Algorithms for Optimization
Problems on
Networks and Graphs

Thesis submitted for the degree “Doctor of Philosophy”

by

Shai Gutner

Under the supervision of

Professor Noga Alon and Professor Yossi Azar

Submitted to the Senate of Tel Aviv University

January 2009
Acknowledgments

I would like to thank Prof. Noga Alon and Prof. Yossi Azar for being my advisors during my Ph.D. studies. Noga and Yossi naturally combine the mission of being a researcher, a teacher, and a mentor. It was in an analysis of algorithms course, given a long time ago by Noga, that I came to realize that my major research interests are algorithms and combinatorics.

This is an opportunity to express my love and appreciation to my wife Michal and my son Alon, the greatest joy of my life. I thank them for their endless support, inspiration and love. This work could not have been completed without the continuous encouragement of my wonderful loving family - my parents Rivka and Menachem, my sisters Shelly and Adi, and my brother Ronny.

I dedicate this thesis to the memory of my beloved late grandfathers - Meir Gutner and Chaim Lebenberg.

I conclude by quoting Heraclitus of Ephesus: “Much learning does not teach understanding.”

Shai
Abstract

There is substantial literature dealing with online and approximation algorithms for network problems. In this work we study various optimization problems, focusing on admission control, approximate counting of small subgraphs within large graphs, the minimum dominating set problem, and the prize collecting generalized Steiner forest problem.

We first address in Chapter 1 the admission control problem in general networks. Communication requests arrive over time, and the online algorithm accepts or rejects each request while maintaining the capacity limitations of the network. We consider preemptive online algorithms whose goal is to minimize the number of rejected requests. Each request arrives together with the path it should be routed on. We show an $O(\log^2 (mc))$-competitive randomized algorithm for the weighted case, where $m$ is the number of edges in the graph and $c$ is the maximum edge capacity. For the unweighted case, we give an $O(\log m \log c)$-competitive randomized algorithm. This settles an open question of Blum, Kalai and Kleinberg raised in [37].

The admission control problem is a generalization of the online set cover with repetitions problem, whose input is a family of $m$ subsets of a ground set of $n$ elements. Elements of the ground set are given to the online algorithm one by one, possibly requesting each element a multiple number of times. The algorithm must cover each element by different subsets, according to the number of times it has been requested.

We give an $O(\log m \log n)$-competitive randomized algorithm for the online set cover with repetitions problem. This matches an $\Omega(\log m \log n)$ lower bound of Feige and Korman [74] for the competitive ratio of any randomized polynomial time algorithm, under the $BPP \neq NP$ assumption.
In Chapter 2 we introduce a new and intriguing variant of families of perfect hash functions. A family of functions from \([n]\) to \([k]\) is an \((\varepsilon, k)\)-balanced family of hash functions, if there exists a positive \(T\) so that for every \(K \subset [n]\) of size \(|K| = k\), the number of functions in the family that are one-to-one on \(K\) is between \((1 - \varepsilon)T\) and \((1 + \varepsilon)T\). The family is perfectly \(k\)-balanced if it is \((0, k)\)-balanced.

We show that any perfectly \(k\)-balanced family is of size at least \(c(k)n^{\lfloor k/2 \rfloor}\), and that for every \(\varepsilon > \frac{1}{\text{poly}(k)}\) there are explicit constructions of \((\varepsilon, k)\)-balanced families of hash functions from \([n]\) to \([k]\) of size \(e^{(1+o(1))k \log n}\). This is tight up to the \(o(1)\)-term in the exponent, and supplies deterministic polynomial time algorithms for approximately counting the number of paths or cycles of a specified length \(k\) (or copies of any graph \(H\) with \(k\) vertices and bounded tree-width) in a given input graph of size \(n\), up to relative error \(\varepsilon\), for all \(k \leq O(\log n)\).

The domination number of a graph \(G = (V, E)\) is the minimum size of a dominating set \(U \subset V\), which satisfies that every vertex in \(V \setminus U\) is adjacent to at least one vertex in \(U\). In Chapter 3 we give a \(k^{O(dk)}n\) time algorithm for finding a dominating set of size at most \(k\) in a \(d\)-degenerated graph with \(n\) vertices. This proves that the dominating set problem is fixed-parameter tractable for degenerated graphs. For graphs that do not contain \(K_h\) as a topological minor, we give an improved algorithm for the problem with running time \((O(h))^{hk}n\). For graphs which are \(K_h\)-minor-free, the running time is further reduced to \((O(\log h))^{hk/2}n\).

The notion of a problem kernel refers to a polynomial time algorithm that achieves some provable reduction of the input size. Given a graph \(G\) whose domination number is \(k\), the objective is to design a polynomial time algorithm that produces a graph \(G'\) whose size depends only on \(k\), and also has domination number equal to \(k\).

