now we prove the inductive step for i, assuming the inductive hypothesis for $i - 1$. By Lemma 3.2.11, $\mathbb{E}\left[|C_i|\right] \leq \mathbb{E}\left[\sum_{v \in C_{i-1}} w_v\right] \cdot O(\log^3 n) \leq \beta(\log n)^{ic}$ by the inductive hypothesis. Also, by Lemma 3.2.12, $\mathbb{E}\left[\sum_{v \in C_i} w_v\right] \leq \mathbb{E}\left[|C_i|\right] \cdot O(\log^4 n) + \mathbb{E}\left[\sum_{v \in C_{i-1}} w_v\right] \cdot O(\log^3 n) \leq \beta(\log n)^{ic+5}$. The bound then follows from an application of the Markov bound.

The second bound follows from a more careful use of the parameters of the two sampling lemmas, Lemmas 3.2.11 and 3.2.12. In fact, it suffices to apply these lemmas with $\varepsilon = e^{\Theta(1/t)}$ and $\delta = 0.1$ for the first level and $\delta = 1/n^3$ for subsequent levels. □

These lemmas, 3.2.13 and 3.2.15, together with the characterization theorem 3.2.3, almost complete the proof of Theorem 3.2.1. It remains to bound the run time of the resulting estimation algorithm, which we do in the next section.

3.2.3 Near-Linear Time Algorithm

We now discuss the time complexity of the algorithm, and show that the Algorithm 10 (Estimation) may be implemented in $n \cdot (\log n)^{O(h)}$ time. We note that as currently described in Algorithm 10, our reconstruction technique takes time $\tilde{O}(hQ_b \cdot n)$ time, where $Q_b = \beta(\log n)^{O(\log_b n)}$ is the sample complexity upper bound from Lemma 3.2.15 (note that, combined with the algorithm of [54], this already gives a $n^{4/3+o(1)}$ time algorithm). The main issue is the computation of the quantities $\delta_{v'}$, as, naively, it requires to iterate over all $k \in [n]$.

To reduce the time complexity of the Algorithm 10, we define the following quantity, which replaces the quantity $\delta_{v'}$ in the description of the algorithm:

$$\delta'_{v'} = \min_{k=e^{i/\log n}:i\in[\log n\cdot\ln(3n/\beta)]} \left(|k| + \min_{k':|k'|\le k} \tau(v', z + jl_{i+1} + k') \right).$$

Lemma 3.2.16. *If we use $\delta'_{v'}$ instead of $\delta_{v'}$ in Algorithm 10, the new algorithm outputs at most a $1 + o(1)$ factor higher value than the original algorithm.*

Proof. First we note that it is sufficient to consider only $k \in [-3n/\beta, 3n/\beta]$, since, if the algorithm uses some k with $|k| > 3n/\beta$, then the resulting output is guaranteed to be $> 3n/\beta$. Also, the estimate may only increase if one restricts the set of possible k's.

Second, if we consider k's that are integer powers of $e^{1/\log n}$, we increase the estimate by only a factor $e^{1/\log n}$. Over $h = O(\log_b n)$ levels, this factor accumulates to only $e^{h/\log n} \le 1 + o(1)$. □

Finally, we mention that computing all $\delta'_{v'}$ may be performed in $O(\log^2 n)$ time after we perform the following (standard) precomputation on the values $\tau(v', z')$ for $z' \in [n]$ and $v' \in C_{i+1}$. For each dyadic interval I, compute $\min_{z\in I} \tau(v, z)$. Then, for each (not necessarily dyadic) interval $I' \subset [n]$, computing $\min_{z'\in I'} \tau(v', z')$ may be done in $O(\log n)$ time. Hence, since we consider only $O(\log n)$ values of k, we obtain $O(\log^2 n)$ time per computation of $\delta'_{v'}$.

Total running time becomes $O(hQ_b \cdot n \cdot \log^2 n) = n \cdot (\log n)^{O(\log_b n)}$.

A more technical issue that we swept under the carpet is that distribution $\mathcal{W}$ defined in Lemma 3.2.12 is a continuous distribution on $[1, n^3]$. However this is not an issue since a $n^{-\Omega(1)}$ discretization suffices to obtain the same result, with only $O(\log n)$ loss in time complexity.

3.3 Query Complexity Lower Bound

We now give a full proof of our lower bound, Theorem 1.2.3. After some preliminaries, this section contains three rather technical parts: tools for analyzing indistinguishability, tools for analyzing edit distance behavior, and a finally a part where we put together all elements of the proof. The precise and most general forms of our lower bound appear in that final part as Theorem 3.3.15 and Theorem 3.3.16.

3.3.1 Preliminaries

We assume throughout that $|\Sigma| \geq 2$. Let x and y be two strings. Define $\underline{\mathrm{ed}}(x, y)$ to be the minimum number of character insertions and deletions needed to transform x into y. Character substitution are not allowed, in contrast to $\mathrm{ed}(x, y)$, but a substitution can be simulated by a deletion followed by an insertion, and thus $\mathrm{ed}(x, y) \leq \underline{\mathrm{ed}}(x, y) \leq 2\,\mathrm{ed}(x, y)$. Observe that

$$\underline{\mathrm{ed}}(x, y) = |x| + |y| - 2\,\mathrm{LCS}(x, y), \tag{3.5}$$

where $\mathrm{LCS}(x, y)$ is the length of the longest common subsequence of x and y.

Alignments. For two strings x, y of length n, an *alignment* is a function $A : [n] \to [n] \cup \{\perp\}$ that is monotonically increasing on $A^{-1}([n])$ and satisfies $x[i] = y[A(i)]$ for all $i \in A^{-1}([n])$. Observe that an alignment between x and y corresponds exactly to a common subsequence to x and y.

Projection. For a string $x \in \Sigma^n$ and $Q \subseteq [n]$, we write $x|_Q$ for the string that is the projection of x on the coordinates in Q. Clearly, $x|_Q \in \Sigma^{|Q|}$. Similarly, if $\mathcal{D}$ is

a probability distribution over strings in Σ^n, we write $\mathcal{D}|_Q$ for the distribution that is the projection of $\mathcal{D}$ on the coordinates in Q. Clearly, $\mathcal{D}|_Q$ is a distribution over strings in $\Sigma^{|Q|}$.

Substitution Product. Suppose that we have a "mother" string $x \in \Sigma^n$ and a mapping $B : \Sigma \to (\Sigma')^{n'}$ of the original alphabet into strings of length n' over a new alphabet Σ'. Define the *substitution product* of x and B, denoted $x \circledast B$, to be the concatenation of $B(x_1), \cdots, B(x_n)$. Letting $B_a = B(a)$ for each $a \in \Sigma$ (i.e. B defines a collection of $|\Sigma|$ strings), we have

$$x \circledast B \stackrel{\text{def}}{=} B_{x_1} B_{x_2} \cdots B_{x_n} \in (\Sigma')^{nn'}.$$

Similarly, for each $a \in \Sigma$, let $\mathcal{D}_a$ be a probability distribution over strings in $(\Sigma')^{n'}$. The *substitution product* of x and $\mathcal{D} \stackrel{\text{def}}{=} (\mathcal{D}_a)_{a \in \Sigma}$, denoted $x \circledast \mathcal{D}$, is defined as the probability distribution over strings in $(\Sigma')^{nn'}$ produced by replacing every symbol x_i, $1 \leq i \leq n$, in x by an independent sample B_i from $\mathcal{D}_{x_i}$.

Finally, let $\mathcal{E}$ be a "mother" probability distribution over strings in Σ^n, and for each $a \in \Sigma$, let $\mathcal{D}_a$ be a probability distribution over strings in $(\Sigma')^{n'}$. The *substitution product* of $\mathcal{E}$ and $\mathcal{D} \stackrel{\text{def}}{=} (\mathcal{D}_a)_{a \in \Sigma}$, denoted $\mathcal{E} \circledast \mathcal{D}$, is defined as the probability distribution over strings in $(\Sigma')^{nn'}$ produced as follows: first sample a string $x \sim \mathcal{E}$, then independently for each $i \in [n]$ sample $B_i \sim \mathcal{D}_{x_i}$, and report the concatenation $B_1 B_2 \dots B_n$.

Shift. For $x \in \Sigma^n$ and integer r, let $S^r(x)$ denote a cyclic shift of x (i.e. rotating x) to the left by r positions. Clearly, $S^r(x) \in \Sigma^n$. Similarly, let $\mathcal{S}_s(x)$ the distribution over strings in Σ^n produced by rotating x by a random offset in $[s]$, i.e. choose $r \in [s]$ uniformly at random and take $S^r(x)$.

For integers i, j, define $i +_n j$ to be the unique $z \in [n]$ such that $z = i + j \pmod{n}$. For a set Q of integers, let $Q +_n j = \{i +_n j : i \in Q\}$.

Fact 3.3.1. *Let $x \in \Sigma^n$ and $Q \subset [n]$. For every integer r, we have $S^r(x)|_Q = x|_{Q +_n r}$.*

Thus, for every integer s, the probability distribution $\mathcal{S}_s(x)|_Q$ is identical to $x|_{Q+_n r}$ for a random $r \in [s]$.

