UNIVERSITY OF CALIFORNIA,
IRVINE

Exact Learning of Sequences from Queries and Trackers

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Pedro Ascensão Ferreira Matias

2021

DEDICATION

To my parents Margarida and Luís, to my sister Ana and to my brother Luisito, for the encouragement in choosing my own path, and for always being there.

– *~~Senhorzinho~~ Doutorzinho*

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

The journey to my Ph.D. degree has been a legitimate emotional rollercoaster coming to a comforting end. With all its ups and downs, slow and fast moments, it certainly qualifies as the hardest, most strenuous, but also most gratifying journey I have completed to date. Yet, it feels like it was only yesterday that I first arrived Irvine, exhausted by that multiple-day roadtrip from San Francisco, but thrilled to start my endeavours as a doctoral student. It is hard to believe that it's been almost 5 years since the start of this adventure... I will forever be intrigued by our biased perception of time, but I cherish in thinking that, after all these years and many more failures, I have grown as a computer scientist, and even more as person. For that, I have several people to thank for, without whom this journey would not have been completed.

As I am writing this, at a time when people have to wear masks to go outside and no public gatherings are allowed, I cannot help but embrace this overwhelming wave of introspection, assaulting me with memories of failures and successes, memories of self-doubt and self-determination, memories of moments shared with dear friends and colleagues, memories of beginnings and endings...

Takes deep breath

First and foremost, I thank my advisor Michael Goodrich for conceding me this journey and for believing in my capacity through the end. I am also thankful for the independence I was given to pursue any research projects that interested me, and for the freedom to collaborate with anyone. I thank David Eppstein for his cooperation in multiple projects and for serving on my defense committee alongside Sandy Irani, my very first contact within UCI and with whom I am also very grateful. I would further like to thank Vijay Vazirani and Cristina Lopes for serving on my advancement committee, as well as all of the faculty in the Center for Algorithms and Theory of Computation. My fellow theory graduate students, past and current – Daniel, Evrim, Hadi, Haleh, James, Jordan, Juan, Karthik, Martha, Nil, Ramtin, Shion, Thorben, Tim and Will – also have my appreciation for endless hours of stimulating whiteboard discussions, theory tea time and many enjoyable conversations.

Since I first became a student, I have profited immensely from mentors, teachers and professors who not only taught me how to *study* and pushed me to become a better version of myself, but who provided me with career-shaping opportunities. For this, I owe many thanks to Helena Pina Marques, Jesuíno Simões, Luís Paquete, Carlos Fonseca, Ernesto Costa and Chien-Chung Huang.

During my doctoral studies, I have learned that my research productivity is always superior when working together with others and, for that, I thank all my co-authors: Ramtin Afshar, Ami Amir, Pratibha Choudhary, Alon Efrat, David Eppstein, Daniel Frishberg, Michael Goodrich, Siddharth Gupta, Hadi Khodabande, Stephen Kobourov, James Liu, Nil Mamano, Martha Osegueda, Valentin Polishchuk and Venkatesh Raman.

Having lived in multiple cities and countries, I was fortunate enough to create many long-lasting friendships throughout my years as a student. As an undergrad in the college town of Coimbra, Portugal, I learned early on the value in tradition and in meaningful friendships and, for that, I owe many words of appreciation to my dearest friends and elements of the *Get LEId* family – Adriana, Bernardo, Carlos, Cerveira, Correia, Couceiro, Fil, Levi, Madeira, Mariana, Mário, Maximino, Micael, Pedroso, Primo, Ricardo, Si, Sá, Tiago, Valença, Valter and Vítor – who call me, to this day, by my freshman nickname *Dezanove*.

As an Erasmus student at Chalmers University of Technology in Göteborg, Sweden (from where I eventually earned my Master's degree), I met wonderful people who I now consider family – Alejandro, Carlos, Fabian, Rachele, Robert, Salvo, Santí, Sérgio and Víctor. They were the first non-Portuguese speaking true friends I made, and I am grateful for learning from them the importance of cultural awareness, open-mindedness, and of broadening one's horizons. I am also thankful for the influence they had (implicit or not) on my decision to pursue a Ph.D.

As a grad student at UCI, I had the fortune of establishing early on a reliable group of friends that I could count to for anything, and with whom I could relate my usual Ph.D. struggles. I am forever indebted to Shiva, who I met at orientation and who became my closest friend throughout my doctoral studies. Our friendship meant everything to me and her kindness and unconditional support made it so much easier to be so far away from home and family. I am also enormously thankful to Efi, Maruf, Sameera and Sumaya – we became very close friends during our first year of the program and have, since, arranged numerous trips and getaways together. I also cannot thank them enough for helping me get through COVID-19 pandemic lockdowns, for which I also thank Anil, Avik, Colin and Prabhu.

I would also like to express my deep appreciation for Tame Impala, Mac DeMarco, Crumb, Kaki King, Psychedelic Pron Crumpets, Khruangbin, Leonard Cohen, The Beatles, Kurt Vile, Courtney Barnett, TOOL, Pink Floyd, Father John Misty, Peach Pit, Portugal. The Man, Anaquim, Capitão Fausto, and so many other artists whose songs I've listened to for endless hours throughout my Ph.D. Thank you for your existence, the world needs more of your music.

Finally, I am deeply grateful for the constant love and support of my wonderful family – Ana, Luisito, Margarida and Luís, I owe you everything. The rest of the Ascensão and Matias families also have my gratitude for their kindness, empathy and compassion. *Obrigado.*

I am grateful to the Donald Bren School of Information and Computer Science for their funding support, as well as to the National Science Foundation who supported part of the work in this dissertation, under grant 1815073.

VITA

Pedro Matias

EDUCATION

Doctor of Philosophy in Computer Science	**2021**
University of California, Irvine	*Irvine, USA*
Master of Science in Computer Science	**2015**
Chalmers University of Technology	*Göteborg, Sweden*
Bachelor of Science in Informatics Engineering	**2013**
University of Coimbra	*Coimbra, Portugal*

EXPERIENCE

Teaching Assistant	**2016–2021**
University of California, Irvine	*Irvine, California*
Software Engineer Intern	**2020**
Facebook	*New York City, New York*
Software Engineering Intern	**2019**
Google	*Kirkland, Washington*
Research Intern	**2018**
Facebook Reality Labs	*Redmond, Washington*
Software Engineering Intern	**2017**
Google	*Cambridge, Massachusetts*
Software Engineering Intern	**2016**
ThousandEyes	*San Francisco, California*

PUBLICATIONS

Conferences

Brief Announcement: Parallel Network Mapping Algorithms **2021**
Ramtin Afshar, Michael T. Goodrich, Pedro Matias, Martha C. Osegueda
Symposium on Parallelism in Algorithms and Architectures (SPAA)

How to Catch Marathon Cheaters: New Approximation Algorithms for Tracking Paths **2021**
Michael T. Goodrich, Siddharth Gupta, Hadi Khodabandeh, Pedro Matias
Algorithms and Data Structures Symposium (WADS)

Reconstructing Biological and Digital Phylogenetic Trees in Parallel **2020**
Ramtin Afshar, Michael T. Goodrich, Pedro Matias, Martha C. Osegueda
European Symposium on Algorithms (ESA)

Reconstructing Binary Trees in Parallel **2020**
Ramtin Afshar, Michael T. Goodrich, Pedro Matias, Martha C. Osegueda
Symposium on Parallelism in Algorithms and Architectures (SPAA)

Adaptive Exact Learning in a Mixed-Up World: Dealing with Periodicity, Errors and Jumbled-Index Queries in String Reconstruction **2020**
Ramtin Afshar, Amihood Amir, Michael T. Goodrich, Pedro Matias Ramtin Afshar, Amihood Amir, Michael T. Goodrich, Pedro Matias
International Symposium on String Processing and Information Retrieval (SPIRE)

New Applications of Nearest-Neighbor Chains: Euclidean TSP and Motorcycle Graphs **2019**
Nil Mamano, Alon Efrat, David Eppstein, Daniel Frishberg, Michael T. Goodrich, Stephen G. Kobourov, Pedro Matias, Valentin Polishchuk
International Symposium on Algorithms and Computation (ISAAC)

Tracking Paths in Planar Graphs **2019**
David Eppstein, Michael T. Goodrich, James A. Liu, Pedro Matias
International Symposium on Algorithms and Computation (ISAAC)

PUBLICATIONS

Journals

MOSAL: software tools for multiobjective sequence alignment **2014**
Luís Paquete, Pedro Matias, Maryam Abbasi, Miguel Pinheiro
Source Code for Biology and Medicine

Improvements on bicriteria pairwise sequence alignment: algorithms and applications **2013**
Maryam Abbasi, Luís Paquete, Arnaud Liefooghe, Miguel Pinheiro, Pedro Matias
Bioinformatics

ABSTRACT OF THE DISSERTATION

Exact Learning of Sequences from Queries and Trackers

By

Pedro Ascensão Ferreira Matias

Doctor of Philosophy in Computer Science

University of California, Irvine, 2021

Distinguished Professor Michael T. Goodrich, Chair

Exact learning aims at unambiguously determining an unknown concept from answers to a carefully crafted set of questions, where each answer, given by an all-knowing oracle, reveals structural properties of the concept being learned. We consider Exact learning of sequences, i.e. collections of objects with an intrinsic total order. In this setting, we introduce a new kind of oracle, whose answers depend on an augmented version of the problem's input, devised strategically in advance by the learner, to enable efficient learning of the unknown concept. In particular, we study the problem of reconstructing the path traversed from s to t in the known input graph, by placing trackers on a subset T of the vertices, whose intersection with any s-t path results in a unique sequence, thus enabling exact reconstruction. This problem has applications to animal migration tracking and securing of large infrastructures. Since minimizing $|T|$ is NP-complete, we devise approximation and FPT algorithms.

We also study the problem of learning an unknown string over a known fixed alphabet. We consider classic oracle queries which have been shown to yield linear query complexity lower bounds, and exploit string periodicity to obtain algorithms with sublinear query complexities. We also study the problem of reconstructing a periodic string that has been corrupted by a small number of errors (e.g. due to noisy transmission channels). Finally, we study lower and upper bounds for learning a general string using jumbled-index queries.

Chapter 1

Introduction

Exact learning is, at its core, a game of guesses much like the well known game of *Mastermind*, played between a *codemaker* and a *codebreaker*, where the codebreaker's goal is to learn, as quickly as possible, the secret code established in advance by the codemaker as a sequence of 4 colors (order matters). The two players take turns, alternating between a codebreaker's guess of the secret code, and the codemaker's feedback on the correctness of the last guess, as illustrated in Fig. 1.1. In this game, it is in the codebreaker's best interest to adjust his/her strategy upon receiving the feedback given after every guess, so as to maximize the amount of new information gained and, consequently, minimize the overall number of guesses (this has been studied extensively in e.g. [6, 91, 107, 154, 176, 213, 297]). Applications of Mastermind include security and privacy around genomic data [175], as well as bank PIN data [161].

More generally, **exact learning**[1] aims at unambiguously determining an underlying unknown *concept*, by making a series of queries to an oracle, the answers of whom reveal partial information on the structure of the concept being learned [27, 28]. Naturally, the learner typically knows the scope of the secret concept, i.e. the class of concepts (possibly of

[1]In some literature exact learning goes by the name of *combinatorial search* [13]

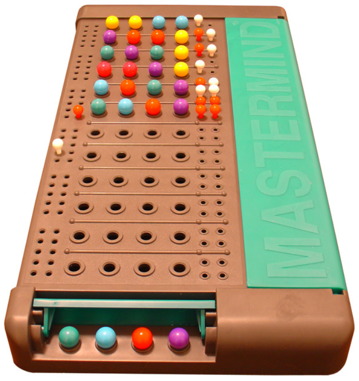


Figure 1.1: Example of a setup of Mastermind [307], where the secret sequence, decided a priori by the codemaker, is hidden from the codebreaker in the bottom row. The feedback given by the codemaker after every codebreaker's guess consists of the number of pegs in the codebreaker's guess sequence which are: (i) correct in both color and position, and (ii) correct in color, but incorrect in position – see [307] for details.

infinite size) which contains the secret. For example, in the case of Mastermind, the learner knows the secret concept is a string of colors over a fixed alphabet. We are interested in devising a learning policy that unveils the secret concept using the least amount of queries possible, i.e. an algorithm of minimum **query-complexity**. This complexity measure comes from machine learning and learning theory (see e.g. [6, 29, 98, 138, 286]), as well as complexity theory (where it is also known as *decision-tree complexity*, see e.g. [58, 310]). Thus, any amount of computation that does not involve making queries is considered free. We refer the reader to Section 1.3 for an overview of literature relevant to exact learning and, more generally, computational learning theory, an active area of research in artificial intelligence and machine learning.

Exact learning comes in two flavors[2]: *adaptive* and *non-adaptive*. In **adaptive** exact learning, we are allowed to choose the next query depending on the set of answers to all the queries made so far. In contrast, **non-adaptive** exact learning requires us to fix in advance the set of queries we wish to ask, before any answer is given. Naturally, the latter setting typically

[2]There are also settings which combine both adaptive and non-adaptive queries, such as the one studied in [127], which consists of stages: each stage corresponds to a set of parallel queries and each query can only depend on answers from previous stages.

requires asking more queries. On the other hand, non-adaptive queries can be made in any order, making parallelization of learning algorithms trivial.

In this dissertation, we restrict the scope of learning concepts to **sequences**, i.e. ordered collections of objects. In particular, we study settings of adaptive and non-adaptive exact learning of *strings* and *paths* in a graph, both of which are outlined in the next sections.

1.1 Learning Strings

We devise adaptive algorithms for learning an unknown string S, of known or unknown size, over a known fixed alphabet, much like the Mastermind example given earlier – see Chapter 2. In contrast to Mastermind, however, we consider the following types of adaptive queries:

1. **Substring** queries, of the form "Is X a substring[3] of S?"
2. **Subsequence** queries, of the form "Is X a subsequence[4] of S?"
3. **Jumbled-index** queries (also known as **histogram-index**), of the form "Is there a substring of S whose letter frequencies match the given vector of frequencies?"

Results. The first two queries above, subtring and subsequence, have been shown to yield linear query complexity lower bounds, as well as asymptotically matching upper bounds. In this dissertation, we exploit **periodicity** in strings to achieve sublinear query complexities for these types of queries [9]. We also study the problem of using substring queries to reconstruct a near-periodic string, i.e. a periodic string that has been corrupted by a small

[3]A *substring* of a string S is a contiguous sequence of letters in S.

[4]A *subsequence* of a string S is a string whose letters appear in S with the same relative order, i.e. a subsequence can be obtained from S by simply dropping some of its letters.

number of errors (e.g. due to noisy transmission channels). As far as we know, jumbled-index queries, which have received much attention of late (e.g. see [7, 19]), have not been studied before in the context of adaptive string reconstruction, so we believe we are the first to consider this oracle in an adaptive learning setting. For this latter query type, we prove that, even if the oracle adversarially provides a location within S of a matching substring, reconstructing S is not always possible in general, no matter how good our adaptive strategy is. Thus, we consider the setting where the oracle gives a relevant location chosen uniformly at random (instead of adversarially), for which we give an efficient randomized algorithm which succeeds in reconstructing S with high probability.

1.2 Learning Paths in a Graph

Given a graph and two vertices, s and t, the goal is to perform the least amount of queries that enables learning an unknown path from s to t. To achieve this, we introduce a new form of oracle, whose answers depend on an augmented version of the problem's input, devised strategically in advance by the learner, to enable efficient learning of the unknown concept. In particular, we study the problem of reconstructing the path traversed from s to t in the known input graph, by placing trackers on a subset T of the vertices, whose intersection with any s-t path results in a unique sequence, thus enabling exact learning [151, 177]. The difficult part using this strategy is, then, the placement of the trackers themselves. Good placements are studied in depth in Chapters 3 and 4. Once a valid set of trackers is determined, learning the s-t path traversed becomes straightforward. A trivial choice for T is the set of all vertices in the graph, however this will most likely not correspond to the most cost-efficient solution. Thus, we aim at finding the most economic solutions, ones that minimize the amount of trackers needed (i.e. minimize $|T|$). Since this task is NP-complete (for the vast majority of input graphs, including planar graphs – see hardness proof in Chapter 3), we provide approximation algorithms, as well as fixed-parameter tractable algorithms – see below a summary of our contributions.

Results. We show that the problem of minimizing $|T|$ is NP-complete even for planar graphs and, for graphs in this class, we provide an efficient algorithm with a 4-approximation ratio. We then improve on this result by devising a $(1 + \epsilon)$-approximation algorithm for H-minor-free graphs, using a separator-based divide-and-conquer approach. Using ideas from this latter algorithm, we design fixed-parameter tractable algorithms for H-minor-free and general graphs with, respectively, linear and quadratic sizes (w.r.t. solution size, or natural parameter). We also present a $O(\log n)$-approximation algorithm for general weighted graphs (where the goal now is to minimize $\sum_{t \in T} w(t)$), using a reduction to Set Cover. We also show that the problem has an underlying dual set-system of bounded VC-dimension, which immediately implies a $O(\log OPT)$-approximation algorithm for general unweighted graphs. Finally, we prove that minimizing $|T|$ is in P when the input graph has bounded-clique width, via Courcelle's theorem.

Our $(1 + \epsilon)$-approximation algorithm for H-minor-free graphs is admittedly much slower than our 4-approximation algorithm for planar graphs, but it has nevertheless theoretical relevance. However, as a consequence of our linear kernel for H-minor-free graphs, we get an efficient $O(1)$-approximation algorithm for this class.

1.3 Literature Overview on Exact Learning and Other Learning Models

Exact learning is, occasionally, referred to as *combinatorial search* – we refer the reader to Aigner's book [13] and Katona's survey [201] for literature using this terminology. As pointed out earlier, exact learning falls into the realm of *computational learning theory*, a branch of artificial intelligence which studies the limits and performance bounds of learning (in its broadest sense, but typically forms of learning specialized to the application), using formal mathematics and tools from theoretical computer science and statistics. For more information on this topic, we refer the reader to Kearns and Vazirani's book [203], as well as Angluin's survey [31]. The approach of exact learning differs from others in computational learning theory in the sense that it uses exclusively deductive reasoning.

The exact learning model most commonly studied is believed to have been proposed by Angluin [27, 28], who initially focused on learning regular languages (i.e. those accepted by a deterministic finite automaton) using membership queries (i.e. those specifying whether a given string is an element of the underlying language), as well as other queries, such as equivalence, superset and subset queries [29]. This model has since been regarded in terms of learning (more generally) n-variable boolean functions $f : \{0,1\}^n \to \{0,1\}$, where e.g. a membership query corresponds to asking the value of $f(\mathbf{x})$, for some assignment $\mathbf{x} \in \{0,1\}^n$. We typically study restricted classes of boolean functions, however, in order to obtain non-exhaustive strategies that exploit the structure of f. For example, if we do not know anything about f in advance, we must ask all 2^n possible membership queries. Similarly, even if we know that f evaluates to 1 for exactly one assignment, we cannot hope to achieve an efficient learning algorithm, even with the help of randomization (at least in a classical setting, without resorting to quantum computers [284]). Important classes of boolean functions include those that depend on few relevant variables – a variable is *relevant* if switching its value in any n-variable assignment $\mathbf{x}$ yields an output that differs from $f(\mathbf{x})$. This class has been studied extensively in both adaptive and non-adaptive settings in e.g. [65, 69, 77, 125, 127, 128, 229, 291]. learning of boolean functions has also been studied using several function representations, including: decision trees (e.g. see [74, 75, 77, 80, 222]), DNF/CNF formulas and circuits (e.g. see [74, 76, 170, 186, 197]) and multivariate polynomials (e.g. see [111, 267, 269]).

Applications. As suggested in e.g. [80], Angluin's model of learning from membership queries has applications in a wide range of fields, including: group testing (in and of itself a greatly motivated field, e.g. see [68, 70, 126]), logic circuit checking (e.g. see [272]), planning of chemical and biological tests, chemical reactions (e.g. see [32]), error detection in hardware and software, pattern recognition, blood testing, multi-access channel communications, molecular biology, VLSI testing, medical diagnosis (e.g. see [134]), AIDS screening, game theory, defective coin detection (e.g. see [14]). Other applications of exact learning can be found in the literature quoted above, as well as in [13, 61, 108, 142, 160, 190, 201, 212, 252]. Some of the learning policies are indeed used in physical laboratories, e.g. see [43, 44, 142, 157, 160].

Specialized Models of Exact Learning

Since its proposal, Angluin's model of exact learning [27, 28] has been repeatedly extended and/or adapted, in the context of a wide variety of fields, to enable efficient learning of concepts belonging to more restricted classes, as motivated by applications in the respective field. Below, we give a few examples – we stress that these hardly constitute an exhaustive enumeration of exact learning specializations.

Graphs. Motivated by problems arising in testing of chemical reactions, Angluin and Chen considered the problem of exact learning the edge set of a graph [32], giving nearly-optimal adaptive learning algorithms using *edge-detecting* queries (i.e. queries specifying whether a given arbitrarily-sized subset of the vertices induces an edge) that simulate whether there exists a pair of molecules in a test tube that react. As suggested by the Angluin and Chen, this problem is equivalent to learning monotone DNF formulas with terms of size 2. Angluin and Chen [32] also provide query-complexity lower bounds for learning non-uniform hypergraphs. Later, they show that uniform hypergraphs are learnable with high probability [33] – see also [1, 88, 145, 146, 190]. Non-adaptive learning of uniform hypergraphs was considered in [2, 78, 96, 169, 190, 237, 238, 289]. Previous related work, motivated by applications in genome sequencing, focused on learning subgraphs belonging to specific classes of graphs, such as Hamiltonian cycles [179–181], matchings [16, 53], or stars and cliques [15]. Query complexities for these, as well as the work of Angluin and Chen [32] have since been improved, although not asymptotically [86, 87] (to the best of our knowledge). Upper and/or lower bounds have been given in the cited papers for both adaptive and non-adaptive settings, although not surprisingly, most efficient algorithms are adaptive.

Other types of queries have been considered in the adaptive setting of exactly learning graphs, as motivated by applications arising in e.g. (i) the discovery of the topology of communication networks, such as the Internet or peer-to-peer networks, or (ii) reconstruction of phylogenetic trees (or, equivalently, hierarchical clustering). These include the following types of queries, returning:

(i) the number of edges induced by a given subset of vertices [98, 178, 180, 181, 262],

(ii) a shortest path between 2 given vertices [198, 199, 262],

(iii) the distance (i.e. length of a shortest path) between 2 given vertices [199, 243, 266] or between a given vertex and all other vertices [52, 156],

(iv) shortest path tree(s) from a given vertex to all vertices [12, 52, 271],

(v) the distance between 2 given leaves in a phylogenetic tree [10, 11, 185, 210, 263, 303],

(vi) the ancestry relationship between 2 given vertices in a rooted tree [10, 11, 301], and

(vii) whether a given vertex lies on a shortest paths between 2 other given vertices (a.k.a. *betweenness* query) [3, 192].

Strings. Exact learning of strings over a fixed alphabet has been considered in both the adaptive and non-adaptive settings, using substring, subsequence and jumbled-index queries (described above). We refer the reader to Chapter 2 for an in-depth review of literature around this topic.

Geometric Concepts. Mass and Turán [234, 235] thoroughly investigated the problem of adaptive exact learning of several types of geometric objects over a discrete domain. More specifically, they provide upper and lower bounds for exactly learning geometric concepts that are defined by points in $\{0, \dots, n-1\}^d$, a d-dimensional space, where each dimension has been discretized into n values. In particular, one of the geometric concepts they studied, which has been studied extensively in the literature, is the class of d-dimensional axis-parallel cuboids, denoted $\textsc{box}_n^d$. They proved that $\Omega(d \log n)$ equivalence queries are required to exactly learn the cuboid from $\textsc{box}_n^d$, where an *equivalence query* returns either (i) that a given hypothesized object (e.g. an element from $\textsc{box}_n^d$) matches the target object (e.g. an element from $\textsc{box}_n^d$), or (ii) a counter-example that invalidates the given hypothesis. This lower bound has since been improved to $\Omega(d^2 \log n / \log d)$ [37]. As for upper bounds, the following simple algorithm requires $O(dn)$ equivalence queries (as explained in [79]): until our hypothesis is correct, make an equivalence query corresponding to the smallest box that is consistent with the examples seen previously. This query complexity has since been improved to $O(2^d \log n)$ [233] and, later, to $O(d^2 \log n)$ [94]. The more general case of learning the union of cuboids in $\textsc{box}_n^d$ using equivalence and/or membership queries (i.e. those returning whether a point is inside the cuboid) have also been studied, e.g. see [79, 90, 92, 93, 236]. Closely related is the problem of learning the union of *non-discretized* cuboids (e.g. see [66]) under the PAC model [203, 294], which we briefly discuss in the next section.

Other Approaches to Learning

PAC-Learning. Another framework for learning, which is prior to Angluin's model of exact learning [27, 28], is the framework of *probably approximately correct learning* (PAC-learning) introduced by Valiant [294]. In the PAC-learning model the learner's goal is that of outputting an "hypothesis" concept that most closely approximates the underlying target concept – this in contrast with exact learning, which requires exact identification of the target concept. More specifically, let U be a universe of examples, each of which having a computational representation, and let $C \subseteq U$ be the target concept to be learned, chosen from a collection of concepts $\mathcal{C}$ known to us in advance. Further, let D be a distribution over U, chosen from a collection of distributions $\mathcal{D}$ also known to us in advance. An algorithm in the PAC-learning model has two input parameters, ϵ and δ, and access to an oracle who, when queried, draws an example $u \in U$ according to D, and returns u along with a label classifying the example as positive ($u \in C$) or negative ($u \notin C$). When the algorithm halts, its output is an hypothesis concept $H \in \mathcal{C}$, such that, with probability at least $(1 - \delta)$, H misclassifies an example x drawn under D (i.e. $x \in H \iff x \notin C$) with probability at most ϵ. An algorithm for PAC-learning shall have query-complexity polynomial in $1/\epsilon$, $1/\delta$ and the sizes of examples and the target concept (e.g. the length of the string representing these). Algorithms for exact learning with polynomial query-complexity, using only equivalence queries, can be transformed into algorithms in the PAC-learning model, as shown by Angluin [29], or in the absolute mistake-bound model [229]. Even though many PAC-learnable discrete concept classes can also be exactly learned using a polynomial number of equivalence queries only, Blum [64] showed that this is not true in general. For more information on PAC-learning and its relationship with exact learning, we refer the reader to Valiant's seminal paper [294], Kearns and Vazirani book [203], as well as Angluin's survey [31].

Inductive Inference. Also related (and prior) to the exact learning model of Angluin [27, 28] is the model of *inductive inference*, where one's goal is that of learning a general rule from a limited set of observations (see Angluin and Smith [34] for a survey). More specifically, the goal of inductive inference is to learn a target concept that generalizes (finite) sets of positive and negative examples, given to us in advance by either an arbitrary or a stochastic source, as in the case of PAC-learning – this differs from non-adaptive exact learning in the sense that one has no choice over the examples. Unlike PAC-learning, whose literature typically revolves around learning boolean functions (more generally), inductive inference is mostly studied in the context of recursive functions or formal grammars and, thus, it shares many techniques from recursion theory.

Chapter 2

Learning Strings

2.1 Introduction

Exact learning involves asking a series of queries so as to learn a configuration or concept uniquely and without errors, e.g., see [30]. For example, imagine a game where a player, Alice, is trying to exactly learn a secret string, S, such as $S =$ `"rumpelstiltskin"`, which is known only to a magic fairy. Alice may ask the fairy questions about S, but only if they are in a form allowed by the fairy, such as "Is X a substring of S?". Any allowable question that Alice asks must be answered truthfully by the fairy. Alice's goal is to learn S by asking the fewest number of allowable questions. Her strategy is *adaptive* if her questions can depend on the answers to previous queries. This exact-learning string-reconstruction problem might at first seem like a contrived game, but it actually has a number of applications.

For example, the magic fairy could represent a corporation with a document database, S, that supports an API allowing users to perform certain online query operations on S, such as keyword searches. Further, this corporation may receive financial compensation for each of its database responses (either directly or through advertisements); hence, the corporation might not want the database's entire contents leaking out. In this case, Alice could represent a rival corporation that is interested in learning the contents of the database, by asking legal queries from its API, so that Alice can setup a competing online query service. An optimal solution to the fairy-querying game would allow Alice to steal the database by asking the fewest number of questions necessary.

As another example, in interactive DNA sequencing, the fairy's string is an unknown DNA sequence, S, and allowable queries are "Is X a substring of S?" Each such question can be answered by a hybridization experiment that exposes copies of S to a mixture containing specific primers to see which ones bind to S, e.g., see [278]. An efficient scheme for Alice to play this fairy-querying game results in an efficient method for sequencing the unknown DNA sequence.

Yet another application comes from computer security and cryptography, dealing with searchable encryption (e.g., [122, 282]), where a database returns encrypted answers in response to queries. In this case, so long as Alice can, for instance, tell encryptions of "yes" apart from encryptions of "no," then the fairy-querying game corresponds to a type of side-channel attack, e.g., see [84, 204, 218, 223, 250, 313].

Thus, we are interested in the exact-learning complexity of adaptively learning an unknown string via queries of various given types, that is, for exactly reconstructing a string from queries. Formally, we are interested in minimizing a *query-complexity* measure, $Q(n)$, which, in our case, is the number of queries of certain types needed in order to exactly learn a string, S. This query-complexity concept comes from machine-learning and complexity theory, e.g., see [6, 30, 57, 99, 139, 286, 311].

2.1.1 Related Work

Motivated by DNA sequencing, Skiena and Sundaram [278] were the first to study exact string reconstruction from adaptive queries. For *substring queries*, of the form "Is X a substring of S?", they give a bound for $Q(n)$ of $(\sigma-1)n + 2\log n + O(\sigma)$, where σ is the alphabet size. For *subsequence queries*, of the form "Is X a subsequence of S?", they prove a bound for $Q(n)$ of $\Theta(n\log\sigma + \sigma\log n)$. Recently, Iwama, Teruyama and Tsuyama [191] study the problem for binary alphabets, which removes the additive logarithmic term in this case. These papers do not consider "mixed-up" strings, however, such as strings that are periodic or periodic with errors. The abundance of repetitions and periodic runs in genomic sequences is well known and has been exploited in the last decades for biologic and medical information (see e.g. [54, 55, 129, 140, 144, 217, 255, 258, 280, 304]). It is somewhat surprising that this phenomenon has not been used to achieve more efficient algorithms. Margaritis and Skiena [242] study a parallel version of exact string reconstruction from queries, which are hybrids of adaptive and non-adaptive strategies, showing, e.g., that a length-n string can be reconstructed in $O(\log^2 n)$ rounds using n substring queries per round. Tsur [290] gives a polynomial approximation algorithm for the 1-round case. As in [278], these papers do not consider bounds for $Q(n)$ based on properties of the string such as its periodicity. Cleve *et. al.* [112] study string reconstruction in a quantum-computing model, showing, for example, that a sublinear number of queries are sufficient for a binary alphabet. This result does not seem to carry over to a classical computing model, however, which is the subject of our paper.

Another type of query we consider is the *jumbled (or histogram)-index* query, first considered in [81, 82, 109, 155] and studied more recently in, e.g. [8, 20, 23, 24, 215, 247]. Jumbled indexing has many applications. It can be used as a tool for de novo peptide identification (as in e.g. [193, 208, 209]), and has been used as a filter for searching an image database [110, 133, 285, 300, 315]. In this query, which has received much study of late, but has not been studied before for adaptive string reconstruction, one is given a Parikh vector, i.e., a vector of frequency counts for each character in an alphabet, and asked if there is a substring of the reference string, S, having these frequency counts and, if so, where it occurs in S. Such reconstruction may aid in narrowing down peptide identification, or focusing on image retrieval.

Another model for string reconstruction, tangential to ours and studied extensively, is the one defined by a non-adaptive oracle, where: we are given a set of answers to queries in advance, and we aim to understand sufficient and necessary conditions on the answers that enable the exact reconstruction of the string. This model differs from the adaptive one considered in this chapter in that it focuses on the study of combinatorial properties of strings, rather than on minimizing the number of queries. Below, we give a detailed review of existing literature on this model, for each type of query considered in this chapter.

Non-adaptive Substring Queries

There is an extensive line of work focusing on the ability to reconstruct a string given the multiset of all its length-L substrings. For $L \geq a \lg n$ ($a > 1$), it is shown in [89, 147, 167] that, as n approaches infinity, almost every length-n string can be recovered. The following variants have also been studied: (i) only a subset of the length-L substrings is given, or each substring is subject to substitution errors of fixed Hamming distance [206, 241]; (ii) the hidden string is an i.i.d. DNA string [36], combined with a random subset of the length-L substrings [248], subject to probabilistic substitution errors [249] or edit errors (of fixed maximum amount) [168]; (iii) the hidden string satisfies several constraints based on its repeat statistics [72, 292] and input substrings are subject to erasure errors[1] [274]; and (iv) when partial reconstruction of the hidden string is sufficient [275]. On a different note, the authors of [83, 158] consider instead the case where the input is a special set of substrings

[1]A letter in the substring is replaced by an ε.

which is derived from the set of maximal substrings.