3.3.2 Tools for Analyzing Indistinguishability

In this section, we introduce tools for analyzing indistinguishability of distributions we construct. We introduce a notion of uniform similarity, show what it implies for query complexity, give quantitative bounds on it for random cyclic shifts of random strings, and show how it composes under the substitution product.

Similarity of Distributions

We first define an auxiliary notion of similarity. Informally, a set of distributions on the same set are similar if the probability of every element in their support is the same up to a small multiplicative factor.

Definition 3.3.2. *Let $\mathcal{D}_1, \ldots, \mathcal{D}_k$ be probability distributions on a finite set Ω. Let $p_i : \Omega \to [0,1]$, $1 \le i \le k$, be the probability mass function for $\mathcal{D}_i$. We say that the distributions are* α-similar *if for every $\omega \in \Omega$,*

$$(1-\alpha) \cdot \max_{i=1,\ldots,k} p_i(\omega) \le \min_{i=1,\ldots,k} p_i(\omega).$$

We now define uniform similarity for distributions on strings. Uniform similarity captures how the similarity between distributions on strings changes as a function of the number of queries.

Definition 3.3.3. *Let $\mathcal{D}_1, \ldots, \mathcal{D}_k$ be probability distributions on Σ^n. We say that they are* uniformly α-similar *if for every subset Q of $[n]$, the distributions $\mathcal{D}_1|_Q, \ldots, \mathcal{D}_k|_Q$ are $\alpha|Q|$-similar.*

Finally, we show that if two distributions on strings are uniformly similar, then an algorithm distinguishing strings drawn from them has to make many queries.

Lemma 3.3.4. *Let $\mathcal{D}_0$ and $\mathcal{D}_1$ be uniformly μ-similar distributions on Σ^n. Let $\mathcal{A}$ be a randomized algorithm that makes q (adaptive) queries to symbols of a string selected*

according to either $\mathcal{D}_0$ or $\mathcal{D}_1$, and outputs either 0 *or* 1*. Let p_j, for $j \in \{0, 1\}$, be the probability that $\mathcal{A}$ outputs* j *when the input is selected according to $\mathcal{D}_j$. Then*

$$\min\{p_0, p_1\} \leq \frac{1 + \mu q}{2}.$$

Proof. Once the random bits of $\mathcal{A}$ are fixed, $\mathcal{A}$ can be seen as a decision tree with depth q the following properties. Every internal node corresponds to a query to a specific position in the input string. Every internal node has $|\Sigma|$ children, and the $|\Sigma|$ edges outgoing to the children are labelled with distinct symbols from Σ. Each leaf is labelled with either 0 or 1; this is the algorithm's output, i.e. the computation ends up in a leaf if and only if the sequence of queries on the path from the root to the leaf gives the sequence described by the edge labels on the path.

Fix for now $\mathcal{A}$'s random bits. Let t be the probability that $\mathcal{A}$ outputs 0 when the input is chosen from $\mathcal{D}_0$, and let t' be defined similarly for $\mathcal{D}_1$. We now show an upper bound on $t - t'$. t is the probability that the computation ends up in a leaf v labelled 0 for an input chosen according to $\mathcal{D}_0$. Consider a specific leaf v labelled with 0. The probability of ending up in the leaf equals the probability of obtaining a specific sequence of symbols for a specific sequence of at most q queries. Let t_v be this probability when the input is selected according to $\mathcal{D}_0$. The same probability for $\mathcal{D}_1$ must be at least $(1 - q\mu)t_v$, due to the uniform μ-similarity of the distributions. By summing over all leaves v labelled with 0, we have $t' \geq (1 - \mu q)t$, and therefore, $t - t' \leq q\mu \cdot t \leq q\mu$.

Note that p_0 is the expectation of t over the choice of $\mathcal{A}$'s random bits. Analogously, $1 - p_1$ is the expectation of t'. Since $t - t'$ is always at most μq, we have $p_0 - (1 - p_1) \leq \mu q$. This implies that $p_0 + p_1 \leq 1 + \mu q$, and $\min\{p_0, p_1\} \leq \frac{1+\mu q}{2}$. □

Random Shifts

In this section, we give quantitative bounds on uniform similarity between distributions created by random cyclic shifts of random strings.

Making a query into a cyclic shift of a string is equivalent to querying the original

string in a position that is shifted, and thus, it is important to understanding what happens to a fixed set of q queries that undergoes different shifts. Our first lemma shows that a sufficiently large set of shifts of q queries can be partitioned into at most q^2 large sets, such that no two shifts in the same set intersect (in the sense that they query the same position).

Lemma 3.3.5. *Let Q be a subset of $[n]$ of size q, and let $Q_i \stackrel{\text{def}}{=} Q +_n i$ be its shift by i modulo n. Every $\mathcal{I} \subset [n]$ of size $t \geq 16q^4 \ln q$ admits a q^2-coloring $C : \mathcal{I} \to [q^2]$ with the following two properties:*

- *For all $i \neq j$ with $Q_i \cap Q_j \neq \emptyset$, we have $C(i) \neq C(j)$.*
- *For all $i \in [n]$, we have $|C^{-1}(i)| \geq n/(2q^4)$.*

Proof. Let $x \in [n]$. There are exactly q different indices i such that $x \in Q_i$. For every Q_i such that $x \in Q_i$, x is an image of a different $y \in Q$ after a cyclic shift. Therefore, each Q_i can intersect with at most $q(q-1)$ other sets Q_j.

Consider the following probabilistic construction of C. For consecutive $i \in \mathcal{I}$, we set $C(i)$ to be a random color in $[q^2]$ among those that were not yet assigned to sets Q_j that intersect Q_i. Each color $c \in [q^2]$ is considered at least t/q^2 times: each time c is selected it makes c not be considered for at most $q(q-1)$ other $i \in \mathcal{I}$. Each time c is considered, it is selected with probability at least $1/q^2$. By the Chernoff bound, the probability that a given color is selected less than $t/(2q^4)$ times is less than

$$\exp\left(-\frac{t}{q^4} \cdot \frac{1}{2^2} \cdot \frac{1}{2}\right) \leq \frac{1}{q^2}.$$

By the union bound, the probability of selecting the required coloring is greater than zero, so it exists. □

Fact 3.3.6. *Let n and k be integers such that $1 \leq k \leq n$. Then $\sum_{i=1}^{k} \binom{n}{i} \leq n^k$.*

The following lemma shows that random shifts of random strings are likely to result in uniformly similar distributions.

Lemma 3.3.7. *Let $n \in \mathbb{Z}_+$ be greater than 1. Let $k \le n$ be a positive integer. Let x_i, $1 \le i \le k$, be uniformly and independently selected strings in Σ^n, where $2 \le |\Sigma| \le n$. With probability $2/3$ over the selection of x_i's, the distributions $\mathcal{S}_s(x_1)$, ..., $\mathcal{S}_s(x_k)$ are uniformly $\frac{1}{A}$-similar, for $A \stackrel{\text{def}}{=} \max\left\{\log_{|\Sigma|} \sqrt[6]{\frac{s}{400 \ln n}}, 1\right\}$.*

Proof. Let $p_{i,Q,\omega}$ be the probability of selecting a sequence $\omega \in \Sigma^{|Q|}$ from the distribution $S_s(x_i)|_Q$, where $Q \subseteq [n]$ and $1 \le i \le k$. We have to prove that with probability at least $2/3$ over the choice of x_i's, it holds that for every $Q \subseteq Q$ and every $\omega \in \Sigma^{|Q|}$,

$$(1 - |Q|/A) \cdot \max_{i=1,\dots,k} p_{i,Q,\omega} \le \min_{i=1,\dots,k} p_{i,Q,\omega}.$$

The above inequality always holds when Q is empty or has at least A elements. Let $Q \subseteq [n]$ be any choice of queries, where $0 < |Q| < A$. By Fact 3.3.6, there are at most n^A such different choices of queries. Let $q \stackrel{\text{def}}{=} |Q|$. Note that $8q^4 \ln q \le 8q^5 \le 8A^5 \le 8|\Sigma|^{5A} \le 8 \cdot \frac{s}{400 \ln n} \le s$. This implies that we can apply Lemma 3.3.5, which yields the following. We can partition all s shifts of Q over x_i that contribute to the distribution $S_s(x_i)|_Q$ into q^2 sets σ_j such that the shifts in each of the sets are disjoint, and each of the sets has size at least $s/(2q^4)$. For each of the sets σ_j, and for each $\omega \in \Sigma^q$, the probability that fewer than $(1 - \frac{q}{2A})|\sigma_j|/|\Sigma|^q$ shifts give ω is bounded by

$$\begin{aligned}
\exp\left(-\frac{1}{2} \cdot \left(\frac{q}{2A}\right)^2 \cdot \frac{|\sigma_j|}{|\Sigma|^q}\right) &\le \exp\left(-\frac{s}{16q^2A^2|\Sigma|^q}\right) \\
&\le \exp\left(-\frac{s}{16A^4|\Sigma|^A}\right) \\
&\le \exp\left(-\frac{s}{16|\Sigma|^{5A}}\right) \\
&\le \exp\left(-\frac{1}{16} \cdot \sqrt[6]{s \cdot (400 \ln n)^5}\right) \\
&\le \exp\left(-9.2\sqrt[6]{s(\ln n)^5}\right),
\end{aligned}$$

where the first bound follows from the Chernoff bound. Analogously, the probability that more than $(1 + \frac{q}{2A})|\sigma_j|/|\Sigma|^q$ shifts give ω is bounded by

$$
\begin{aligned}
\exp\left(-\frac{1}{4}\cdot\left(\frac{q}{2A}\right)^2\cdot\frac{|\sigma_j|}{|\Sigma|^q}\right) &\leq \exp\left(-\frac{s}{32q^2A^2|\Sigma|^q}\right)\\
&\leq \exp\left(-\frac{s}{32A^4|\Sigma|^A}\right)\\
&\leq \exp\left(-\frac{s}{32|\Sigma|^{5A}}\right)\\
&\leq \exp\left(-\frac{1}{32}\cdot\sqrt[6]{s\cdot(400\ln n)^5}\right)\\
&\leq \exp\left(-4.6\sqrt[6]{s(\ln n)^5}\right),
\end{aligned}
$$

where the first inequality follows from the version of the Chernoff bound that uses the fact that $\frac{q}{2A}\leq\frac{1}{2}\leq 2e-1$.

We now apply the union bound to all x_i, all choices of $Q\subseteq[n]$ with $|Q|<A$, all corresponding sets σ_j, and all settings of $\omega\in\Sigma^{|Q|}$ to bound the probability that $p_{i,Q,\omega}$ does not lie between $|\Sigma|^{-|Q|}\cdot(1-\frac{q}{2A})$ and $|\Sigma|^{-|Q|}\cdot(1+\frac{q}{2A})$. Assuming that $A>1$ (otherwise, the lemma holds trivially), note first that

$$
\begin{aligned}
n\cdot n^A\cdot A^2\cdot|\Sigma|^A &\leq n^{5A}\\
&\leq \exp(5A\ln n)\\
&\leq \exp(5|\Sigma|^A\ln n)\\
&\leq \exp\left(5\sqrt[6]{\frac{s(\ln n)^5}{400}}\right)\\
&\leq \exp\left(2.4\cdot\sqrt[6]{s(\ln n)^5}\right).
\end{aligned}
$$

Our bound is

$$
\exp\left(2.4\cdot\sqrt[6]{s(\ln n)^5}\right)\cdot\left(\exp\left(-9.2\cdot\sqrt[6]{s(\ln n)^5}\right)+\exp\left(-4.6\cdot\sqrt[6]{s(\ln n)^5}\right)\right)
$$

$$
\leq \exp\left(-6.8\cdot\sqrt[6]{s(\ln n)^5}\right)+\exp\left(-2.2\cdot\sqrt[6]{s(\ln n)^5}\right)\leq 0.01+0.2\leq 1/3.
$$

Therefore, all $p_{i,Q,\omega}$ of interest lie in the desired range with probability at least 2/3.

Then, we know that for any Q of size less than A, and any $\omega \in \Sigma^{|Q|}$, we have

$$\begin{aligned}\left(1 - \frac{|Q|}{A}\right) \cdot \max_{i=1,\ldots,k} p_{i,Q,\omega} &\leq \cdot \left(1 - \frac{|Q|}{A}\right) \cdot \left(1 + \frac{|Q|}{2A}\right) \cdot |\Sigma|^{-|Q|} \\ &= \left(1 - \frac{|Q|}{2A} - \frac{|Q|^2}{2A^2}\right) \cdot |\Sigma|^{-|Q|} \\ &\leq \left(1 - \frac{|Q|}{2A}\right) \cdot |\Sigma|^{-|Q|} \\ &\leq \min_{i=1,\ldots,k} p_{i,Q,\omega}.\end{aligned}$$

This implies that $\mathcal{S}_s(x_1), \ldots, \mathcal{S}_s(x_k)$ are uniformly $\frac{1}{A}$-similar with probability at least 2/3. □

Amplification of Uniform Similarity via Substitution Product

One of the key parts of our proof is the following lemma that shows that the substitution product of uniformly similar distributions amplifies uniform similarity.

Lemma 3.3.8. *Let $\mathcal{D}_a$ for $a \in \Sigma$, be uniformly α-similar distributions on $(\Sigma')^{n'}$. Let $\mathcal{D} \stackrel{\text{def}}{=} (\mathcal{D}_a)_{a\in\Sigma}$. Let $\mathcal{E}_1, \ldots, \mathcal{E}_k$ be uniformly β-similar probability distributions on Σ^n, for some $\beta \in [0, 1]$. Then the k distributions $(\mathcal{E}_1 \circledast \mathcal{D}), \ldots, (\mathcal{E}_k \circledast \mathcal{D})$ are uniformly $\alpha\beta$-similar.*

Proof. Fix $t, t' \in [k]$, let X be a random sequence selected according to $\mathcal{E}_t \circledast \mathcal{D}$, and let Y be a random sequence selected according to $\mathcal{E}_{t'} \circledast \mathcal{D}$. Fix a set $S \subseteq [n \cdot n']$ of indices, and the corresponding sequence s of $|S|$ symbols from Σ'. To prove the lemma, it suffices to show that

$$\Pr[X|_S = s] \geq (1 - \alpha\beta|S|) \cdot \Pr[Y|_S = s], \tag{3.6}$$

since in particular the inequality holds for t that minimizes $\Pr[X|_S = s]$, and for t' that maximizes $\Pr[Y|_S = s]$.

Recall that each $(\mathcal{E}_j \circledast \mathcal{D})$ is generated by first selecting a string x according to $\mathcal{E}_j$, and then concatenating n blocks, where the i-th block is independently selected

from $\mathcal{D}_{x_i}$. For $i \in [n]$ and $b \in \Sigma$, let $p_{i,b}$ be the probability of drawing from $\mathcal{D}_b$ a sequence that when used as the i-th block, matches s on the indices in S (if the block is not queried, set $p_{i,b} = 1$). Let q_i be the number of indices in S that belong to the i-th block. Since $\mathcal{D}_b$ for $b \in \Sigma$ are α-similar, for every $i \in [n]$, it holds that $(1 - \alpha q_i) \cdot \max_{b\in\Sigma} p_{i,b} \leq \min_{b\in\Sigma} p_{i,b}$. For every $i \in [n]$, define $\alpha_i^\star \stackrel{\text{def}}{=} \min_{b\in\Sigma} p_{i,b}$ and $\beta_i^\star \stackrel{\text{def}}{=} \max_{b\in\Sigma} p_{i,b}$. We thus have

$$(1 - \alpha q_i)\beta_i^\star \leq \alpha_i^\star. \tag{3.7}$$

The following process outputs 1 with probability $\Pr[Y|_S = s]$. Whenever we say that the process outputs a value, 0 or 1, it also terminates. First, for every block $i \in [n]$, the process independently picks a random real r_i in $[0, 1]$. It also independently draws a random sequence $c \in \Sigma^n$ according to $\mathcal{E}_{t'}$. If $r_i > \beta_i^\star$ for at least one i, the process outputs 0. Otherwise, let $Q = \{i \in [n] : \; r_i > \alpha_i^\star\}$. If $r_i \leq p_{i,c_i}$ for all $i \in Q$, the process outputs 1. Otherwise, it outputs 0. The correspondence between the probability of outputting 1 and $\Pr[Y|_S = s]$ directly follows from the fact that each of the random variables r_i simulates selecting a sequence that matches s on indices in S with the right success probability, i.e., p_{i,c_i}, and the fact that block substitutions are independent. The important difference, which we exploit later, is that not all symbols of c have always impact on whether the above process outputs 0 or 1.

For every $Q \subseteq [n]$, let p'_Q be the probability that the above process selected Q. Furthermore, let $p''_{Q,c}$ be the conditional probability of outputting 1, given that the process selected a given $Q \subseteq [n]$, and a given $c \in \Sigma^n$. It holds

$$\Pr\left[Y|_S = s\right] = \sum_{Q\subseteq[n]} p'_Q \cdot \mathbb{E}_{c\leftarrow\mathcal{E}_{t'}}\left[p''_{Q,c}\right].$$

Notice that for two different $c_1, c_2 \in \Sigma^n$, we have $p''_{Q,c_1} = p''_{Q,c_1}$ if $c_1|_Q = c_2|_Q$, since this probability only depends on the symbols at indices in Q. Thus, for $\tilde{c} \in \Sigma^{|Q|}$ we

can define $\tilde{p}_{Q,\tilde{c}}$ to be equal to $p''_{Q,c}$ for any $c \in \Sigma$ such that $c|_Q = \tilde{c}$. We can now write

$$\Pr\left[Y|_S = s\right] = \sum_{Q \subseteq [n]} p'_Q \cdot \mathbb{E}_{\tilde{c} \leftarrow \mathcal{E}_{t'}|_Q}\left[\tilde{p}_{Q,\tilde{c}}\right],$$

and analogously,

$$\Pr\left[X|_S = s\right] = \sum_{Q \subseteq [n]} p'_Q \cdot \mathbb{E}_{\tilde{c} \leftarrow \mathcal{E}_{t}|_Q}\left[\tilde{p}_{Q,\tilde{c}}\right].$$