Non-adaptive Subsequence Queries

Perhaps the most studied problem in this category is the k-deck problem: given the multiset of all length-k subsequences of a length-n string S, what is the smallest value of k that enables the unique reconstruction of S? This problem was introduced in [196], who showed an upper bound of $\lfloor n/2 \rfloor$. This bound was improved to $(1 + o(1))\sqrt{(n \ln n)}$ in [270] and, in the same year, to $\lfloor 16/7\sqrt{n} \rfloor + 5$ in [221]. The first non-trivial lower bound, of $\lg / \lg \lg n$, was given in [312] and later on, was improved to $\lg n$ in [240] and to $e^{\Omega(\sqrt{\log n})}$ in [143]. Recently, Gabrys and Milenkovic [166] considered an extension of the k-deck problem, where one is also given a number of special subsequences of length $n - t$, $t > 0$; they provide lower and upper bounds that have a dependence on t. Also related to the k-deck problem is the work of Simon [276], on which subsequences are considered to be of length *at most* k. Another relevant problem is trace reconstruction. The input to this problem is a set of traces, distorted versions of the hidden string obtained by deletion (i.e. subsequences) or other types of errors, when sending it through a noisy channel. Similarly, the goal is to recover the hidden string S, either exactly or with some accuracy or probability, using the least amount of traces. To the best of our knowledge, this problem was first studied in [225], who provided bounds for the number of input traces, when subject to a worse case fixed number of substitutions, transpositions, deletions or insertion errors. In the case of exclusively dealing with deletions, where each letter is deleted with some fixed probability q, Batu, Kannan, Khanna and McGregor [49] showed that reconstruction is possible w.h.p. for $q = O(1/\log n)$ and $O(\log n)$ traces, when S is chosen uniformly at random. Moreover, they show that, for arbitrary S and for $q = O(1/n^{1/2+\epsilon})$, $O(1/\epsilon)$ traces are sufficient to reconstruct a close approximation of S and $O(n \log n)$ traces are sufficient to recover S exactly. Later Kannan and McGregor [200] extended these results to the case where insertion errors are also allowed, showing that for deletion/insertion error probabilities of $q = O(1/\log^2 n)$ and $O(\log n)$ traces, S can be recovered w.h.p. assuming it is chosen uniformly at random. Similarly, they show that an arbitrary S can be recovered w.h.p., for $q = O(1/n^{1/2+\epsilon})$ and $O(1)$ traces of length at most n^{ϵ}. Later, Viswanathan and Swaminathan [298] improved on this, by showing that deletion/insert error probabilities of $q = O(1/\log n)$ are sufficient to reconstruct S, chosen uniformly at random. They also show that $\Omega(\log n)$ traces are necessary to reconstruct $1 - o(1)$ length-n strings with high probability. Holenstein, Mitzenmacher, Panigrahy and Wieder [188] showed that, for the case of deletion errors only, of probability $q = O(1)$, reconstruction is possible w.h.p. using poly(n) traces, when S is chosen uniformly at random. Finally, Sala, Gabrys, Schoeny, Mazooji and Dolecek [268] studied lower bounds on the number of input traces formed from a worst-case number of insertion errors, where S is a member of specific error-correcting codes, i.e. sets of strings constructed strategically to allow recovering them from a noisy channel is modified.

Non-adaptive Jumbled-Index Queries

Acharya, Das, Milenkovic, Orlitsky and Pan [4, 5], study a non-adaptive version of the problem of enumerating candidate strings from the *composition multiset* of the underlying string. The composition multiset corresponds to the set of answers to all possible queries of the following type: given a Parikh vector, how many times does a matching substring occur in the hidden string? Under this model, they extend polynomial techniques used for the turnpike problem (see [124, 277]) to give: (i) sufficient (but not necessary) conditions for the ability to uniquely reconstruct a string, (ii) a sufficient characterization of unreconstructable strings and (iii) a backtracking algorithm that enumerates the set of all candidate strings, whose cardinality they lower and upper bound.

2.1.2 Our Results

We provide new and improved results for exactly reconstructing strings from adaptive substring, subsequence, and jumbled-index queries. For example, we believe we are the first to characterize query complexities for exactly reconstructing periodic strings from adaptive queries, including the following results for reconstructing a length-n periodic (i.e., "mixed-up") string, $S = p^k p'$, of smallest period p, where p' is a prefix of p and the alphabet has size σ:

- It requires at least $|p| \lg \sigma$ substring or subsequence queries.
- It can be done with $\sigma|p| + \lceil \lg |p| \rceil$ substring queries, if n is known.
- It can be done with $O(\sigma|p| + \log n)$ substring queries, if n is unknown.
- It can be done with $\sigma\lceil \lg n \rceil + 2|p|\lceil \lg \sigma \rceil$ subsequence queries, for known n.
- It can be done with $2\sigma\lceil \lg n \rceil + 2|p|\lceil \lg \sigma \rceil$ subsequence queries, if n is unknown.

Perhaps our most technical result is that we show that we can reconstruct a length-n string, S, within Hamming distance d of a periodic string $S' = p^k p'$, of smallest period p, using $O(\min(\sigma n,\, d\sigma|p| + d|p| \log \frac{n}{d+1}))$ substring queries, if n is unknown. We also show that we can exactly reconstruct a general length-n string, S, using $2\sigma\lceil \lg n \rceil + n\lceil \lg \sigma \rceil$ subsequence queries, if n is unknown. Such queries are another "mixed-up" setting, since there can be multiple subsequence matches for a given string. Our bound improves the previous best, decades-old result, by Skiena and Sundaram [278], who prove a query complexity of $2\sigma \lg n + 1.59n \lg \sigma + 5\sigma$ for this case. If n is known, then $\sigma\lceil \lg n \rceil + n\lceil \lg \sigma \rceil$ subsequence queries suffice. We believe we are the first to study string reconstruction using jumbled-index queries, which are yet another "mixed-up" setting, since they simply count the frequency of each character occurring in a substring. We prove the following results:

- We can reconstruct a length-n string with $O(\sigma n)$ yes/no extended jumbled-index queries, which include a count for an end-of-string character, \$.
- For jumbled-index queries that return an index of a matching substring, string reconstruction is not possible if this index is chosen adversarially, but is possible using $O(\sigma + n \log n)$ queries if it is chosen uniformly at random.

2.1.3 Preliminaries

We consider strings over the alphabet $\Sigma = \{a_1, a_2, \ldots, a_\sigma\}$ of σ letters. The size of a string X is denoted by $|X|$. We use $X[i]$ to denote the i^{th} letter of X and $X[i..j]$ to refer to the substring of X starting at its i^{th} and ending at its j^{th} letter (e.g., $X = X[1..|X|]$). We may ignore i when expressing a prefix $X[..j]$ of X. Similarly, $X[i..]$ is a suffix of X. Occasionally, we will express concatenation of strings X and Y by $X \cdot Y$ (instead of XY) to emphasize some property of the string. A string X concatenated with itself k (resp. infinitely many) times can be expressed as X^k (resp. X^∞). The reversal of a string X is denoted by X^R.

A string, S, has *period* p if $S = p^k p'$, such that $k > 0$ is an integer and p' is a (possibly empty) prefix of p. Further, a string S is *periodic* if it has a period that repeats at least twice, i.e. $S = p^k p'$ and $k > 1$[2]. The following is a well known result concerning the periodicity of a string, due to Fine and Wilf [159], which we will need later on.

Lemma 2.1 (Periodicity Lemma [159]). *If p, q are periods of a string X of length $|X| \geq |p| + |q| - \gcd(|p|, |q|)$, then X also has a period of size $\gcd(|p|, |q|)$.*

A *doubling search* is the operation used to determine a number n from a (typically unbounded) range of possibilities. It involves doubling a query value, m, until it is greater than n, followed by a binary search to determine n itself. Its time complexity is $2\lfloor \lg n \rfloor + 1$.

[2]Our algorithms assume that S is periodic ($k > 1$), while the Periodicity Lemma (2.1) only requires a string to have a period ($k > 0$).

A more sophisticated version of this procedure exists (see [56]) that actually improves the time complexity into

$$\lfloor \lg L^{(0)} \rfloor + \lfloor \lg L^{(1)} n \rfloor + \cdots + \lfloor \lg L^{(t-1)} n \rfloor + 2 \lfloor \lg L^{(t)} n \rfloor + 1,$$

where $L^{(j)}(n) = \lfloor \lg L^{(j-1)}(n) \rfloor + 1$ and $L^{(0)}(n) = n$, for which there exists an optimized value of t. For simplicity, we use the traditional algorithm, which is asymptotically equivalent.

2.2 Substring Queries

In this section, we study query complexities for a string, S, subject to yes/no *substring* queries, IsSubstr, i.e. queries of "Is X a substring of S?". We focus on the cases where S corresponds to an originally periodic string, that may have lost its periodicity property due to error corruption. The nature of the errors is context-dependent. For example, corruption may be caused by transmission errors, measurement errors, malicious tampering, or even by the aging process of a natural phenomenon. There are multiple ways to model errors in strings. Examples include:

- *Hamming* distance model: corruption is caused by allowing the substitution of a letter in the string by a different letter of the alphabet
- *Edit* distance model: a generalization of the Hamming distance that also allows the insertion or deletion of a letter (see [224])
- *Swap* distance model: the operation allowed consists of swapping two adjacent letters in the string (see [232, 299])
- *Interchange* (or *Caley*) distance model: it generalizes swap distance, by allowing the swap of any two letters, not necessarily adjacent (see [21, 26, 85, 194])

In this chapter, we consider Hamming distance. We say that S is a *d-corrupted periodic string* if there exists a periodic string S' of period p, such that $|S| = |S'|$ and $\delta(S', S) \leq d$, where δ is the Hamming distance. We refer to p as an *approximate period* of S. Notice that, depending on d, there might exist multiple strings S' with approximate periods of S. We are interested in reconstructing S, as opposed to S', since we can use one of the existing algorithms to enumerate all possible strings S' (see [18, 25, 216]), without incurring additional queries.

Our main result in this section is the following.

Theorem 2.1. *We can reconstruct a length-n d-corrupted periodic string S using*

$$O\left(\min\left(\sigma n, d\sigma|p| + d|p|\log\frac{n}{d+1}\right)\right) \text{ queries,}$$

for known d, unknown $|p|$, regardless of whether we know n, where p is a smallest approximate period of S.

The algorithm of Theorem 2.1 is a more elaborate version of a reconstruction algorithm for the special case of $d = 0$, i.e. when no errors occurred and $S = S'$, and when n is not known in advance.

Theorem 2.2. *We can reconstruct a length-n periodic string, $S = p^k p'$, of smallest period p, using $O(\sigma|p| + \log n)$ substring queries, assuming both n and $|p|$ are unknown in advance.*

The algorithm of Theorem 2.2, in turn, builds from a simple reconstruction algorithm that handles the case where n is known in advance and $d = 0$.

For clarity, we will present our results in increasing order of complexity, from the least general result of $d = 0$ and known n, to the most general result of arbitrary d and unknown n.

Algorithm 1: Reconstructing a periodic string $S = p^k p'$ of known size n and smallest period p, for $k > 1$.

```
1. Let $q = \varepsilon$
2. repeat
       Append a letter to $q$                                        $(\sigma - 1)$
   until IsSubstr($q^{g(q)-1}$)                  (1 per iteration; $|p|$ iterations)
3. Let $T = q^{g(q)-1}$
4. While $T$ is a substring of $S$, append a letter to $T$                         }
5. While $|T| < n$ and $T$ is a substring of $S$, prepend a letter to $T$          } $(\sigma(2|p| - 1))$
6. Output $T$                                                                      }
```

2.2.1 Uncorrupted Periodic Strings of Known Size

We first give a simple algorithm to reconstruct a periodic string $S = p^k p'$ of smallest period p and known size with query complexity $O(\sigma|p|)$, and then show how to improve this algorithm to have query complexity $\sigma|p|$ plus lower-order terms. Our algorithms use two primitives developed by Skiena and Sundaram [278], which we call "*append*" and "*prepend*" a letter. In the append (resp., prepend) primitive, we start with a known substring q of S, and we ask queries $\mathsf{IsSubstr}(qa_i)$ (resp., $\mathsf{IsSubstr}(a_iq)$), for each $a_i \in \Sigma$. Note that if we know that one of the qa_i (resp., a_iq) strings must be a substring, we can save one query, so that appending or prepending a letter uses at most $\sigma - 1$ queries in this case.

In our simple algorithm, we iteratively grow a candidate period, q, using the append primitive until $q^{g(q)-1}$ is a substring, where $g(x) = \lfloor n/|x| \rfloor$. Notice that q may be an "unlucky" cyclic rotation of p, which only repeats $g(p) - 1$ times, and we need to account for this possibility. Thus, once we get a substring corresponding to $q^{g(q)-1}$, we then append/prepend letters until we recover all of S. For reference, see Algorithm 1, where the number of queries is shown in parentheses for steps involving queries.

Theorem 2.3. *We can reconstruct a length-n periodic string $S = p^k p'$, of smallest period p, using $O(\sigma|p|)$ substring queries, assuming n is known in advance and $|p|$ is unknown.*

Proof. The main loop in Algorithm 1 will always terminate, because S is periodic and any cyclic permutation of p is a substring, when concatenated at least $g(p) - 1$ times. It is easy to see that the procedure of iteratively appending letters to q must result in a cyclic permutation of p, unless the main loop stops earlier. After the main loop, there are at most $2|p|-1$ letters left to be recovered, so the overall query complexity is at most $\sigma|p|+\sigma(2|p|-1)$, which is $O(\sigma|p|)$. □

With a little more effort, we can improve the constant factor in the query complexity. The main challenge to achieving this improvement is that, after the main loop in Algorithm 1, q may not correspond to a cyclic rotation of p. For example, in $S = ababbabaab \cdot ababbabaab \cdot ababbabaab$, we may get $q = abababa$, while the actual period is $p = ababbabaab$. However, we show that, when $k = n/|p| > 3$, the following implication holds indeed: if $q^{g(q)-1}$ is a substring, then q must be a cyclic rotation of p.

We begin by giving the details for our improved algorithm for reconstructing a periodic length-n string S, when n is known, which is shown in Algorithm 2.

Algorithm 2: Reconstructing a periodic string $S = p^k p'$ of known size n and smallest period p, for $k > 3$.

```
   start
1.   Let $q = \varepsilon$
2.   repeat
3.       Append a letter to $q$                                  $(\sigma - 1)$
     until IsSubstr($q^{g(q)-1}$)                (1 per iteration; $|p|$ iterations)
4.   Let $p$ = TrueRotation($q$)                                 $(\lceil \lg |q| \rceil)$
5.   Determine $p'$ and output $p^k p'$

   function TrueRotation($q$)
     Find, using binary search, the largest suffix $q[j..]$, such that
      IsSubstr($q[j..] \cdot q^{g(q)-1}$)                        $(\lceil \lg |q| \rceil)$
     Return $q[j..] \cdot q[..j-1]$
```

Remark 2.1. *A string, p, is a period of a string X of length $|X| \geq i|p|$ if and only if p^j is a period of X, for all $j \in \{1, 2, \ldots, i\}$.*

Theorem 2.4. *We can reconstruct a length-n periodic string $S = p^k p'$, of smallest period p, using at most $\sigma|p| + \lceil \lg |p| \rceil$ substring queries, assuming that: n is known in advance, $k > 3$ and $|p|$ is unknown.*

Proof. Consider Algorithm 2. We claim that, immediately after the main loop, the candidate period q is indeed a cyclic rotation of the true period p. The remainder of the proof then follows from this.

So let us prove our claim. Let q be the string immediately after the main loop and let $T = q^{\lfloor n/|q| \rfloor - 1}$. If $|q| = |p|$, then q is clearly a cyclic rotation of p. Besides, $|q|$ cannot be greater than $|p|$, because the letter-by-letter construction of q would have implied a halt of the main loop when q had size $|p|$: any cyclic rotation of p must repeat at least $\lfloor n/|p| \rfloor - 1$ times. So let us consider the case $|q| < |p|$. Since $k > 3$, we have that $n \geq 4|p|$. Moreover, since $T = q^{\lfloor n/|q| \rfloor - 1}$, we know that $|T| \geq n - (2|q| - 1)$ and, thus, $|T| \geq 2|p|$. Since T is a substring of S, T must have a second period of size $|p|$. Moreover,

$$
\begin{aligned}
|T| &\geq 2|p| \\
&\geq |p| + |q| \\
&\geq |p| + |q| - \gcd(|p|, |q|)
\end{aligned}
$$

Thus, by the Periodicity Lemma (2.1), T has a period p_T of size $\gcd(|p|, |q|)$. Therefore, S must have a period of size $|p_T|$, and thus, S must have a period of size $|q|$ (by Remark 2.1), which contradicts the fact that p is the smallest period of S. □

Our analysis above is tight in the sense that, for $k = 3$, it no longer holds: recall the example given above, where $S = ababab aab \cdot abababaab \cdot abababaab$ and $q = abababa$.

Notice that any reconstruction algorithm requires at least $|p| \lg \sigma$ queries; this follows from an information-theoretic argument.

Theorem 2.5. *Reconstructing a length-n string, $S = p^k p'$, of smallest period p, requires at least $|p| \lg \sigma$ IsSubstr queries, even if n and $|p|$ are known.*

Proof. There are $\sigma^{|p|}$ possible periods for S. Since each period corresponds to a different output of a reconstruction algorithm, A, and each query is binary, we can model any such algorithm, A, as a binary decision tree, where each internal node corresponds to an IsSubstr query. Each of the $\sigma^{|p|}$ possible periods must correspond to at least one leaf of A; hence, the minimum height of A is $\lg(\sigma^{|p|})$. □

In the next section we consider the case where the underlying string is of unknown size.

2.2.2 Uncorrupted Periodic Strings of Unknown Size

As in Section 2.2.1, we iteratively grow a candidate period q and attempt to recover S by concatenating q with itself in the appropriate way. The difficulty when n is unknown is that we can no longer confidently predict $g(q)$. Thus, we can no longer issue a single query to test if q is the right period. An immediate solution is to use a doubling search. Unfortunately, this introduces a multiplicative $O(\log n)$ term into the query complexity. To avoid it, we show how we can take advantage of the Periodicity Lemma (2.1) to amortize the extra work needed to recover S.

Let us describe the algorithm (see Algorithm 3 for reference). We start with an empty candidate period q. At each iteration, we add a letter to q, using the append primitive and, using a doubling search, determine the *run-length* t of q, i.e. the maximum integer t such that q^t is a substring of S. If $t = 1$, we advance to the next iteration and repeat this process. If, on the other hand, $t > 1$, we use q to determine the largest substring T that has a period of size $|q|$. This can be done efficiently, using doubling searches, by determining the largest suffix l of q and the largest prefix r of q, such that IsSubstr$(l \cdot q^t \cdot r)$. Once T is

Algorithm 3: Reconstructing a periodic string $S = p^k p'$, of smallest period p and unknown size n, for $k > 1$.

```
    start
1.      Let $q = \varepsilon$
2.      repeat
3.          Append or prepend a letter to $q$          $(\sigma - 1$; potentially, $2\sigma - 1$ when $k \leq 2)$
4.          Determine the run-length $t$ of $q$                              $(2\lfloor \lg t \rfloor + 1)$
5.          if $t = 1$ then Let $T = q$
6.          else
7.              Let $l$ be the largest suffix of $q$ such that IsSubstr$(l \cdot q^t)$        $(2\lfloor \lg |l| \rfloor + 1)$
8.              Let $r$ be the largest prefix of $q$ such that IsSubstr$(l \cdot q^t \cdot r)$   $(2\lfloor \lg |r| \rfloor + 1)$
9.              Let $T = l \cdot q^t \cdot r$
10.             Let $q = T[..|T| - |q| + 1]$
        until IsValid$(T)$                                                    $(2\sigma)$
11.     Output $T$

    function IsValid$(T)$                                                     $(2\sigma)$
        Let $x$ be the letter to the left of $T$ or $\varepsilon$ if there is none      $(\sigma)$
        Let $y$ be the letter to the right of $T$ or $\varepsilon$ if there is none     $(\sigma)$
        Return $x == \varepsilon$ and $y == \varepsilon$
```

determined, we check whether it corresponds to S by checking if there is any letter preceding and succeeding T (see IsValid subroutine). If T corresponds to S, we output it. Otherwise, we update q to be any largest substring of T whose size is assuredly less than $|p|$: using Periodicity Lemma (2.1), we argue in Lemma 2.2 below that, if q is not a cyclic rotation of p, then p must be as large as *almost* the entire substring T; more specifically, it must be the case that $|p| > |T| - |q| + 1$. Thus, we update q to be a length-$(|T| - |q| + 1)$ prefix of T (any other substring of T would also work). We use this fact to get a faster convergence to a cyclic rotation of p, while making sure that we do not overshoot $|p|$. Indeed, this observation will enable us to incur a $O(\log n)$ additive factor, instead of a multiplicative one. After updating q, we advance to the next iteration, where a new letter is appended to q, and repeat this process until $T = S$.

Lemma 2.2. *Let T be the largest proper substring of $S = p^k p'$, of smallest period p, such that: $|q|$ is the length of the smallest period of T. Then, $|p| > |T| - |q| + 1$.*

Proof. Let us assume, by contradiction, that $|p| \leq |T| - |q| + 1$. Then, $|T| \geq |q| + |p| - 1$ and, thus, $|T| \geq |q| + |p| - \gcd(|q|, |p|)$. In addition, if p is a period of S, then T must have a period of size $|p|$. So, by the Periodicity Lemma (2.1), T also has a period of size $\gcd(|q|, |p|)$. Moreover, since T is the largest proper substring of S, $|p|$ is not a multiple of $|q|$. Therefore, T must have a period shorter than $|q|$, a contradiction. □

When $k \leq 2$, our algorithm behaves similarly to the letter-by-letter algorithm of Skiena and Sundaram [278] – after finding a cyclic rotation q of p, our algorithm will continue adding letters to q until $q = S$, this time using both the append and prepend primitives.

Next, we give the details of the correctness and query complexity of Algorithm 3. Let $q_1, q_2, \ldots, q_m$ be the sequence of m candidate periods of increasing length, each of which is the result of the append/prepend primitive at the beginning of every iteration (line 3 of Algorithm 3), e.g. $|q_1| = 1$. Notice that each q_i may be expanded (in line 10), so the difference $|q_i| - |q_{i-1}|$ may not necessarily be 1. In addition, let us use t_i to denote the run-length of q_i computed in line 4.

Lemma 2.3. *Algorithm 3 successfully returns $S = p^k p'$, of smallest period p, if there exists an iteration $i \in \{1, 2, \ldots, m\}$, such that q_i is a cyclic rotation of p.*

Proof. If $t_i > 1$, then it is easy to see that the string T, computed in line 9 in iteration i, must correspond to S. If $t_i = 1$, then the algorithm essentially switches to the letter-by-letter algorithm, appending or prepending letters until the end, when $q_m = S$. Correctness of the stopping condition follows from the correctness of IsValid. □

We now show that, indeed, at some iteration i, the candidate period q_i is a cyclic rotation of p.

Lemma 2.4. *There exists an iteration $i \in \{1, 2, \ldots, m\}$, such that q_i is a cyclic rotation of p.*

Proof. Let us assume that there is no such iteration i. Then, since all the q_i's are increasing in length, it must be the case that there exists an iteration $j \in \{1, 2, \ldots, m-1\}$, such that: $|q_j| < |p|$, but $|q_{j+1}| > |p|$. However, it follows from Lemma 2.2 (when $t_j > 1$) and the fact that we add a single letter to q_j (when $t_j = 1$) that p must be at least as large as q_{j+1}, a contradiction. □

Let us now argue about query complexity. The following lemma shows that we can charge the logarithmic factors, incurred in each iteration j, to the work that would have been required to find the letters introduced in q_{j+1}. This establishes the amortization in query complexity. We denote the number of queries in iteration j of Algorithm 3 by $\mathcal{Q}(j)$.

Lemma 2.5. *The number of queries $\mathcal{Q}(j)$ performed in iteration j of Algorithm 3 is at most $\sigma(|q_{j+1}| - |q_j|) + O(\sigma)$, for $j < m$, or $O(\sigma + \log n)$, for $j = m$.*

Proof. Let l_j and r_j denote, respectively, the lengths of the prefix l and suffix r computed in lines 7 and 8 of Algorithm 3 in iteration j. The query complexity in any iteration j is

$$\mathcal{Q}(j) \leq 2\lfloor \lg t_j \rfloor + 1 + 2\lfloor \lg l_j \rfloor + 1 + 2\lfloor \lg r_j \rfloor + 1 + 4\sigma$$

Let us assume that $t_j > 1$, since otherwise the query complexity is $O(\sigma)$ and, therefore, agrees with the query complexity that is stated in the lemma.

When $j = m$, it must be the case that q_m is a cyclic rotation of p, and therefore has size $|p|$. Thus, we spend at most: (i) σ queries when appending the p^{th} letter, (ii) $2\lfloor \lg n/|p| \rfloor + 1$ queries to determine the run-length t_m, and (iii) $2(2\lfloor \lg |p| \rfloor + 1)$ queries to determine the suffix and prefix of lengths l_m and r_m, respectively. Notice that the combined log factors result in no less than $\Theta(\log n)$. Thus, when $j = m$, the overall query complexity is $O(\sigma + \log n)$.

When $j < m$, we have the following:

$$\begin{aligned}
&& \lg t_j \leq t_j - 1 && (t_j > 1)\\
&\implies& \lg t_j \leq (t_j - 2)q_j + 1 && (q_j \geq 1)\\
&\implies& \lg t_j + \lg l_j + \lg r_j \leq (t_j - 2)q_j + l + r + 1 && (\lg x < x)\\
&\implies& 2(\lg t_j + \lg l_j + \lg r_j) \leq 2((t_j - 2)q_j + l + r + 1) &&\\
&\implies& \mathcal{Q}(j) \leq 2((t_j - 2)q_j + l + r) + 2 + 3 + O(\sigma) && (\text{def. of } \mathcal{Q}(j))\\
&\implies& \mathcal{Q}(j) \leq 2((t_j - 2)q_j + l + r + 2) + O(\sigma) &&\\
&\implies& \mathcal{Q}(j) \leq 2(|q_{j+1}| - |q_j|) + O(\sigma) && (\star)\\
&\implies& \mathcal{Q}(j) \leq \sigma(|q_{j+1}| - |q_j|) + O(\sigma) && (\sigma \geq 2),
\end{aligned}$$

where $(\star)$ follows from the fact that, when $t_j > 1$, $|q_{j+1}| = (t_j - 1)|q_j| + l + r + 2$. □

Finally, we are in conditions of proving Theorem 2.2, recalled below for convenience:

Theorem 2.2. *We can reconstruct a length-n periodic string, $S = p^k p'$, of smallest period p, using $O(\sigma|p| + \log n)$ substring queries, assuming both n and $|p|$ are unknown in advance.*

Proof. Correctness follows from Lemmas 2.3 and 2.4. As for the query complexity, it follows from Lemma 2.5, that the overall query complexity of Algorithm 3 is

$$\sum_{j=1}^{m} \mathcal{Q}(j)$$

Let i be the iteration in which $|q_i| = |p|$ (see Lemma 2.4) and let us consider the queries done up to and after iteration $i-1$. Thus, by Lemma 2.5:

$$\begin{aligned}\sum_{j=1}^{m}\mathcal{Q}(j) &= \sum_{j=1}^{i-1}\Big(\sigma(|q_{j+1}|-|q_j|)+O(\sigma)\Big)+\sum_{j=i}^{m}\mathcal{Q}(j)\\ &= O(\sigma|p|)+\sum_{j=i}^{m}\mathcal{Q}(j),\end{aligned}$$

where the last equality follows from the telescoping nature of the first summation. As for the second summation, regarding the queries done after iteration $i-1$, we consider two cases. If $i=m$, then we spend either $O(\sigma)$ queries if $t_i=1$, or $O(\sigma+\log n)$ queries if $t_i>1$, by Lemma 2.5. If, on the other hand, $i<m$, then notice that it must have been the case that $t_j=1$ for all $j\in\{i,i+1,\ldots,m\}$. Thus, the total number of letters in S left to recover at the end of iteration $i-1$ is at most $2|q_i|-1=2|p|-1$, each of which is added during each iteration $j\in\{i,i+1,\ldots,m\}$ using $O(\sigma|p|)$ queries in total. Thus, whether or not $i=m$, the overall query complexity is

$$\sum_{j=1}^{m}\mathcal{Q}(j)=O(\sigma|p|+\log n)$$

□

2.2.3 Corrupted Periodic Strings

Let us assume throughout the remainder of this section that S is a d-corrupted periodic string of approximate period p. Recall that S is a d-corrupted periodic string if there exists a periodic string S' of period p, such that $|S|=|S'|$ and $\delta(S',S)\leq d$, where δ is the Hamming distance. Again, the main idea of the algorithm described in this section consists of: (1) determining a cyclic rotation of a true period (in this case, there might be multiple true periods), by iteratively growing a candidate period q, and (2) using q to recover S accordingly. However, in the presence of errors, each of these steps becomes more difficult to realize efficiently. For example, in the first step, we might be growing a candidate period q that includes an error. So, in order to rightfully reject the hypothesis that q is at most as large as some approximate period p, our algorithm should be able to tell the difference between (i) $|p| = |q|$ and q includes an error and (ii) $|p| > |q|$. Otherwise, the algorithm will keep on growing q until it is equal to S, possibly incurring σn queries. In addition, the second step of using q to determine S requires more work, since the presence of errors discards the possibility of simply concatenating q with itself the required number of times. Because of these issues, it is crucial that our algorithm understands when a candidate period is or not free of errors. Thus, the algorithm relies on the following.

Lemma 2.6. *Let A be any length-$(2d+1)|p|$ substring of a d-corrupted periodic string S of approximate period p, corresponding to the concatenation of length-$|p|$ substrings $q_1, q_2, \ldots, q_{2d+1}$. Then, a cyclic rotation of p must be the only substring q_j appearing at least $d+1$ times in $q_1, q_2, \ldots, q_{2d+1}$.*

Proof. Clearly, there is some q_i that is a cyclic rotation of p. Moreover, there is some q_j that appears at least $d+1$ times in $q_1, q_2, \ldots, q_{2d+1}$, or the number of errors would exceed d, by the pigeonhole principle. If $i \neq j$, then each occurrence of q_j, contributes at least 1 error, resulting in at least $d+1$ errors, a contradiction. Finally, q_j must be the only string with $d+1$ appearances in $q_1, q_2, \ldots, q_{2d+1}$, by the pigeonhole principle. $\square$

Algorithm 4: Reconstructing a d-corrupted periodic string S.

```
1. Let $A = \varepsilon$
2. repeat
3.     Append/prepend $\min(2d+1, |S| - |A|)$ letters to $A$        $(\sigma(2d+2))$
4.     Let $q$ be the candidate period that is a substring of $A$,
         as determined by Lemma 2.6
5.     $(\mathsf{success}, T) = \mathsf{Expand}(q)$        $(O(d\sigma + d\log\frac{n}{d+1}))$
   until success
6. Output $T$
```

```
Function Expand(q)                                                    $(O(d\sigma + d\log\frac{n}{d+1}))$
1. Let $T = q$, done = False
2. while $\delta(T, q^\infty[..|T|]) \le d$ and not done do
3.     Find the largest substring $R$, such that IsSubstr($T \cdot R$)          $(2\lfloor\lg|R|\rfloor + 1)$
4.     Find the largest substring $L$, such that IsSubstr($L \cdot T \cdot R$)      $(2\lfloor\lg|L|\rfloor + 1)$
5.     Let $r$ ($l$) be the letter to the right (left) of $L \cdot T \cdot R$ or $\varepsilon$ if there is none    $(2\sigma)$
6.     Let done = ($r == \varepsilon$ and $l == \varepsilon$)
7.     Let $T = l \cdot L \cdot T \cdot R \cdot r$
8. if $\delta(T, q^\infty[..|T|]) > d$ then return (False, _)
9. return (True, $T$)
```

Let us give the details for our algorithm, which is able to recover S, even when its size n is unknown (see Algorithm 4 for reference). We maintain an initially empty substring, A, of S, by extending it with $2d+1$ letters in each iteration, using the append and prepend primitives (as described in Section 2.2.1), potentially incurring an extra σ queries for detecting a left or right endpoint of S. In the case that $n = |S| < |p|(2d+1)$, the last iteration requires only $\min(2d+1, |S| - |A|)$ new letters. Thus, after adding letters to A in the i^{th} iteration, A is a substring of S of size at most $i(2d+1)$. Before advancing to the next iteration, we determine the only possible length-i candidate period q that could have originated A with at most d errors (by Lemma 2.6). At this point we do not know if some approximate period p has size $|p| = i$, so we try to use q to recover the rest of the string, halting whenever the total number of errors exceeds d, in which case we advance to the next iteration and repeat this process for a new candidate period of size $i+1$. This logic is in the subroutine Expand(q), described next (see the pseudo-code for reference). It initializes a string T to q and expands it by doing the following at each iteration:

1. Appending to T the largest periodic substring of period $\overrightarrow{q}$, where $\overrightarrow{q}$ is the appropriate cyclic rotation of q that aligns with the right-endpoint of T. This can be done efficiently by determining the maximum value of x, using a doubling search, for which

$$\mathsf{IsSubstr}(T \cdot (\overrightarrow{q}^\infty[..\, x])),$$

incurring $2\lfloor \lg x \rfloor + 1$ queries. The cyclic rotation $\overrightarrow{q}$ can be determined with no additional queries, by maintaining the value x', which is the value of x in the previous iteration, i.e. $\overrightarrow{q}$ is the cyclic rotation of q starting at the index $(x' \mod |q| + 2)$ of q.

2. Prepending to T the largest periodic substring of period $\overleftarrow{q}$, where $\overleftarrow{q}$ is the appropriate cyclic rotation of q that aligns with the left-endpoint of T. This can be done efficiently by determining the maximum value of y, using a doubling search, for which

$$\mathsf{IsSubstr}(((\overleftarrow{q}^R)^\infty[..\, y])^R \cdot T),$$

incurring $2\lfloor \lg y \rfloor + 1$ queries. The cyclic rotation $\overleftarrow{q}$ can be determined with no additional queries in a similar fashion to $\overrightarrow{q}$.

3. Determining, if they exist, the letters immediately to the left and to the right of T, using 2σ queries, and adding them to T.

The expansion process in $\mathsf{Expand}(q)$ halts when either the total number of errors with respect to q, $\delta(T, q^\infty[..|T|])$, exceeds d (in which case we advance to the next iteration), or when $T = S$ (in which case we return T).

Remark 2.2. *$\mathsf{Expand}(q)$ successfully returns S if and only if q is a cyclic rotation of some approximate period.*

Lemma 2.7. *The number of queries performed during any call to Expand is $O(d\sigma + d \log \frac{n}{d+1})$.*

Proof. Each call to Expand uses at most $2(d+1)\sigma$ queries to determine the corrupted letters, as well as the left/right endpoints of S – the total number of iterations of the while loop in Expand is $d + 1$, since every iteration except the last introduces at least 2 errors in T, and each iteration incurs 2σ queries.

In addition, the number of queries used by $\mathsf{Expand}(q)$ during the doubling searches is

$$\sum_{j=1}^{|q|} \left(2\lfloor \lg R_j \rfloor + 2\lfloor \lg L_j \rfloor + 2\right),$$

where R_j and L_j denote, respectively, the lengths of the substrings determined via doubling searches in lines 3 and 4, during the j^{th} call to Expand. Since the total number of iterations is $d+1$, there is at most $d+2$ such R_j's and L_j's. Moreover, the above summation is maximized when all the R_j's and L_j's have the same average value of at most $(n-d)/(d+1)$. This follows from Jensen's inequality and concavity of log. Thus, the overall time complexity is

$$O\left(d\sigma + d \log \frac{n}{d+1}\right).$$

□

Correctness and query complexity of our algorithm follows from Remark 2.2 and Lemmas 2.6 and 2.7, giving us the following result:

Theorem 2.6. *We can reconstruct a length-n d-corrupted periodic string S using $O(d\sigma|p| + d|p| \log \frac{n}{d+1})$ queries, for known d, unknown $|p|$, regardless of whether we know n, where p is a smallest approximate period of S.*

Proof. At the $|p|^{\text{th}}$ iteration of the main loop, A has size $(2d+1)|p|$ and, by Lemma 2.6, q must correspond to a cyclic rotation of some approximate period p. Correctness of reconstruction then follows from Remark 2.2.