Due to the uniform β-similarity of $\mathcal{E}_{t'}$ and $\mathcal{E}_t$, we know that for every $Q \subset [n]$, the probability of selecting each $\tilde{c} \in \Sigma^{|Q|}$ from $\mathcal{E}_t|_Q$ is at least $(1 - \beta|Q|)$ times the probability of selecting the same $\tilde{c}$ from $\mathcal{E}_{t'}|_Q$. This implies that

$$\mathbb{E}_{\tilde{c} \leftarrow \mathcal{E}_{t}|_Q}\left[\tilde{p}_{Q,\tilde{c}}\right] \geq (1 - \beta|Q|) \cdot \mathbb{E}_{\tilde{c} \leftarrow \mathcal{E}_{t'}|_Q}\left[\tilde{p}_{Q,\tilde{c}}\right].$$

We obtain

$$\begin{aligned}
\Pr\left[Y|_S = s\right] - \Pr\left[X|_S = s\right] &= \sum_{Q \subseteq [n]} p'_Q \cdot \left(\mathbb{E}_{\tilde{c} \leftarrow \mathcal{E}_{t'}|_Q}\left[\tilde{p}_{Q,\tilde{c}}\right] - \mathbb{E}_{\tilde{c} \leftarrow \mathcal{E}_{t}|_Q}\left[\tilde{p}_{Q,\tilde{c}}\right]\right). \\
&\leq \sum_{Q \subseteq [n]} p'_Q \cdot \beta|Q| \cdot \mathbb{E}_{\tilde{c} \leftarrow \mathcal{E}_{t'}|_Q}\left[\tilde{p}_{Q,\tilde{c}}\right] \\
&= \beta \cdot \sum_{Q \subseteq [n]} p'_Q \cdot |Q| \cdot \mathbb{E}_{c \leftarrow \mathcal{E}_{t'}}\left[p''_{Q,c}\right] \\
&= \beta \cdot \mathbb{E}_{c \leftarrow \mathcal{E}_{t'}}\left[\sum_{Q \subseteq [n]} p'_Q \cdot p''_{Q,c} \cdot |Q|\right]. \qquad (3.8)
\end{aligned}$$

Fix now any $c \in \Sigma^n$ for which the process outputs 1 with positive probability. The expected size of Q for the fixed c, given that the process outputs 1, can be written as

$$\mathbb{E}\left[|Q| \;\middle|\; \text{process outputs } 1\right] = \frac{\sum_{Q \subseteq [n]} p'_Q \cdot p''_{Q,c} \cdot |Q|}{\sum_{Q \subseteq [n]} p'_Q \cdot p''_{Q,c}}$$

The probability that a given $i \in [n]$ belongs to Q for the fixed c, given that the process outputs 1 equals $\frac{p_{i,c_i} - \alpha_i^\star}{p_{i,c_i}}$. This follows from the two facts (a) if the process outputs 1 then r_i is uniformly distributed on $[0, p_{i,c_i}]$; and (b) $i \in Q$ if and only if $r_i \in (\alpha_i^\star, \beta_i^\star]$.

We have

$$\frac{p_{i,c_i} - \alpha_i^\star}{p_{i,c_i}} \leq \frac{\beta_i^\star - \alpha_i^\star}{\beta_i^\star} \leq \frac{\alpha q_i \cdot \beta_i^\star}{\beta_i^\star} = \alpha q_i. \tag{3.9}$$

By the linearity of expectation, the expected size of Q in this setting is at most $\sum_{i\in[n]} \alpha q_i = \alpha \cdot |S|$. Therefore,

$$\sum_{Q\subseteq[n]} p'_Q \cdot p''_{Q,c} \cdot |Q| \leq \alpha \cdot |S| \cdot \sum_{Q\subseteq[n]} p'_Q \cdot p''_{Q,c}. \tag{3.10}$$

Note that the inequality trivially holds also for c for which the process always outputs 0; both sides of the inequality equal 0.

By plugging (3.10) into (3.8), we obtain

$$\begin{aligned}
\Pr[Y|_S = s] - \Pr[X|_S = s] &\leq \beta \cdot \mathbb{E}_{c\leftarrow\mathcal{E}_{t'}}\left[\alpha \cdot |S| \cdot \sum_{Q\subseteq[n]} p'_Q \cdot p''_{Q,c}\right] \\
&= \alpha\beta \cdot |S| \cdot \mathbb{E}_{c\leftarrow\mathcal{E}_{t'}}\left[\sum_{Q\subseteq[n]} p'_Q \cdot p''_{Q,c}\right] \\
&= \alpha\beta \cdot |S| \cdot \Pr[Y|_S = s].
\end{aligned}$$

This proves (3.6) and completes the proof of the lemma. □

3.3.3 Tools for Analyzing Edit Distance

This section provides tools to analyze how the edit distance changes under a under substitution product. We present two separate results with different guarantees, one is more useful for a large alphabet, the other for a small alphabet. The latter is used in the final step of reduction to binary alphabet.

Distance between random strings

The next bound is well-known, see also [23, 14, 61]. We reproduce it here for completeness.

Lemma 3.3.9. *Let $x, y \in \Sigma^n$ be chosen uniformly at random. Then*

$$\Pr\left[\mathrm{LCS}(x,y) \geq 5n/\sqrt{|\Sigma|}\right] \leq e^{-5n/\sqrt{|\Sigma|}}.$$

Proof. Let $c \stackrel{\text{def}}{=} 5 > e^{1.5}$ and $t \stackrel{\text{def}}{=} cn/\sqrt{|\Sigma|}$. The number of potential alignments of size t between two strings of length n is at most $\binom{n}{t}^2 \leq (\frac{ne}{t})^{2t}$. Each of them indeed becomes an alignment of x, y (i.e. symbols that are supposed to align are equal) with probability at most $1/|\Sigma|^t$. Applying a union bound,

$$\Pr[\mathrm{LCS}(x,y) \geq t] \leq (\tfrac{ne}{t})^{2t}/|\Sigma|^t \leq (e^2 c^{-2}|\Sigma|)^t \cdot |\Sigma|^{-t} \leq e^{-t}. \qquad \square$$

Distance under substitution product (large alphabet)

We proceed to analyze how the edit distance between two strings, say $\mathrm{ed}(x,y)$, changes when we perform a substitution product, i.e. $\mathrm{ed}(x \circledast B, y \circledast B)$. The bounds we obtain are additive, and are thus most effective when the edit distance $\mathrm{ed}(x,y)$ is large (linear in the strings length). Furthermore, they depend on $\lambda_B \in [0,1]$, which denotes the maximum normalized LCS between distinct images of $B : \Sigma \to (\Sigma')^{n'}$, hence they are most effective when λ_B is small, essentially requiring a large alphabet Σ'.

Theorem 3.3.10. *Let $x, y \in \Sigma^n$ and $B : \Sigma \to (\Sigma')^{n'}$. Then*

$$n' \cdot \underline{\mathrm{ed}}(x,y) - 8nn'\sqrt{\lambda_B} \;\leq\; \underline{\mathrm{ed}}(x \circledast B, y \circledast B) \;\leq\; n' \cdot \underline{\mathrm{ed}}(x,y),$$

where $\lambda_B \stackrel{\text{def}}{=} \max\left\{\frac{\mathrm{LCS}(B(a),B(b))}{n'} : a \neq b \in \Sigma\right\}$.

Before proving the theorem, we state a corollary that will turn to be most useful. The corollary follows from Theorem 3.3.10 by letting $\Sigma' = \Sigma$, and using Lemma 3.3.9 together with a union bound over all pairs $B(a), B(b)$ (while assuming $n' \geq |\Sigma|$).

Corollary 3.3.11. *Assume $|\Sigma| \geq 2$ and $n' \geq |\Sigma|$ is sufficiently large (i.e. at least some absolute constant c'). Let $B : \Sigma \to (\Sigma)^{n'}$ be a random function, i.e. for each $a \in \Sigma$ choose $B(a)$ uniformly at random. Then with probability at least $1 - 2^{-n'/|\Sigma|}$,*

for all n and all $x, y \in \Sigma^n$,

$$0 \;\le\; n' \cdot \underline{\text{ed}}(x, y) - \underline{\text{ed}}(x \circledast B, y \circledast B) \;\le\; O(nn'/|\Sigma|^{1/4}).$$

Proof of Theorem 3.3.10. By using the direct connection (3.5) between $\underline{\text{ed}}(x, y)$ and $\text{LCS}(x, y)$, it clearly suffices to prove

$$n' \cdot \text{LCS}(x, y) \le \text{LCS}(x \circledast B, y \circledast B) \le n' \cdot \text{LCS}(x, y) + 4nn'\sqrt{\lambda_B}. \tag{3.11}$$

Throughout, we assume the natural partitioning of x, y into n blocks of length n'.

The first inequality above is immediate. Indeed, give an (optimal) alignment between x and y, do the following; for each (i, j) such that x_i is aligned with y_j, align the entire i-th block in $x \circledast B$ with the entire j-th block in $y \circledast B$. It is easily verified that the result is indeed an alignment and has size $n' \cdot \underline{\text{ed}}(x, y)$.

To prove the second inequality above, fix an optimal alignment A between $x \circledast B$ and $y \circledast B$; we shall construct an $\hat{A}$ alignment for x, y in three stages, namely, first pruning A into A', then pruning it further into A'', and finally constructing $\hat{A}$. Define the *span* of a block b in either $x \circledast B$ or $y \circledast B$ (under the current alignment) to be the number of blocks in the other string to which it is aligned in at least one position (e.g. the span of block i in $x \circledast B$ is the number of blocks j for which at least one position p in block i satisfies that $A(p)$ is in block j.)

Now iterate the following step: "unalign" a block (in either $x \circledast B$ or $y \circledast B$) completely whenever its span is greater than $s \stackrel{\text{def}}{=} 2/\sqrt{\lambda_B}$. Let A' be the resulting alignment; its size is $|A'| \ge |A| - 4nn'/s$ because each iteration is triggered by a distinct block, the total span of all these blocks is at most $4n$, hence the total number of iterations is at most $4n/s$.

Next, iterate the following step (starting with A' as the current alignment): remove alignments between two blocks (one in $x \circledast B$ and one in $y \circledast B$) if, in one of the two blocks, at most $\lambda_B n'$ positions are aligned to the other block. Let A'' be the resulting alignment; its size is $|A''| \ge |A'| - ns \cdot \lambda_B n'$ because each iteration is triggered by a distinct pair of blocks, out of at most ns pairs (by the span bound above).

This alignment A'' has size $|A''| \geq |A| - 4nn'/s - nn's\lambda_B$. Furthermore, if between two blocks, say block i in $x \circledast B$ and block j in $y \circledast B$, the number of aligned positions is at least one, then this number is actually greater than $\lambda_B n'$ (by construction of A'') and thus $x[i] = y[j]$ (by definition of $\lambda_B n'$).

Finally, construct an alignment $\hat{A}$ between x and y, where initially, $\hat{A}(i) = \perp$ for all $i \in [n]$. Think of the alignment A'' as the set of aligned positions, namely $\{(p,q) \in [n] \times [n] : A''(p) = q\}$. Let $\mathrm{blk}_{x \circledast B}(p)$ denote the number of the block in $x \circledast B$ which contains p, and similarly for positions q in $y \circledast B$. Now scan A'', as a set of pairs, in lexicographic order. More specifically, initialize (p,q) to be the first edge in A'', and iterate the following step: assign $\hat{A}(\mathrm{blk}_{x \circledast B}(p)) = \mathrm{blk}_{y \circledast B}(q)$, and advance (p,q) according to the lexicographic order so that both coordinates now belong to new blocks, i.e. set it to be the next pair $(p', q') \in A''$ for which both $\mathrm{blk}_{x \circledast B}(p') > \mathrm{blk}_{x \circledast B}(p)$ and $\mathrm{blk}_{y \circledast B}(q') > \mathrm{blk}_{y \circledast B}(q)$. We claim that $\hat{A}$ is an alignment between x and y. To see this, consider the moment when we assign some $\hat{A}(i) = j$. Then the corresponding blocks in $x \circledast B$ and $y \circledast B$ contain at least one pair of positions that are aligned under A'', and thus, as argued above, $x[i] = y[j]$. In addition, all subsequent assignments of the form $\hat{A}(i') = j'$ satisfy that both $i' > i$ and $j' > j$. Hence $\hat{A}$ is indeed an alignment.

En route to bounding the size of $\hat{A}$, we claim that each iteration scans (i.e. advances the current pair by) at most n' pairs from A''. To see this, consider an iteration where we assign some $\hat{A}(i) = j$. Every pair $(p,q) \in A''$ that is scanned in this iteration satisfies that either $i = \mathrm{blk}_{x \circledast B}(p)$ or $j = \mathrm{blk}_{x \circledast B}(p)$. Each of these two requirements can be satisfied by at most n' pairs, and together at most $2n'$ pairs are scanned. By the fact that A'' is monotone, it can be easily verified that at least one of the two requirements must be satisfied by all scanned pairs, hence the total number of scanned pairs is at most n'.

Using the claim, we get that $|\hat{A}| \leq |A''|/n'$ (recall that each iteration also makes one assignment to $\hat{A}$). It immediately follows that

$$n' \cdot \mathrm{LCS}(x,y) \geq n' \cdot |\hat{A}| \geq |A''| \geq |A| - 4nn'/s - nn's\lambda_B = \mathrm{LCS}(x \circledast B, y \circledast B) - 4nn'\sqrt{\lambda_B},$$

which completes the proof of (3.11) and of Theorem 3.3.10. ☐

Distance under substitution product (any alphabet)

We give another analysis for how the edit distance between two strings, say $\mathrm{ed}(x, y)$, changes when we perform a substitution product, i.e. $\mathrm{ed}(x \circledast B, y \circledast B)$. The bounds we obtain here are multiplicative, and may be used as a final step of alphabet reduction (say, from a large alphabet to the binary one).

Theorem 3.3.12. *Let $B : \Sigma \to (\Sigma')^{n'}$, and suppose that (i) for every $a \neq b \in \Sigma$, we have*

$$\mathrm{LCS}(B_a, B_b) \leq \tfrac{15}{16} n';$$

and (ii) for every $a, b, c \in \Sigma$ (possibly equal), and every substring B' of (the concatenation) $B_b B_c$ that has length n' and overlaps each of B_b and B_c by at least $n'/10$, we have

$$\mathrm{LCS}(B_a, B') \leq 0.98 n'.$$

Then for all $x, y \in \Sigma^n$,

$$c_1 n' \cdot \underline{\mathrm{ed}}(x, y) \leq \underline{\mathrm{ed}}(x \circledast B, y \circledast B) \leq n' \cdot \underline{\mathrm{ed}}(x, y), \tag{3.12}$$

where $0 < c_1 < 1$ is an absolute constant.

Before proving the theorem, let us show that it is applicable for a random mapping B, by proving two extensions of Lemma 3.3.9. Unlike the latter, the lemmas below are effective also for small alphabet size.

Lemma 3.3.13. *Suppose $|\Sigma| \geq 2$ and let $x, y \in \Sigma^n$ be chosen uniformly at random. Then with probability at least $1 - |\Sigma|^{-l/8}$, the following holds: for every substring x' in x of length $l \geq 24$, and every length l substring y' in B_b, we have*

$$\mathrm{LCS}(x', y') \leq \tfrac{15}{16} l.$$

Proof. Set $\alpha \stackrel{\text{def}}{=} 1/16$. Fix l and the positions of x' inside x and of y' inside y. Then x' and y' are chosen at random from Σ^l, hence

$$\Pr[\mathrm{LCS}(x',y') \geq (1-\alpha)l] \leq \binom{l}{(1-\alpha)l}^2 |\Sigma|^{-(1-\alpha)l} \leq (\tfrac{e}{\alpha})^{2\alpha l}|\Sigma|^{-(1-\alpha)l} \leq |\Sigma|^{-l/4},$$

where the last inequality uses $|\Sigma| \geq 2$.

Now apply a union bound over all possible positions of x' and y' and all values of l. It follows that the probability that x and y contain length l substrings x' and y' (respectively) with $\mathrm{LCS}(x',y') \geq (1-\alpha)l$ is at most $|\Sigma|^3 \cdot |\Sigma|^{-l/4} \leq |\Sigma'|^{-l/8}$, if only l is sufficiently large. □

The next lemma is an easy consequence of Lemma 3.3.13. It follows by applying a union bound and observing that disjoint substrings of $B(a)$ are independent.

Lemma 3.3.14. *Let $B : \Sigma \to (\Sigma')^{n'}$ be chosen uniformly at random for $|\Sigma'| \geq 2$ and $n' \geq 1000 \log |\Sigma|$. Then with probability at least $1 - |\Sigma'|^{-\Omega(n')}$, B satisfies the properties (i) and (ii) described in Theorem 3.3.12.*

Proof of Theorem 3.3.12. The last inequality in (3.12) is straightforward. Indeed, whenever x_i is aligned against y_j, we have $x_i = y_j$ and $B(x_i) = B(y_j)$, hence we can align the corresponding blocks in $x \circledast B$ and $y \circledast B$. We immediately get that $\mathrm{LCS}(x \circledast B, y \circledast B) \geq n' \cdot \mathrm{LCS}(x,y)$.

Let us now prove the first inequality. Denote $R \stackrel{\text{def}}{=} \underline{\mathrm{ed}}(x \circledast B, y \circledast B)$, and fix a corresponding alignment between the two strings. The string $x \circledast B$ is naturally partitioned into n blocks of length n'. The total number of coordinates in $x \circledast B$ that are unaligned (to $y \circledast B$) is exactly $R/2$, which is $R/2n$ in an average block.

We now prune this alignment in two steps. First, "unaliagn" each block in $x \circledast B$ with at least $(nn'/100R) \cdot (R/2n) = n'/200$ unaligned coordinates. By averaging (or Markov's inequality), this step applies to at most $100R/nn'$-fraction of the n blocks.

Next, define the *gap* of a block in $x \circledast B$ to be the difference (in the positions) between the first and last positions in $y \circledast B$ that are aligned against a coordinate in $x \circledast B$. The second pruning step is to unalign every block in $x \circledast B$ whose gap is at

least $1.01n'$. Every such block can be identified with a set of at least $n'/100$ unaligned positions in $y \circledast B$ (sandwiched inside the gap), hence these sets (for different blocks) are all disjoint, and the number of such blocks is at most $(R/2)/(n'/100) = 50R/n'$.