The overall query complexity consists of the queries used to expand A in each iteration and the queries used in the calls to the subroutine Expand. The former requires at most $(2d+2)\sigma|p|$ queries overall, and the latter requires at most $O(d\sigma|p| + d|p| \log \frac{n}{d+1})$, by Lemma 2.7.

Thus, the overall query complexity is

$$O\left(d\sigma|p| + d|p|\log\frac{n}{d+1}\right).$$

□

If n is known, we could save the queries used to check the left and right endpoints of S in line 5 of Expand, but this does not alter the query complexity asymptotically.

We assume a small enough number of errors, following [25]. Algorithm 4 is an improvement to the $O(\sigma n)$ letter-by-letter algorithm of Skiena and Sundaram [278] for general strings of known and unknown size, when $d = O(\sigma n/(\sigma|p| + |p|\log n))$. In particular, if $d = O(k/(1 + \log n))$, then Algorithm 4 is an improvement, where $k = \lfloor n/|p| \rfloor$. Thus, our algorithm performs no worse if there is, on average, at most 1 error in every other $O(1 + \log n)^{\text{th}}$ non-overlapping occurrence of p. If the number of errors is not small enough, then one should run the letter-by-letter algorithm intercalated with ours, to get an upper bound of $O(\sigma n)$ queries, giving us Theorem 2.1, which we referred to at the beginning of Section 2.2, recalled here for convenience.

Theorem 2.1. *We can reconstruct a length-n d-corrupted periodic string S using*

$$O\left(\min\left(\sigma n, d\sigma|p| + d|p|\log\frac{n}{d+1}\right)\right) \text{ queries,}$$

for known d, unknown $|p|$, regardless of whether we know n, where p is a smallest approximate period of S.

2.3 Subsequence Queries

We study the query complexity for a length-n string, S, subject to yes/no *subsequence* queries, IsSubseq, i.e., queries of the form "Is X a subsequence of S?"

We begin with a simple lower bound.

Theorem 2.7. *Reconstructing a length-n periodic string, $S = p^k p'$, of smallest period p, requires at least $|p| \lg \sigma$ IsSubseq queries, even if n and $|p|$ are known.*

Proof. The proof follows that of Theorem 2.5 for substring queries, which can be found in Section 2.2. □

Let us next describe an algorithm for reconstructing a periodic length-n periodic string, $S = p^k p'$, of smallest period p. We begin by performing either binary searches (if n is known) or doubling search (if n is unknown), using queries of the form IsSubseq(a^i) to determine the number of a's in S, for each $a \in \Sigma$. From all of these queries, we can determine the value of n if it was previously unknown. This part of our algorithm requires either $\sigma\lceil \lg n \rceil$ or $2\sigma\lceil \lg n \rceil$ queries in total, depending on whether we knew n at the outset.

If the number of a's in S is n, for any $a \in \Sigma$, then we are done, so let us assume the number of a's in S is less than n, for each $a \in \Sigma$. Thus, when we complete all our doubling/binary searches, for each letter, $a \in \Sigma$ that occurs a nonzero number of times in S, we have a maximal subsequence, S_a, of S, consisting of a's. Moreover, since S is periodic with a period that repeats k times, each S_a is periodic with a period that repeats k times. Unfortunately, at this point in the algorithm, we may not be able to determine k. So next we create a binary merge tree, T, with each of its leaves associated with a nonempty subsequence, S_a, much in the style of the well-known merge-sort algorithm, so that T has height $\lceil \lg \sigma \rceil$. We then perform a bottom-up merge-like procedure in T using IsSubseq queries, as follows.

Let v be an internal node in T, with children x and y for which we have inductively determined periodic subsequences, S_x and S_y, respectively, of S. Let $n_x = |S_x|$ and $n_y = |S_y|$. To create the subsequence, S_v, for v, we need to perform a merge procedure to interleave S_x and S_y. To do this, we maintain indices i and j in S_x and S_y, respectively, such that we have already determined an interleaving, $S_v[..i+j]$, of $S_x[..i]$ and $S_y[..j]$. Initially, $i = j = 0$. We then perform the query $\mathsf{IsSubseq}(S_v[..i+j] \cdot S_x[i+1] \cdot S_y[j+1..n_y])$. Suppose the answer to this query is "yes". In this case, we set $S_v[..i+j+1] = S_v[..i+j] \cdot S_x[i+1]$ and we increment i. If, on the other hand, the answer to the above query is "no", then we set $S_v[..i+j+1] = S_v[..i+j] \cdot S_y[j+1]$, because in this case we know that $\mathsf{IsSubseq}(S_v[..i+j] \cdot S_y[j+1] \cdot S_x[i+1..n_x])$ would return "yes". If this latter condition occurs, then we increment j.

Let q_v denote this new interleaving prefix, $S_v[..i+j]$, and let $\hat{k} = \lfloor n/|q_v| \rfloor$. If ${q_v}^{\hat{k}}{q_v}'$ is a plausible interleaving of S_x and S_y, where ${q_v}'$ is a prefix of q_v, then we next ask the query $\mathsf{IsSubseq}({q_v}^{\hat{k}}{q_v}')$. If the answer is "yes", then we set $S_v = {q_v}^{\hat{k}}{q_v}'$ and this completes the merge. Otherwise, we continue incrementally interleaving S_x and S_y, using the current values of i and j, by iterating the procedure described above. Clearly, this merge procedure asks at most $2|q_v|$ queries in total.

Lemma 2.8. *Let p_v be the subsequence of p consisting of the letters from S_v. Then $|q_v| \leq |p_v|$.*

Proof. The letter-by-letter construction of q_v ensures that q_v is the smallest period of S_v. Since p_v is itself a period of S_v no smaller than the smallest, then $|p_v| \geq |q_v|$. □

Theorem 2.8. *We can determine a length-n periodic string, $S = p^k p'$, of smallest period p of unknown size, using $2\sigma\lceil \lg n \rceil + 2|p|\lceil \lg \sigma \rceil$ IsSubseq queries, if n is unknown. If n is known, then $\sigma\lceil \lg n \rceil + 2|p|\lceil \lg \sigma \rceil$ IsSubseq queries suffice.*

Proof. The total query complexity includes: (i) the letter decomposition S_a for all $a \in \Sigma$, during the the first stage and (ii) the merge-like composition of all subsequences S_a, during the second stage. If n is known, the first stage requires $|\Sigma|$ binary searches, incurring $\sigma\lceil\lg n\rceil$ queries. Otherwise, it requires $|\Sigma|$ doubling searches, amounting to $2\sigma\lceil\lg n\rceil$ queries. Regarding the second stage, we claim that any level l of the binary merge tree, T, incurs a total of at most $2|p|$ queries, which amounts to a total of at most $2|p|\lceil\lg\sigma\rceil$ queries, when taking into account all the $\lceil\lg\sigma\rceil$ levels of T. Let $T(l)$ be the set of all nodes in T at level l. Then,

$$\sum_{v\in T(l)} |q_v| \leq \sum_{v\in T(l)} |p_v| = |p|.$$

This follows from Lemma 2.8 and the fact that all $\{S_v \mid v \in T(l)\}$ are pairwise letter-disjoint. Since the merge of an internal node v requires a cost of $2|q_v|$, the total cost incurred in any level l of T is at most $2|p|$. □

A simple modification of our algorithm also implies the following.

Theorem 2.9. *We can determine a length-n string, S, using $2\sigma\lceil\lg n\rceil + n\lceil\lg\sigma\rceil$ IsSubseq queries, without knowing the value of n in advance. If n is known, then $\sigma\lceil\lg n\rceil + n\lceil\lg\sigma\rceil$ IsSubseq queries suffice.*

Proof. Modify our subsequence-querying algorithm given in Section 2.3 to remove the queries for strings of the form ${q_v}^{\hat{k}}{q_v}'$. The proof follows by an analysis similar to that for Theorem 2.8. □

This latter theorem improves a result of Skiena and Sundaram [278], who prove a query bound of $2\sigma\lg n + 1.59n\lg\sigma + 5\sigma$ when n is unknown.

2.4 Jumbled-index Queries

Jumbled-indexing involves preprocessing a given string, S, so as to determine whether there exists a substring of S whose letter frequencies match the given *Parikh vector*, i.e., a vector $\psi = (f_1, \ldots, f_\sigma)$ such that f_i is the number of occurrences in S of $a_i \in \Sigma$, e.g., see [8, 20, 23, 24, 215, 247]. In this section, we study the query complexity for reconstructing an unknown length-n string, S, using jumbled-index queries. As observed by Acharya, Das, Milenkovic, Orlitsky and Pan [4, 5], strings and their reversals have the same "composition multiset". This immediately implies the following negative result, which we prove regardless for completeness.

Lemma 2.9. *If S is not a palindrome, then S cannot be reconstructed by yes/no jumbled-index queries, which return whether there is a substring in S with a given Parikh vector.*

Proof. Suppose $S \neq S^R$, where S^R denotes the reversal of S. For any substring, T, of S, there is, of course, a corresponding substring, T^R, of S^R. Moreover, T and T^R have the same Parikh vector. Thus, S and S^R have the same set of responses to yes/no jumbled-index queries; hence, any set of yes/no jumbled-index queries cannot distinguish S from S^R. □

Given that simple yes/no jumbled-index queries are not sufficient for string reconstruction, let us consider an extended type of yes/no jumbled-index query.

- *Jumbled-Indexing with End-of-string symbol "\$"* (JIE): given an *extended* Parikh vector, $\psi = (f_1, \ldots, f_\sigma, f_\$)$, for the letters in Σ and an end-of-string symbol, \$, which is not in Σ, this query returns a yes/no response as to whether there is a substring of $S\$$ with extended Parikh vector ψ.

Unlike the yes/no jumbled-index queries, this variant enables full reconstruction.

Theorem 2.10. *We can reconstruct a length-n string, S, using $(\sigma - 1)n$ JIE queries, if n is known, or $\sigma(n+1)$ JIE queries, if n is unknown.*

Proof. Our method is to use a letter-by-letter reconstruction algorithm via an adaption of the prepend-a-letter primitive for substring queries. Suppose n is unknown. Let ψ be an extended Parikh vector for a known suffix, s, of $S\$$; initially, $\psi = (0, 0, \ldots, 0, 1)$ and $s = \$$. Then we perform a jumbled-index query for ψ_i, for each $a_i \in \Sigma$, where $\psi_i = \psi$ except that ψ_i adds 1 to the f_i value in ψ. If one of these, say, ψ_i, returns "yes", then we prepend a_i to our known suffix and we repeat this procedure using ψ_i for ψ. If all of these queries return "no", then we are done. If n is known, on the other hand, then we can skip this last test of all-no responses and we can also save at least one query with each iteration, with the algorithm otherwise being the same. $\square$

We can also consider jumbled-index queries that return an index of a matching substring for a given Parikh vector, if such a substring exists. Though related, notice that this type of query is not subsumed by the query studied in Acharya, Das, Milenkovic, Orlitsky and Pan [4, 5], which returns the number of occurrences (instead of position) of matching substrings in S. There is some ambiguity, however, if there is more than one matching substring; hence, we should consider how to handle such multiple matches. For example, if a jumbled-index query returns the indices of all matching substrings, then σ queries are clearly sufficient to reconstruct any length-n string, for any n, without knowing the value of n in advance. Thus, let us consider two more-interesting types of jumbled-index queries.

- *Adversarial Jumbled-Indexing* (AJI): given a Parikh vector, $\psi = (f_1, \ldots, f_\sigma)$, this query returns, in an adversarial manner, one of the starting indices of a matching substring, if such a string exists. If there is no matching substring, this query returns False.

- *Random Jumbled-Indexing* (RJI): given a Parikh vector, $\psi = (f_1, \ldots, f_\sigma)$, this query returns, uniformly at random, one of the indices of a substring with Parikh vector ψ if such a substring exists in S. If there is no such substring, this query returns False.

Unfortunately, for the AJI variant, there are some strings that cannot be fully reconstructed, but this is admittedly not obvious. In fact, the unreconstructability characterization of [4, 5] fails for AJI queries, because the symmetry property used in their construction of pairwise "equicomposable" strings inherently yields matching substrings with symmetric (e.g. different) positions in S.

Nevertheless, we give a construction of an infinite family of pairwise undistinguishable strings, i.e. two strings such that, for every possible query, there exists an answer (positive or negative) that is common to both strings. Clearly, the adversarial strategy is to output these common answers when given either of these strings. In particular, for all $b \geq 1$, consider the two binary strings of length $4b+14$ given below, which differ only in the middle section, consisting of `01` in the first string and `10` in the second:

$$S_1 = \texttt{101101(10)}^b\texttt{01(10)}^b\texttt{010010}$$
$$S_2 = \texttt{101101(10)}^b\texttt{10(10)}^b\texttt{010010}$$

Theorem 2.11. *The strings S_1 and S_2 cannot be distinguished using AJI queries, for $b \geq 1$.*

Proof. Let $n = 4b+14$ be the size of the strings. We refer to responses that would be common to both S_1 and S_2 as *helpless* answers. Let us think of a positive answer i to a query (k, l) in terms of the space occupied by its matching substring, denoted $\langle i,\ i+k+l-1\rangle$. We note that an answer that does not span the middle section or that spans it in its entirety must be helpless.

Notice that the first half of either string is the symmetric complement of the second half. This implies the following: (i) an answer $\langle i,\ j\rangle$ to a query (k, l) exists if and only if an answer

$\langle n-j+1,\ n-i+1\rangle$ exists for the query (l,k) and (ii) an answer is negative to (k,l) if and only if an answer is negative to (l,k). Therefore, we can restrict ourselves to queries of the form $(k,k+c)$, where $c \geq 0$. We break this down to the following cases:

1. **Queries of type** (k,k).

   We say that an answer is *k-centered* if it is of the type $\langle n/2-(k-1),\ n/2+1+(k-1)\rangle$. Since any k-centered answer contains the middle section, it must be helpless. Thus, it is enough to show, by induction, that all queries (k,k) have k-centered answers. Clearly, this holds for the base case $(1,1)$, so let us assume that there exists a $(k-1)$-centered answer a to the query $(k-1,k-1)$. Then, because the first half of either string is the symmetric complement of the second half, the letters preceding and succeeding a must be the complement of each other. Thus, the k-centered answer must be valid for the query (k,k).

2. **Queries of type** $(k,k+1)$.

   Take the k-centered answer and either extend it with one letter to the left, or one letter to the right. Exactly one of these options is a valid answer to $(k,k+1)$ (by the symmetric-complement property of the strings) and either are helpless, since they span the middle section.

3. **Queries of type** $(k,k+2)$.

   Consider, as a base case, the answer $\langle 2,\ 3\rangle$ to the query $(0,2)$. Clearly, it is a helpless answer. Given that the letters at positions $4+2j$ and $5+2j$ are complements of each other, $\langle 2,\ 3+2j\rangle$ is a valid answer to the query $(j,j+2)$, for all $0 \leq j \leq b+2$. For greater values of j, the answer is helpless regardless, since it corresponds to a substring of length greater than $2b+7$, half of the string length and, therefore, it spans the middle section.

4. **Queries of type** $(k,k+c)$**, for** $c \geq 3$.

It is enough to analyze answers that partially span the middle section (i.e. in exactly 1 letter), since otherwise answers are automatically helpless. Let Δ_i denote the number of `1`'s minus the number of `0`'s for the answer $\langle i,\ n/2\rangle$, with respect to S_2, for all $0 \leq i \leq n/2$ (e.g. $\Delta_{n/2} = 1$, corresponds to the first letter in the middle). A simple passage from right to left, for increasing values of i, reveals that there exist no value of i for which $\Delta_i = 4$, so we do not need to handle the case $c \geq 4$. Moreover, the only values of i for which $\Delta_i = 3$ are $i = 2$ and $i = 0$, which correspond to answers for the queries $(b+1, b+4)$ and $(b+2, b+5)$, respectively. However, these queries have helpless answers: $\langle 0,\ 2b+4\rangle$ in the former and $\langle 2,\ 2b+8\rangle$ in the latter. For the first string, a similar exercise reveals that there exist no answers that partially overlap the middle section and whose difference between the number of `1`'s and the number of `0`'s is at least 3.

□

In contrast, the query variant RJI can be used to reconstruct any length-n string, S, without knowing the value of n in advance. In particular, it is possible to reconstruct any length-n string, S, using $O(\sigma + n \log n)$ RJI queries with high probability. Our algorithm for doing this involves a reduction to a multi-window coupon-collector problem.

Let ψ_i be a Parikh vector that is all 0's except for a count of 1 for the letter $a_i \in \Sigma$. Note that an RJI query using ψ_i will return one of the n_i locations in S with an a_i uniformly at random (if $n_i > 0$). If $n_i = 0$, for any $i = 1, 2, \ldots, \sigma$, we learn this fact immediately after one RJI query for ψ_i, so let us assume, w.l.o.g., that $n_i > 0$, for all $i = 1, 2, \ldots, \sigma$, after performing an initial σ number of RJI queries.

Recall that in the *coupon-collector* problem, a collector visits a coupon window each day and requests a coupon from an agent, who chooses one of n coupons uniformly at random and gives it to the collector, e.g., see [245]. The expected number of days required for the collector to get all n coupons is nH_n, where H_n is the n^{th} Harmonic number. But this assumes the collector knows when they have received all n coupons (i.e., the collector knows the value of n).

In a coupon-collector formulation of our reconstruction problem, we instead have σ coupon windows, one for each letter $a_i \in \Sigma$, where each window i has n_i coupons that differ from the coupons for the other windows, and we do not know the value of any n_i. Each day the collector must choose one of the coupon windows, i, and request one of its coupons (corresponding to an RJI query for ψ_i), which is chosen uniformly at random from the n_i coupons for window i. We are interested in a strategy and analysis for the collector to collect all $n = n_1 + n_2 + \cdots + n_\sigma$ coupons, with high probability (i.e., with probability at least $1 - 1/n$).

Note that although we do not know the value of any n_i, we can nonetheless test whether the collector has collected all n coupons. In particular, suppose we have received RJI responses for all indices, $1, 2, \ldots, n$, for letters in S, and let n_i be the number of a_i's we have found so far. Let $\psi' = (n_1, n_2, \ldots, n_\sigma)$, and let ψ'_i be equal to ψ' except that we increment n_i by 1. If an RJI query for each ψ'_i returns False, then we know we have fully reconstructed S. Thus, if $n = 1$, then we can determine this and S after 2σ RJI queries, so let us assume that $n \geq 2$. Further, we can assume we have a bound, $N \geq 2$, which is at least n and at most twice n, by a simple doubling strategy, where we double N any time a test for n fails and we set N equal to any RJI query response that is larger than N. Therefore, the remaining problem is to solve the multi-window coupon-collector problem.

Our strategy for the multi-window coupon-collector problem is simply to visit the coupon windows in phases, so that in phase i we repeatedly visit window i until we are confident we have all of its n_i coupons, for which the following lemma will prove useful.

Lemma 2.10. *Let T_i be the number of trips to window i needed to collect all its $n_i \geq 1$*

coupons. Then, for any real number β:

$$\Pr\left(T_i > \beta n_i \ln N\right) \leq \frac{n_i}{N^\beta}.$$

Proof. Adapting a proof from [305], let $Z_{j,r}$ denote the event that the j-th coupon was not picked in the first r trips to window i. Then

$$\Pr\left(Z_{j,r}\right) = \left(1 - \frac{1}{n_i}\right)^r \leq e^{-r/n_i}.$$

Thus, for $r = \beta n_i \ln N$, we have $\Pr(Z_{j,r}) \leq e^{-(\beta n_i \ln N)/n_i} = N^{-\beta}$. Therefore, by a union bound,

$$\Pr\left(T_i > \beta n_i \ln N\right) = \Pr\left(\bigcup_j Z_{j,\beta n_i \ln N}\right) \leq n_i \cdot \Pr\left(Z_{1,\beta n_i \ln N}\right) \leq \frac{n_i}{N^\beta}.$$

□

Our strategy, then, is to let $\beta \geq 2$ be constant, and in phase i, implement a doubling strategy where we perform $\beta N_i \log N$ RJI queries for ψ_i, such that N_i is an upper bound estimate for n_i, which we double each time we get more than N_i distinct responses to our queries in this phase. So by the end of the phase i, $n_i \leq N_i \leq 2n_i$. This gives us:

Theorem 2.12. *A string, S, of unknown size, n, can be reconstructed using $O(\sigma + n \log n)$ RJI queries, with high probability.*

Proof. After an initial $O(\sigma)$ queries to determine which letters from Σ appear in S, the total number of remaining queries performed by our method is at most

$$2\sum_{i=1}^{\sigma} 2\beta N_i \ln N = 4\sum_{i=1}^{\sigma} \beta N_i \ln N,$$

by the doubling strategy applied to each letter, $a_i \in \Sigma$, and then globally for N. Further,

$$\sum_{i=1}^{\sigma} \beta N_i \ln N \leq \sum_{i=1}^{\sigma} 2\beta n_i \ln N \leq \sum_{i=1}^{\sigma} 3\beta n_i \ln n = 3\beta n \ln n,$$

with probability at least

$$1 - \sum_{i=1}^{\sigma} \frac{n_i}{N^\beta} = 1 - \frac{\sum_{i=1}^{\sigma} n_i}{N^\beta} = 1 - \frac{n}{N^\beta} \geq 1 - \frac{1}{n},$$

by Lemma 2.10, since $\beta \geq 2$. □

2.5 Conclusion and Open Questions

We have studied the reconstruction of strings under the following settings, by giving efficient reconstruction algorithms and proving lower bounds: (i) periodic strings of known and unknown sizes, with and without mismatch errors, using substring queries; (ii) periodic strings of known and unknown sizes, using subsequence queries and (iii) general strings, using variations of jumbled-indexing queries. For the non-optimal algorithms given here, it would be nice to know whether there exist matching lower bounds, or whether there exist faster algorithms.

Regarding corrupted periodic strings, different applications suffer from different types of corruption. In particular, the following error metrics have been considered in the literature: Pseudo-local metrics such as swap distance [22] or Interchange (Cayley) distance [25]; and the Levenshtein edit distance [224]. It would be interesting to see whether our reconstruction algorithms can be adapted to these more general error distances.

The next step is to reconstruct strings that have more complex syntactic regularities than periods, such as *covers* [35]. A length m substring C of a string T of length n, is said to be a *cover* of T, if $n > m$ and every letter of T lies within some occurrence of C. We would like to efficiently reconstruct a coverable string, without knowing its cover a-priori.

Data compression schemes such as, Lempel-Ziv [316, 317] are known to compress any stationary and ergodic source down to the entropy rate of the source per source symbol, provided the input source sequence is sufficiently long. These schemes rely heavily on encoding repeated substrings by their starting index and length. In this sense, a periodic string is highly compressible. We would like to extend our ideas to reconstruct a general string in time proportional to its LZ compression.

The type of query used for reconstruction is a key factor in the reconstruction complexity. Much as the error distance, the query type is also application-dependent. A reasonable query type is the *less than matching*. Let S_1 and S_2 be strings of length n over an ordered alphabet. We say that S_1 is *less than* S_2 if $S_1[i] < S_2$, $\forall i = 1, \ldots, n$. Other matchings that have been researched in the literature, are the *order preserving matching* [97, 121, 207], and the *parameterized matching* [41, 42]. In the order preserving matching, we say that two strings match if the relative order of their elements is the same, for example $1, 2, 3, 2, 1$ matches such strings as $1, 100, 101, 100, 1$, or $56, 61, 366, 61, 56$, i.e., any string where the fist element is smaller than the second, which is smaller than the third, where the fourth is equal to the second, and the fifth equals the first. Two equal-length strings S_1, S_2 over alphabet Σ are said to *parameterize match*, if there is a bijection $f : \Sigma \to \Sigma$ such that $S_1 = f(S_2)$. Using these more powerful queries, can we reconstruct a string more efficiently?

Finally, given the impossibility result on reconstructing strings using Adversarial Jumbled-Indexing queries, it would be interesting to know whether there exists an efficient algorithm that enumerates all of the undistinguishable strings.

Chapter 3

Learning Paths in Planar Graphs

3.1 Introduction

In most modern marathons, each runner is provided with a small RFID tag, which is worn on the runner's shoe or embedded in the runner's bib. RFID readers are placed throughout the course and are used to track the progress of the runners [100, 302]. In spite these measures, some runners try to cheat by taking shortcuts [306]. To detect all possible course-cutting, we are interested in the combinatorial optimization problem of placing the minimum number of RFID readers in the environment of a marathon to determine every possible path from the start to the finish, including paths that deviate from the official course, just from the sequence of RFID readers that are crossed by a runner taking a given path. In addition to detecting marathon course-cutting, solutions to this optimization problem could also allow for a type of marathon where each runner could be allowed to map out their own path from the start to finish so long as their path is at least the required length.

Formally, we model a city road network [150, 152, 153] through which a marathon will be run as an undirected graph, $G = (V, E)$, where V is the set of road intersections and possible

RFID reader locations in the city, as well as the placements of the start and finish lines, and E is the set of road segments joining two points in V without having any other elements of V in its interior. Given a start-finish pair, (s, t), of vertices in G, a *tracking set* for (s, t) is a subset, T, of V, such that for any s-t path[1] P in G, the sequence $\mathcal{S}^T(P)$ of vertices in T traversed by P uniquely identifies P. In other words, T is a tracking set if $\mathcal{S}^T(P) \neq \mathcal{S}^T(Q)$ for all distinct s-t paths P and Q. We formally define the optimization problem, which is called the *tracking paths* problem, as follows:

TRACKING(G, s, t):	
Input:	An undirected simple graph $G = (V, E)$ and vertices $s, t \in V$.
Question:	A smallest tracking set for (s, t) in G.

We denote by PLANAR-TRACKING the problem of tracking paths when the input graph is planar.

Banik, Katz, Packer and Simakov [48] and Banik, Choudhary, Lokshtanov, Raman and Saurabh [46, 47] introduced the tracking paths problem, motivated by applications in surveillance and monitoring. Examples of surveillance applications include the following: (i) vehicle tracking in road networks; (ii) habitat monitoring; (iii) intruder tracking and securing large infrastructures; (iv) tracing back of illicit Internet activities by tracking data packets. Another application would be to determine the nodes in a network that have been compromised by a spreading infection, given an incomplete transmission history of a pathogen. This information can be helpful in identifying and attenuating the negative impact caused by biological and non-biological infectious agents, such as:

- Contagious diseases (e.g. COVID-19, Severe Acute Respiratory Syndrome) which could lead to epidemics [173].

[1] In this chapter, paths do not repeat vertices. We denote a path from u to v by u-v.

- Fake news and hate speech being disseminated in social networks, as well as violations of privacy (e.g. sharing without permission highly sensitive content owned by a user, such as intimate pictures).
- Computer viruses, which spread throughout servers scattered across the Internet.

Some of these applications have been studied empirically or using heuristics in [60, 182, 259, 279] or, in the case of infections, [39, 246, 251, 273]. To the best of our knowledge, the above-mentioned references [46–48] were the first to approach the problem of tracking paths from a theory perspective. In this work, we extend some of their work and give new algorithms. Our main results apply to graphs that can: be embedded on the plane (of interest to surveillance in road networks and similar infrastructures) or that have bounded clique-width (mainly of theoretical interest).

3.1.1 Related Work

Banik, Choudhary, Lokshtanov, Raman and Saurabh first introduced Tracking in [46], where it is shown to be NP-hard by reducing from Vertex Cover, which seems unlikely to work in the planar case. Although not immediately obvious, they also show that Tracking is in NP, by observing that every tracking set is also a *feedback vertex set*, i.e. a set of vertices whose removal yields an acyclic graph. Finally, they present a fixed-parameter tractable (FPT) algorithm (parameterized by the solution size k) for the decision version of the problem, where they obtain a kernel of $O(k^7)$ edges.

The concept of tracking set, however, first appeared in Banik, Katz, Packer and Simakov [48], where they considered a variant of Tracking that only concerns shortest s-t paths, essentially modeling the input as a directed acyclic graph (DAG). Using a similar reduction from Vertex Cover, they show that this variant cannot be approximated within a factor of 1.3606, unless P=NP. They also give a 2-approximation for the planar version of tracking shortest paths, but they omit any hardness results for this variant.

More recently, Bilò, L. Gualà, Leucci and Proietti [62] generalized the version of the problem concerning shortest s-t paths, into the case of multiple source-destination pairs, for which they claim the first $O(\sqrt{n \log n})$-approximation algorithm for general graphs. They also study a version of this multiple source-destination pairs problem in which the set of trackers itself (excluding the order in which they are visited) is enough to distinguish between s-t shortest paths[2]. In this setting, they claim a $O(\sqrt{n})$-approximation algorithm and they show that it is NP-hard even for cubic planar graphs. The hardness construction intrinsically relies on the multiplicity of source-destination pairs and, therefore, cannot be adapted to the problem studied in this chapter. They also give an FPT algorithm (with respect to the maximum number of vertices at the same distance from the source) for the problem concerning a single source-destination pair that was introduced in [48].

In [45], Banik and Choudhary generalize TRACKING into a problem on set systems[3], which are characterized by a universe (e.g. vertex set) and a family of subsets of the universe (e.g. s-t paths). They show that this generalized version of the problem is fixed-parameter tractable, by establishing a correspondence with the well known Test Cover problem.

Our results. We give a 4-approximation for PLANAR-TRACKING (Section 3.2) and prove that it is NP-complete (Section 3.3). In addition, we show that TRACKING can be solved in cubic time for graphs of bounded clique-width and linear time if the clique decomposition of bounded width is given in advance (Section 3.4).

[2]Notice that when tracking shortest paths only and using a single source-destination pair, these two versions of the problem are the same.

[3]Also called hypergraphs.

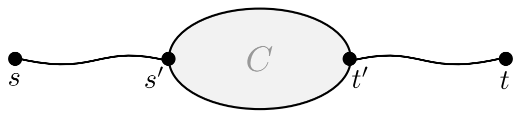


Figure 3.1: Entry-exit pair illustration, with entry vertex s' and exit vertex t'.

3.1.2 Definitions

Definition 3.1 (Entry-exit pair)**.** *Let (G, s, t) be an instance of* Tracking*. An* entry-exit pair *is, with respect to some simple cycle C in $G = (V, E)$, an ordered pair (s', t') of vertices in C that satisfy the following conditions:*

1. *There exists a path s-s' from s to the* entry *vertex s'*
2. *There exists a path t'-t from the* exit *vertex t' to t*
3. *Paths s-s' and t'-t are vertex-disjoint*
4. *Path s-s' (resp. t'-t) and C share exactly one vertex: s' (resp. t').*

Essentially, an entry-exit pair (s', t') with respect to a cycle C (see Fig. 3.1) represents two alternative s-t paths and, thus, requires tracking at least one of them. We say that (s', t') is *tracked with respect to C* if and only if $C \setminus \{s', t'\}$ contains a tracker. In addition, C is *tracked* if and only if there is no entry-exit pair with respect to C that is untracked. If a cycle contains either (i) 3 trackers or (ii) s or t and 1 tracker in a non-entry/non-exit vertex, then it must be tracked. We say that these cycles are *trivially tracked*.

An alternative characterization of a tracking set, first given by Banik, Choudhary, Lokshtanov, Raman and Saurabh [47, Lemma 2], is the following.

Lemma 3.1 ([47])**.** *For a graph $G = (V, E)$, a subset $T \subseteq V$ is a tracking set if and only if every simple cycle C in G is tracked with respect to T.*

3.2 Approximation algorithm

Theorem 3.1. *There exists a 4-approximation algorithm for* PLANAR-TRACKING*.*

The overall idea for the approximation algorithm builds on the following two insights:

1. The cardinality of the optimal solution cannot be much smaller than the number of faces in the graph.
2. The average number of trackers per face does not need to be very large.

The first idea gives us a lower bound on OPT, the cardinality of an optimal solution. The second gives us an upper bound on ALG, the cardinality of our approximation algorithm.

3.2.1 Lower bound on OPT

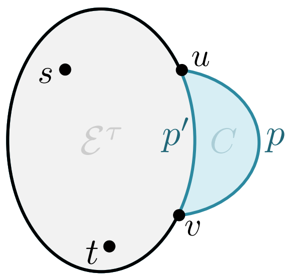
We consider the following reduction, which takes care of disconnected components, or components that are "attached" to the graph by a cut vertex that is not in an s-t path. We say that a reduction is *safe*, if it does not eliminate any untracked cycles.

Reduction 1. While there exists an edge or vertex that does not participate in any s-t path, remove it from the graph.

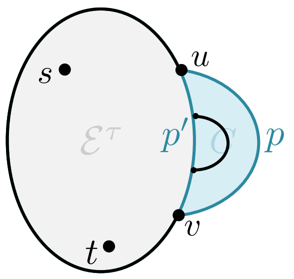
Lemma 3.2. *Reduction 1 is safe and can be done in polynomial time.*

Proof. See proof in [47]. □

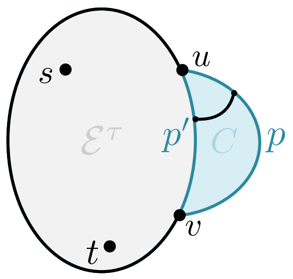
Lemma 3.3. *After Reduction 1, every simple cycle in the graph contains at least one entry-exit pair. This holds for non-planar graphs as well.*



(a) Valid



(b) Invalid



(c) Invalid

Figure 3.2: Illustration of valid and invalid choices of vertices u and v in the proof of Lemma 3.4. Cycle C must correspond to a face in the fixed embedding $\mathcal{E}$.