Now consider one of the remaining blocks (at least $n - 100R/n' - 50R/n'$ blocks). By our pruning, for each such block i we can find a corresponding substring of length n' in $y \circledast B$ with at least $n' - n'/200 - n'/100 > 0.98n'$ aligned pairs (between these two substrings). Using the property (ii) of B, the corresponding substring in $y \circledast B$ must have overlap of at least $0.9n'$ with some block of $y \circledast B$ (recall that $y \circledast B$ is also naturally partitioned into length n' blocks). Thus, for each such block i in $x \circledast B$ there is a corresponding block j in $y \circledast B$, such that these two blocks contain at least $0.9n' - 0.02n' = 0.88n'$ aligned pairs. By the property (i) of B, it follows that the corresponding coordinates in x and in y are equal, i.e. $x_i = y_j$. Observe that distinct blocks i in $x \circledast B$ are matched in this way to distinct blocks j in $y \circledast B$ (because the initial substrings in $y \circledast B$ were non-overlapping, and they each more than $n'/2$ overlap with a distinct block j).

It is easily verified that the above process gives an alignment between x and y. Recall that the number of coordinates in x that are not aligned in this process is at most $150R/n'$, hence $\underline{\text{ed}}(x, y) \leq 300R/n'$, and this completes the proof. □

3.3.4 The Lower Bound

We now put all the elements of our proof together. We start by describing hard distributions, and then prove their properties. We also give a slightly more precise version of the lower bound for polynomial approximation factors in a separate subsection.

The Construction of Hard Distributions

We give a probabilistic construction for the hard distributions. We have two basic parameters, n which is roughly the length of strings, and α which is the approximation factor. We require that $2 < \alpha \ll n/\log n$. The strings length is actually smaller than n (for n large enough), but our query complexity lower bound hold also for length n,

e.g., by a simple argument of padding by a fixed string.

We now define the hard distributions.

1. Fix an alphabet Σ of size $\lceil 5^2 \cdot 2^{16} \cdot \log_\alpha^4 n \rceil$.

2. Set:

   - $T \stackrel{\text{def}}{=} \lceil 1000 \cdot \log |\Sigma| \rceil$.
   - $\beta \stackrel{\text{def}}{=} \begin{cases} \alpha, & \text{if } \alpha < n^{1/3}, \\ \frac{n}{\alpha \ln n}, & \text{otherwise.} \end{cases}$
   - $s \stackrel{\text{def}}{=} \lceil 400\beta \ln n \cdot |\Sigma|^{12} \rceil$, thus $s = O(\beta \cdot \log n \cdot \log_\alpha^{48} n)$.
   - $B \stackrel{\text{def}}{=} \lceil 8\alpha s \cdot \log_\alpha n \rceil$, implying that $B = O(\alpha\beta \log n \cdot \log_\alpha^{49} n)$. Notice that $B < \frac{n}{T}$ for n large enough. If $\alpha < n^{1/3}$, then $B = \tilde{O}(n^{2/3})$. Otherwise, $\log_\alpha n \leq 3$, $\log |\Sigma| = O(1)$, and $B = o(n)$.

3. Select at random $|\Sigma|$ strings of length B, denoted x_a for $a \in \Sigma$.

4. Define $|\Sigma|$ corresponding distributions $\mathcal{D}_a$. For each $a \in \Sigma$, let

   $$\mathcal{D}_a \stackrel{\text{def}}{=} \mathcal{S}_s(x_a),$$

   and set

   $$\mathcal{D} \stackrel{\text{def}}{=} (\mathcal{D}_a)_{a \in \Sigma}.$$

5. Define by induction on ia a collection of distributions $\mathcal{E}_{i,a}$ for $a \in \Sigma$. As the base case, set

   $$\mathcal{E}_{1,a} \stackrel{\text{def}}{=} \mathcal{D}_a.$$

   For $i > 1$, set

   $$\mathcal{E}_{i,a} \stackrel{\text{def}}{=} \mathcal{E}_{i-1,a} \circledast \mathcal{D}.$$

6. Let $i_\star \stackrel{\text{def}}{=} \lfloor \log_B \frac{n}{T} \rfloor$. Note that the distributions $\mathcal{E}_{i_\star,a}$ are defined on strings of length $B^{i_\star}$, which is is of course at most $\frac{n}{T}$, but due to an earlier observation, we also know that $i_\star \geq 1$, for n large enough.

7. Fix distinct $a_\star, b_\star \in \Sigma$. Let $\mathcal{F}_0 \stackrel{\text{def}}{=} \mathcal{E}_{i_\star,a_\star}$ and $\mathcal{F}_1 \stackrel{\text{def}}{=} \mathcal{E}_{i_\star,b_\star}$.

8. Pick a random mapping $R : \Sigma \to \{0,1\}^T$. Let $\mathcal{F}'_0 \stackrel{\text{def}}{=} \mathcal{F}_0 \circledast R$ and $\mathcal{F}'_1 \stackrel{\text{def}}{=} \mathcal{F}_1 \circledast R$. Note that the strings drawn from $\mathcal{F}'_0$ and $\mathcal{F}'_1$ are of length at most n.

Notice the construction is probabilistic only because of step #3 (the base strings x_a), and #8 (the randomized reduction to binary alphabet).

Proof of the Query Complexity Lower Bound

The next theorem shows that:

- Every two strings selected from the same distribution $\mathcal{F}_i$ are always close in edit distance.
- With non-zero probability (recall the construction is probabilistic), distribution $\mathcal{F}_0$ produces strings that are far, in edit distance, from strings produced by $\mathcal{F}_1$, yet distinguishing between these cases requires many queries.

Essentially the same properties hold also for $\mathcal{F}'_0$ and $\mathcal{F}'_1$.

Theorem 3.3.15. *Consider a randomized algorithm that is given full access to a string in Σ^n, and query access to another string in Σ^n. Let $2 < \alpha \leq o(n/\log n)$. If the algorithm distinguishes, with probability at least $2/3$, edit distance $\geq n/2$ from $\leq n/(4\alpha)$, then it makes*

$$\left(2 + \Omega\left(\frac{\log \alpha}{\log\log n}\right)\right)^{\max\left\{1, \Omega\left(\frac{\log n}{\log \alpha + \log\log n}\right)\right\}}$$

queries for $\alpha < n^{1/3}$, and $\Omega\left(\log \frac{n}{\alpha \ln n}\right)$ queries for $\alpha \geq n^{1/3}$. The bound holds even for $|\Sigma| = O(\log_\alpha^4 n)$.

For $\Sigma = \{0,1\}$, the same number of queries is required to distinguish edit distance $\geq c_1 n/2$ and $\leq c_1 n/(4\alpha)$, where $c_1 \in (0,1)$ is the constant from Theorem 3.3.12.

Proof. We use the construction described in Section 3.3.4. Recall that $i_\star \geq 1$, for n large enough, and that $i_\star \leq \log_B n$.

Let $F : \Sigma \to \Sigma^B$ be defined as $F(a) \stackrel{\text{def}}{=} x_a$ for every $a \in \Sigma$. We define $y_{i,a}$ inductively. Let $y_{1,a} \stackrel{\text{def}}{=} x_a$ for every $a \in \Sigma$, then for $i > 1$ define $y_{i,a} \stackrel{\text{def}}{=} y_{i-1,a} \circledast F$.

We now claim that for every word z with non-zero probability in $\mathcal{E}_{i,a}$ for $a \in \Sigma$, we have

$$\frac{\mathrm{ed}(z, y_{i,a})}{B^i} \leq \frac{i \cdot 2 \cdot s}{B} \leq \frac{i}{4\alpha \log_\alpha n}.$$

This follows by induction on i, since every rotation by s can be "reversed" with at most s insertions and s deletions. In particular,

$$\frac{\mathrm{ed}(z, y_{i_\star,a})}{B^{i_\star}} \leq \frac{\log_B n}{4\alpha \log_\alpha n} = \frac{\log \alpha}{4\alpha \log B} \leq \frac{1}{4\alpha},$$

where the last inequality is because $\alpha \leq B$.

It follows from Lemma 3.3.9 and the union bound that with probability

$$1 - |\Sigma|^2 \cdot e^{-5B/\sqrt{|\Sigma|}} \geq 1 - |\Sigma|^2 \cdot e^{-5|\Sigma|} \geq 1 - e^{-3|\Sigma|} \geq 1 - e^{-3} \geq 2/3$$

(over the choice of F, i.e. x_a for $a \in \Sigma$), that for all $a \neq b \in \Sigma$ we have $\mathrm{LCS}(x_a, x_b) \leq 5B/\sqrt{|\Sigma|}$, that is, the value corresponding to $\sqrt{\lambda_B}$ in Lemma 3.3.10 is at most $\sqrt{5/\sqrt{|\Sigma|}} \leq 1/(16 \log_\alpha n)$. We assume henceforth this event occurs. Then by Lemma 3.3.10 and induction, we have that for all $a \neq b$,

$$\underline{\mathrm{ed}}(y_{i,a}, y_{i,b}) \geq B^i \left(2 - \frac{i}{2\log_\alpha n}\right)$$

which gives

$$\begin{aligned} \mathrm{ed}(y_{i_\star,a_\star}, y_{i_\star,b_\star}) &\geq \frac{1}{2}\underline{\mathrm{ed}}(y_{i_\star,a_\star}, y_{i_\star,b_\star}) \geq B^{i_\star}\left(1 - \frac{i_\star}{4\log_\alpha n}\right) \geq B^{i_\star}\left(1 - \frac{\log \alpha}{4 \log B}\right) \\ &\geq B^{i_\star}\left(1 - \frac{1}{4}\right) = \frac{3}{4}B^{i_\star}. \end{aligned}$$

Consider now an algorithm that is given full access to the string $y_{i_\star,a_\star}$ and query access to some other string z. If z comes from $\mathcal{F}_0 = \mathcal{E}_{i_\star,a_\star}$, then $\mathrm{ed}(y_{i_\star,a_\star}, z) \leq \frac{B^{i_\star}}{4\alpha}$. If z comes from $\mathcal{F}_1 = \mathcal{E}_{i_\star,b_\star}$, then $\mathrm{ed}(y_{i_\star,a_\star}, z) \geq \frac{3}{4}B^{i_\star} - \frac{1}{4\alpha}B^{i_\star} \geq \frac{1}{2}B^{i_\star}$ by the triangle

inequality.