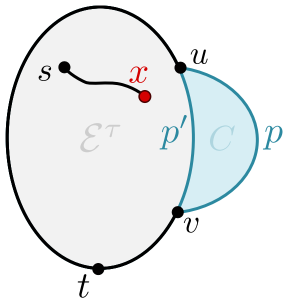
Proof. Let C by some simple cycle in the graph. After Reduction 1, there must exist an s-t path that shares an edge with C. The first and last vertices on this path that belong to C correspond to an entry-exit pair. □

Lemma 3.4. *In an embedded undirected planar graph G that results from Reduction 1, $OPT \geq (|F| - 1)/2$, where F is the set of faces of G.*

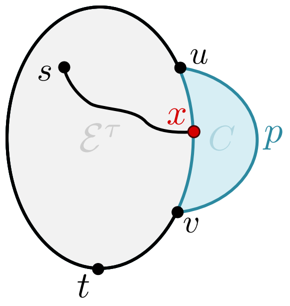
On a high level, the proof of Lemma 3.4 is done by keeping a set of "active" trackers while reconstructing a planar embedding $\mathcal{E}$ of G: we start, as a base case, with any simple s-t path in $\mathcal{E}$ and iteratively add faces to it until it matches $\mathcal{E}$. Given a fixed, optimal tracking set T^*, the addition of each face requires either (i) adding a new tracker from T^* to the active set, or (ii) deactivating an active tracker, rendering it useless for distinguishing paths on future faces. As a consequence, each tracker charges at most two faces: the one adding the tracker and the one deactivating it. This demonstrates that $|T^*| \geq (|F| - 1)/2$.

Let $\textsc{outer}(\mathcal{E}^\tau)$ be the set of outer-edges of our planar reconstructing embedding $\mathcal{E}^\tau$ at time τ. At time $\tau = 0$, our embedding corresponds to an s-t path and, for all $0 \leq \tau \leq |F| - 1$, we add exactly one face C from $\mathcal{E}$ to $\mathcal{E}^\tau$, by connecting two vertices u and v in $\textsc{outer}(\mathcal{E}^\tau)$ with a simple path p (see Fig. 3.2). In doing so, we erase an u-v path p' in $\textsc{outer}(\mathcal{E}^\tau)$, so we have that $\textsc{outer}(\mathcal{E}^{\tau+1}) = \textsc{outer}(\mathcal{E}^\tau) \setminus p' \cup p$. In the end, $\mathcal{E}^{|F|} = \mathcal{E}$.

By Lemma 3.3, there is at least one entry-exit pair in $\mathcal{E}$ with respect to face C, so any



(a) Tracker on x (red) is inactive due to the violation of Condition (i).



(b) Tracker on x (red) is inactive due to the violation of Condition (ii). Notice that x is entry for exit vertices u and v (with respect to C), therefore C needs another tracker.

Figure 3.3: Examples of inactive trackers used in the proof of Lemma 3.4.

tracking set must contain a tracker on some vertex of C. During the reconstruction process, we maintain a list of trackers in sets A and A', such that $(A \cup A') \subseteq T^*$, where A contains active trackers and A' contains inactive ones. A tracker in vertex v is *active* at time τ if and only if it meets both of the following conditions:

Condition (i) $v \in \textsc{outer}(\mathcal{E}^\tau)$

Condition (ii) There is no s-v path in $\mathcal{E}^\tau$ that traverses vertices in $\textsc{outer}(\mathcal{E}^\tau)$

Intuitively, an active tracker can be used to track future faces, although no more than one (see below). An inactive tracker, on the other hand, either cannot be used to track future faces (Condition (i)), or its corresponding vertex is entry/exit for some future face (Condition (ii)), in which case we require yet another tracker on that face (see Fig. 3.3). Condition (ii) is necessary for dealing with embeddings of G where at least one of s and t is not in the outer face.

Proof of Lemma 3.4. First, we argue that each time we add a face during the reconstruction process described above, we either (i) need to increase the number of active trackers (by adding it to either A or A'), or (ii) we can get away by re-using and, therefore, deactivating an active tracker.

We assume for the rest of the argument that t is on the outer face, because such a planar embedding is always possible to construct.

Let C be the face added a time τ by connecting vertices u and v in $\textsc{outer}(\mathcal{E}^{\tau})$, as specified above. In addition, let T^* be any optimal solution (i.e. $|T^*| = OPT$). We consider two cases, depending on the existence of a tracker in C at time τ:

Case 1: $\boldsymbol{C \cap A = \emptyset}$ at time τ.

By Lemma 3.3, there exists a vertex $x \in C$ such that $x \in T^*$. We place a tracker on x. If $x \in \textsc{outer}(\mathcal{E}^{\tau+1})$ we add x to A, otherwise we add it to A'.

Case 2: $\boldsymbol{C \cap A \neq \emptyset}$ at time τ.

Let $y \in C \cap A$ be a vertex of C with a tracker. We again consider two cases:

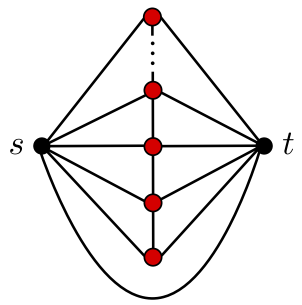
(i) $\boldsymbol{y \notin \{u, v\}}$. Then, $y \in \textsc{outer}(\mathcal{E}^{\tau})$ but $y \notin \textsc{outer}(\mathcal{E}^{\tau+1})$, which amounts to moving y from A to A'.

(ii) $\boldsymbol{y \in \{u, v\}}$. If (u, v) is an entry-exit pair with respect to C, or if the tracker in y is not active, then there exists $x' \in C \setminus \{u, v\}$ such that $x' \in T^*$. Similarly to Case 1, we place a tracker on x', which corresponds to adding x' either to A or A'.

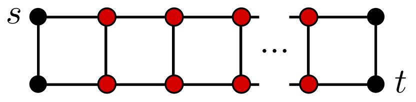
Otherwise, the tracker in y is active and (u, v) is not an entry-exit pair with respect to C. Let us assume without loss of generality that $y = u$. Then, the addition of C deactivates the tracker in u by definition of active tracker (Condition (ii) is now violated), so we move u from A to A'.

Every tracker in A' is charged by at most two faces: one for adding an active tracker to A and another for deactivating it and moving it to A'. Therefore, $|F| - 1 \leq |A| + 2|A'|$. Since $|A| + |A'| \leq |T^*|$, it follows that $|F| - 1 \leq 2OPT$.

$\square$



(a) Example of a planar graph where $OPT = |F|/2$ (in red).



(b) Example of a planar graph where $ALG = 2(|F| - 2)$ (in red).

Figure 3.4: Tight examples for the lower bound on OPT (left) and the upper bound on ALG (right) in planar graphs.

A tight example for the lower bound on OPT is illustrated in Fig. 3.4a.

3.2.2 Upper bound on ALG

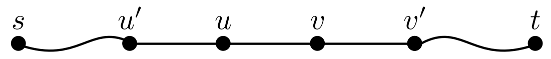
We say that an undirected planar graph is *reduced* if it cannot be further reduced by Reduction 1 or any of the following reductions.

Reduction 2. While there exist two adjacent vertices of degree 2, remove one of them (and its edges) and add an edge connecting its neighbors.

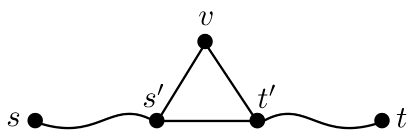
Reduction 3. While there exists vertex $v \notin \{s, t\}$ of degree 2 in a 3-cycle, place a tracker on v and remove it and its edges from the graph.

Reduction 4. While there exist non-adjacent vertices $u, v \notin \{s, t\}$ of degree 2 in a 4-cycle, place a tracker on either u or v and remove it and its edges from the graph.

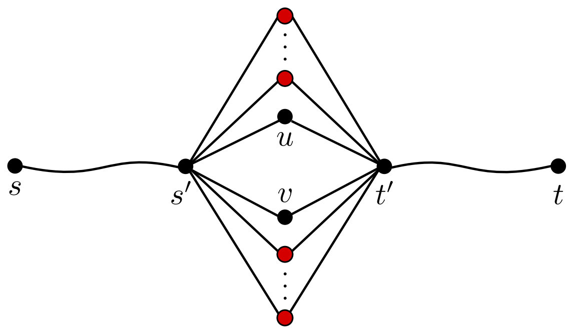
Notice that all reduction rules are valid for general graphs, not only planar ones. In addition, Reductions 2, 3 and 4 can be applied interchangeably and in any order until none of them is applicable, but we will see that they need to be carried out after Reduction 1. Fortunately, we will not be required to re-apply Reduction 1 after performing the remaining reductions.



(a) Illustration of Reduction 2, where $\deg(u) = \deg(v) = 2$.



(b) Illustration of Reduction 3, where $\deg(s') \geq 3$, $\deg(t') \geq 3$ and $\deg(v) = 2$.



(c) Illustration of Reduction 4, where $\deg(s') \geq 3$, $\deg(t') \geq 3$ and $\deg(u) = \deg(v) = 2$.

Figure 3.5: Illustration of Reductions 2, 3 and 4.

We denote the degree of a vertex v by $\deg(v)$, where the underlying graph can be determined from its context.

Claim 3.1. *If vertex v is on an entry-exit pair, then* $\deg(v) > 2$.

Proof. Trivial by Definition 3.1. □

Claim 3.2. *Reductions 2, 3 and 4 maintain the property that every cycle in G contains at least one entry-exit pair (see Lemma 3.3).*

Proof. Reductions 2, 3 and 4 only erase faces and vertices of degree 2, which cannot be in entry-exit pairs (by Claim 3.1), so every simple cycle of the graph still contains an entry-exit pair. □

Lemma 3.5. *Reduction 2 is safe and can be done in polynomial time, if done after Reduction 1.*

Proof. Banik, Choudhary, Lokshtanov, Raman and Saurabh [47, Lemma 16] showed that having three adjacent vertices of degree 2 is redundant. Here, we extend their proof and show that we can also get rid of the second adjacent vertex of degree 2. Let u and v be the two adjacent vertices of degree 2, and let u' be the other neighbor of u. Notice that

v, u' are not adjacent, otherwise the u, v edge would then not belong to an s-t path and thus be removed by Reduction 1. Hence, no parallel edges are added to the graph after this reduction. We argue that this reduction is safe by showing that there exists a minimum tracking set that does not include u and v simultaneously. Take any minimum tracking set T that includes u and v. We can always move the tracker from u to u'; this remains a tracking set, because u immediately follows or precedes u' on any s-t path. This can only decrease T's cardinality. This reduction can be done in time polynomial in n, by repeatedly checking for the existence of adjacent vertices of degree 2. □

Lemma 3.6. *Reduction 3 is safe and can be done in polynomial time, if done after Reduction 1.*

Proof. Let $v \notin \{s, t\}$ be the vertex of degree 2 in the triangle $\Delta vs't'$ (see Fig. 3.5b). Then, it must be the case that s' and t' form an entry-exit pair. This follows from (i) the fact that there exists an entry-exit pair in $\Delta vs't'$ (by Lemma 3.3) and (ii) the fact that v cannot be in an entry-exit pair (by Claim 3.1). Then, by Lemma 3.1, any feasible solution must place a tracker on v. Therefore, v and its edges can be removed, since v is neither a cut-vertex, nor in any entry-exit pair, so that its removal does not eliminate an untracked cycle. Clearly, this reduction can be done in $O(\text{poly}(n))$ time. □

Lemma 3.7. *Reduction 4 is safe and can be done in polynomial time, if done after Reduction 1.*

Proof. Let $u, v \notin \{s, t\}$ be the two vertices of degree 2 that connect the same pair of vertices s' and t'. Similarly to the proof of Lemma 3.6, s' and t' must form an entry-exit pair. So, by Lemma 3.1, any feasible solution must place a tracker in either u or v. By symmetry, we can track and remove u and its edges. Therefore, Reduction 4 is safe by Claim 3.1 and the facts that u is neither a cut-vertex nor in an entry-exit pair. □

Remark 3.1. *None of Reductions 2, 3 and 4 compromise planarity.*

```
Algorithm $\mathcal{A}$
Input: Undirected planar graph $G = (V, E)$ and vertices $s \in V$ and $t \in V$.
Output: Tracking set.

1. Perform Reduction 1 in $G$.
2. Perform Reductions 2, 3 and 4 repeatedly until $G$ is reduced.
3. Output remaining vertices of degree at least 3 (except $s$ or $t$)
```

Lemma 3.8. *Algorithm $\mathcal{A}$ outputs a tracking set for the input graph G.*

Proof. By Lemmas 3.2, 3.5, 3.6 and 3.7, Reductions 1-4 are safe, so let us assume without loss of generality that G is reduced. Then, every cycle of G of 5 or more vertices is trivially tracked, because it must contain at least 3 vertices of degree at least 3 (by Reduction 2). Similarly, every 3- or 4-cycle must contain at least 3 vertices of degree at least 3 by Reduction 2 and Reductions 3 and 4 (respectively). □

Lemma 3.9. *Algorithm $\mathcal{A}$ outputs a tracking set of size at most $2(|F| - 2)$, where F is the set of faces of the input graph G.*

Proof. Notice that each tracker added during Reductions 3 and 4 is associated with the removal of one face from G. Therefore, it is enough to show that the lemma holds with respect to a reduced graph G. Let us partition V into $V = (V_2 \cup V_{\geq 3})$, where V_2 and $V_{\geq 3}$ consist of the vertices of degree 2 and degree at least 3, respectively (notice that there cannot be vertices of degree 1). The lemma statement follows from Lemma 3.8 and the fact that $|F| \geq \frac{|V_{\geq 3}|}{2} + 2$. This inequality can be derived by plugging the inequality $2|E| \geq 3|V_{\geq 3}| + 2|V_2|$ in Euler's formula for planar graphs: $|V| - |E| + |F| = 2$, where E is the set of edges of G. □

A tight example is illustrated in Fig. 3.4b.

Proof of Theorem 3.1. By Lemmas 3.4 and 3.9, Algorithm $\mathcal{A}$ is a 4-approximation to PLANAR-TRACKING. □

3.3 Hardness of tracking paths

We show that PLANAR-TRACKING is NP-hard, by reducing from PLANAR-3-SAT, a special version of the satisfiability problem, shown to be NP-complete by Lichtenstein [226].

In 3-SAT, we are given a set $\mathcal{X} = \{x_1, \ldots, x_p\}$ of variables and a 3-CNF formula ϕ, where each clause in ϕ is a disjunction of exactly three distinct literals with respect to $\mathcal{X}$. The goal is to find a boolean assignment to all variables in $\mathcal{X}$ that satisfies ϕ. Consider the bipartite graph with a vertex for each clause C in ϕ and each variable $x_i \in \mathcal{X}$, and edges (x_i, C) if and only if C contains x_i or its negation $\overline{x_i}$. Lichtenstein [226] showed that PLANAR-3-SAT, the subset of instances of 3-SAT whose underlying bipartite graph is planar, remains NP-complete. In particular, the definition of PLANAR-3-SAT requires that a cycle can be drawn connecting all of the variables while maintaining planarity. Later, Knuth and Raghunatan [214] exploited this condition to show that we can always draw the underlying bipartite graph of a PLANAR-3-SAT instance in a *rectilinear* fashion without crossings (example in Fig. 3.6): variables are arranged in a horizontal line and clauses are horizontal line segments with vertical legs to represent the literals present in the clause. Vertical legs attach to the appropriate variables and are labeled *red* for negated literals and *blue*, otherwise. In particular, a given clause is drawn completely above or below the line of variables.

We convert a planar rectilinear drawing $\mathcal{D}$ of an instance of PLANAR-3-SAT, with formula ϕ and a set $\mathcal{X}$ of variables, into a planar drawing $\mathcal{G}$ corresponding to the instance of PLANAR-TRACKING. The reduction is straightforward:

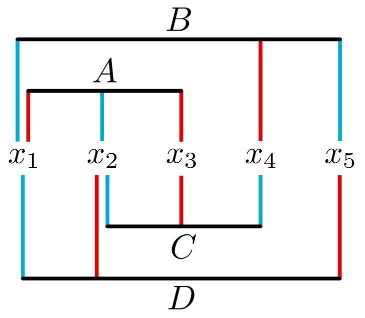


Figure 3.6: Example of a PLANAR-3-SAT instance drawn in a rectilinear fashion with clauses $A = (\overline{x_1} \vee x_2 \vee \overline{x_3})$, $B = (x_1 \vee \overline{x_4} \vee x_5)$, $C = (x_2 \vee \overline{x_3} \vee x_4)$, $D = (x_1 \vee \overline{x_2} \vee \overline{x_5})$.

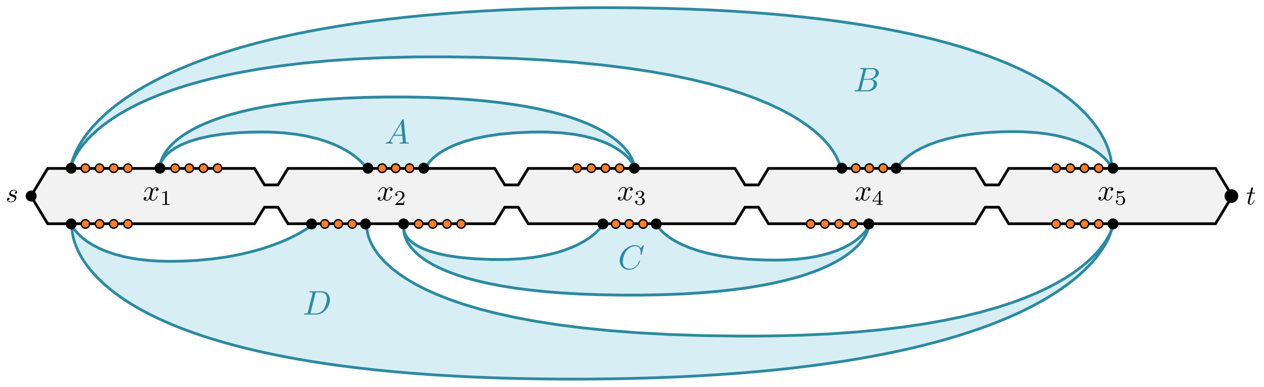


Figure 3.7: Example of a PLANAR-TRACKING instance. The orange vertices ensure that no cycle is untracked (see Fig. 3.9).

(i) transform each variable x_i in $\mathcal{D}$ into a gadget containing m_i copies of literal vertices x_i and $\overline{x_i}$;

(ii) transform each 3-legged clause into a face containing corresponding literals vertices and an entry-exit pair;

(iii) choose the boolean assignment according to the placement of trackers, such that a clause is satisfied if and only if its corresponding face is tracked.

The union of all the variable and clause gadgets constitutes $\mathcal{G}$ (see example in Fig. 3.7). Details of each gadget are given below. For simplicity, we avoid introducing too many subscripts and we rely on pictures to describe the gadgets.

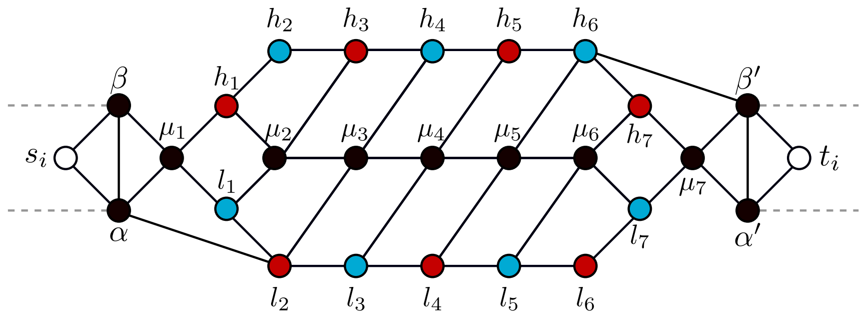


Figure 3.8: Illustration of x_i's gadget, containing m_i vertices for x_i (in blue) and m_i vertices for $\overline{x_i}$ (in red). Vertices colored black require trackers in any minimum tracking set. The dashed edges are added to force trackers in s_i and t_i.

3.3.1 Variable gadget

Each variable gadget converts a variable x_i in $\mathcal{D}$ into a connected subgraph corresponding to Fig. 3.8, with length parameterized by m_i. We refer to the set $\{h_k, \mu_k, l_k\}$ as column k and we refer to the vertices $\{h_1, \ldots, h_{m_i}\} \cup \{l_1, \ldots, l_{m_i}\}$ as *literal vertices*.

Each variable gadget is linked with the next one by setting $t_i = s_{i+1}$, to form a horizontal chain of gadgets, where $s = s_1$ and $t = t_p$. For convenience, we force trackers in all the s_i (except s), by drawing edges between the α' (β') of a variable gadget and the α (β) of the next variable gadget in the chain. We refer to the resulting drawing as the *spine*.

There are exactly two minimum tracking sets associated with the variable gadget with source s_i and destination t_i. One of them corresponds to a *true assignment* of x_i and the other one to a *false assignment*. Both of them require tracking the vertices in $R = \{\alpha, \alpha', \beta, \beta', \mu_1, \mu_{m_i}\}$, as well as the remaining μ_k. In addition, the true assignment tracks the even-indexed h_k and odd-indexed l_k, while the false assignment tracks the odd-indexed h_k and even-indexed l_k. This requires $2m_i + 4$ trackers in total. We will show that these true and false assignments are the only minimum tracking sets. To do that, we first prove a couple of helpful lemmas and make a few observations:

- Each vertex in R belongs to a triangle, such that the other two vertices form an entry-exit pair, so must be tracked.

- Each square face, besides the two including h_2 and l_{m_i-1}, requires 3 tracked vertices.
- Two adjacent vertices cannot both be untracked.

Lemma 3.10. *The true/false assignments of x_i correspond to tracking sets with respect to x_i's gadget with source s_i and destination t_i.*

Proof. In a true/false assignment of x_i, every face of the gadget is trivially tracked, except the faces including s_i/t_i (which are clearly tracked) and the faces including h_2 and l_{m_i-1}. Though the latter faces may only contain 2 trackers (depending on x_i's truth value and/or m_i's parity), they are nevertheless tracked because one of these trackers cannot be in an entry-exit pair. The remaining cycles, i.e. the ones which are not faces, are also trivially tracked: since these cycles must have size at least 6, they must contain 3 trackers given the observation that no adjacent vertices are untracked. □

Lemma 3.11. *In a minimum tracking set, each column has exactly 2 trackers.*

Proof. Since the true/false assignments achieve this property, we only have to show that each column requires at least two trackers. Assume that a minimum tracking set has only one tracker in column $1 < k < m_i$; it must be on μ_k. Then all vertices on columns $k-1, k+1$ must be tracked. For the average number of trackers per column to be at most 2, the number of trackers per column must be an alternating sequence of $\dots 3, 1, 3, 1, \dots$, with column 1 and/or column m_i only having 1 tracker, which contradicts the square face property. □

We are now in conditions of proving our claim regarding the variable gadget.

Lemma 3.12. *The true and false assignments are the only minimum tracking sets.*

Proof. Assume that some μ_k is not tracked in a minimum tracking set, for $1 < k < m_i$. By Lemma 3.11, we only have to show that this causes a column to have 3 trackers. If $k = m_i - 1$ then column m_i must have three trackers. Otherwise, $\mu_{k+1}, h_{k+1}, h_{k+2}$ must be tracked, since they share a square face with μ_k. If l_{k+1} is tracked we are done, otherwise, l_{k+2} must be tracked. Additionally, μ_{k+2} must be tracked, either by the square face property, or in the case where $k = m_i - 2$, because it is in R. Then, column $k + 2$ has 3 trackers.

Now, if all the μ_k are tracked, then the only minimum tracking sets are the true and false assignments, by the observation that two adjacent vertices cannot both be untracked and Lemma 3.10. □

3.3.2 Clause gadget

Let $C = (\ell_a \vee \ell_b \vee \ell_c)$ be a clause in ϕ with literals corresponding to variables $x_a, x_b, x_c \in \mathcal{X}$. Its gadget, depicted in Fig. 3.9, is a face F_C consisting of:

- literal vertex α from x_a's gadget, corresponding to literal ℓ_a;
- adjacent literal vertices $\beta_1, \overline{\beta_1}, \beta_2, \overline{\beta_2}, \beta_3$ from x_b's gadget, corresponding to literals alternating between ℓ_b and $\overline{\ell_b}$;
- literal vertex γ from x_c's gadget, corresponding to literal ℓ_c;
- edges $(\alpha, \gamma), (\alpha, \beta_1), (\beta_3, \gamma)$ and the edges from x_b's gadget connecting all of the β_k and $\overline{\beta_k}$.

We can increase the lengths of the variable gadgets to any polynomial that provides enough literal vertices for all clauses. Since $\mathcal{D}$ is planar, there are no crossings between clauses. We also impose the following restrictions:

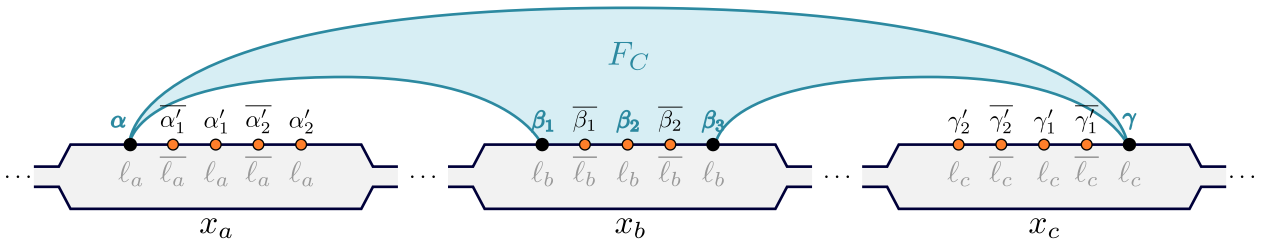


Figure 3.9: Illustration of the gadget for clause $C = (\ell_a \vee \ell_b \vee \ell_c)$, where the vertices in each variable gadget are all adjacent. The entry-exit $(\overline{\beta_1}, \overline{\beta_2})$ are responsible for satisfying C.

1. α cannot be one of $\{h_1, l_{m_a}\}$; this ensures that the only faces in x_a's gadget that do not require 3 trackers do not become untracked. We apply the equivalent restriction to β_1, β_3 and γ.

2. The $\alpha'_k/\overline{\alpha'_k}$, (see Fig. 3.9) cannot belong to any other clause gadget; these correspond to the 4 literal vertices following α in x_a's gadget and reserving them ensures that non-clause faces, between nested clauses, are tracked. We apply the equivalent restriction to the $\gamma'_k/\overline{\gamma'_k}$.

3. All literal vertices in a clause need to be on the same side of the spine; this restriction is trivial because $\mathcal{D}$ is rectilinear, but it simplifies the analysis.

Lemma 3.13. *Clause C is satisfied if and only if its corresponding gadget face F_C is tracked.*

Proof. The entry-exit $(\overline{\beta_1}, \overline{\beta_2})$ is the only one, with respect to F_C, that is tracked if and only if C is satisfied. The remaining entry-exit pairs are tracked by either a β_k or a $\overline{\beta_k}$. □

Theorem 3.2. *There exists a polynomial time reduction from* Planar-3-sat *to* Planar-Tracking.

Proof. Let $(G, \mathcal{G})$ be an instance of Planar-Tracking that results from applying the transformation described above to an instance $(\phi, \mathcal{D})$ of Planar-3-sat, where $\mathcal{G}$ and $\mathcal{D}$ are the underlying planar embeddings. We show that ϕ is satisfiable if and only if G has a tracking set of size $T = (\sum_{i=1}^{p} 2m_i + 4) + p - 1$.

($\Leftarrow$) Choose the truth assignment of each variable according to the given tracking set. The implication follows from Lemma 3.12 and the fact that if some clause in ϕ was not satisfied, then its gadget face would have been untracked, a contradiction.

($\Rightarrow$) Place T trackers on the literal vertices that correspond to the satisfiable truth assignment of every variable and on all non-literal vertices (except s and t). We show that this corresponds to a tracking set by arguing that every cycle C in G is tracked. We distinguish between two cases:

Case 1: C contains no clause edges.

Then C is tracked by (almost) the same argument given in Lemma 3.10 that shows that a truth assignment of x_i corresponds to a tracking set with respect to x_i's gadget. Notice that, because of Restriction 1, clause edges are only added to faces that require 3 trackers, so this does not change the argument for the faces which do not require 3 trackers. The only differences are: (i) the addition of the edges that force trackers in every s_i, which only helps the argument, and (ii) the fact that C may span multiple variable gadgets, in which case C must traverse at least 3 trackers on the non-literal vertices between two variable gadgets.

Case 2: C contains clause edges.

Notice that, by construction of G, C must have at least one spine edge. If C corresponds to a clause face, then it must be tracked by Lemma 3.13. Otherwise, we show that C contains at least 3 trackers and, thus, is trivially tracked. Let us think of C as alternating non-empty paths of two types: *clause paths*, which only contain clause edges and *spine paths*, which only contain spine edges. To avoid dealing with complex cycles, we observe that each spine path in C must contain at least 1 tracker; this follows from the fact at least one of the 2 vertices sharing a spine edge must have a tracker (see variable gadget). Thus, let us assume that C

contains no more than 2 spine/clause paths, or otherwise C immediately contains 3 trackers. Notice that if one of the spine paths spans 2 or more variable gadgets, then it must traverse at least 3 trackers on the non-literal vertices between two variable gadgets. Since every clause in ϕ contains exactly 3 distinct literals and C is simple, the only cases where none of the spine paths span multiple variable gadgets are the following:

(i) C contains exactly 2 clause paths in different sides of the spine.

Then, the 2 spine paths connecting the two sides of the spine must traverse 2 trackers each (see variable gadget). Therefore, C contains at least 4 trackers.

(ii) C contains exactly 2 clause paths on the same side of the spine, which both start or both end at the same variable gadget, one nested in the other.

Then, by Restriction 2 one of the spine paths contains at least 6 vertices, half of which must be tracked.

□

Corollary 3.1. Planar-Tracking *is NP-hard.*

It remains to show that Planar-Tracking is in NP; Banik, Choudhary, Lokshtanov, Raman and Saurabh [47] prove this in the more general case of Tracking.

3.4 Bounded clique-width graphs

We show that Tracking can be solved in linear time when the input graph has bounded clique-width, by applying Courcelle's theorem [113, 116, 118], a powerful meta-theorem that establishes fixed-parameter tractability of *any* graph property that is expressible in monadic second order logic.

Clique-width, first introduced by Courcelle, Engelfriet and Rozenberg [117] and revisited by Courcelle and Olariu [120], is an important graph parameter that, intuitively, measures the closeness of a graph to a cograph – a graph with no induced 4-vertex paths. It is closely related to *tree-width*, another influential graph parameter that measures closeness of a graph to a tree and that was first introduced by Bertelé and Brioschi [59] and later rediscovered by Halin [183] and Robertson and Seymour [264]. While both parameters are determined based on specific hierarchical decompositions of a graph, the clique-width is strictly more powerful in the sense that the class of graphs of bounded clique-width includes all graphs of bounded tree-width, but not vice-versa. Details on the relationship between these parameters can be found in Courcelle and Engelfriet [116]. Graphs of bounded clique-width include series-parallel graphs, outerplanar graphs, pseudoforests, cographs, distance-hereditary graphs, etc.

$\mathbf{MSO_1}$ vs $\mathbf{MSO_2}$. Second order logic extends first order logic, by allowing quantification over relations (of any fixed arity) on the elements of the domain of discourse. Monadic second order logic itself only allows quantification over unary relations (subsets of the domain of discourse) and, in the logic of graphs, it comes in two flavors: MSO_1 and MSO_2. The only distinction between these is that the latter allows edges to be elements of the domain of discourse (and thus be quantified over), while the former does not. Besides the quantifiers ($\forall$ and $\exists$) and the standard logic operations $\neg, \wedge, \vee, \rightarrow$, both logics include predicates for equality ($=$) and relation membership ($\in$). In addition, MSO_1 includes a predicate ($\sim$) that determines vertex adjacency and MSO_2 includes a predicate for vertex-edge incidence. MSO_2 is more expressive, for example: Hamiltonicity can be expressed using MSO_2, but not using MSO_1. Details on the distinction between the two logics can be found in [116].

Courcelle's theorem. Courcelle, Makowsky and Rotics [113, 118] showed that any graph property expressed in MSO_1 or MSO_2 is FPT under clique-width and tree-width (respectively). More specifically, they showed that any MSO_1-, MSO_2-expressible property can be tested in $f(k,l) \cdot n$ and $g(k',l') \cdot (n+m)$ time (respectively) for graphs of clique-width k and tree-width k', where: f and g are computable functions, l and l' are the lengths of the logic formulas, n is the number of vertices and m is the number of edges. The result for MSO_2 is valid in optimization problems with linear evaluation functions [119], and so is the result for MSO_1 [118]. Examples of constructing FPT graph algorithms parameterized by clique-width or tree-width, which are based on automata, are given in [114, 115]. While it is possible to construct tree decompositions of width k' in linear time [67], there is no FPT algorithm for finding clique decompositions of clique-width $k > 3$. Fortunately, it is possible to construct a clique decomposition of width exponential in k in cubic time [254]. In this section, we will take advantage of the latter.

We give an alternative definition for tracking set that is easier to express using the logic of graphs.

Lemma 3.14 (Tracking set)**.** *For an undirected graph $G = (V, E)$, a subset $T \subseteq V$ is a tracking set if and only if there is* no *s-t path $P_{st} = P_{ss'} \cup P_{s't'} \cup P_{t't}$ for $s', t' \in V$ and corresponding s-s', s'-t' and t'-t paths, such that:*

1. *There exists an alternative s'-t' path $P'_{s't'} \neq P_{s't'}$ and*
2. *$T \cap (P_{s't'} \cup P'_{s't'}) \subseteq \{s', t'\}$.*

Proof. This follows directly from Lemma 3.1. □

Next, we present the logic formulas required. We use lowercase letters to quantify over vertices and uppercase letters to quantify over sets of vertices. Further, we use s and t as free variables and, for convenience, we also use set intersection ($\cap$), union ($\cup$) and containment ($\subseteq$), without explicitly expressing these operations using MSO_1.

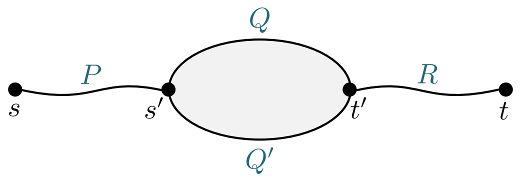


Figure 3.10: Illustration of variable sets P, Q, Q' and R as well as vertex variables s, s', t' and t used in expressing IsTrackingSet using MSO$_1$.

$$
\begin{aligned}
&\textsc{IsTrackingSet}(T, s, t)\\
&\iff \nexists P, Q, R\ [\ \exists s', t'\ [\quad \textsc{HasPath}(P, s, s') \ \wedge\ \textsc{HasPath}(Q, s', t') \ \wedge\ \textsc{HasPath}(R, t', t)\\
&\qquad\qquad\qquad\qquad\qquad \wedge\ P \cap Q = \{s'\} \ \wedge\ Q \cap R = \{t'\} \ \wedge\ P \cap R = \emptyset\\
&\qquad\qquad\qquad\qquad\qquad \wedge\ \exists Q' \neq Q \quad [\textsc{HasPath}(Q', s', t') \ \wedge\ T \cap (Q \cup Q') \subseteq \{s', t'\}\ ]]]
\end{aligned}
$$

The first two lines of the above equivalence establish that P, Q and R form an s-t path. The last line restricts s' and t' to be an entry-exit pair with respect to the cycle $Q \cup Q'$ (see Fig. 3.10) and, in addition, establishes that the cycle $Q \cup Q'$ is not tracked.