We now show that the algorithm has to make many queries to learn whether z is drawn from $\mathcal{F}_0$ or from $\mathcal{F}_1$. By Lemma 3.3.7, with probability at least 2/3 over the choice of x_a's, $\mathcal{E}_{1,a}$'s are uniformly $\frac{1}{A}$-similar, for

$$A \stackrel{\text{def}}{=} \log_{|\Sigma|} \sqrt[6]{\frac{s}{400 \ln B}} \geq \log_{|\Sigma|} \sqrt[6]{\beta \cdot |\Sigma|^{12}} = 2 + \frac{\log \beta}{6 \log |\Sigma|}.$$

Note that both the above statement regarding $\frac{1}{A}$-similarity as well as the earlier requirement that $\mathrm{LCS}(x_a, x_b)$ be small for all $a \neq b$, are satisfied with non-zero probability.

Observe that $\log |\Sigma| = \Theta(1 + \log(\frac{\log n}{\log \alpha}))$. For $\alpha < n^{1/3}$,

$$A = 2 + \Omega\left(\frac{\log \alpha}{1 + \log\left(\frac{\log n}{\log \alpha}\right)}\right) = 2 + \Omega\left(\frac{\log \alpha}{\log \log n}\right).$$

For $\alpha \geq n^{1/3}$,

$$A \geq 2 + \Omega\left(\frac{\log \frac{n}{\alpha \ln n}}{1 + \log\left(\frac{\log n}{\log \alpha}\right)}\right) \geq \Omega\left(\log \frac{n}{\alpha \ln n}\right),$$

where the last transition follows since $\frac{\log n}{\log \alpha} = \Theta(1)$ and $\alpha = o(n/\log n)$.

By using Lemma 3.3.8 over $\mathcal{E}_{i,a}$'s, we have that $\mathcal{E}_{i,a}$'s are uniformly $\frac{1}{A^i}$-similar. It now follows from Lemma 3.3.4 that an algorithm that distinguishes whether its input z is drawn from $\mathcal{F}_0 = \mathcal{E}_{i_\star,a_\star}$ or from $\mathcal{F}_1 = \mathcal{E}_{i_\star,b_\star}$ with probability at least 2/3, must make at least $A^{i_\star}/3$ queries to z. Consider first the case of $\alpha < n^{1/3}$. We have $i_\star = \Omega\left(\frac{\log n}{\log B}\right) = \Omega\left(\frac{\log n}{\log \alpha + \log \log n}\right)$. The number of queries we obtain is

$$\left(2 + \Omega\left(\frac{\log \alpha}{\log \log n}\right)\right)^{\max\left\{1, \Omega\left(\frac{\log n}{\log \alpha + \log \log n}\right)\right\}}.$$

For $\alpha \geq n^{1/3}$ we have $i_\star \geq 1$, and the algorithm must make $\Omega\left(\log \frac{n}{\alpha \ln n}\right)$ queries. This finishes the prove of the first part of the theorem, which states a lower bound for an alphabet of size $\Theta(\log_\alpha^4 n)$.

For the second part of the theorem regarding alphabet $\Sigma = \{0,1\}$, we use the distributions from the first part, but we employ the mapping $\mathcal{R} : \Sigma \to \{0,1\}^T$ to replace every symbol in Σ with a binary string of length T. Lemma 3.3.14 and Theorem 3.3.12 state that if R is chosen at random, then with non-zero probability, R preserves (normalized) edit distance up to a multiplicative c_1. Using such a mapping R and α/c_1 instead of α in the entire proof, we obtain the desired gap in edit distance between $\mathcal{F}_0'$ and $\mathcal{F}_1'$. The number of required queries remains the same after the mapping, because every symbol in a string obtained from $\mathcal{F}_0'$ or $\mathcal{F}_1'$ is a function of a single symbol from a string obtained from $\mathcal{F}_0$ or $\mathcal{F}_1$, respectively. An algorithm using few queries to distinguish $\mathcal{F}_0'$ from $\mathcal{F}_1'$ would therefore imply an algorithm with similar query complexity to distinguish $\mathcal{F}_0$ from $\mathcal{F}_1$, which is not possible. □

A More Precise Lower Bound for Polynomial Approximation Factors

We now state a more precise statement that specifies the exponent for polynomial approximation factors.

Theorem 3.3.16. *Let λ be a fixed constant in $(0,1)$. Let t be the largest positive integer such that $\lambda \cdot t < 1$.*

Consider an algorithm that is given a string in Σ^n, and query access to another string in Σ^n. If the algorithm correctly distinguishes edit distance $\geq n/2$ and $\leq n/(4n^\lambda)$ with probability at least $2/3$, then it needs $\Omega(\log^t n)$ queries, even for $|\Sigma| = O(1)$.

For $\Sigma = \{0,1\}$, the same number of queries is required to distinguish edit distance $\geq c_1 n/2$ and $\leq c_1 n/(4n^\lambda)$, where $c_1 \in (0,1)$ is the constant from Theorem 3.3.12.

Proof. The proof is a modification of the proof of Lemma 3.3.15. We reuse the same construction with the following differences:

- We set $\alpha \stackrel{\text{def}}{=} n^\lambda$. This is our approximation factor.
- We set $\beta \stackrel{\text{def}}{=} n^{\frac{1}{2}\left(\frac{1}{t}-\lambda\right)}$. This is up to a logarithmic factor the shift at every level of recursion

T, s, B, $|\Sigma|$ are defined in the same way as functions of α and β. Note that $B = \Theta\left(n^{\frac{1}{2}\left(\frac{1}{t}+\lambda\right)} \log n\right)$ and $T = \Theta(1)$. This implies that for sufficiently large n, $i_\star = \lfloor \log_B \frac{n}{T} \rfloor = t$, because $B^t = \tilde{\Theta}\left(n^{\frac{1+\lambda t}{2}}\right) = o(n)$, and $B^{t+1} = \tilde{\Theta}\left(n^{\frac{1}{2}+\frac{1}{2t}+\frac{\lambda(t+1)}{2}}\right) = \tilde{\Omega}\left(n^{1+\frac{1}{2t}}\right) = \omega(n)$.

As in the proof of Lemma 3.3.15, we achieve the desired separation in edit distance. Recall that the number of queries an algorithm must make is $\Omega(A^{i_\star})$, where

$$A \geq 2 + \frac{\log \beta}{6 \log |\Sigma|} = \Omega(\log n).$$

Thus, the number of required queries equals $\Omega(\log^t n)$. $\square$

Bibliography

[1] The size of the World Wide Web. `http://www.worldwidewebsize.com/`. Accessed Aug 24, 2010.

[2] Nir Ailon, Bernard Chazelle, Seshadhri Comandur, and Ding Liu. Estimating the distance to a monotone function. *Random Structures and Algorithms*, 31:371–383, 2007. Previously appeared in RANDOM'04.

[3] Noga Alon. On constant time approximation of parameters of bounded degree graphs. Manuscript., 2010.

[4] Noga Alon, Paul D. Seymour, and Robin Thomas. A separator theorem for graphs with an excluded minor and its applications. In *STOC*, pages 293–299, 1990.

[5] Noga Alon and Asaf Shapira. A characterization of the (natural) graph properties testable with one-sided error. *SIAM J. Comput.*, 37(6):1703–1727, 2008.

[6] Jesse Alpert and Nissan Hajaj. We knew the web was big.... In *The Official Google Blog*, July 25, 2008. Accessed Aug 24, 2010.

[7] Alexandr Andoni, T.S. Jayram, and Mihai Pătraşcu. Lower bounds for edit distance and product metrics via Poincaré-type inequalities. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2010.

[8] Alexandr Andoni and Robert Krauthgamer. The computational hardness of estimating edit distance. *SIAM Journal on Computing*, 39(6):2398–2429, 2010. Previously appeared in FOCS'07.

[9] Alexandr Andoni and Huy L. Nguyen. Near-tight bounds for testing Ulam distance. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2010.

[10] Alexandr Andoni and Krzysztof Onak. Approximating edit distance in near-linear time. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 199–204, 2009.

[11] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and hardness of approximation problems. In *FOCS*, pages 14–23, 1992.

[12] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs; a new characterization of np. In *FOCS*, pages 2–13, 1992.