The primitive HasPath(X, a, b), whose input consists of a set $X \subseteq V$ and vertices $a, b \in V$, verifies the existence of a simple path between a and b that only uses vertices in X. We define it as follows:

$$
\begin{aligned}
&\textsc{HasPath}(X, a, b)\\
&\iff \nexists X_1, X_2 \subseteq X\ [\ X_1 \cup X_2 = X \ \wedge\ a \in X_1 \ \wedge\ b \in X_2\\
&\qquad\qquad\qquad\qquad \wedge\ \neg\, (\exists u \in X_1 \ \wedge\ \exists v \in X_2\ [u \sim v\ ])]
\end{aligned}
$$

Remark 3.2. HasPath(X, a, b) *is correctly expressed under MSO$_1$ and it correctly verifies that there exists an a-b path using only vertices in X.*

Remark 3.3. IsTrackingSet *is correctly expressed under MSO$_1$ and it correctly verifies that the given subset of vertices is a tracking set.*

Theorem 3.3. Tracking(G, s, t) *can be solved in polynomial time if G has bounded clique-width. Moreover, if a clique decomposition of bounded width is given, it can be solved in* *linear time.*

Proof. This follows directly from Lemma 3.14, Remarks 3.2 and 3.3, and the linear time algorithm given by Courcelle [118] for any optimization problem on graphs of bounded clique-width, whose decomposition is given in advance. □

3.5 Conclusion and Open Questions

We showed that PLANAR-TRACKING is NP-complete and we give a 4-approximation algorithm. We also show that, for graphs of bounded-clique width, TRACKING can be solved in linear time by applying Courcelle's theorem, as long as its clique decomposition is given in advance. A natural direction of future study would be to improve the approximation ratio of PLANAR-TRACKING, establish constant approximation factors for graphs of larger genus or, more generally, for TRACKING. We address these two latter issues in Chapter 4.

Another open question is to establish the difficulty of TRACKING on directed graphs: on one side planar directed acyclic graphs are not known to be NP-hard and, on the other side, it is not known whether its general or planar versions are in NP (Fig. 4.9 in Chapter 4). Finally, it would be interesting to find efficient algorithms for graphs of bounded tree-width or clique-width without resorting to the finite automaton approach used in Courcelle's theorem.

Chapter 4

Learning Paths in Minor-Free Graphs

4.1 Introduction

We consider the problem of learning paths in a graph, introduced in Chapter 3 in the context of planar graphs. For more information on background and on what motivated us to study this problem, we refer the reader to Chapter 3. In this chapter, we provide provably good solutions to the same problem on larger classes of graphs, including H-minor-free and general graphs. The former is an important generalization of planar graphs (see Fig. 4.1) that could prove useful when studying this problem in the context of *road networks*, which may admit non-constant genus [152]. Like planar graphs, the family of H-minor-free graphs is minor-closed and, thus, is characterized by a finite set of forbidden minors, those graphs which cannot be minor of the graph in the family. But, as pointed out in [152], we do not have evidence of any particular forbidden minors for road networks. Still, studying Tracking for the class of H-minor-free graphs can provide valuable insight into better approximation ratios for more general graphs.

We design a $(1 + \epsilon)$-approximation algorithm and an FPT algorithm with a linear kernel for

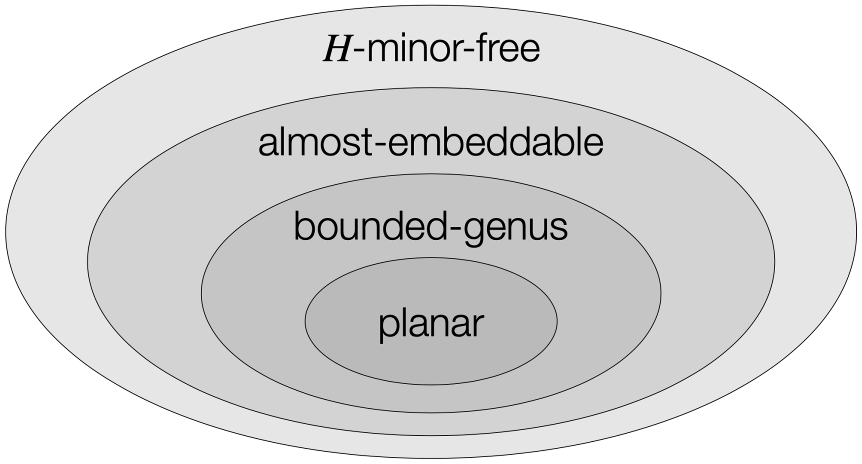


Figure 4.1: Non-exhaustive illustration of the hierarchy of known classes of sparse graphs, ranging from planar graphs, at the center, to H-minor-free graphs, circling all other classes. Planar graphs are those with genus 0. Bounded-genus graphs have positive, but $O(1)$ genus. Almost-embeddable graphs are, essentially, bounded-genus graphs with a constant number of vortices and apex vertices. H-minor-free graphs are clique-sums of almost-embeddable graphs, as characterized by Robertson and Seymour [265].

H-minor-free graphs. For general graphs, we give the first approximation algorithm for the vertex-weighted version of the problem, with a ratio of $O(\lg n)$. We also show that the dual set system for the unweighted version of the problem has bounded VC-dimension, which immediately gives us a $O(\lg OPT)$-approximation in this setting, where OPT is the size of an optimal tracking set.

Let us denote by WeightedTracking the vertex-weighted version, whose goal is to find a tracking set of least total weight. Further, we denote by k-Tracking the decision version of Tracking, which asks whether there exists a tracking set of size at most k (for any given integer k). For conciseness, we refer to the "tracking set of G", when s and t are clear from context.

Directly Related Work. Motivated from the additional applications of tracking animals in migration networks, tracking intruders in buildings, and tracking malicious packets in computer networks, the tracking paths problem was first introduced by Banik, Choudhary, Lokshtanov, Raman and Saurabh [46, 47], who showed that the decision version of this problem is NP-complete, by a reduction from Vertex Cover and showing containment in NP by observing that every tracking set must be a feedback vertex set in a subgraph which excludes redundant components. Recall that a *feedback vertex set* (FVS) is a set of vertices whose removal results in an acyclic graph. In addition, they give the first fixed-parameter tractable (FPT) algorithm, obtaining a kernel with $O(k^7)$ edges, when parameterized by the solution size k. This has since been improved to $O(k^2)$ edges for general graphs and $O(k)$ edges for planar graphs [102]. Other parameterizations also yield FPT algorithms for Tracking, including the size of vertex cover and the size of cluster vertex deletion set [103]. Eppstein, Goodrich, Liu and Matias [151] show that the tracking paths problem remains NP-hard even when the input graph is planar and they give a 4-approximation in this case. Until now, the only other known approximation algorithm was a $2(\Delta + 1)$-approximation for degree-Δ graphs [101], which is achieved by taking a 2-approximate FVS and all its neighbors. Optimal polynomial time algorithms have been derived when the graph has bounded clique-width [151] (linear time if the corresponding clique-decomposition is given in advance), as well as for chordal and tournament graphs [101].

Tracking Shortest Paths. The tracking paths problem is related to an earlier *tracking set for shortest paths* (TSSP) problem, which was first studied by Banik, Katz, Packer and Simakov [48]. In the TSSP problem, only shortest s-t paths need to be tracked, thus allowing one to model the input as a directed acyclic graph. They show that this variant is APX-hard (and thus, NP-hard), and they present a 2-approximation for the planar version of the TSSP problem, which is a variant for which we still have no hardness results. Bilò, Gualà, Leucci and Proietti [62] generalized the TSSP problem, allowing for the existence of multiple start-finish pairs and requiring tracking sets to distinguish between any two shortest paths between any two start-finish pairs. In this setting, they give a $O(\sqrt{n \log n})$-approximation algorithm. They further study a version of this problem in which the set of trackers (ignoring the order in which they are traversed) is itself enough to distinguish between any two start-finish shortest paths[1], and they present a $O(\sqrt{n})$-approximation algorithm in this case. Moreover, they prove that both of these settings are NP-hard even for cubic planar graphs, by a reduction from Vertex Cover. Finally, They also give an FPT algorithm (parameterized by the maximum number of vertices at the same distance from the start s) for the case of a single start-finish pair, which is the original TSSP problem introduced by Banik, Katz, Packer and Simakov [48].

Other Related Results. Other related work includes work by Banik and Choudhary [45], who consider a version of Tracking on hypergraphs. which asks for the smallest subset of vertices whose intersection with each hyperedge is unique. They prove fixed-parameter tractability of this problem, by showing a correspondence with the Test Cover problem.

When tracking shortest paths, Banik, Katz, Packer and Simakov [48] also provide a data structure of size $O(n|T|)$ for an n-vertex graph and a tracking set T which, given the subset of visited trackers (order does not matter), reconstructs the traversed shortest s-t path P in $O(|P|)$ time. They also give an optimal polynomial time algorithm for the related problem *Catching the Intruder*, which asks for the smallest subset of vertices T such that every shortest s-t path that visits a vertex of T if and only if it visits a vertex of a given set of forbidden vertices.

When tracking all paths, Choudhary [101] also present a reconstruction algorithm, which, given the set of trackers and the sequence of traversed trackers, reconstructs the corresponding s-t path in polynomial time for a fixed number of trackers. Moreover, they consider a version Tracking where one places trackers on weighted edges, instead of (unweighted) vertices. They show that this problem can be solved optimally in polynomial time, by proving that it is equivalent to finding a minimum weighted feedback edge set (i.e. a set of edges

[1]Notice that, when there is a single start-finish pair, the order of traversed trackers is no longer advantageous, rendering these two versions of the problem equivalent.

whose removal leaves an acyclic graph), which in turn is equivalent to finding a minimum spanning tree.

Also related is the NP-hard problem of finding a minimum *Loop Cutset* (see e.g. [283]), which asks for a subset of vertices in a directed graph of minimum weight which covers every cycle (ignoring edges direction), where a cycle C is covered if we select at least one vertex in C with positive outdegree in C. Becker and Geiger [51] showed that this problem admits a 2-approximation, by reducing it to the minimum weight FVS problem (unfortunately, this reduction does not translate to Tracking). Later, Becker, Bar-Yehuda and Geiger [50] gave a randomized algorithm that outputs the optimal loop cutset with probability at least $1-(1-\frac{1}{6^k})^{c6^k}$ after $O(c \cdot 6^k n)$ steps, where $c > 1$ is a constant chosen by the user, n is the number of vertices and k is the size of an optimal loop cutset. Finding small loop cutsets is a crucial step in Pearl's method of conditioning [256, 257], an algorithm for probabilistic inference in Bayesian networks (i.e. it computes posterior distribution of variables given new evidence, according to Bayes' Rule).

Our Contributions. Our results are summarized below:

1. **Linear kernel for H-minor-free graphs**. Previously, we only knew of a linear kernel for planar graphs [102]. This result also immediately implies an efficient $O(1)$-approximation.

2. **$(1+\epsilon)$-approximation for H-minor-free graphs**. Previous best was a 4-approximation for planar graphs [151].

3. **$O(\lg OPT)$-approximation for Tracking**, where OPT denotes the cardinality of an optimal tracking set. This is the first algorithm for general graphs with a non-trivial approximation ratio. Previously, we only knew of a $O(\sqrt{n \lg n})$-approximation for tracking shortest paths only and its algorithm requires solving a complex dynamic

program [62].

4. $O(\lg n)$**-approximation for** WeightedTracking. This is the first approximation for weighted graphs, among all variants of Tracking above mentioned.

5. **Improvements to the quadratic kernel for general graphs of** [102]. We simplify the kernelization algorithm and reduce the constants in the kernel size, while also completing the case analysis in a proof of a lemma central to the kernelization of [102].

4.2 Preliminaries

Notation and Definitions. A graph is *simple* if it does not contain any self-loops or parallel edges. We denote by $V(G)$ and $E(G)$ the set vertices and edges, respectively, of a graph G. Let us use $G - U$ (resp. $G - u$) to denote the subgraph of G induced by $V(G) \setminus U$ (resp. $V(G) \setminus \{u\}$). The degree of a graph G is the largest degree $\deg(v)$ among all vertices v in G. The neighborhood $N_G(u)$ of a vertex u w.r.t. G is the set vertices of G adjacent to u (when it is clear from the context, the subscript in $N_G(u)$ is omitted). The neighborhood $N_G(U)$ of a vertex set U is simply the union of the neighborhoods for all vertices u in U. We denote a bipartite graph by $(U \cup V, E)$, with vertex set partitioned into U and V, and edge set $E \subseteq U \times V$. The block-cut tree of a graph is the tree of biconnected components, and a cut-vertex is a vertex shared by at least two biconnected components, such that its removal disconnects the graph (see [189] for more information). Finally, a graph that can be obtained from a graph G by a sequence of edge contractions, edge deletions or vertex deletions is a *minor* of G.

Approximation Algorithms. An algorithm is an *α-approximation* (algorithm) if it returns a solution X whose cardinality is within α of an optimal solution X^*, i.e. $|X| \leq \alpha|X^*|$

for minimization problems and $|X| \geq \alpha|X^*|$ for maximization problems. We call X an α-*approximate* solution.

Fixed-parameter Tractability. A decision problem parameterized by k admits a *kernel* if there exists a *kernelization* algorithm that outputs, in time polynomial in both k and the size of the instance, a decision-equivalent instance (the kernel) whose size is bounded by $f(k)$, for some computable function f. If the kernel size is linear (resp. quadratic) in k, we say that the problem admits a *linear* (resp. *quadratic*) kernel. It is well known that a problem admits a kernel if and only if it is fixed-parameter tractable [123], i.e. it can be solved in $g(k) \cdot n^{O(1)}$ time, for some parameter k, computable function g and problem size n. A kernelization algorithm typically consists of the application of a fixed set of reduction rules, some of which are *rejection* rules – these return trivial NO-instances, of $O(1)$ size, indicating a negative answer to the decision problem. For more information on kernelization algorithms and parameterized complexity, we refer the reader to [123, 141, 253].

4.3 Structural Properties

Definition 4.1 (Entry-exit subgraph)**.** *Let* (G, s, t) *be an instance of* TRACKING*. An* entry-exit subgraph *is a triple* (G', s', t')*, where* G' *is a subgraph of* G*, and* (s', t') *is the* entry-exit pair *corresponding to vertices in* C *that satisfy the following conditions:*

1. *There exists a path* s-s' *from* s *to the* entry *vertex* s'
2. *There exists a path* t'-t *from the* exit *vertex* t' *to* t
3. *Paths* s-s' *and* t'-t *are vertex-disjoint*
4. *Path* s-s' *(resp.* t'-t*) and* G' *share exactly one vertex:* s' *(resp.* t'*).*

Notice that the same subgraph G' of G may contain multiple entry-exit pairs, hence the triple notation (G', s', t').

Definition 4.2 (Entry-exit cycle)**.** *An* entry-exit cycle *is an entry-exit subgraph* (C, s', t')*, where* C *is a cycle (see Fig. 4.2).*

Intuitively, an entry-exit cycle (C, s', t') (see Fig. 4.2) represents a choice between two s-t paths and, thus, requires tracking at least one of them.

We say that a vertex v *tracks* (C, s', t') if $v \in C \setminus \{s', t'\}$. Moreover, we say that (C, s', t') is *tracked* if there exists a tracker in a vertex that tracks it. A cycle C is *tracked* if all entry-exit cycles with entry-exit pairs in C are tracked. If C contains either (i) 3 trackers or (ii) s or t and 1 tracker in a non-entry/non-exit vertex, then it must be tracked. We say that these cycles are *trivially tracked.*

We rely on the following alternative characterization of a tracking set, due to Banik, Choudhary, Lokshtanov, Raman and Saurabh [47, Lemma 2], which establishes TRACKING as a covering problem.

Lemma 4.1 ([47])**.** *For a graph* $G = (V, E)$*, a subset* $T \subseteq V$ *is a tracking set if and only if every simple cycle* C *in* G *is tracked with respect to* T*.*

Reduction Rules. Let us recall some reduction rules previously used to obtain polynomial kernels [47, 102] and approximation algorithms [48, 101, 103, 151].

Rule 1. [47] If there exists an edge or vertex that does not participate in any s-t path, remove it from the graph.

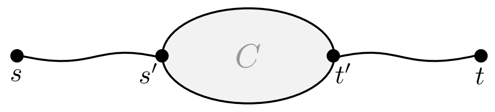


Figure 4.2: Entry-exit pair illustration, with entry vertex s' and exit vertex t'.

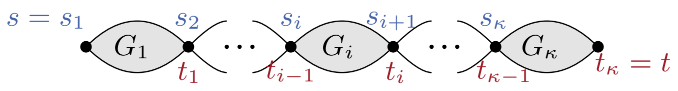


Figure 4.3: The block-cut tree of a graph G reduced by Rule 1 (see Remark 4.4). An optimal tracking set for (G, s, t) is the disjoint union of all optimal tracking sets for all instances (G_i, s_i, t_i).

Rule 2. [102] If the degree of s (or t) is 1 and $N(s) \neq \{t\}$ ($N(t) \neq \{s\}$), then remove s (t), and label the vertex adjacent to it as s (t).

Rule 3. [151] If there exist adjacent vertices $a, b \notin \{s, t\}$ such that $\deg(a) = \deg(b) = 2$, then contract the edge ab.

Definition 4.3. *We say that an undirected graph G is* reduced by Rule X *if it cannot be further by reduced Rule X. Further, we say that G is* reduced *if it is reduced by Rules 1, 2 and 3.*

After exhaustive application of Rules 1 and 2, the graph is either a single edge, (s, t), or all its vertices have degree at least 2. Henceforth, we assume the latter, since the problem becomes trivial in the former case. Rule 3, which precludes the existence of adjacent vertices of degree 2, is used to bound the overall number of degree-2 vertices.

Let us highlight a few additional useful consequences of Rule 1.

Remark 4.1 ([47])**.** *Let G be a graph reduced by Rule 1. Then, every subgraph of G containing at least one edge has at least one entry-exit pair.*

Remark 4.2 ([47])**.** *Let G be a graph reduced by Rule 1. Then, any tracking set of G is also an FVS of G.*

Remark 4.3. *Let G be a graph reduced by Rule 1. Then the block-cut tree of G is an s-t path (see Fig. 4.3).*

In other words, the latter remark says that the graph G that results from exhaustively applying Rule 1 consists of a sequence of $\kappa \geq 1$ biconnected components attached together by cut-vertices in a way that is analogous to series composition in series-parallel graphs. Thus, we can turn an instance (G, s, t) of TRACKING into one or more subproblems on biconnected graphs, (G_i, s_i, t_i), one for each biconnected component, for all $i \in \{1, \dots, \kappa\}$. The cut vertices of G define the start-finish pairs of each subproblem, where $t_i = s_{i+1}$ (for all $i \in \{1, \dots, \kappa - 1\}$) and $s_1 = s$, $t_\kappa = t$. as depicted in Fig. 4.3.

Intuitively, a cut-vertex is included in all s-t paths, so placing a tracker on it would be helpless, giving us the following:

Remark 4.4. *Let G be a graph reduced by Rule 1. Then, an optimal tracking set for (G, s, t) is the* disjoint union *of optimal tracking sets for all (G_i, s_i, t_i).*

Proof. By Remark 4.3, an optimal tracking set must contain the union of optimal tracking sets for all (G_i, s_i, t_i). Thus, the remark follows if we show that no optimal tracking set includes a cut-vertex. By Remark 4.3, any cut-vertex v of G disconnects the start s from the finish t, when removed. It follows that v cannot track any entry-exit cycle, since it will always be entry/exit for any entry-exit cycle containing it. □

Lower Bounds. We expand on a result by Choudhary and Raman [102], which provides a lower bound on the size of a tracking set, based on the presence of a tree-sink structure in the graph.

Definition 4.4 ([102])**.** *A* tree-sink *in a graph G is a pair (Tr, x), where Tr is a subtree of G with at least two vertices and x, the* sink*, a vertex not in Tr that is adjacent to all the leaves*[2] *of Tr in G. We use $G(Tr, x)$ to denote the subgraph induced by (Tr, x). (Notice that this definition does not preclude the adjacency between non-leaf vertices and x, as illustrated in Fig. 4.4.)*

[2]We consider a leaf in an unrooted tree to be any vertex of degree 1.

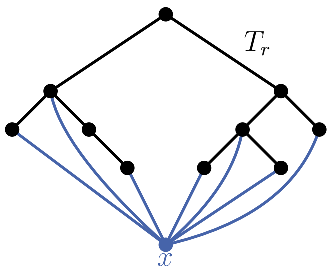


Figure 4.4: Illustration of a tree-sink (Tr, x).

Lemma 4.2 ([102])**.** *Let (Tr, x) be a tree-sink in a reduced graph G, such that $|N_{Tr}(x)| = \delta$. Further let (s', t') be an entry-exit pair of $G(Tr, x)$. Then, if $x \in \{s', t'\}$, any tracking set of G contains at least $\delta - 1$ vertices in $V(Tr)$.*

The above lemma is a generalization of the lower bound given by the maximum number of vertex-disjoint paths between any two vertices [47]. We generalize it further to obtain a more useful lower bound, established as the maximum degree among non-cut vertices.

Lemma 4.3. *Let G' be a subgraph of a reduced graph G and x a vertex in G', such that $G' - x$ is connected and $N_{G'}(x) = \delta$. Then, any tracking set of G contains at least $\delta - 2$ vertices in $G' - x$.*

Proof. Let Tr be a tree in $G' - x$ whose leaves are all adjacent to x (i.e. contained in $N_{G'}(x)$). Such tree can be constructed by trimming a spanning tree of $G' - x$: iteratively remove any leaf that is not adjacent to x in G'. Clearly, (Tr, x) is a tree-sink in G', and by Remark 4.1, $G(Tr, x)$ has at least one entry-exit pair. If x is in any such entry-exit pair, the lemma follows directly from Lemma 4.2, so let us assume otherwise hereafter.

Consider the entry-exit pair (s', t') of $G(Tr, x)$ and let us root Tr at s'. As in [102, Lemma 8], consider the subtrees Tr_1, Tr_2 of Tr determined by the edge separator connecting t' to its parent vertex in Tr. In particular, let $Tr_1 = Tr - Tr(t')$ and $Tr_2 = Tr(t')$, where $Tr(v)$ denotes the subtree of Tr rooted at $v \in V(Tr)$. To ensure that every leaf in Tr_1 is adjacent to x, we again repeatedly remove any leaf of Tr_1 that is not adjacent to x (these would correspond to ancestors of t' in Tr). Consider the two complementing cases, illustrated in Fig. 4.5: (1) both Tr_1 and Tr_2 have at least one leaf adjacent to x and (2) one of Tr_1, Tr_2 has no leaf adjacent to x. The latter case was neglected in [102, Lemma 8].

Case 1. Since there is an edge from x to Tr_2, there exists an $x - t'$ path which does not intersect Tr_1. Thus, (Tr_1, x) constitutes a tree-sink with entry-exit pair (s', x). Similarly, since there is an edge from x to Tr_2, (Tr_2, x) constitutes a tree-sink with entry-exit pair (x, t'). The lemma follows from applying Lemma 4.2 to either: (a) each of the tree-sinks (Tr_1, x) and (Tr_2, x), when both Tr_1 and Tr_2 contain at least two vertices; or (b) to the tree-sink that contains exactly $\delta - 1$ leaves adjacent to x (the other tree-sink must be a single vertex when (a) does not hold).

Case 2. Since Tr is rooted at s' and every leaf of Tr is adjacent to x, it must be the case that Tr_1 has no leaf adjacent to x. Thus, (Tr_2, x) is a tree-sink containing all of $N_{G'}(x)$, so let us apply inductively the same reasoning we did earlier (with roles of s', t' reversed), whereby we consider a subdivision of Tr_2 into 2 subtrees as established by the existence of an entry-exit pair (s'', t'') of $G(Tr_2, x)$ – for simplicity, we assume that $t'' = t'$, since t' is exit for a tree containing Tr_2. The lemma follows directly by the inductive hypothesis that any tracking set of G contains at least $\delta - 2$ vertices in Tr_2. The base case is a tree-sink corresponding to a star that contains all of $N_{G'}(x)$, and whose leaves are all adjacent to x. In this case, an entry-exit pair must belong to the star's root and one of its leaves (x cannot be in an entry-exit pair, given our initial assumption that x did not belong to any entry-exit pair of Tr, a supertree of this one). It follows that Case 1 applies to the base case setting.

□

Corollary 4.1. *Let δ be the degree of a non-cut vertex in a reduced graph G. Then, any tracking set of G has size at least $\delta - 2$.*

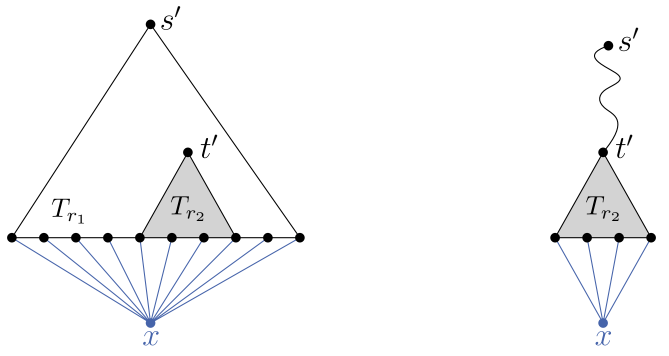


Figure 4.5: Illustration of proof of Lemma 4.3, with Case 1 on the left (both Tr_1 and Tr_2 have leaves adjacent to x) and Case 2 on the right (just Tr_2 has leaves adjacent to x).

Lemma 4.3 above generalizes [102, Lemma 8], and its proof completes the case analysis given in [102, Lemma 8]. We use it in Sections 4.4 and 4.5.

4.4 H-Minor-Free Graphs

A graph is *H-minor-free* if it does not contain a fixed graph H as a minor. In this section, we present a linear kernel for H-minor-free graphs and use this kernel, as well as some ideas intrinsic to its construction, to design an efficient polynomial-time approximation scheme (EPTAS). An EPTAS is a $(1 \pm \epsilon)$-approximate algorithm whose running time is $O(n^c)$ for an input of size n and a constant c independent of ϵ. This is substantially better than a polynomial-time approximation scheme (PTAS), whose running time can be as bad as $O(n^{f(1/\epsilon)})$, for any computable function f independent of n.

Unlike the minimum FVS problem, which also consists of covering cycles, Tracking is not minor-closed [102] (i.e., an optimal solution for a minor of Gmay require more trackers than an optimal solution for G), so the powerful framework of bidimensionality [130, 131, 162] cannot be used to obtain either linear kernels [163, 230] or PTASs for H-minor-free graphs [132]. Moreover, Tracking does not possess the "local" properties required by Baker's technique to develop EPTASs for planar graphs [40], or apex-minor-free graphs [148, 149].

Intuitively, these local properties can be verified by locally checking constant-sized neighborhoods; nonlocal problems include minimum FVS and minimum connected dominating set, in contrast to problems such as minimum vertex cover, maximum independent set, or minimum dominating set which possess these local properties. Demaine and Hajiaghayi [132] developed extensions to Baker's approach to obtain PTASs to other nonlocal problems (such as connected dominating set), but these are not applicable to minimum FVS problem, as pointed by the authors.

4.4.1 Linear Kernel

We first present a linear kernel for H-minor-free graphs. The following theorem about the sparsity of H-minor-free graphs will be helpful throughout the section.

Theorem 4.1 (Mader [239]). *Any simple H-minor-free graph with n vertices has at most $\sigma_H n$ edges, where σ_H depends solely on $|V(H)|$.*

Since its introduction in [239], the value of the constant σ_H has been improved [219, 220, 287]. The current value of σ_H is $(\alpha + o(1))|V(H)|\sqrt{\ln |V(H)|}$ [288], where $\alpha \approx 0.319$.

We now give the following lemma concerning a relationship between the sizes of the vertex sets in certain bipartite minor-free graphs.

Lemma 4.4. *Let $B = (U \cup V, E)$ be a simple H-minor-free bipartite graph, such that:*

(i) every vertex in V has degree at least 2, and

(ii) there exist at most δ neighbors in common between any pair u_1, u_2 in U, i.e., $|N(u_1) \cap N(u_2)| \leq \delta$ for all $u_1, u_2 \in U$.

Then $|V| \leq \delta\sigma_H|U|$.

Proof. To show the bound on the size of vertex set V, we construct a new graph from B as follows. Replace every vertex v in V and its incident edges in B by an edge connecting any two of its neighbors. Observe that, this operation results in an H-minor-free graph as it is equivalent to contraction of any edge incident to v, followed by the deletion of all but one of the remaining edges incident to v). The resulting graph has vertex set U, exactly $|V|$ edges, and at most δ parallel edges between any pair of vertices (this follows from (ii)). By Theorem 4.1, any simple H-minor-free graph with $|U|$ vertices has at most $\sigma_H|U|$ edges and, thus at most $\delta\sigma_H|U|$ edges when there exist at most δ parallel edges between any pair of vertices. □

Next, we give a lemma which will be useful throughout the chapter.

Lemma 4.5. *Let F be an FVS of a reduced graph G. Then $|V(G-F)| \leq 4|X|-5$, where X is the cut set defined by $(F, G-F)$, consisting of edges with endpoints in both F and $G-F$.*

Proof. Let us partition $V(G-F)$ into V_1, V_2, $V_{\geq 3}$ corresponding to the sets of vertices whose degree in $G-F$ is (respectively) 1 (a.k.a. leaves), 2 or at least 3. Further, let V^X denote the set of vertices in $V(G-F)$ which are endpoints of an edge in X, i.e., the set of vertices in $V(G-F)$ adjacent to a vertex in F.

Since each vertex in V_1 must be adjacent to a vertex in F (G contains no degree-1 vertices since it's reduced), we have that $V_1 \subseteq V^X$ and, thus,

$$|V_1| \leq |X|$$

Moreover, $|V_{\geq 3}| \leq |V_1| - 2$ (this is a well known fact applicable to any forest). Thus,

$$|V_{\geq 3}| \leq |X| - 2$$

Next, we bound $|V_2|$. Consider the set $V_2 \setminus V^X$ of vertices in V_2 which are not adjacent to any vertex in F. By Rule 3, the set $V_2 \setminus V^X$ induces an independent set. Hence, $|V_2 \setminus V^X|$ is at most the number of edges in the forest that results from replacing each $v \in V_2 \setminus V^X$ by an edge connecting v's neighbors, giving us

$$|V_2 \setminus V^X| \leq |V_1| + |V_{\geq 3}| + |V_2 \cap V^X| - 1,$$

which implies

$$|V_2| \leq |V_1| + 2|V_2 \cap V^X| + |V_{\geq 3}| - 1$$

Thus,

$$\begin{aligned} |V(G - F)| &\leq 2V_1 + 2|V_2 \cap V^X| + 2|V_{\geq 3}| - 1 \\ &= 2|V_1 \cap V^X| + 2|V_2 \cap V^X| + 2|V_{\geq 3}| - 1 \qquad (V_1 \subseteq V^X) \\ &\leq 2|X| + 2|V_{\geq 3}| - 1 \\ &\leq 4|X| - 5 \end{aligned}$$

□

We will use Lemmas 4.4 and 4.5 above to give, in the next lemma, a linear kernel for a biconnected reduced H-minor-free graph.

Lemma 4.6. *Let G be a biconnected reduced H-minor-free graph with start s and finish t. Then, G has at most $(16\sigma_H^2 + 8\sigma_H + 1)OPT - 5$ vertices and at most $(20\sigma_H^2 + 11\sigma_H)OPT - 6$ edges, where OPT denotes the size of an optimal tracking set of G.*

Proof. Let T^* be an optimal tracking set of (G, s, t), i.e., $|T^*| = OPT$. Note that $G - T^*$ is a forest, since T^* is an FVS of G. We assume that $|T^*| \geq 2$, since otherwise one could check, in polynomial time, which vertex of G belongs to T^*. We now give some claims about the structure of G:

Claim 1: Let u_1, u_2 be two vertices in T^*. There exist at most 2 trees in $G - T^*$ that are adjacent[3] to both u_1 and u_2.

Claim 2: Every tree in $G - T^*$ is adjacent to at least 2 vertices in T^*.

Claim 3: Every tree in $G - T^*$ contains at most 2 vertices adjacent to the same vertex in T^*.

The first claim follows from Lemma 4.3. If there existed 3 or more trees adjacent to both u_1 and u_2, then the graph G', induced by u_1, u_2 and the trees, would require at least 1 tracker in $V(G') \setminus \{u_1\}$ and 1 tracker in $V(G') \setminus \{u_2\}$, contradicting the feasibility of T^*. The last claim also follows from Lemma 4.3 in a similar fashion. The second claim follows from the fact that G is biconnected and hence contains no cut-vertices.

To show the bound on the size of the vertex set and the edge set of G, we construct a new graph as follows. We first contract each tree Tr in $G - T^*$ into a *tree vertex* v_{Tr}. Let F be the set of all tree vertices. Note that this operation may create parallel edges between a vertex in T^* and a tree vertex, but never between two vertices in T^* or F. Furthermore, we remove any edges between vertices in T^*. The resulting graph is bipartite, with vertex set partitioned into T^* and F, and is H-minor-free (since the class of minor-free graphs is minor-closed). By Claims 1 and 2, any 2 vertices in T^* have at most 2 common neighbors, and every vertex in F is adjacent to at least 2 vertices in T^*. Hence, by Lemma 4.4,

$$|F| \leq 2\sigma_H |T^*|.$$

As a consequence of Claim 3, there are at most 2 parallel edges between a vertex in T^* and

[3]In this context, a tree is adjacent to v if it includes a vertex that is adjacent to v.

a vertex in F. Thus, by Theorem 4.1, the set of edges, X, in the bipartite graph is at most

$$2 \cdot \sigma_H(|F| + |T^*|) \leq (4\sigma_H^2 + 2\sigma_H)|T^*|.$$

Notice that X is the cut set defined by $(T^*, G - T^*)$, consisting of edges with endpoints in both T^* and $G - T^*$. Hence, by Lemma 4.5, $|V(G - T^*)| \leq 4|X| - 5$, giving us:

$$|V(G)| \leq (16\sigma_H^2 + 8\sigma_H + 1)|T^*| - 5.$$

The edges of G consist of (a) edges in $G - T^*$ (at most $|V(G - T^*)| - 1$), (b) the cut set X, and (c) edges with both endpoints in T^* (at most $\sigma_H|T^*|$ by Theorem 4.1). Thus,

$$\begin{aligned} |E(G)| &\leq (4|X| - 6) + |X| + (\sigma_H|T^*|) \\ &\leq (20\sigma_H^2 + 11\sigma_H)|T^*| - 6. \end{aligned}$$

□

By Remark 4.4 and the application of the above lemma to each biconnected component of a reduced graph, we obtain the following.

Lemma 4.7. *Let G be a reduced H-minor-free graph with start s and finish t. Then G has at most $(16\sigma_H^2 + 8\sigma_H + 1)OPT - 5$ vertices and at most $(20\sigma_H^2 + 11\sigma_H)OPT - 6$ edges, where OPT denotes the size of an optimal tracking set of G.*

Proof. The proof follows from Remark 4.4 and from applying Lemma 4.6 to each biconnected component of G, as in the proof of Lemma 4.14. □

Theorem 4.2. *k-*Tracking *admits a linear kernel when restricted to H-minor-free graphs.*

Corollary 4.2. Tracking *admits a $O(1)$-approximation for H-minor-free graphs.*

Even though we develop a $(1+\epsilon)$-approximation in the next section, the latter corollary can be more useful in practice, when running time is a concern.

4.4.2 EPTAS

Given the unsuitability of bidimensionality and Baker's technique discussed earlier, we shall resort to the use of balanced separators. Our algorithm relies on an appropriate decomposition of the graph into regions, which can be accomplished by recursively finding small *balanced separators*, sets of vertices whose removal partitions the graph into two roughly equal-sized parts.

For simplicity, we define balanced separators in the context of unweighted vertices, but this can be generalized to a weighted setting.

Definition 4.5 (Balanced separator)**.** *Let G be a graph, and let $X \subseteq V(G)$ be a subset of its vertices. We say that X is a* balanced separator *if $V(G) \setminus X$ can be partitioned into two sets A, B each of which has size at most $2/3|V(G)|$, such that no edge joins a vertex in A with a vertex in B.*

Ungar [293] first showed that every n-vertex planar graph has a balanced separator of size $O(\sqrt{n}\lg^{3/2} n)$. This was later improved by Lipton and Tarjan [227] to $\sqrt{8n}$, and Goodrich [174] showed how to compute these recursively in linear time.

This result is widely known as the *Planar Separator Theorem* and its impact cannot be overstated, with applications in the design of approximation algorithms using the divide-and-conquer paradigm for various NP-Complete problems [95, 228]. This theorem has been further refined [105, 135, 136, 164, 171, 244, 281] and generalized to bounded-genus graphs [137, 172, 205] as well as to H-minor-free graphs [17, 63, 202, 260, 261, 309].

Theorem 4.3 (Minor-free Separator Theorem [17])**.** *Let G be an H-minor-free graph with*

n vertices, where H is a simple graph with $h \geq 1$ vertices. Then a balanced separator for G of size at most $c^1_H\sqrt{n}$ can be found in $O(h^{O(1)}n^{O(1)})$ time, where c^1_H is a positive constant depending solely on h.

We use the Minor-free Separator Theorem recursively to decompose the graph into a set $\mathcal{R}$ of edge-disjoint subgraphs, called *regions*. The vertices of a region $R \in \mathcal{R}$ which belong to at least one other region are called *boundary vertices* and the set of these vertices is denoted by $\partial(R)$. The remaining vertices of R are called *interior vertices* and are denote by $int(R)$.

Definition 4.6 (Relaxed r-division). *A* relaxed r-division *of an n-vertex graph G is a decomposition of G into $\Theta(n/r)$ regions, each of which has at most r vertices, such that the total number boundary vertices is $O(n/\sqrt{r})$, for a positive integer r.*

A relaxed r-division was perhaps among the first applications of Lipton and Tarjan's Planar Separator Theorem, and it was used to obtain EPTASs for maximum independent set [228] and for minimum vertex cover [95]. A relaxed r-division is a relaxed version of an r-division, a decomposition introduced by Frederickson [165] which additionally requires every region to have $O(\sqrt{r})$ boundary vertices. Both decompositions can be constructed in $O(n \lg n)$ time, by recursively splitting the graph using balanced separators. Typically, an r-division is constructed from a relaxed r-division – it is shown in [165] that, after constructing a relaxed division, one can further split the big regions (the ones violating the extra condition) without asymptotically increasing the total number of regions or boundary vertices. Even though r-divisions were introduced for planar graphs [165], its derivation can easily be generalized to any class of graphs that is characterized by the existence of sublinear balanced separators, including H-minor-free graphs.

Theorem 4.4 (Minor-free Separator Theorem (4.3) + Frederickson [165]). *There is an $O(n \lg n)$ algorithm that, given an H-minor-free graph G and a positive integer r, computes a relaxed r-division of G.*

Proof. Follows from Minor-free Separator Theorem (4.3) and Frederickson [165]. □

A relaxed r-division is simpler to construct and will be sufficient for our purposes, so we will use these instead.

Algorithm. Our strategy will be to (i) construct a relaxed r-division of a linear kernel K of the input H-minor-free graph G (such that K is itself a $O(1)$-approximate tracking set), (ii) solve optimally for each region, and (iii) combine the solutions for each region into a solution for the original graph with quality comparable to that of an optimal solution. This approach has been used to obtain EPTASs for minimum FVS [71, 314], and (without the need for a linear kernelization) maximum independent set [228], as well as minimum vertex cover [95]. However, and in contrast to these problems, the step of constructing a close to optimal solution from the solutions of each region is not obvious. Indeed, the difficulty of this step emerges from the very "nonlocal" structure of Tracking, which requires special attention to the location of the start-finish (s, t) within the graph, in addition to requiring tracking cycles that escape out of local neighborhoods (as in minimum FVS). The latter issue complicates not only the intermediate step of solving optimally for each region, but also the step of combining solutions for each region – an entry-exit cycle spanning two regions may not be tracked by just the boundary vertices it traverses, depending on the position of s and t, as illustrated in Fig. 4.6; this is in contrast to minimum FVS, which requires only one tracker per cycle rendering the combining step trivial.

Our EPTAS is as follows:

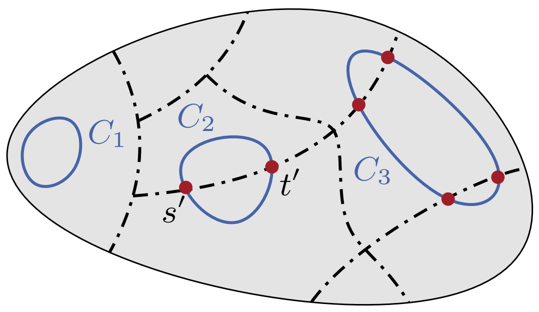


Figure 4.6: Illustration of a relaxed r-division $\mathcal{R}$ (boundaries in dashed lines) and the types of cycles tracked by the output tracking set T (considered in Lemma 4.9). C_1-type cycles, which span a single region $R \in \mathcal{R}$ are tracked by $OPT(R)$. C_2-type cycles, which span exactly 2 regions, are not guaranteedly tracked by the 2 boundary vertices they traverse, since these may correspond to an entry-exit pair (s', t'). C_3-type cycles, which span at least 3 regions, are trivially tracked by the ≥ 3 boundary vertices they traverse.

1. Compute a linear kernel K of G by reducing it with Rules 1, 2, 3, such that an optimal tracking set of K is a constant fraction of K (see Corollary 4.2).

2. Compute a relaxed r-division of K with $r = (2c_H^1 c_H^2 (c_H^3 + 1)/\epsilon)^2$, for any choice of $\epsilon > 0$ and constants $c_H^1, c_H^2, c_H^3 > 0$ specified later. Let $\mathcal{R}$ be the set of resulting regions.

3. For each region R in $\mathcal{R}$, compute an optimal tracking set $OPT(R)$ for the subset of entry-cycles (with respect to (s,t)) which are completely contained in R.

4. Output $T = \bigcup_{R \in \mathcal{R}} (OPT(R) \cup \partial(R) \cup \mathcal{N}(R))$.

Here, $\mathcal{N}(R) := N_{\Pi(R)}(\partial(\Pi(R)))$ defines an appropriate neighborhood of the boundary vertices of R, where $\Pi(R)$ is the subgraph of R consisting of the union of each path in R that: (i) is not an edge, (ii) has $\partial(R)$ vertices as endpoints, and (iii) traverses no *internal* vertices that are in $OPT(R)$. We let $\partial(\Pi(R)) := \partial(R) \cap \Pi(R)$. See Fig. 4.7.

Notice that $\bigcup_{R \in \mathcal{R}} (OPT(R) \cup \partial(R))$ may not constitute a tracking set of K, because there might exist entry-exit cycles spanning exactly 2 regions, whose entry-exit pairs are the only vertices with trackers, rendering them untracked (see Fig. 4.6).

We will now give the details of the algorithm and its correctness. We refer to the Reduction Rules defined in Section 4.3. As a reminder, after exhaustive application of Rules 1 and 2, the graph is either a single edge between s and t, or all its vertices have degree at least 2. Henceforth, we will assume the latter, since a minimum tracking set is trivial in the former. Rule 3, which precludes the existence of adjacent vertices of degree 2, is used to bound the overall number of degree-2 vertices.

Notice that none of the reduction rules introduce trackers, so there is no lifting required at the end of our algorithm, i.e., adding back any trackers introduced during the reduction.

Observation 4.1. *No entry-exit cycles are removed during Rules 1, 2 or 3, so a tracking set of the resulting kernel K is a tracking set of the input graph G. Therefore, any minimum tracking set of K is also a minimum tracking set of G.*

Next, we explain how to compute in polynomial time optimal tracking sets for each region in a relaxed r-division of a kernel K.

Lemma 4.8. *Let $\mathcal{C}(R)$ be the set of all entry-exit cycles in G whose vertices are a subset of $V(R)$, where R is a subgraph of G. Then one can compute a minimum subset of $V(R)$ that tracks every entry-cycle of $\mathcal{C}(R)$ in $O(2^{|V(R)|} \cdot n^{O(1)})$ time.*

Proof. It suffices to enumerate all the $2^{|V(R)|}$ possible subsets and, for each, verify in $O(n^{O(1)})$

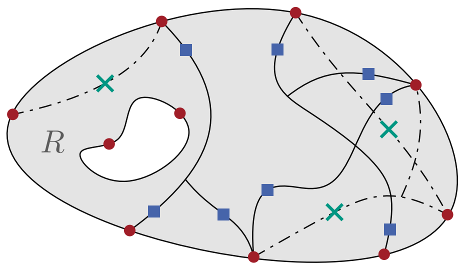


Figure 4.7: Illustration of $\Pi(R)$ and of $\mathcal{N}(R)$ for a region R in a relaxed r-division $\mathcal{R}$. Vertices in $\partial(R)$ are depicted in red circles. $\Pi(R)$ consists of the union of all boundary-to-boundary paths in R (solid black), which are not edges and do not traverse $OPT(R)$ (green crosses). The dashed lines represent paths in $R - \Pi(R)$. $\mathcal{N}(R)$ is depicted in blue squares.

time whether every entry-exit cycle of $\mathcal{C}(R)$ is tracked. The verification step can be done in a way similar to the verification algorithm given by Banik, Choudhary, Lokshtanov, Raman and Saurabh [47] to show that the problem is in NP: from the observation that every tracking set X is an FVS (see Remark 4.2), it follows that there is at most $O(n^{O(1)})$ entry-exit cycles not tracked by X (see also Lemma 4.17). □

Let us now argue that our algorithm computes a $(1+\epsilon)$-approximate tracking set. Let $T = \bigcup_{R \in \mathcal{R}} (OPT(R) \cup \partial(R) \cup \mathcal{N}(R))$ be the output of the algorithm, for the relaxed r-division $\mathcal{R}$ of a kernel K of G.

Lemma 4.9. *T is a tracking set of the input graph G.*

Proof. It is enough to argue that T tracks every cycle C of K (by Lemma 4.1 and Observation 4.1). We consider 3 types of cycles, illustrated in Fig. 4.6. If C spans exactly 1 region R in $\mathcal{R}$, then it is guaranteed to be tracked by feasibility of $OPT(R)$ (see Lemma 4.8). If C spans at least 3 regions in $\mathcal{R}$, then C is trivially tracked by the trackers in, at least, 3 boundary vertices. Otherwise, let C be a cycle spanning exactly 2 regions. We argue that C is also trivially tracked. Let R_1 and R_2 be the two regions spanned by C and let us assume that C traverses exactly 2 boundary vertices b_1 and b_2 (if it traverses more boundary vertices, it must be trivially tracked, and if it traverses less, then it could not span more than one region). If C contains a tracker in $OPT(R_1) \cup OPT(R_2) \setminus \{b_1, b_2\}$, we are done. Otherwise, C is the union of a path in $\Pi(R_1)$ and a path in $\Pi(R_2)$, by definition of Π. In this case, however, C must have a third tracker placed in $N(\{b_1, b_2\}) \cap \{int(R_1) \cup int(R_2)\}$ (notice that C contains at least one non-boundary vertex that is a neighbor of b_1 or b_2, since there are no parallel edges). □

Let us denote by OPT the size of an optimal tracking set of the input graph G. To argue that $|T| \leq (1+\epsilon)OPT$, we will need to argue that the set of trackers in the special neighborhoods defined by $\mathcal{N}(R)$, for all regions R, have small cardinalities, i.e., roughly equal to $O(\epsilon OPT)$. This is the key argument to our EPTAS, which the next lemma addresses. Its proof is not immediately obvious, since the number of neighbors of all boundary vertices could be $\Omega(OPT)$, a consequence of the quadratic gap between $|\partial(R)|$ and $|V(R)|$. Intuitively, if these special neighborhoods were too large, then the following properties could not both be applicable: feasibility of $OPT(R)$ for every region R, and classification of G as minor-free. Our proof below is of combinatorial nature, so it will not quite capture the topological intuition behind it, but it is relatively succinct and avoids complex case analysis.

We will use Lemma 4.2 to derive a few structural properties of $\Pi(R)$, for each $R \in \mathcal{R}$, and exploit these to bound $|\mathcal{N}(R)|$. As a reminder, $\Pi(R)$ is defined to be the subgraph of R that is the union of each path in R that: is not an edge, has $\partial(R)$ vertices as endpoints, and traverses no *internal* vertices that are in $OPT(R)$, where $OPT(R)$ is an optimal tracking set of just the entry-exit cycles of region R.

Lemma 4.10. *$|\mathcal{N}(R)| \leq c_H^3 |\partial(\Pi(R))|$, where $c_H^3 \geq 9\sigma_H^2 + 3\sigma_H$.*

Proof. The proof is similar in spirit to that of Lemma 4.6. Clearly, $\partial(\Pi(R))$ is an FVS for $\Pi(R)$ (if it were not, there would be untracked cycles in R contradicting feasibility of $OPT(R)$), so $\Pi(R) - \partial(\Pi(R))$ is a forest. We assume w.l.o.g. that $|\partial(\Pi(R))| \geq 2$. Below, we make some claims about the structure of $\Pi(R)$:

Claim 1: Let b_1, b_2 be two vertices in $\partial(\Pi(R))$. There exist at most 3 trees in $\Pi(R) - \partial(\Pi(R))$ that are adjacent[4] to both b_1 and b_2.

Claim 2: Every tree in $\Pi(R) - \partial(\Pi(R))$ is adjacent to at least 2 vertices in $\partial(\Pi(R))$.

Claim 3: Every tree in $\Pi(R) - \partial(\Pi(R))$ contains at most 3 vertices adjacent to the same vertex in $\partial(\Pi(R))$.

[4]In this context, a tree is adjacent to v if it includes a vertex that is adjacent to v.

The first claim follows from Lemma 4.2 (if there existed 4 or more trees adjacent to both b_1 and b_2, there would have to be a tracker from $OPT(R)$ in one of the trees, a contradiction). The last claim follows from Lemma 4.2 in a similar fashion. The second claim follows from the definition of $\Pi(R)$ and Rules 1,2.

Let us contract each tree Tr in $\Pi(R) - \partial(\Pi(R))$ into a *tree vertex* v_{Tr} and let F be the set of all tree vertices. Notice that this may create parallel edges between a vertex in $\partial(\Pi(R))$ and a tree vertex, but never between two vertices in $\partial(\Pi(R))$ or F. In addition, let us remove any edges between vertices in $\partial(\Pi(R))$. The resulting graph is bipartite, with vertex set partitioned into $\partial(\Pi(R))$ and F, and is H-minor-free (since the class of minor-free graphs is minor-closed). By Claims 1 and 2, at most 3 vertices in F share the same pair of neighbors, and every vertex in F has degree at least 2. Hence, by Lemma 4.4,

$$|F| \leq 3\sigma_H|\partial(\Pi(R))|$$

As a consequence of Claim 3, there are at most 3 parallel edges between a vertex in $\partial(\Pi(R))$ and a vertex in F. Thus, by Theorem 4.1, the set of edges in the bipartite graph is at most

$$3 \cdot \sigma_H(|F| + |\partial(\Pi(R))|) \leq (9\sigma_H^2 + 3\sigma_H)|\partial(\Pi(R))|$$

The lemma follows from the fact that the edges in the bipartite graph, including the parallel ones, have a 1-1 correspondence with the vertices in $\mathcal{N}(R)$. □

Before proving that the output of our algorithm is a $(1+\epsilon)$-approximate tracking set, let us first recall a result from Frederickson [165, Lemma 1][5], which concerns the sum, for each boundary vertex b of the number of regions $\Delta(b)$ containing b in a relaxed r-division $\mathcal{R}$ of a planar graph. Even though this result was given in the context of planar graphs,

[5] A more detailed proof of [165, Lemma 1] is given in [211] for separators based on edge weights.

it can easily be generalized to any graph whose subgraphs G' admit balanced separators of size $O(\sqrt{|V(G')|})$. We denote the set of all boundary vertices by $\partial(\mathcal{R})$. Further, let $B(\mathcal{R}) = \sum_{b \in \partial(\mathcal{R})} (\Delta(b) - 1)$.

Lemma 4.11 ([165]). *Let $\mathcal{R}$ be a relaxed r-division of an n-vertex graph whose subgraphs G' admit balanced separators of size at most $c\sqrt{|V(G')|}$. Then $B(\mathcal{R}) \leq c \cdot n/\sqrt{r}$, for a constant c independent of r and n.*

We will use the latter lemma to bound the overall number of trackers in the next theorem.

Theorem 4.5. *There exists a $(1+\epsilon)$-approximation algorithm for* Tracking *on H-minor-free graphs (for a fixed H), running in $O(2^{O(1/\epsilon^2)} n^{O(1)})$ time, for a given $\epsilon > 0$.*

Proof. Consider the algorithm given at the beginning of the section. As a reminder, let $T = \bigcup_{R \in \mathcal{R}} (OPT(R) \cup \partial(R) \cup \mathcal{N}(R))$ be the output of the algorithm, for a relaxed r-division $\mathcal{R}$ of a kernel K of G, where $OPT(R)$ is the optimal tracking set computed with respect to entry-exit cycles in R. By Lemma 4.9, T is a tracking set. Next, we argue about the approximation ratio. By a union bound,

$$|T| \leq |\partial(\mathcal{R})| + \sum_{R \in \mathcal{R}} |OPT(R)| + \sum_{R \in \mathcal{R}} |\mathcal{N}(R)|.$$

Let $n' = |V(K)|$ be the number of vertices in K. Clearly, $|\partial(\mathcal{R})| \leq B(\mathcal{R})$. Moreover, we have that $\sum_{R \in \mathcal{R}} |\partial(R)| \leq 2B(\mathcal{R})$, so by Lemma 4.10, we have:

$$\sum_{R \in \mathcal{R}} |\mathcal{N}(R)| \leq 2c_H^3 B(\mathcal{R}).$$

Let T^* be an optimal tracking set of K, i.e., $|T^*| = OPT$ (by Observation 4.1). Since T^* is a tracking set, but not necessarily an optimal one, for all entry-exit cycles within any region

$R \in \mathcal{R}$, we have that $|OPT(R)| \leq |T^* \cap V(R)|$. Thus,

$$\sum_{R \in \mathcal{R}} |OPT(R)| \leq OPT + B(\mathcal{R}).$$

Overall, for $r = (2c_H^1 c_H^2 (c_H^3 + 1)/\epsilon)^2$,

$$\begin{aligned}
|T| &\leq OPT + 2(c_H^3 + 1)B(\mathcal{R}) \\
&\leq OPT + 2c_H^1(c_H^3 + 1)n'/\sqrt{r} && \text{(Lemma 4.11, Theorem 4.3)} \\
&\leq OPT + 2c_H^1 c_H^2 (c_H^3 + 1) OPT/\sqrt{r} && \text{(Lemma 4.7, } c_H^2 \geq 16\sigma_H^2 + 8\sigma_H + 1\text{)} \\
&= (1 + \epsilon) OPT.
\end{aligned}$$

The first step of the algorithm, concerning the kernelization, takes $O(n^{O(1)})$ time, since it consists of applying Rules 1, 2, 3. The second step, which computes a relaxed r-division can be done in $O(n \lg n)$ time [165], using Lipton and Tarjan's linear time algorithm [227] to find a balanced separator. Computing each $OPT(R)$ in step 3 can be done in $O(2^r \cdot n^{O(1)})$ time, by Lemma 4.8, and thus $O(2^r \cdot n^{O(1)}/r)$ time for all $R \in \mathcal{R}$. Finally, computing $\Pi(R)$ and $\mathcal{N}(R)$ can be done in $O(n^{O(1)})$ time, and thus $O(n^{O(1)}/r)$, for all $R \in \mathcal{R}$. Overall, the total time complexity is is bounded by

$$\begin{aligned}
&O(2^r \cdot n^{O(1)}) \\
= &O(2^{O(1/\epsilon^2)} \cdot n^{O(1)}).
\end{aligned}$$

□

4.5 General Graphs

In this section, we derive an $O(\lg n)$-approximation algorithm for WEIGHTEDTRACKING on general graphs, as well as an $O(\lg OPT)$-approximation algorithm for TRACKING. To the best of our knowledge, the only known approximation ratio for general graphs until now is $O(\sqrt{n \lg n})$ for tracking shortest paths only (as opposed to all paths) with multiple start-finish pairs, a result by Bilò, Gualà, Leucci and Proietti [62]. In addition, we improve the quadratic kernel of Choudhary and Raman [102] for general graphs and complete the case-analysis of [102, Lemma 8].

4.5.1 Quadratic Kernel

In this section, we focus on k-TRACKING, the decision version of TRACKING which asks whether there exists a tracking set of size at most k. We consider a parameterization with parameter k itself (often called the *natural* parameter) and give a kernelization algorithm that produces a quadratic kernel for general graphs, by building on the quadratic kernel of Choudhary and Raman [102]. While simpler, our proof of the kernel size completes the case analysis (see Lemma 4.3) for one of the lemmas central to the kernelization algorithm of [102] (specifically, [102, Lemma 8]). Moreover, our kernelization algorithm yields a kernel size with considerably smaller constants. We achieve this by expanding on the notion of tree-sink structures (see Section 4.3), allowing us to bound the maximum degree among non-cut vertices in the kernel (see Corollary 4.1).

Our kernelization algorithm is simply the exhaustive application of Rules 1, 2 and 3 (see Section 4.3) in no particular order, followed by application of Rule 4, and then of Rule 5:

Rule 4. If there exists a non-cut vertex of degree more than $k + 2$, return a trivial NO-instance.

Rule 5. If the number of vertices (resp. edges) is more than $4k^2+9k-5$ (resp. $5k^2+11k-6$), return a trivial NO-instance.

Lemma 4.12. *Rule 4 is safe and can be done in polynomial-time.*

Proof. Follows from Corollary 4.1. □

Next, we show that the last rule is also safe.

Lemma 4.13. *Let G be a biconnected reduced graph, with start s and finish t. Then, G has at most $4OPT^2 + 9OPT - 5$ vertices and at most $5OPT^2 + 11OPT - 6$ edges, where OPT denotes the size of an optimal tracking set of G.*

Proof. Let T^* be an optimal tracking set of G, i.e., $|T^*| = OPT$. Since every tracking set is an FVS of a reduced graph (see Remark 4.2), we can apply Lemma 4.5 and obtain $|V(G-T^*)| \leq 4|X| - 5$, where X is the set of edges with endpoints in both T^* and $G - T^*$. By the fact that G is biconnected and by Corollary 4.1, G has maximum degree $OPT + 2$ and, hence, $|X| \leq OPT(OPT+2)$. It follows that

$$|V(G)| \leq 4OPT^2 + 9OPT - 5$$

The edges of G consist of edges with no endpoint in T^* (at most $|V(G-T^*) - 1|$) and edges with at least one endpoint in T^* (at most $OPT(OPT+2)$ by Corollary 4.1), giving us

$$|E(G)| \leq 5OPT^2 + 11OPT - 6$$

□

We can now apply the latter lemma individually to each biconnected component, giving us the following.

Lemma 4.14. *Any reduced graph G with start s and finish t has at most $4OPT^2+9OPT-5$ vertices and at most $5OPT^2+11OPT-6$ edges, where OPT denotes the size of an optimal tracking set of G.*

Proof. Let G_i denote the i^{th} biconnected component of G, with entry-exit vertices s_i, t_i (see Remark 4.4). Further, let OPT_i denote the size of a minimum tracking set of (G_i, s_i, t_i). It follows from Remark 4.4 that $OPT = \sum_i OPT_i$. Moreover, Lemma 4.13 gives us $|V(G_i)| \leq p(OPT_i)$, where $p(x) = 4x^2 + 9x - 5$. Thus,

$$\begin{aligned}
|V(G)| &\leq \sum_i |V(G_i)| \\
&\leq \sum_i p(OPT_i) && \text{(Lemma 4.13)} \\
&\leq p\left(\sum_i OPT_i\right) && \text{(p is degree-2 polynomial)} \\
&= p(OPT) && \text{(Remark 4.4)}
\end{aligned}$$

The number of edges in G can be upper bounded in a similar manner. □

The latter lemma immediately implies the safety of Rule 5, as well as an $O(\sqrt{n})$-approximation algorithm (output all the vertices in the kernel).

Lemma 4.15. *Rule 5 is safe and can be done in polynomial-time.*

Correctness of Rules 1, 2 and 3 ([47, 102, 151]), as well as of Rules 4 and 5 (Lemmas 4.12 and 4.15) immediately give us the following.

Theorem 4.6. k-Tracking *admits a kernel of size bounded by $4k^2 + 9k - 5$ vertices and $5k^2 + 11k - 6$ edges.*

The latter theorem improves a quadratic kernel of Choudhary and Raman [102], whose size is bounded by $140k^2 - 45k$ vertices and $180k^2 + 65k$ edges.

4.5.2 Approximation Algorithm

We reduce an instance (G, s, t, w') of WEIGHTEDTRACKING, for a weight function $w' : V(G) \to \mathbb{Q}$, into an instance $(\mathcal{U}, \mathcal{X}, w)$ of SETCOVER, which asks for the sub-collection of $\mathcal{X}$ of minimum total weight, whose union equals the universe $\mathcal{U}$. Here, $(\mathcal{U}, \mathcal{X})$ defines a set system, i.e., a collection $\mathcal{X}$ of subsets of a set $\mathcal{U}$, and w is the weight function $w : \mathcal{X} \to \mathbb{Q}$. It is well known that there exist greedy polynomial-time algorithms achieving approximation ratios of $(1 + \ln M)$ [106, 195, 231] or of $(1 + \Delta)$ [187, 296, 308], where M is the size of the largest set in $\mathcal{X}$ and Δ is the maximum number, over all elements u in $\mathcal{U}$, of sets in $\mathcal{X}$ that contain u. At first glance, this reduction does not seem to yield a useful approximation ratio, given the exponential number of entry-exit cycles that need to be covered, but we will show that one can limit this number to a polynomial of fixed degree.

Let $\mathcal{C}$ be the set of all entry-exit cycles in our input graph G, which we assume w.l.o.g. to be reduced by Rule 1. Further, let $\mathcal{C}_F$ be the set of all entry-exit cycles in G, each of which contains at most 2 vertices from the subset $F \subseteq V$. That is, $\mathcal{C}_F := \{(C, s', t') \in \mathcal{C} : |C \cap F| \leq 2\}$. Our algorithm is as follows.

1. Compute a 2-approximate FVS F of G (see [38, 51, 104]).

2. Use the greedy algorithm of [106, 195, 231] to compute an approximate set covering, $S \subseteq V(G)$, for an instance $(\mathcal{U}, \mathcal{X}, w)$ of SetCover where:

(i) the universe, $\mathcal{U}$, of elements to be covered is $\mathcal{C}_F$

(ii) the collection of covering sets, $\mathcal{X}$, is a 1-1 correspondence with $V(G)$, where each covering set with corresponding vertex v is the subset of $\mathcal{C}_F$ which are tracked by v, that is,

$$\mathcal{X} = \{\{(C, s', t') \in \mathcal{C}_F \mid v \text{ tracks } (C, s', t')\}\}_{v \in V(G)}.$$

(iii) the weight function w is the weight function w' defined for WeightedTracking, given the 1-1 correspondence between $\mathcal{X}$ and $V(G)$.

3. Output $T = S \cup F$.

Theorem 4.7. WeightedTracking *admits an* $O(\lg n)$*-approximation algorithm.*

Let us argue about feasibility first. Let $T = S \cup F$ be the output of the algorithm for weighted graphs described in Section 4.5.

Lemma 4.16. T *is a tracking set of* G*.*

Proof. By Lemma 4.1, it is enough to argue that every entry-exit cycle of G is tracked by T. Let F be the FVS computed in the first step of the algorithm and S the set cover computed in the second step. Since $F \subseteq T$, any entry-exit cycle containing at least 3 vertices from F is trivially tracked. All remaining entry-exit cycles are tracked by $S \subseteq T$, by definition. □

Next, we argue about the approximation ratio. The lemma below follows from a proof by Banik *et al.* [47] that Tracking is in NP.

Lemma 4.17. *$|\mathcal{C}_F| \leq O(n^8)$, where n is the number of vertices of the input graph G.*

Proof. Since F is an FVS and G is reduced by Rule 1, every entry-exit cycle of $\mathcal{C}_F$ includes at least 1 vertex from F (see Remark 4.2) and, by definition, at most 2 vertices from F. Since $G - F$ is a forest, there exists at most 1 path between every pair of vertices in $G - F$. Further, each vertex f in F has at most $n - |F|$ neighbors in $G - F$, so there are at most $\binom{n-|F|}{2}$ cycles which contain f and no other vertex from F. Thus, the number of cycles containing exactly 1 vertex from F is at most

$$|F|\binom{n-|F|}{2} \leq n^3$$

Let us now argue about cycles containing exactly 2 vertices f_1 and f_2 from F. Any such cycle is defined by a pair of paths P and Q between f_1 and f_2 traversing vertices in $V(G - F) \cup \{f_1, f_2\}$. Let us first handle cycles where one of the paths P, Q is a single edge, say Q (because G is simple, the other path, P, must consist of at least 2 edges). Clearly, every path P is identified by a path in $G - F$ connecting a neighbor p_1 of f_1 to a neighbor p_2 of f_2. Therefore, there exist at most $(n - |F|)^2$ such paths P[6] and, thus, at most $(n - |F|)^2$ cycles where f_1 and f_2 are connected by an edge. Similarly, for cycles where Q is not an edge, every path Q is identified by a path in $G - F$ connecting a neighbor $q_1 \neq p_1$ of f_1 to a neighbor $q_2 \neq p_2$ of f_2. Hence, there are at most $(n - |F| - 1)^2$ such paths Q and, thus, at most $(n - |F|)^2 \cdot (n - |F| - 1)^2 \leq (n - |F|)^4$ cycles where f_1 and f_2 are not connected by an edge. Taking into account all pairs f_1, f_2 in F, the number of cycles containing exactly 2 vertices from F is at most

$$\binom{|F|}{2}\left((n - |F|)^2 + (n - |F|)^4\right) = O(n^6)$$

[6] In contrast to cycles containing a single vertex from F, the neighbors p_1, p_2 connected by P may be the same vertex, hence we allow repetitions when counting the number of pairs of neighbors.

For every cycle C, there exist at most $|V(C)|(|V(C)|-1) \leq n^2$ entry-exit pairs. Therefore, the number of entry-exit cycles of C_F is at most

$$n^2(n^3 + O(n^6)) = O(n^8)$$

$\square$

Let us denote by OPT the size of an optimal tracking set of G, and let $w(T) = \sum_{a \in T} w(a)$ be the total weight of T.

Lemma 4.18. *$w(T) = O(\lg n)OPT$.*

Proof. By union bound, we have that $w(T) \leq w(S) + w(F)$, where S and F are the sets computed in steps 1 and 2, respectively, of the above algorithm.

Since F is a 2-approximate FVS and every optimal tracking set is also an FVS (by Remark 4.2), we have that

$$w(F) \leq 2OPT$$

Let OPT_F be the size of an optimal solution to the covering problem of step 2, concerning all entry-exit cycles in $\mathcal{C}_F$. Since every optimal tracking set must track all entry-exit cycles in $\mathcal{C}_F$, we have that $OPT \geq OPT_F$. Further, the well known greedy algorithm for SetCover gives us an approximation ratio of at most $(1 + \ln|\mathcal{C}_F|)$. Thus, by Lemma 4.17, $w(S) = O(\lg n)OPT_F$ and therefore,

$$w(S) = O(\lg n)OPT$$

The lemma follows. $\square$

Theorem 4.7, regarding the $O(\lg n)$-approximation for WeightedTracking, follows from Lemmas 4.16 and 4.18.

Unweighted Graphs. We show that the dual of the above set cover formulation has bounded VC-dimension [184, 295]. This immediately improves the approximation ratio to $O(\lg OPT)$ for Tracking (unweighted version) as a consequence of a result by Brönnimann and Goodrich [73], which establishes an approximation-ratio of $O(d \lg(dc))$ for unweighted set cover instances with dual VC-dimension d and optimal covers of size at most c.

Let $(\mathcal{U}, \mathcal{X})$ be a set system and Y a subset of $\mathcal{U}$. We say that Y is *shattered* if $\mathcal{X} \cap Y = 2^Y$, where $\mathcal{X} \cap Y := \{X \cap Y \mid X \in \mathcal{X}\}$. In other words, Y is shattered if the set of intersections of Y with each $X \in \mathcal{X}$ contains all the possible subsets of Y. The set system $(\mathcal{U}, \mathcal{X})$ has *VC-dimension* d if d is the largest integer for which there exists a subset $Y \subseteq \mathcal{U}$, of cardinality $|Y| = d$, that can be shattered.