[13] Mihai Badoiu, Artur Czumaj, Piotr Indyk, and Christian Sohler. Facility location in sublinear time. In *ICALP*, pages 866–877, 2005.

[14] R. A. Baeza-Yates, R. Gavaldà, G. Navarro, and R. Scheihing. Bounding the expected length of longest common subsequences and forests. *Theory Comput. Syst.*, 32(4):435–452, 1999.

[15] Ziv Bar-Yossef, T. S. Jayram, Robert Krauthgamer, and Ravi Kumar. Approximating edit distance efficiently. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 550–559, 2004.

[16] Tuğkan Batu, Funda Ergün, Joe Kilian, Avner Magen, Sofya Raskhodnikova, Ronitt Rubinfeld, and Rahul Sami. A sublinear algorithm for weakly approximating edit distance. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 316–324, 2003.

[17] Tuğkan Batu, Funda Ergün, and Cenk Sahinalp. Oblivious string embeddings and edit distance approximations. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 792–801, 2006.

[18] Itai Benjamini, Oded Schramm, and Asaf Shapira. Every minor-closed property of sparse graphs is testable. In *STOC*, pages 393–402, 2008.

[19] Philip Bille and Martin Farach-Colton. Fast and compact regular expression matching. *Theoretical Computer Science*, 409(28):486–496, 2008.

[20] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. *J. Comput. Syst. Sci.*, 47(3):549–595, 1993.

[21] Andrej Bogdanov, Kenji Obata, and Luca Trevisan. A lower bound for testing 3-colorability in bounded-degree graphs. In *FOCS*, pages 93–102, 2002.

[22] B. Chazelle, R. Rubinfeld, and L. Trevisan. Approximating the minimum spanning tree weight in sublinear time. *ICALP*, 2001.

[23] V. Chvatal and D. Sankoff. Longest common subsequences of two random sequences. *J. Appl. Probability*, 12:306–315, 1975.

[24] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001.

[25] Graham Cormode. *Sequence Distance Embeddings*. Ph.D. Thesis, University of Warwick. 2003.

[26] Graham Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. *ACM Trans. Algorithms*, 3(1), 2007. Special issue on SODA'02.

[27] Graham Cormode, Mike Paterson, Suleyman Cenk Sahinalp, and Uzi Vishkin. Communication complexity of document exchange. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 197–206, 2000.

[28] Artur Czumaj, Asaf Shapira, and Christian Sohler. Testing hereditary properties of nonexpanding bounded-degree graphs. *SIAM J. Comput.*, 38(6):2499–2510, 2009.

[29] Artur Czumaj and Christian Sohler. Estimating the weight of metric minimum spanning trees in sublinear-time. In *STOC*, pages 175–183, 2004.

[30] Andrzej Czygrinow and Michal Hańćkowiak. Distributed algorithm for better approximation of the maximum matching. In *COCOON*, pages 242–251, 2003.

[31] Andrzej Czygrinow and Michal Hanckowiak. Distributed approximation algorithms for weighted problems in minor-closed families. In *COCOON*, pages 515–525, 2007.

[32] Andrzej Czygrinow, Michal Hańćkowiak, and Edyta Szymańska. A fast distributed algorithm for approximating the maximum matching. In *ESA*, pages 252–263, 2004.

[33] Andrzej Czygrinow, Michal Hańćkowiak, and Wojciech Wawrzyniak. Fast distributed approximations in planar graphs. In *DISC*, pages 78–92, 2008.

[34] Gábor Elek. L^2-spectral invariants and convergent sequences of finite graphs. *Journal of Functional Analysis*, 254(10):2667 – 2689, 2008.

[35] Gábor Elek. Parameter testing with bounded degree graphs of subexponential growth. arXiv:0711.2800v3, 2009.

[36] Funda Ergün, Sampath Kannan, Ravi Kumar, Ronitt Rubinfeld, and Manesh Viswanathan. Spot-checkers. *J. Comput. Syst. Sci.*, 60(3):717–751, 2000.

[37] M. R. Garey and David S. Johnson. The rectilinear Steiner tree problem is NP complete. *SIAM Journal of Applied Mathematics*, 32:826–834, 1977.

[38] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[39] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 339–348, 1996.

[40] Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002.

[41] Dan Gusfield. *Algorithms on strings, trees, and sequences.* Cambridge University Press, Cambridge, 1997.

[42] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[43] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[44] Piotr Indyk. Sublinear time algorithms for metric space problems. In *STOC*, pages 428–434, 1999.

[45] Piotr Indyk. Algorithmic aspects of geometric embeddings (tutorial). In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 10–33, 2001.

[46] Piotr Indyk and Jiří Matoušek. Low distortion embeddings of finite metric spaces. *CRC Handbook of Discrete and Computational Geometry*, 2003.

[47] Piotr Indyk and David Woodruff. Optimal approximations of the frequency moments of data streams. *Proceedings of the Symposium on Theory of Computing (STOC)*, 2005.

[48] David S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. Syst. Sci.*, 9(3):256–278, 1974.

[49] Kyomin Jung and Devavrat Shah. Algorithmically efficient networks. Manuscript, `http://web.kaist.ac.kr/~kyomin/Algorithmically_Efficient_Networks.pdf`. Accessed May 19, 2010.

[50] Subhash Khot and Assaf Naor. Nonembeddability theorems via Fourier analysis. *Math. Ann.*, 334(4):821–852, 2006. Preliminary version appeared in FOCS'05.

[51] Robert Krauthgamer and Yuval Rabani. Improved lower bounds for embeddings into L_1. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1010–1017, 2006.

[52] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. The price of being near-sighted. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 980–989, New York, NY, USA, 2006. ACM.

[53] Eyal Kushilevitz, Rafail Ostrovsky, and Yuval Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. *SIAM J. Comput.*, 30(2):457–474, 2000. Preliminary version appeared in STOC'98.

[54] Gad M. Landau, Eugene W. Myers, and Jeanette P. Schmidt. Incremental string comparison. *SIAM J. Comput.*, 27(2):557–582, 1998.

[55] Christoph Lenzen, Yvonne Anne Oswald, and Roger Wattenhofer. What can be approximated locally? Case study: Dominating sets in planar graphs. In *SPAA*, pages 46–54, 2008.

[56] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals (in russian). *Doklady Akademii Nauk SSSR*, 4(163):845–848, 1965. Appeared in English as: V. I. Levenshtein, Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady* 10(8), 707–710, 1966.

[57] P. Li, T. Hastie, and K. W. Church. Nonlinear estimators and tail bounds for dimension reduction in l1 using cauchy random projections. *Journal of Machine Learning Research (JMLR)*, 2007.

[58] Richard J. Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36:177–189, 1979.

[59] Richard J. Lipton and Robert Endre Tarjan. Applications of a planar separator theorem. *SIAM J. Comput.*, 9(3):615–627, 1980.

[60] László Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13:383–390, 1975.

[61] G. S. Lueker. Improved bounds on the average length of longest common subsequences. *J. ACM*, 56(3):1–38, 2009.

[62] Sharon Marko and Dana Ron. Approximating the distance to properties in bounded-degree and general sparse graphs. *ACM Transactions on Algorithms*, 5(2), 2009.

[63] William J. Masek and Mike Paterson. A faster algorithm computing string edit distances. *J. Comput. Syst. Sci.*, 20(1):18–31, 1980.

[64] S. Muthukrishnan and Cenk Sahinalp. Approximate nearest neighbors and sequence comparison with block operations. *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 416–424, 2000.

[65] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, 2001.

[66] Huy N. Nguyen and Krzysztof Onak. Constant-time approximation algorithms via local improvements. In *FOCS*, pages 327–336, 2008.

[67] Krzysztof Onak. An efficient algorithm for approximating the vertex cover size. Manuscript., 2010.

[68] Rafail Ostrovsky and Yuval Rabani. Low distortion embedding for edit distance. *J. ACM*, 54(5), 2007. Preliminary version appeared in STOC'05.

[69] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, 1998.

[70] Michal Parnas and Dana Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theoretical Computer Science*, 381(1-3):183–196, 2007.

[71] Michal Parnas, Dana Ron, and Ronitt Rubinfeld. Tolerant property testing and distance approximation. *J. Comput. Syst. Sci.*, 72(6):1012–1042, 2006.

[72] Seth Pettie and Peter Sanders. A simpler linear time 2/3-epsilon approximation for maximum weight matching. *Inf. Process. Lett.*, 91(6):271–276, 2004.

[73] Neil Robertson and Paul D. Seymour. Graph minors. XIII. The disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63(1):65 – 110, 1995.

[74] Neil Robertson and Paul D. Seymour. Graph minors. XX. Wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325 – 357, 2004. Special Issue Dedicated to Professor W.T. Tutte.

[75] S. Roweis and L. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290:2323–2326, 2000.

[76] Süleyman Cenk Sahinalp. Edit distance under block operations. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms*. Springer, 2008.

[77] Michael Saks and Xiaodong Sun. Space lower bounds for distance approximation in the data stream model. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 360–369, 2002.

[78] Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168 – 173, 1974.

[79] Yuichi Yoshida, Masaki Yamamoto, and Hiro Ito. An improved constant-time approximation algorithm for maximum matchings. In *STOC*, 2009.

[80] Mark Zuckerberg. 500 million stories. In *The Facebook Blog*, July 21, 2010. Accessed Aug 24, 2010.