The dual problem of an unweighted instance $(\mathcal{U}, \mathcal{X})$ of SetCover is finding a *hitting set* of minimum size, where a hitting set is a subset of $\mathcal{U}$ that has a non-empty intersection with every set in $\mathcal{X}$. In our case, it corresponds to finding the smallest subset of entry-exit cycles that covers every vertex, where a vertex is covered if it tracks least one entry-exit cycle in the subset. This is equivalent to an unweighted instance of SetCover with set system $(V, \mathcal{C}_F^*)$, where $V = V(G)$ and $\mathcal{C}_F^* := \{V(C) \setminus \{s', t'\} : (C, s', t') \in \mathcal{C}_F\}$ is the collection of sets, one for each entry-exit cycle, of vertices which can track that entry-exit cycle.

Lemma 4.19. *The set system $(V, \mathcal{C}_F^*)$ has VC-dimension at most 9.*

Proof. We show that there exists no subset $Y \subseteq V$ of size $|Y| \geq 10$ that can be shattered by $\mathcal{C}_F^*$. Since every element of $\mathcal{C}_F^*$ contains at most 2 vertices from F (by definition of $\mathcal{C}_F$), we cannot have more than 2 vertices from F in Y (since we would then require an entry-exit cycle containing at least 3 vertices in F to shatter Y). Thus, the lemma follows if we show

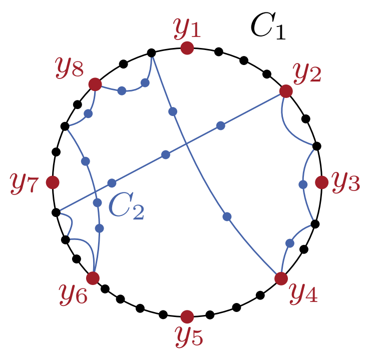


Figure 4.8: Illustration of proof of Lemma 4.19, that the dual VC-dimension is bounded. If a set $Y = \{y_1, y_2, \ldots, y_8\}$ is to be shattered by $\mathcal{C}_F^*$, then there must exist cycles C_1, C_2 traversing, respectively, Y and every other vertex of Y. However, for large enough Y ($|Y| \geq 8$), the existence of C_1, C_2 contradicts that F is an FVS.

that no subset $Y \subseteq V \setminus F$ of size $|Y| \geq 8$ can be shattered by $\mathcal{C}_F^*$. Let us assume, by contradiction, that this is possible. Then, if $Y \subseteq V \setminus F$ is to be shattered by $\mathcal{C}_F^*$, there must exist 2 entry-exit cycles (C_1, s_1', t_1') and (C_2, s_2', t_2') in $\mathcal{C}_F$ (see Fig. 4.8), such that:

- C_1 traverses all vertices of Y, say in the order $y_1, y_2, \ldots, y_{|Y|}$ (for all $y_j \in Y$),
- C_2 traverses every other vertex of Y traversed by C_1, say $Y' = \{y_2, y_4, \ldots, y_{|Y|}\}$, but not necessarily in the same order (we assume w.l.o.g. $|Y|$ is even).

Consider the graph consisting of the union of the cycles C_1, C_2. Let us contract every shared edge between C_1, C_2. Note that C_1 remains a cycle that traverses Y and C_2 remains a cycle that traverses Y' but not any vertex of $Y \setminus Y'$. So we can safely assume that C_1 and C_2 do not share any edges. Thus, the union of C_1, C_2 is a graph with $|C_1| + |C_2| - |Y|/2$ vertices and $|C_1| + |C_2|$ edges. Since both entry-exit cycles are in $\mathcal{C}_F$, each of C_1, C_2 shares at most 2 vertices with F. Let us remove such vertices, say there's $k \leq 4$ of them. The result is a graph with $|C_1| + |C_2| - |Y|/2 - k$ vertices and, at best, $|C_1| + |C_2| - 2k$ edges (the removed vertices cannot be in Y, so they have degree 2). In order for this graph to be acyclic (since F is an FVS by Remark 4.2, and our contractions preserve cycles) we would then require $|Y| < 8$ (since any acyclic graph with n vertices has at most $n - 1$ edges), a contradiction. □

The above lemma, combined with the result of Brönnimann and Goodrich [73] gives us the following.

Theorem 4.8. Tracking *admits an $O(\lg OPT)$-approximation algorithm, where OPT is the size of an optimal tracking set.*

4.6 Conclusion and Open Questions

We have designed a $(1+\epsilon)$-approximation algorithm, as well as an FPT algorithm with a linear kernel for H-minor-free graphs, generalizing our result for planar graphs in Chapter 3. We also gave the first approximation algorithm of ratio $O(\lg n)$ for the vertex-weighted version of this problem, and we showed that the set system for the unweighted version has bounded dual VC-dimension, which immediately implied a $O(\lg OPT)$-approximation in the unweighted setting.

A natural direction for future work would be to obtain a $(1+\epsilon)$-approximation for vertex-weighted H-minor-free graphs, or show that this is not likely, under some reasonable assumption, such as P$\neq$NP. Our EPTAS is not very practical in terms of running time, so we question whether there exist more efficient solutions in this setting.

Another next step would be characterizing the tractability of Tracking and the extent to which it can be approximated, when the input graph is directed. If the graph is a tournament, we know that the problem is in P, but we do not know whether the more general problem is even in NP – our proof of containment for the undirected case relies on the fact that every tracking set is a feedback vertex set (for the graph containing all relevant cycles), but this is no longer the case in the directed case (see Fig. 4.9).

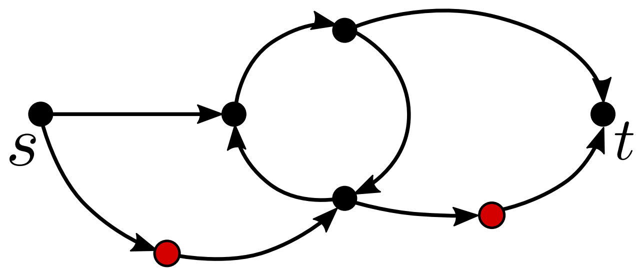


Figure 4.9: Illustration of a tracking set that is not a feedback vertex set in a directed graph. For this reason, one cannot reuse the proof of containment in NP when the input graph is directed, and thus the question of whether it is in NP is still open.

Chapter 5

Conclusion

We have studied the problem of exact learning of sequences, under two different settings: learning of strings of a fixed alphabet, and learning of paths in a fixed graph.

When learning strings (Chapter 2), we exploited periodicity to give algorithms that learn, with sublinear query-complexities, a hidden periodic string using substring and subsequence oracles. We also gave an algorithm that recovers an near-periodic string using a sublinear amount of substring queries. When learning general strings using jumbled-indexing queries, we show that even if the location of a matching substring is given to us in an adversarial way, one cannot learn the secret string in general, so we gave an efficient randomized learning algorithm that recovers the string with high probability, when the location of a matching substring is given uniformly at random.

To learn paths in a graph, we considered an oracle whose answers may depend on the learner's placement of trackers in the input graph. We proved that the problem of minimizing the number of trackers remains NP-complete when the input graph is planar and we gave a 4-approximation algorithm in this setting, as well as an optimal polynomial-time algorithm for bounded clique-width graphs (Chapter 3). Later, we designed a $(1+\epsilon)$-approximation algorithm, as well as an FPT algorithm with a linear kernel for H-minor-free graphs (Chapter 4), generalizing our result for planar graphs. We also gave the first approximation algorithm of ratio $O(\lg n)$ for the vertex-weighted version of this problem, and we showed that the set system for the unweighted version has bounded dual VC-dimension, which immediately implied a $O(\lg OPT)$-approximation in the unweighted setting, where OPT is the size of an optimal tracking set.

Bibliography

[1] H. Abasi, N. H. Bshouty, and H. Mazzawi. On exact learning monotone DNF from membership queries. In P. Auer, A. Clark, T. Zeugmann, and S. Zilles, editors, *Algorithmic Learning Theory - 25th International Conference, ALT 2014, Bled, Slovenia, October 8-10, 2014. Proceedings*, volume 8776 of *Lecture Notes in Computer Science*, pages 111–124. Springer, 2014.

[2] H. Abasi, N. H. Bshouty, and H. Mazzawi. Non-adaptive learning of a hidden hypergraph. *Theor. Comput. Sci.*, 716:15–27, 2018.

[3] M. Abrahamsen, G. Bodwin, E. Rotenberg, and M. Stöckel. Graph reconstruction with a betweenness oracle. In N. Ollinger and H. Vollmer, editors, *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, volume 47 of *LIPIcs*, pages 5:1–5:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

[4] J. Acharya, H. Das, O. Milenkovic, A. Orlitsky, and S. Pan. Quadratic-backtracking algorithm for string reconstruction from substring compositions. In *2014 IEEE International Symposium on Information Theory, Honolulu, HI, USA, June 29 - July 4, 2014*, pages 1296–1300. IEEE, 2014.

[5] J. Acharya, H. Das, O. Milenkovic, A. Orlitsky, and S. Pan. String reconstruction from substring compositions. *SIAM J. Discret. Math.*, 29(3):1340–1371, 2015.

[6] P. Afshani, M. Agrawal, B. Doerr, C. Doerr, K. G. Larsen, and K. Mehlhorn. The query complexity of finding a hidden permutation. In A. Brodnik, A. López-Ortiz, V. Raman, and A. Viola, editors, *Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*, volume 8066 of *Lecture Notes in Computer Science*, pages 1–11. Springer, 2013.

[7] P. Afshani, I. van Duijn, R. Killmann, and J. S. Nielsen. A lower bound for jumbled indexing. In S. Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 592–606. SIAM, 2020.

[8] P. Afshani, I. van Duijn, R. Killmann, and J. S. Nielsen. A lower bound for jumbled indexing. In *2020 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 592–606, 2020.

[9] R. Afshar, A. Amir, M. T. Goodrich, and P. Matias. Adaptive exact learning in a mixed-up world: Dealing with periodicity, errors and jumbled-index queries in string reconstruction. In C. Boucher and S. V. Thankachan, editors, *String Processing and Information Retrieval - 27th International Symposium, SPIRE 2020, Orlando, FL, USA, October 13-15, 2020, Proceedings*, volume 12303 of *Lecture Notes in Computer Science*, pages 155–174. Springer, 2020.

[10] R. Afshar, M. T. Goodrich, P. Matias, and M. C. Osegueda. Reconstructing binary trees in parallel. In C. Scheideler and M. Spear, editors, *SPAA '20: 32nd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, July 15-17, 2020*, pages 491–492. ACM, 2020.

[11] R. Afshar, M. T. Goodrich, P. Matias, and M. C. Osegueda. Reconstructing biological and digital phylogenetic trees in parallel. In F. Grandoni, G. Herman, and P. Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 3:1–3:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[12] R. Afshar, M. T. Goodrich, P. Matias, and M. C. Osegueda. Brief announcement: Parallel network mapping algorithms. In *SPAA '21: Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '21), July 6–8, 2021, Virtual Event, USA*. ACM, 2021.

[13] M. Aigner. *Combinatorial search.* John Wiley & Sons, Inc., 1988.

[14] M. Aigner and A. Li. Searching for counterfeit coins. *Graphs Comb.*, 13(1):9–20, 1997.

[15] N. Alon and V. Asodi. Learning a hidden subgraph. *SIAM J. Discret. Math.*, 18(4):697–712, 2005.

[16] N. Alon, R. Beigel, S. Kasif, S. Rudich, and B. Sudakov. Learning a hidden matching. *SIAM J. Comput.*, 33(2):487–501, 2004.

[17] N. Alon, P. D. Seymour, and R. Thomas. A separator theorem for graphs with an excluded minor and its applications. In H. Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 293–299. ACM, 1990.

[18] A. Amir, M. Amit, G.M.Landau, and D. Sokol. Period recovery of strings over the hamming and edit distances. *Theortetical Computer Science*, 710:2–18, 2018.

[19] A. Amir, A. Apostolico, T. Hirst, G. M. Landau, N. Lewenstein, and L. Rozenberg. Algorithms for jumbled indexing, jumbled border and jumbled square on run-length encoded strings. *Theor. Comput. Sci.*, 656:146–159, 2016.

[20] A. Amir, A. Apostolico, T. Hirst, G. M. Landau, N. Lewenstein, and L. Rozenberg. Algorithms for jumbled indexing, jumbled border and jumbled square on run-length encoded strings. *Theoretical Computer Science*, 656:146–159, 2016.

[21] A. Amir, Y. Aumann, G. Benson, A. Levy, O. Lipsky, E. Porat, S. Skiena, and U. Vishne. Pattern matching with address errors: Rearrangement distances. *J. Comput. Syst. Sci.*, 75(6):359–370, 2009.

[22] A. Amir, Y. Aumann, G. Landau, M. Lewenstein, and N. Lewenstein. Pattern matching with swaps. *Journal of Algorithms*, 37:247–266, 2000. (Preliminary version appeared at FOCS 97.).

[23] A. Amir, A. Butman, and E. Porat. On the relationship between histogram indexing and block-mass indexing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 372(2016):20130132, 2014.

[24] A. Amir, T. M. Chan, M. Lewenstein, and N. Lewenstein. On hardness of jumbled indexing. In J. Esparza, P. Fraigniaud, T. Husfeldt, and E. Koutsoupias, editors, *Int. Colloq. on Automata, Languages, and Programming (ICALP)*, pages 114–125, Berlin, Heidelberg, 2014. Springer.

[25] A. Amir, E. Eisenberg, A. Levy, E. Porat, and N. Shapira. Cycle detection and correction. *ACM Trans. Alg.*, 9(1):13, 2012.

[26] A. Amir, T. Hartman, O. Kapah, A. Levy, and E. Porat. On the cost of interchange rearrangement in strings. In L. Arge, M. Hoffmann, and E. Welzl, editors, *Algorithms - ESA 2007, 15th Annual European Symposium, Eilat, Israel, October 8-10, 2007, Proceedings*, volume 4698 of *Lecture Notes in Computer Science*, pages 99–110. Springer, 2007.

[27] D. Angluin. A note on the number of queries needed to identify regular languages. *Inf. Control.*, 51(1):76–87, 1981.

[28] D. Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987.

[29] D. Angluin. Queries and concept learning. *Mach. Learn.*, 2(4):319–342, 1987.

[30] D. Angluin. Queries and concept learning. *Machine learning*, 2(4):319–342, 1988.

[31] D. Angluin. Computational learning theory: Survey and selected bibliography. In S. R. Kosaraju, M. Fellows, A. Wigderson, and J. A. Ellis, editors, *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 351–369. ACM, 1992.

[32] D. Angluin and J. Chen. Learning a hidden graph using o(log n) queries per edge. In J. Shawe-Taylor and Y. Singer, editors, *Learning Theory, 17th Annual Conference on Learning Theory, COLT 2004, Banff, Canada, July 1-4, 2004, Proceedings*, volume 3120 of *Lecture Notes in Computer Science*, pages 210–223. Springer, 2004.

[33] D. Angluin and J. Chen. Learning a hidden hypergraph. *J. Mach. Learn. Res.*, 7:2215–2236, 2006.

[34] D. Angluin and C. H. Smith. Inductive inference: Theory and methods. *ACM Comput. Surv.*, 15(3):237–269, 1983.

[35] A. Apostolico, C. Iliopoulos, and M. Farach. Optimal superprimativity testing for strings. *Information Processing Letters*, 39:17–20, 1991.

[36] R. Arratia, D. Martin, G. Reinert, and M. S. Waterman. Poisson process approximation for sequence repeats and sequencing by hybridization. *J. Comput. Biol.*, 3(3):425–463, 1996.

[37] P. Auer. On-line learning of rectangles in noisy environments. In L. Pitt, editor, *Proceedings of the Sixth Annual ACM Conference on Computational Learning Theory, COLT 1993, Santa Cruz, CA, USA, July 26-28, 1993*, pages 253–261. ACM, 1993.

[38] V. Bafna, P. Berman, and T. Fujito. Constant ratio approximations of the weighted feedback vertex set problem for undirected graphs. In J. Staples, P. Eades, N. Katoh, and A. Moffat, editors, *ISAAC*, volume 1004 of *Lecture Notes in Computer Science*, pages 142–151. Springer, 1995.

[39] N. T. Bailey. *The Mathematical Theory of Infectious Diseases and its Applications.* Charles Griffin & Company Ltd, High Wycombe, United Kingdom, 2nd edition, 1975.

[40] B. S. Baker. Approximation algorithms for np-complete problems on planar graphs. *J. ACM*, 41(1):153–180, 1994.

[41] B. S. Baker. Parameterized pattern matching: Algorithms and applications. *Journal of Computer and System Sciences*, 52(1):28–42, 1996.

[42] B. S. Baker. Parameterized duplication in strings: Algorithms and an application to software maintenance. *SIAM Journal on Computing*, 26(5):1343–1362, 1997.

[43] D. Balding, W. Bruno, D. Torney, and E. Knill. A comparative survey of non-adaptive pooling designs. In *Genetic mapping and DNA sequencing*, pages 133–154. Springer, 1996.

[44] D. J. Balding and D. C. Torney. Optimal pooling designs with error detection. *J. Comb. Theory, Ser. A*, 74(1):131–140, 1996.

[45] A. Banik and P. Choudhary. Fixed-parameter tractable algorithms for tracking set problems. In B. S. Panda and P. P. Goswami, editors, *Algorithms and Discrete Applied Mathematics - 4th International Conference, CALDAM 2018, Guwahati, India, February 15-17, 2018, Proceedings*, volume 10743 of *Lecture Notes in Computer Science*, pages 93–104. Springer, 2018.

[46] A. Banik, P. Choudhary, D. Lokshtanov, V. Raman, and S. Saurabh. A polynomial sized kernel for tracking paths problem. In M. A. Bender, M. Farach-Colton, and M. A. Mosteiro, editors, *LATIN 2018: Theoretical Informatics - 13th Latin American Symposium, Buenos Aires, Argentina, April 16-19, 2018, Proceedings*, volume 10807 of *Lecture Notes in Computer Science*, pages 94–107. Springer, 2018.

[47] A. Banik, P. Choudhary, D. Lokshtanov, V. Raman, and S. Saurabh. A polynomial sized kernel for tracking paths problem. *Algorithmica*, 82(1):41–63, 2020.

[48] A. Banik, M. J. Katz, E. Packer, and M. Simakov. Tracking paths. In D. Fotakis, A. Pagourtzis, and V. T. Paschos, editors, *Algorithms and Complexity - 10th International Conference, CIAC 2017, Athens, Greece, May 24-26, 2017, Proceedings*, volume 10236 of *Lecture Notes in Computer Science*, pages 67–79, 2017.

[49] T. Batu, S. Kannan, S. Khanna, and A. McGregor. Reconstructing strings from random traces. In J. I. Munro, editor, *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*, pages 910–918. SIAM, 2004.

[50] A. Becker, R. Bar-Yehuda, and D. Geiger. Randomized algorithms for the loop cutset problem. *J. Artif. Intell. Res.*, 12:219–234, 2000.

[51] A. Becker and D. Geiger. Approximation algorithms for the loop cutset problem. In R. L. de Mántaras and D. Poole, editors, *UAI '94: Proceedings of the Tenth Annual Conference on Uncertainty in Artificial Intelligence, Seattle, Washington, USA, July 29-31, 1994*, pages 60–68. Morgan Kaufmann, 1994.

[52] Z. Beerliova, F. Eberhard, T. Erlebach, A. Hall, M. Hoffmann, M. Mihalák, and L. S. Ram. Network discovery and verification. *IEEE J. Sel. Areas Commun.*, 24(12):2168–2181, 2006.

[53] R. Beigel, N. Alon, S. Kasif, M. S. Apaydin, and L. Fortnow. An optimal procedure for gap closing in whole genome shotgun sequencing. In T. Lengauer, editor, *Proceedings of the Fifth Annual International Conference on Computational Biology, RECOMB 2001, Montréal, Québec, Canada, April 22-25, 2001*, pages 22–30. ACM, 2001.

[54] G. Benson. Tandem repeats finder: a program to analyze dna sequence. *Nucleic Acids Research*, 27(2):573–580, 1999.

[55] G. Benson and M. Waterman. A method for fast database search for all k-nucleotide repeats. *Nucleic Acids Research*, 22:4828–4836, 1994.

[56] J. L. Bentley and A. C. Yao. An almost optimal algorithm for unbounded searching. *Inf. Process. Lett.*, 5(3):82–87, 1976.

[57] A. Bernasconi, C. Damm, and I. Shparlinski. Circuit and decision tree complexity of some number theoretic problems. *Information and Computation*, 168(2):113 – 124, 2001.

[58] A. Bernasconi, C. Damm, and I. E. Shparlinski. Circuit and decision tree complexity of some number theoretic problems. *Inf. Comput.*, 168(2):113–124, 2001.

[59] U. Bertele and F. Brioschi. *Nonserial Dynamic Programming*. Academic Press, 1972.

[60] S. Bhatti and J. Xu. Survey of target tracking protocols using wireless sensor network. In *2009 Fifth International Conference on Wireless and Mobile Communications*, pages 110–115. IEEE, 2009.

[61] E. Biglieri and L. Györfi. *Multiple access channels: theory and practice*, volume 10. IOS press, 2007.

[62] D. Bilò, L. Gualà, S. Leucci, and G. Proietti. Tracking routes in communication networks. *Theor. Comput. Sci.*, 844:1–15, 2020.

[63] P. Biswal, J. R. Lee, and S. Rao. Eigenvalue bounds, spectral partitioning, and metrical deformations via flows. *J. ACM*, 57(3):13:1–13:23, 2010.

[64] A. Blum. Separating distribution-free and mistake-bound learning models over the boolean domain. *SIAM J. Comput.*, 23(5):990–1000, 1994.

[65] A. Blum, L. Hellerstein, and N. Littlestone. Learning in the presence of finitely or infinitely many irrelevant attributes. *J. Comput. Syst. Sci.*, 50(1):32–40, 1995.

[66] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Learnability and the vapnik-chervonenkis dimension. *J. ACM*, 36(4):929–965, 1989.

[67] H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996.

[68] A. D. Bonis, L. Gargano, and U. Vaccaro. Group testing with unreliable tests. *Inf. Sci.*, 96(1&2):1–14, 1997.

[69] A. D. Bonis, L. Gasieniec, and U. Vaccaro. Optimal two-stage algorithms for group testing problems. *SIAM J. Comput.*, 34(5):1253–1270, 2005.

[70] A. D. Bonis and U. Vaccaro. Improved algorithms for group testing with inhibitors. *Inf. Process. Lett.*, 67(2):57–64, 1998.

[71] G. Borradaile, H. Le, and B. Zheng. Engineering a PTAS for minimum feedback vertex set in planar graphs. In I. S. Kotsireas, P. M. Pardalos, K. E. Parsopoulos, D. Souravlias, and A. Tsokas, editors, *Analysis of Experimental Algorithms - Special Event, SEA^2 2019, Kalamata, Greece, June 24-29, 2019, Revised Selected Papers SEA^2*, volume 11544 of *Lecture Notes in Computer Science*, pages 98–113. Springer, 2019.

[72] G. Bresler, M. Bresler, and D. Tse. Optimal assembly for high throughput shotgun sequencing. In *BMC Bioinformatics*, volume 14, page S18. Springer, 2013.

[73] H. Brönnimann and M. T. Goodrich. Almost optimal set covers in finite vc-dimension. *Discret. Comput. Geom.*, 14(4):463–479, 1995.

[74] N. H. Bshouty. Exact learning boolean functions via the monotone theory. *Electron. Colloquium Comput. Complex.*, 2(8), 1995.

[75] N. H. Bshouty. Exact learning from an honest teacher that answers membership queries. *Theor. Comput. Sci.*, 733:4–43, 2018.

[76] N. H. Bshouty, R. Cleve, R. Gavaldà, S. Kannan, and C. Tamon. Oracles and queries that are sufficient for exact learning. *J. Comput. Syst. Sci.*, 52(3):421–433, 1996.

[77] N. H. Bshouty and A. Costa. Exact learning of juntas from membership queries. *Theor. Comput. Sci.*, 742:82–97, 2018.

[78] N. H. Bshouty and A. Gabizon. Almost optimal cover-free families. In D. Fotakis, A. Pagourtzis, and V. T. Paschos, editors, *Algorithms and Complexity - 10th International Conference, CIAC 2017, Athens, Greece, May 24-26, 2017, Proceedings*, volume 10236 of *Lecture Notes in Computer Science*, pages 140–151, 2017.

[79] N. H. Bshouty, P. W. Goldberg, S. A. Goldman, and H. D. Mathias. Exact learning of discretized geometric concepts. *SIAM J. Comput.*, 28(2):674–699, 1998.

[80] N. H. Bshouty and C. A. Haddad-Zaknoon. Adaptive exact learning of decision trees from membership queries. In A. Garivier and S. Kale, editors, *Algorithmic Learning Theory, ALT 2019, 22-24 March 2019, Chicago, Illinois, USA*, volume 98 of *Proceedings of Machine Learning Research*, pages 207–234. PMLR, 2019.

[81] P. Burcsi, F. Cicalese, G. Fici, and Z. Lipták. Algorithms for jumbled pattern matching in strings. *Int. J. Found. Comput. Sci.*, 23(2):357–374, 2012.

[82] A. Butman, R. Eres, and G. M. Landau. Scaled and permuted string matching. *Inf. Process. Lett.*, 92(6):293–297, 2004.

[83] A. Carpi and A. de Luca. Words and special factors. *Theor. Comput. Sci.*, 259(1-2):145–182, 2001.

[84] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart. Leakage-abuse attacks against searchable encryption. In *In Proc. of the 22nd ACM SIGSAC Conference on Computer and Communications Security, CCS*, pages 668–679, 2015.

[85] A. Cayley. Lxxvii. note on the theory of permutations. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 34(232):527–529, 1849.

[86] H. Chang, H. Chen, H. Fu, and C. Shi. Reconstruction of hidden graphs and threshold group testing. *J. Comb. Optim.*, 22(2):270–281, 2011.

[87] H. Chang, H. Fu, and C. Shih. Learning a hidden graph. *Optim. Lett.*, 8(8):2341–2348, 2014.

[88] H. Chang, H. Fu, and C. Shih. Learning a hidden uniform hypergraph. *Optim. Lett.*, 12(1):55–62, 2018.

[89] Z. Chang, J. Chrisnata, M. F. Ezerman, and H. M. Kiah. Rates of DNA sequence profiles for practical values of read lengths. *IEEE Trans. Inf. Theory*, 63(11):7166–7177, 2017.

[90] Z. Chen. Learning unions of two rectangles in the plane with equivalence queries. In L. Pitt, editor, *Proceedings of the Sixth Annual ACM Conference on Computational Learning Theory, COLT 1993, Santa Cruz, CA, USA, July 26-28, 1993*, pages 243–252. ACM, 1993.

[91] Z. Chen, C. Cunha, and S. Homer. Finding a hidden code by asking questions. In J. Cai and C. K. Wong, editors, *Computing and Combinatorics, Second Annual International Conference, COCOON '96, Hong Kong, June 17-19, 1996, Proceedings*, volume 1090 of *Lecture Notes in Computer Science*, pages 50–55. Springer, 1996.

[92] Z. Chen and S. Homer. Learning unions of rectangles with queries. *Unpublished manuscript, July*, 1993.

[93] Z. Chen and S. Homer. The bounded injury priority method and the learnability of unions of rectangles. *Ann. Pure Appl. Log.*, 77(2):143–168, 1996.

[94] Z. Chen and W. Maass. On-line learning of rectangles. In D. Haussler, editor, *Proceedings of the Fifth Annual ACM Conference on Computational Learning Theory, COLT 1992, Pittsburgh, PA, USA, July 27-29, 1992*, pages 16–28. ACM, 1992.

[95] N. Chiba, T. Nishizeki, and N. Saito. Applications of the lipton and tarjan's planar separator theorem. *J. Inf. Process*, 4(4):203–207, 1981.

[96] F. Y. L. Chin, H. C. M. Leung, and S. Yiu. Non-adaptive complex group testing with multiple positive sets. *Theor. Comput. Sci.*, 505:11–18, 2013.

[97] S. Cho, J. C. Na, K. Park, and J. S. Sim. Fast order-preserving pattern matching. In *Proc. 7th conf. Combinatorial Optimization and Applications COCOA*, volume 8287 of *Lecture Notes in Computer Science*, pages 295–305. Springer, 2013.

[98] S. Choi and J. H. Kim. Optimal query complexity bounds for finding graphs. *Artif. Intell.*, 174(9-10):551–569, 2010.

[99] S.-S. Choi and J. H. Kim. Optimal query complexity bounds for finding graphs. *Artificial Intelligence*, 174(9):551 – 569, 2010.

[100] C. Chokchai. Low cost and high performance UHF RFID system using Arduino based on IoT applications for marathon competition. In *2018 21st International Symposium on Wireless Personal Multimedia Communications (WPMC)*, pages 15–20, 2018.

[101] P. Choudhary. Polynomial time algorithms for tracking path problems. In L. Gasieniec, R. Klasing, and T. Radzik, editors, *Combinatorial Algorithms - 31st International Workshop, IWOCA 2020, Bordeaux, France, June 8-10, 2020, Proceedings*, volume 12126 of *Lecture Notes in Computer Science*, pages 166–179. Springer, 2020.

[102] P. Choudhary and V. Raman. Improved kernels for tracking path problems. *CoRR*, abs/2001.03161, 2020.

[103] P. Choudhary and V. Raman. Structural parameterizations of tracking paths problem. In G. Cordasco, L. Gargano, and A. A. Rescigno, editors, *Proceedings of the 21st Italian Conference on Theoretical Computer Science, Ischia, Italy, September 14-16, 2020*, volume 2756 of *CEUR Workshop Proceedings*, pages 15–27. CEUR-WS.org, 2020.

[104] F. A. Chudak, M. X. Goemans, D. S. Hochbaum, and D. P. Williamson. A primal-dual interpretation of two 2-approximation algorithms for the feedback vertex set problem in undirected graphs. *Oper. Res. Lett.*, 22(4-5):111–118, 1998.

[105] F. R. Chung. *Separator theorems and their applications.* Universität Bonn. Institut für Ökonometrie und Operations Research, 1988.

[106] V. Chvátal. A greedy heuristic for the set-covering problem. *Math. Oper. Res.*, 4(3):233–235, 1979.

[107] V. Chvátal. Mastermind. *Comb.*, 3(3):325–329, 1983.

[108] F. Cicalese. Group testing. In *Fault-Tolerant Search Algorithms*, pages 139–173. Springer, 2013.

[109] F. Cicalese, G. Fici, and Z. Lipták. Searching for jumbled patterns in strings. In J. Holub and J. Zdárek, editors, *Proceedings of the Prague Stringology Conference 2009, Prague, Czech Republic, August 31 - September 2, 2009*, pages 105–117. Prague Stringology Club, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, 2009.

[110] L. Cieplinski. MPEG-7 color descriptors and their applications. In W. Skarbek, editor, *Proc. 9th Intl. Conf. on Computer Analysis of Images and Patterns CAIP*, volume 2124 of *LNCS*, pages 11–20. Springer, 2001.

[111] M. Clausen, A. W. M. Dress, J. Grabmeier, and M. Karpinski. On zero-testing and interpolation of k-sparse multivariate polynomials over finite fields. *Theor. Comput. Sci.*, 84(2):151–164, 1991.

[112] R. Cleve, K. Iwama, F. Le Gall, H. Nishimura, S. Tani, J. Teruyama, and S. Yamashita. Reconstructing strings from substrings with quantum queries. In F. V. Fomin and P. Kaski, editors, *Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 388–397, Berlin, Heidelberg, 2012. Springer.

[113] B. Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990.

[114] B. Courcelle and I. Durand. Automata for the verification of monadic second-order graph properties. *J. Applied Logic*, 10(4):368–409, 2012.

[115] B. Courcelle and I. Durand. Computations by fly-automata beyond monadic second-order logic. *Theor. Comput. Sci.*, 619:32–67, 2016.

[116] B. Courcelle and J. Engelfriet. *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*, volume 138 of *Encyclopedia of mathematics and its applications.* Cambridge University Press, 2012.

[117] B. Courcelle, J. Engelfriet, and G. Rozenberg. Handle-rewriting hypergraph grammars. *Journal of Computer and System Sciences*, 46(2):218–270, 1993.

[118] B. Courcelle, J. A. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, 33(2):125–150, 2000.

[119] B. Courcelle and M. Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theoretical Computer Science*, 109(1-2):49–82, 1993.

[120] B. Courcelle and S. Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000.

[121] M. Crochemore, C. S. Iliopoulos, T. Kociumaka, M. Kubica, A. Langiu, S. P. Pissis, J. Radoszewski, W. Rytter, and T. Walen. Order-preserving indexing. *Theoretcial Computer Science*, 613:122–135, 2016.

[122] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. *Journal of Computer Security*, 19(5):895–934, 2011.

[123] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms.* Springer, 2015.

[124] T. Dakic. *On the turnpike problem.* Simon Fraser University BC, Canada, 2000.

[125] P. Damaschke. Computational aspects of parallel attribute-efficient learning. In M. M. Richter, C. H. Smith, R. Wiehagen, and T. Zeugmann, editors, *Algorithmic Learning Theory, 9th International Conference, ALT '98, Otzenhausen, Germany, October 8-10, 1998, Proceedings*, volume 1501 of *Lecture Notes in Computer Science*, pages 103–111. Springer, 1998.

[126] P. Damaschke. Randomized group testing for mutually obscuring defectives. *Inf. Process. Lett.*, 67(3):131–135, 1998.

[127] P. Damaschke. Adaptive versus nonadaptive attribute-efficient learning. *Mach. Learn.*, 41(2):197–215, 2000.

[128] P. Damaschke. On parallel attribute-efficient learning. *J. Comput. Syst. Sci.*, 67(1):46–62, 2003.

[129] P. Deininger. SINEs: short interspersed repeated DNA elements in higher eukaryotes. In D. Berg and M. Howe, editors, *Mobile DNA*, chapter 27, pages 619–636. American Society for Microbiology, 1989.

[130] E. D. Demaine, F. V. Fomin, M. T. Hajiaghayi, and D. M. Thilikos. Subexponential parameterized algorithms on graphs of bounded-genus and H-minor-free graphs. In J. I. Munro, editor, *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*, pages 830–839. SIAM, 2004.

[131] E. D. Demaine and M. Hajiaghayi. The bidimensionality theory and its algorithmic applications. *Comput. J.*, 51(3):292–302, 2008.

[132] E. D. Demaine and M. T. Hajiaghayi. Bidimensionality: new connections between FPT algorithms and ptass. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, British Columbia, Canada, January 23-25, 2005*, pages 590–601. SIAM, 2005.

[133] T. Deselaers, D. Keysers, and H. Ney. Features for image retrieval: an experimental comparison. *Inf. Retr.*, 11(2):77–107, 2008.

[134] A. Dhagat and L. Hellerstein. PAC learning with irrelevant attributes. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 64–74. IEEE Computer Society, 1994.

[135] H. Djidjev and S. M. Venkatesan. Reduced constants for simple cycle graph separation. *Acta Informatica*, 34(3):231–243, 1997.

[136] H. N. Djidjev. On the problem of partitioning planar graphs. *SIAM Journal on Algebraic Discrete Methods*, 3(2):229–240, 1982.

[137] H. N. Djidjev. A linear algorithm for partitioning graphs of fixed genus. *Serdica. Bulgariacae mathematicae publicationes*, 11(4):369–387, 1985.

[138] S. Dobzinski and J. Vondrák. From query complexity to computational complexity. In H. J. Karloff and T. Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 1107–1116. ACM, 2012.

[139] S. Dobzinski and J. Vondrak. From query complexity to computational complexity. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, STOC '12, pages 1107–1116, New York, NY, USA, 2012. ACM.

[140] N. O. Domaniç and F. P. Preparata. A novel approach to the detection of genomic approximate tandem repeats in the levenshtein metric. *Journal of Computational Biology*, 14(7):873–891, 2007.

[141] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999.

[142] D. Du, F. K. Hwang, and F. Hwang. *Combinatorial group testing and its applications*, volume 12. World Scientific, 2000.

[143] M. Dudík and L. J. Schulman. Reconstruction from subsequences. *J. Comb. Theory, Ser. A*, 103(2):337–348, 2003.

[144] J. Dudley, M.-T. Lin, D. Le, and J.R.Eshleman. Microsatellite instability as a biomarker for pd-1 blockade. *Clinical Cancer Research*, 22(4):813–820, 2016.

[145] A. G. D'yachkov, I. V. Vorobyev, N. A. Polyanskii, and V. Y. Shchukin. Adaptive learning a hidden hypergraph. *CoRR*, abs/1607.00507, 2016.

[146] A. G. D'yachkov, I. V. Vorobyev, N. A. Polyanskii, and V. Y. Shchukin. On multistage learning a hidden hypergraph. In *IEEE International Symposium on Information Theory, ISIT 2016, Barcelona, Spain, July 10-15, 2016*, pages 1178–1182. IEEE, 2016.

[147] O. Elishco, R. Gabrys, M. Médard, and E. Yaakobi. Repeat-free codes. In *IEEE International Symposium on Information Theory, ISIT 2019, Paris, France, July 7-12, 2019*, pages 932–936. IEEE, 2019.

[148] D. Eppstein. Subgraph isomorphism in planar graphs and related problems. *J. Graph Algorithms Appl.*, 3(3):1–27, 1999.

[149] D. Eppstein. Diameter and treewidth in minor-closed graph families. *Algorithmica*, 27(3):275–291, 2000.

[150] D. Eppstein and M. T. Goodrich. Studying (non-planar) road networks through an algorithmic lens. In *16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS. ACM, 2008.

[151] D. Eppstein, M. T. Goodrich, J. A. Liu, and P. Matias. Tracking paths in planar graphs. In P. Lu and G. Zhang, editors, *30th International Symposium on Algorithms and Computation, ISAAC 2019, December 8-11, 2019, Shanghai University of Finance and Economics, Shanghai, China*, volume 149 of *LIPIcs*, pages 54:1–54:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[152] D. Eppstein and S. Gupta. Crossing patterns in nonplanar road networks. In *25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS. ACM, 2017.

[153] D. Eppstein and H. Khodabandeh. On the edge crossings of the greedy spanner. *CoRR*, abs/2002.05854, 2020.

[154] P. Erdos and A. Rényi. On two problems of information theory. *Magyar Tud. Akad. Mat. Kutató Int. Közl*, 8:229–243, 1963.

[155] R. Eres, G. M. Landau, and L. Parida. Permutation pattern discovery in biosequences. *J. Comput. Biol.*, 11(6):1050–1060, 2004.

[156] T. Erlebach, A. Hall, M. Hoffmann, and M. Mihalák. Network discovery and verification with distance queries. In T. Calamoneri, I. Finocchi, and G. F. Italiano, editors, *Algorithms and Complexity, 6th Italian Conference, CIAC 2006, Rome, Italy, May*

29-31, 2006, Proceedings, volume 3998 of *Lecture Notes in Computer Science*, pages 69–80. Springer, 2006.

[157] M. Farach, S. Kannan, E. Knill, and S. Muthukrishnan. Group testing problems with sequences in experimental molecular biology. In B. Carpentieri, A. D. Santis, U. Vaccaro, and J. A. Storer, editors, *Compression and Complexity of SEQUENCES 1997, Positano, Amalfitan Coast, Salerno, Italy, June 11-13, 1997, Proceedings*, pages 357–367. IEEE, 1997.

[158] G. Fici, F. Mignosi, A. Restivo, and M. Sciortino. Word assembly through minimal forbidden words. *Theor. Comput. Sci.*, 359(1-3):214–230, 2006.

[159] N. J. Fine and H. S. Wilf. Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society*, 16(1):109–114, 1965.

[160] P. Fischer, N. Klasner, and I. Wegener. On the cut-off point for combinatorial group testing. *Discret. Appl. Math.*, 91(1-3):83–92, 1999.

[161] R. Focardi and F. L. Luccio. Cracking bank pins by playing mastermind. In P. Boldi and L. Gargano, editors, *Fun with Algorithms, 5th International Conference, FUN 2010, Ischia, Italy, June 2-4, 2010. Proceedings*, volume 6099 of *Lecture Notes in Computer Science*, pages 202–213. Springer, 2010.

[162] F. V. Fomin, E. D. Demaine, M. T. Hajiaghayi, and D. M. Thilikos. Bidimensionality. In *Encyclopedia of Algorithms*, pages 203–207. 2016.

[163] F. V. Fomin, D. Lokshtanov, S. Saurabh, and D. M. Thilikos. Bidimensionality and kernels. In M. Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 503–510. SIAM, 2010.

[164] E. Fox-Epstein, S. Mozes, P. M. Phothilimthana, and C. Sommer. Short and simple cycle separators in planar graphs. *ACM J. Exp. Algorithmics*, 21(1):2.2:1–2.2:24, 2016.

[165] G. N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. Comput.*, 16(6):1004–1022, 1987.

[166] R. Gabrys and O. Milenkovic. The hybrid k-deck problem: Reconstructing sequences from short and long traces. In *2017 IEEE International Symposium on Information Theory, ISIT 2017, Aachen, Germany, June 25-30, 2017*, pages 1306–1310. IEEE, 2017.

[167] R. Gabrys and O. Milenkovic. Unique reconstruction of coded sequences from multiset substring spectra. In *2018 IEEE International Symposium on Information Theory, ISIT 2018, Vail, CO, USA, June 17-22, 2018*, pages 2540–2544. IEEE, 2018.

[168] S. Ganguly, E. Mossel, and M. Z. Rácz. Sequence assembly from corrupted shotgun reads. In *IEEE International Symposium on Information Theory, ISIT 2016, Barcelona, Spain, July 10-15, 2016*, pages 265–269. IEEE, 2016.

[169] H. Gao, F. K. Hwang, M. T. Thai, W. Wu, and T. Znati. Construction of *d(H)*-disjunct matrix for group testing in hypergraphs. *J. Comb. Optim.*, 12(3):297–301, 2006.

[170] R. Gavalda. On the power of equivalence queries. 1994.

[171] H. Gazit and G. L. Miller. Planar separators and the euclidean norm. In T. Asano, T. Ibaraki, H. Imai, and T. Nishizeki, editors, *Algorithms, International Symposium SIGAL '90, Tokyo, Japan, August 16-18, 1990, Proceedings*, volume 450 of *Lecture Notes in Computer Science*, pages 338–347. Springer, 1990.

[172] J. R. Gilbert, J. P. Hutchinson, and R. E. Tarjan. A separator theorem for graphs of bounded genus. *J. Algorithms*, 5(3):391–407, 1984.

[173] K.-T. Goh, J. Cutter, B.-H. Heng, S. Ma, B. K. Koh, C. Kwok, C.-M. Toh, and S.-K. Chew. Epidemiology and control of SARS in Singapore. *Annals of the Academy of Medicine, Singapore*, 35(5):301, 2006.

[174] M. T. Goodrich. Planar separators and parallel polygon triangulation. *Journal of Computer and System Sciences*, 51(3):374–389, 1995.

[175] M. T. Goodrich. The mastermind attack on genomic data. In *30th IEEE Symposium on Security and Privacy (S&P 2009), 17-20 May 2009, Oakland, California, USA*, pages 204–218. IEEE Computer Society, 2009.

[176] M. T. Goodrich. On the algorithmic complexity of the mastermind game with black-peg results. *Inf. Process. Lett.*, 109(13):675–678, 2009.

[177] M. T. Goodrich, S. Gupta, H. Khodabandeh, and P. Matias. How to catch marathon cheaters: New approximation algorithms for tracking paths. *CoRR*, abs/2104.12337, 2021.

[178] V. Grebinski. On the power of additive combinatorial search model. In W. Hsu and M. Kao, editors, *Computing and Combinatorics, 4th Annual International Conference, COCOON '98, Taipei, Taiwan, R.o.C., August 12-14, 1998, Proceedings*, volume 1449 of *Lecture Notes in Computer Science*, pages 194–203. Springer, 1998.

[179] V. Grebinski and G. Kucherov. Optimal query bounds for reconstructing a hamiltonian cycle in complete graphs. In *Fifth Israel Symposium on Theory of Computing and Systems, ISTCS 1997, Ramat-Gan, Israel, June 17-19, 1997, Proceedings*, pages 166–173. IEEE Computer Society, 1997.

[180] V. Grebinski and G. Kucherov. Reconstructing a hamiltonian cycle by querying the graph: Application to DNA physical mapping. *Discret. Appl. Math.*, 88(1-3):147–165, 1998.

[181] V. Grebinski and G. Kucherov. Optimal reconstruction of graphs under the additive model. *Algorithmica*, 28(1):104–124, 2000.

[182] R. Gupta and S. R. Das. Tracking moving targets in a smart sensor network. In *2003 IEEE 58th Vehicular Technology Conference. VTC 2003-Fall*, volume 5, pages 3035–3039. IEEE, 2003.

[183] R. Halin. S-functions for graphs. *Journal of Geometry*, 8(1-2):171–186, 1976.

[184] D. Haussler and E. Welzl. epsilon-nets and simplex range queries. *Discret. Comput. Geom.*, 2:127–151, 1987.

[185] J. J. Hein. An optimal algorithm to reconstruct trees from additive distance data. *Bulletin of mathematical biology*, 51(5):597–603, 1989.

[186] M. Hermo and A. Ozaki. Exact learning: On the boundary between horn and CNF. *ACM Trans. Comput. Theory*, 12(1):4:1–4:25, 2020.

[187] D. S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM J. Comput.*, 11(3):555–556, 1982.

[188] T. Holenstein, M. Mitzenmacher, R. Panigrahy, and U. Wieder. Trace reconstruction with constant deletion probability and related results. In S. Teng, editor, *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 389–398. SIAM, 2008.

[189] J. E. Hopcroft and R. E. Tarjan. Efficient algorithms for graph manipulation [H] (algorithm 447). *Commun. ACM*, 16(6):372–378, 1973.

[190] F. K.-m. Hwang and D.-z. Du. *Pooling designs and nonadaptive group testing: important tools for DNA sequencing*, volume 18. World Scientific, 2006.

[191] K. Iwama, J. Teruyama, and S. Tsuyama. Reconstructing strings from substrings: Optimal randomized and average-case algorithms, 2018.

[192] M. V. Janardhanan. Graph verification with a betweenness oracle. In S. Hanneke and L. Reyzin, editors, *International Conference on Algorithmic Learning Theory, ALT 2017, 15-17 October 2017, Kyoto University, Kyoto, Japan*, volume 76 of *Proceedings of Machine Learning Research*, pages 238–249. PMLR, 2017.

[193] K. Jeong, N. Bandeira, S. Kim, and P. A. Pevzner. Gapped spectral dictionaries and their applications for database searches of tandem mass spectra. *Mol Cell Proteomics*, 2011. M110.002220.

[194] M. Jerrum. The complexity of finding minimum-length generator sequences. *Theor. Comput. Sci.*, 36:265–289, 1985.

[195] D. S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. Syst. Sci.*, 9(3):256–278, 1974.

[196] L. Kalashnik. The reconstruction of a word from fragments. *Numerical mathematics and computer technology*, pages 56–57, 1973.

[197] S. Kannan. On the query complexity of learning. In L. Pitt, editor, *Proceedings of the Sixth Annual ACM Conference on Computational Learning Theory, COLT 1993, Santa Cruz, CA, USA, July 26-28, 1993*, pages 58–66. ACM, 1993.

[198] S. Kannan, C. Mathieu, and H. Zhou. Near-linear query complexity for graph inference. In M. M. Halldórsson, K. Iwama, N. Kobayashi, and B. Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, volume 9134 of *Lecture Notes in Computer Science*, pages 773–784. Springer, 2015.

[199] S. Kannan, C. Mathieu, and H. Zhou. Graph reconstruction and verification. *ACM Trans. Algorithms*, 14(4):40:1–40:30, 2018.

[200] S. Kannan and A. McGregor. More on reconstructing strings from random traces: insertions and deletions. In *Proceedings of the 2005 IEEE International Symposium on Information Theory, ISIT 2005, Adelaide, South Australia, Australia, 4-9 September 2005*, pages 297–301. IEEE, 2005.

[201] G. O. Katona. Combinatorial search problems. In *A survey of combinatorial theory*, pages 285–308. Elsevier, 1973.

[202] K. Kawarabayashi and B. A. Reed. A separator theorem in minor-closed classes. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 153–162. IEEE Computer Society, 2010.

[203] M. J. Kearns and U. V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, 1994.

[204] G. Kellaris, G. Kollios, K. Nissim, and A. O'Neill. Generic attacks on secure outsourced databases. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1329–1340, 2016.

[205] J. A. Kelner. Spectral partitioning, eigenvalue bounds, and circle packings for graphs of bounded genus. *SIAM J. Comput.*, 35(4):882–902, 2006.

[206] H. M. Kiah, G. J. Puleo, and O. Milenkovic. Codes for DNA sequence profiles. *IEEE Trans. Inf. Theory*, 62(6):3125–3146, 2016.

[207] J. Kim, A. Amir, J. C. Na, K. Park, and J. S. Sim. On representations of ternary order relations in numeric strings. In *Proc. 2nd International Conference on Algorithms for Big Data (ICABD)*, volume 1146 of *CEUR Workshop Proceedings*, pages 46–52, 2014.

[208] S. Kim, N. Bandeira, and P. A. Pevzner. Spectral profiles: A novel representation of tandem mass spectra and its applications for de novo peptide sequencing and identification. *Mol Cell Proteomics*, 8:1391–1400, 2009.

[209] S. Kim, N. Gupta, N. Bandeira, and P. A. Pevzner. Spectral dictionaries: Integrating de novo peptide sequencing with database search of tandem mass spectra. *Mol Cell Proteomics*, 8(1):53–69, 2009.

[210] V. King, L. Zhang, and Y. Zhou. On the complexity of distance-based evolutionary tree reconstruction. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA*, pages 444–453. ACM/SIAM, 2003.

[211] P. Klein and S. Mozes. Optimization algorithms for planar graphs. http://planarity.org/, 2020. Accessed 13-Dec-2020.

[212] E. Knill. Lower bounds for identifying subset members with subset queries. In K. L. Clarkson, editor, *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, 22-24 January 1995. San Francisco, California, USA*, pages 369–377. ACM/SIAM, 1995.

[213] D. E. Knuth. The computer as master mind. *Journal of Recreational Mathematics*, 9(1):1–6, 1976.

[214] D. E. Knuth and A. Raghunathan. The problem of compatible representatives. *SIAM Journal on Discrete Mathematics*, 5(3):422–427, 1992.

[215] T. Kociumaka, J. Radoszewski, and W. Rytter. Efficient indexes for jumbled pattern matching with constant-sized alphabet. In H. L. Bodlaender and G. F. Italiano, editors, *European Symp. on Algorithms (ESA)*, pages 625–636, Berlin, Heidelberg, 2013. Springer.

[216] T. Kociumaka, J. Radoszewski, W. Rytter, J. Straszyński, T. Waleń, and W. Zuba. Faster recovery of approximate periods over edit distance. In *Proc. 25th International Symposium on String Processing and Information Retrieval (SPIRE)*, LNCS, pages 233–240. Springer, 2018.

[217] R. Kolpakov and G. Kucherov. mreps: efficient and flexible detection of tandem repeats in DNA. *Nucleic Acids Res.*, 31:3672–3678, 2003. http://www.loria.fr/mreps/.

[218] E. M. Kornaropoulos, C. Papamanthou, and R. Tamassia. Data recovery on encrypted databases with k-nearest neighbor query leakage. In *Proc. IEEE Symposium on Security and Privacy, SP*, pages 245–262, 2019.

[219] A. V. Kostochka. The minimum hadwiger number for graphs with a given mean degree of vertices. *Metody Diskret. Analiz.*, (38):37–58, 1982.

[220] A. V. Kostochka. Lower bound of the hadwiger number of graphs by their average degree. *Comb.*, 4(4):307–316, 1984.

[221] I. Krasikov and Y. Roditty. On a reconstruction problem for sequences,. *J. Comb. Theory, Ser. A*, 77(2):344–348, 1997.

[222] E. Kushilevitz and Y. Mansour. Learning decision trees using the fourier spectrum. *SIAM J. Comput.*, 22(6):1331–1348, 1993.

[223] M. Lacharité, B. Minaud, and K. G. Paterson. Improved reconstruction attacks on encrypted data using range query leakage. In *Proc. IEEE Symposium on Security and Privacy, SP*, pages 297–314, 2018.

[224] V. I. Levenshtein. Binary codes capable of correcting, deletions, insertions and reversals. *Soviet Phys. Dokl.*, 10:707–710, 1966.

[225] V. I. Levenshtein. Efficient reconstruction of sequences. *IEEE Trans. Inf. Theory*, 47(1):2–22, 2001.

[226] D. Lichtenstein. Planar formulae and their uses. *SIAM Journal on Computing*, 11(2):329–343, 1982.

[227] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

[228] R. J. Lipton and R. E. Tarjan. Applications of a planar separator theorem. *SIAM J. Comput.*, 9(3):615–627, 1980.

[229] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Mach. Learn.*, 2(4):285–318, 1987.

[230] D. Lokshtanov. Kernelization, bidimensionality and kernels. In *Encyclopedia of Algorithms*, pages 1006–1011. 2016.

[231] L. Lovász. On the ratio of optimal integral and fractional covers. *Discret. Math.*, 13(4):383–390, 1975.

[232] R. Lowrance and R. A. Wagner. An extension of the string-to-string correction problem. *J. ACM*, 22(2):177–183, 1975.

[233] W. Maass and G. Turán. On the complexity of learning from counterexamples (extended abstract). In *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989*, pages 262–267. IEEE Computer Society, 1989.

[234] W. Maass and G. Turán. Lower bound methods and separation results for on-line learning models. *Mach. Learn.*, 9:107–145, 1992.

[235] W. Maass and G. Turán. Algorithms and lower bounds for on-line learning of geometrical concepts. *Mach. Learn.*, 14(1):251–269, 1994.

[236] W. Maass and M. K. Warmuth. Efficient learning with virtual threshold gates. *Inf. Comput.*, 141(1):66–83, 1998.

[237] A. J. Macula and L. J. Popyack. A group testing method for finding patterns in data. *Discret. Appl. Math.*, 144(1-2):149–157, 2004.

[238] A. J. Macula, V. V. Rykov, and S. Yekhanin. Trivial two-stage group testing for complexes using almost disjunct matrices. *Discret. Appl. Math.*, 137(1):97–107, 2004.

[239] W. Mader. Homomorphiesätze für graphen. *Mathematische Annalen*, 178(2):154–168, 1968.

[240] B. Manvel, A. Meyerowitz, A. J. Schwenk, K. Smith, and P. K. Stockmeyer. Reconstruction of sequences. *Discret. Math.*, 94(3):209–219, 1991.

[241] S. Marcovich and E. Yaakobi. Reconstruction of strings from their substrings spectrum. *CoRR*, abs/1912.11108, 2019.

[242] D. Margaritis and S. S. Skiena. Reconstructing strings from substrings in rounds. In *IEEE 36th Symp. on Foundations of Computer Science (FOCS)*, pages 613–620, Oct 1995.

[243] C. Mathieu and H. Zhou. Graph reconstruction via distance oracles. In F. V. Fomin, R. Freivalds, M. Z. Kwiatkowska, and D. Peleg, editors, *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, volume 7965 of *Lecture Notes in Computer Science*, pages 733–744. Springer, 2013.

[244] G. L. Miller. Finding small simple cycle separators for 2-connected planar graphs. *J. Comput. Syst. Sci.*, 32(3):265–279, 1986.

[245] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press, 2 edition, 2017.

[246] C. Moore and M. E. Newman. Epidemics and percolation in small-world networks. *Physical Review E*, 61(5):5678, 2000.

[247] T. M. Moosa and M. S. Rahman. Indexing permutations for binary strings. *Information Processing Letters*, 110(18):795–798, 2010.

[248] A. S. Motahari, G. Bresler, and D. N. C. Tse. Information theory of DNA shotgun sequencing. *IEEE Trans. Inf. Theory*, 59(10):6273–6289, 2013.

[249] A. S. Motahari, K. Ramchandran, D. Tse, and N. Ma. Optimal DNA shotgun sequencing: Noisy reads are as good as noiseless reads. In *Proceedings of the 2013 IEEE International Symposium on Information Theory, Istanbul, Turkey, July 7-12, 2013*, pages 1640–1644. IEEE, 2013.

[250] M. Naveed, S. Kamara, and C. V. Wright. Inference attacks on property-preserving encrypted databases. In *In Proc. of the 22nd ACM SIGSAC Conference on Computer and Communications Security,CCS*, pages 644–655, 2015.

[251] M. E. Newman. Spread of epidemic disease on networks. *Physical Review E*, 66(1):016128, 2002.

[252] H. Q. Ngo and D. Du. A survey on combinatorial group testing algorithms with applications to DNA library screening. In D. Du, P. M. Pardalos, and J. Wang, editors, *Discrete Mathematical Problems with Medical Applications, Proceedings of a DIMACS Workshop, December 8-10, 1999*, volume 55 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 171–182. DIMACS/AMS, 1999.

[253] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford University Press, 2006.

[254] S.-I. Oum. Approximating rank-width and clique-width quickly. *ACM Transactions on Algorithms*, 5(1):10, 2008.

[255] V. Parisi, V. D. Fonzo, and F. Aluffi-Pentini. STRING: finding tandem repeats in DNA sequences. *Bioinformatics*, 19(14):1733–1738, 2003.

[256] J. Pearl. Fusion, propagation, and structuring in belief networks. *Artif. Intell.*, 29(3):241–288, 1986.

[257] J. Pearl. *Probabilistic reasoning in intelligent systems - networks of plausible inference.* Morgan Kaufmann series in representation and reasoning. Morgan Kaufmann, 1989.

[258] M. Pellegrini, M. E. Renda, and A. Vecchio. Trstalker: an efficient heuristic for finding fuzzy tandem repeats. *Bioinformatics [ISMB]*, 26(12):358–366, 2010.

[259] T. Peng, C. Leckie, and K. Ramamohanarao. Survey of network-based defense mechanisms countering the DoS and DDoS problems. *ACM Computing Surveys*, 39(1):3, 2007.

[260] S. A. Plotkin, S. Rao, and W. D. Smith. Shallow excluded minors and improved graph decompositions. In D. D. Sleator, editor, *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms. 23-25 January 1994, Arlington, Virginia, USA*, pages 462–470. ACM/SIAM, 1994.

[261] B. A. Reed and D. R. Wood. A linear-time algorithm to find a separator in a graph excluding a minor. *ACM Trans. Algorithms*, 5(4):39:1–39:16, 2009.

[262] L. Reyzin and N. Srivastava. Learning and verifying graphs using queries with a focus on edge counting. In M. Hutter, R. A. Servedio, and E. Takimoto, editors, *Algorithmic Learning Theory, 18th International Conference, ALT 2007, Sendai, Japan, October 1-4, 2007, Proceedings*, volume 4754 of *Lecture Notes in Computer Science*, pages 285–297. Springer, 2007.

[263] L. Reyzin and N. Srivastava. On the longest path algorithm for reconstructing trees from distance matrices. *Inf. Process. Lett.*, 101(3):98–100, 2007.

[264] N. Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986.

[265] N. Robertson and P. D. Seymour. Graph minors. XVI. excluding a non-planar graph. *J. Comb. Theory, Ser. B*, 89(1):43–76, 2003.

[266] G. Rong, W. Li, Y. Yang, and J. Wang. Reconstruction and verification of chordal graphs with a distance oracle. *Theor. Comput. Sci.*, 859:48–56, 2021.

[267] R. M. Roth and G. M. Benedek. Interpolation and approximation of sparse multivariate polynomials over GF(2). *SIAM J. Comput.*, 20(2):291–314, 1991.

[268] F. Sala, R. Gabrys, C. Schoeny, K. Mazooji, and L. Dolecek. Exact sequence reconstruction for insertion-correcting codes. In *IEEE International Symposium on Information Theory, ISIT 2016, Barcelona, Spain, July 10-15, 2016*, pages 615–619. IEEE, 2016.

[269] R. E. Schapire and L. Sellie. Learning sparse multivariate polynomials over a field with queries and counterexamples. *J. Comput. Syst. Sci.*, 52(2):201–213, 1996.

[270] A. D. Scott. Reconstructing sequences. *Discret. Math.*, 175(1-3):231–238, 1997.

[271] S. Sen and V. N. Muralidhara. The covert set-cover problem with application to network discovery. In M. S. Rahman and S. Fujita, editors, *WALCOM: Algorithms and Computation, 4th International Workshop, WALCOM 2010, Dhaka, Bangladesh, February 10-12, 2010. Proceedings*, volume 5942 of *Lecture Notes in Computer Science*, pages 228–239. Springer, 2010.

[272] G. Seroussi and N. H. Bshouty. Vector sets for exhaustive testing of logic circuits. *IEEE Trans. Inf. Theory*, 34(3):513–522, 1988.

[273] D. Shah and T. Zaman. Rumors in a network: Who's the culprit? *IEEE Transactions on Information Theory*, 57(8):5163–5181, 2011.

[274] I. Shomorony, T. A. Courtade, and D. N. C. Tse. Do read errors matter for genome assembly? In *IEEE International Symposium on Information Theory, ISIT 2015, Hong Kong, China, June 14-19, 2015*, pages 919–923. IEEE, 2015.

[275] I. Shomorony, G. M. Kamath, F. Xia, T. A. Courtade, and D. N. C. Tse. Partial DNA assembly: A rate-distortion perspective. In *IEEE International Symposium on Information Theory, ISIT 2016, Barcelona, Spain, July 10-15, 2016*, pages 1799–1803. IEEE, 2016.

[276] I. Simon. Piecewise testable events. In H. Barkhage, editor, *Automata Theory and Formal Languages, 2nd GI Conference, Kaiserslautern, May 20-23, 1975*, volume 33 of *Lecture Notes in Computer Science*, pages 214–222. Springer, 1975.

[277] S. Skiena, W. D. Smith, and P. Lemke. Reconstructing sets from interpoint distances (extended abstract). In R. Seidel, editor, *Proceedings of the Sixth Annual Symposium on Computational Geometry, Berkeley, CA, USA, June 6-8, 1990*, pages 332–339. ACM, 1990.

[278] S. Skiena and G. Sundaram. Reconstructing strings from substrings. *Journal of Computational Biology*, 2(2):333–353, 1995.

[279] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, S. T. Kent, and W. T. Strayer. Hash-based IP traceback. *ACM SIGCOMM Computer Communication Review*, 31(4):3–14, 2001.

[280] D. Sokol. Tredd - a database for tandem repeats over the edit distance. *Database: The Journal of Biological Databases and Curation*, 2010(baq003), 2010.

[281] D. A. Spielman and S. Teng. Disk packings and planar separators. In S. Whitesides, editor, *Proceedings of the Twelfth Annual Symposium on Computational Geometry, Philadelphia, PA, USA, May 24-26, 1996*, pages 349–358. ACM, 1996.

[282] E. Stefanov, C. Papamanthou, and E. Shi. Practical dynamic searchable encryption with small leakage. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*, 2014.

[283] H. J. Suermondt and G. F. Cooper. Probabilistic inference in multiply connected belief networks using loop cutsets. *Int. J. Approx. Reason.*, 4(4):283–306, 1990.

[284] A. Ta-Shma. Classical versus quantum communication complexity. *SIGACT News*, 30(3):25–34, 1999.

[285] K. Tan, B. C. Ooi, and C. Y. Yee. An evaluation of color-spatial retrieval techniques for large image databases. *Multim. Tools Appl.*, 14(1):55–78, 2001.

[286] G. Tardos. Query complexity, or why is it difficult to separate $NP^A \cap coNP^A$ from P^A by random oracles A? *Combinatorica*, 9(4):385–392, Dec 1989.

[287] A. Thomason. An extremal function for contractions of graphs. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 95, pages 261–265. Cambridge University Press, 1984.

[288] A. Thomason. The extremal function for complete minors. *J. Comb. Theory, Ser. B*, 81(2):318–338, 2001.

[289] D. C. Torney. Sets pooling designs. *Annals of Combinatorics*, 3(1):95–101, 1999.

[290] D. Tsur. Tight bounds for string reconstruction using substring queries. In C. Chekuri, K. Jansen, J. D. P. Rolim, and L. Trevisan, editors, *Algorithms and Techniques for Approximation, Randomization and Combinatorial Optimization*, pages 448–459. Springer, 2005.

[291] R. Uehara, K. Tsuchida, and I. Wegener. Optimal attribute-efficient learning of disjunction, parity and threshold functions. In S. Ben-David, editor, *Computational Learning Theory, Third European Conference, EuroCOLT '97, Jerusalem, Israel, March 17-19, 1997, Proceedings*, volume 1208 of *Lecture Notes in Computer Science*, pages 171–184. Springer, 1997.

[292] E. Ukkonen. Approximate string matching with q-grams and maximal matches. *Theor. Comput. Sci.*, 92(1):191–211, 1992.

[293] P. Ungar. A theorem on planar graphs. *Journal of the London Mathematical Society*, 1(4):256–262, 1951.

[294] L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984.

[295] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of complexity*, pages 11–30. Springer, 2015.

[296] V. V. Vazirani. *Approximation algorithms*. Springer, 2001.

[297] G. Viglietta. Hardness of mastermind. In E. Kranakis, D. Krizanc, and F. L. Luccio, editors, *Fun with Algorithms - 6th International Conference, FUN 2012, Venice, Italy, June 4-6, 2012. Proceedings*, volume 7288 of *Lecture Notes in Computer Science*, pages 368–378. Springer, 2012.

[298] K. Viswanathan and R. Swaminathan. Improved string reconstruction over insertion-deletion channels. In S. Teng, editor, *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 399–408. SIAM, 2008.

[299] R. A. Wagner. On the complexity of the extended string-to-string correction problem. In W. C. Rounds, N. Martin, J. W. Carlyle, and M. A. Harrison, editors, *Proceedings of the 7th Annual ACM Symposium on Theory of Computing, May 5-7, 1975, Albuquerque, New Mexico, USA*, pages 218–223. ACM, 1975.

[300] J. Wang and X. Hua. Interactive image search by color map. *ACM Trans. Intell. Syst. Technol.*, 3(1):12:1–12:23, 2011.

[301] Z. Wang and J. Honorio. Reconstructing a bounded-degree directed tree using path queries. In *57th Annual Allerton Conference on Communication, Control, and Computing, Allerton 2019, Monticello, IL, USA, September 24-27, 2019*, pages 506–513. IEEE, 2019.

[302] R. Want. An introduction to RFID technology. *IEEE Pervasive Computing*, 5(1):25–33, 2006.

[303] M. S. Waterman, T. F. Smith, M. Singh, and W. A. Beyer. Additive evolutionary trees. *Journal of theoretical Biology*, 64(2):199–213, 1977.

[304] Y. Wexler, Z. Yakhini, Y. Kashi, and D. Geiger. Finding approximate tandem repeats in genomic sequences. In *RECOMB*, pages 223–232, 2004.

[305] Wikipedia contributors. Coupon collector's problem. https://en.wikipedia.org/wiki/Coupon_collector%27s_problem, 2019. accessed 30-Jan-2020.

[306] Wikipedia contributors. Marathon course-cutting, 2019. [Online; accessed 16-Feb-2021].

[307] Wikipedia contributors. Mastermind (board game) — Wikipedia, the free encyclopedia, 2021. [Online; accessed 25-February-2021].

[308] D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

[309] C. Wulff-Nilsen. Separator theorems for minor-free and shallow minor-free graphs with applications. In R. Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 37–46. IEEE Computer Society, 2011.

[310] A. C. Yao. Decision tree complexity and betti numbers. *J. Comput. Syst. Sci.*, 55(1):36–43, 1997.

[311] A. C.-C. Yao. Decision tree complexity and Betti numbers. In *Proceedings of the Twenty-sixth Annual ACM Symposium on Theory of Computing*, STOC '94, pages 615–624, New York, NY, USA, 1994. ACM.

[312] A. Zenkin and V. K. Leont'ev. On a non-classical recognition problem. *USSR Computational Mathematics and Mathematical Physics*, 24(3):189–193, 1984.

[313] Y. Zhang, J. Katz, and C. Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In *25th USENIX Security Symposium, USENIX Security 16*, pages 707–720, 2016.

[314] B. Zheng. *Approximation Schemes in Planar Graphs.* PhD thesis, Oregon State University, 2018. https://ir.library.oregonstate.edu/concern/graduate_thesis_or_dissertations/7w62ff609 (Accessed 07-Jan-2021).

[315] W. Zhou, H. Li, and Q. Tian. Recent advance in content-based image retrieval: A literature survey. *CoRR*, abs/1706.06064, 2017.

[316] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Information Theory*, 23(3):337–343, 1977.

[317] J. Ziv and A. Lempel. Compression of individual sequences via variable rate coding. *IEEE Trans. on Information Theory*, IT-24:530–536, 1978.