We present the first nontrivial result for the general case of graphs with an excluded minor, as follows. For every fixed \(h\), given a graph \(G\) with \(n\) vertices that does not contain \(K_h\) as a topological minor, our \(O(n^{3.5} + k^{O(1)})\) time algorithm constructs a subgraph \(G'\) of \(G\), such that if the domination number of \(G\) is \(k\), then the domination number of \(G'\) is also \(k\) and \(G'\) has at most \(k^c\) vertices, where \(c\) is a constant that depends only on \(h\). This result is improved
for graphs that do not contain $K_{3,h}$ as a topological minor, using a simpler algorithm that constructs a subgraph with at most $ck$ vertices, where $c$ is a constant that depends only on $h$. Using the polynomial kernel construction, we give an $O(n^{3.5} + 2^{O(\sqrt{k})})$ time algorithm for finding a dominating set of size at most $k$ in an $H$-minor-free graph with $n$ vertices.

Chapter 4 deals with approximation algorithms for the prize collecting generalized Steiner forest problem, defined as follows. The input is an undirected graph $G = (V, E)$, a collection $T = \{T_1, \ldots, T_k\}$, each a subset of $V$ of size at least 2, a weight function $w : E \rightarrow \mathbb{R}^+$, and a penalty function $p : T \rightarrow \mathbb{R}^+$. The goal is to find a forest $F$ that minimizes the cost of the edges of $F$ plus the penalties paid for subsets $T_i$ whose vertices are not all connected by $F$. Our main result is a $(3 - \frac{1}{n})$-approximation for the prize collecting generalized Steiner forest problem, where $n \geq 2$ is the number of vertices in the graph. The approximation algorithm is obtained by applying the local ratio method, and is much simpler than the best known combinatorial algorithm for this problem.

Our approach gives a $(2 - \frac{1}{n-1})$-approximation for the prize collecting Steiner tree problem (all subsets $T_i$ are of size 2 and there is some root vertex $r$ that belongs to all of them). This latter algorithm is in fact the local ratio version of the primal-dual algorithm of Goemans and Williamson [56]. Another special case of our main algorithm is Bar-Yehuda’s local ratio $(2 - \frac{2}{n})$-approximation for the generalized Steiner forest problem (all the penalties are infinity) [31].
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgments</td>
<td>iii</td>
</tr>
<tr>
<td>Abstract</td>
<td>v</td>
</tr>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1 Admission Control to Minimize Rejections</td>
<td>15</td>
</tr>
<tr>
<td>1.1 Introduction</td>
<td>16</td>
</tr>
<tr>
<td>1.2 A Fractional Algorithm for the Admission Control Problem</td>
<td>20</td>
</tr>
<tr>
<td>1.3 A Randomized Algorithm for the Admission Control Problem</td>
<td>25</td>
</tr>
<tr>
<td>1.4 Reducing Online Set Cover to Admission Control</td>
<td>27</td>
</tr>
<tr>
<td>1.5 A Deterministic Bicriteria Algorithm for the Online Set Cover Problem</td>
<td>28</td>
</tr>
<tr>
<td>1.6 Concluding Remarks</td>
<td>32</td>
</tr>
<tr>
<td>2 Balanced Families of Perfect Hash Functions</td>
<td>33</td>
</tr>
<tr>
<td>2.1 Introduction</td>
<td>34</td>
</tr>
<tr>
<td>2.1.1 Motivation and Background</td>
<td>34</td>
</tr>
<tr>
<td>2.1.2 Related Work</td>
<td>35</td>
</tr>
<tr>
<td>2.1.3 The New Results</td>
<td>36</td>
</tr>
<tr>
<td>2.1.4 Methods and Organization</td>
<td>37</td>
</tr>
<tr>
<td>2.2 The Ingredients of the Construction</td>
<td>38</td>
</tr>
<tr>
<td>2.3 Perfectly Balanced Families</td>
<td>39</td>
</tr>
<tr>
<td>2.4 Probabilistic Constructions</td>
<td>43</td>
</tr>
<tr>
<td>2.5 Composition Lemmas</td>
<td>45</td>
</tr>
</tbody>
</table>
2.6 Explicit Constructions ........................................... 46
2.7 Approximate Counting of Paths and Cycles .................. 50
2.8 Expanders ................................................................ 52
2.9 Partially Independent Variables ............................... 53
2.10 Optimal Balanced Families ..................................... 57
2.11 Concluding Remarks .............................................. 59

3 The Dominating Set Problem on Graphs with an Excluded Minor 63
3.1 Introduction .......................................................... 64
3.2 Preliminaries .......................................................... 69
3.3 Algorithms for the Dominating Set Problem ................ 71
  3.3.1 Degenerated Graphs ........................................... 71
  3.3.2 Graphs with an Excluded Minor ........................... 72
  3.3.3 The Weighted Case ............................................. 77
3.4 Dominating Sets in Degenerated Graphs ...................... 77
3.5 Bounds on the Number of Central Vertices ................... 79
3.6 Polynomial Kernel for Graphs with an Excluded Minor ....... 83
3.7 Linear Kernel for Graphs with no Topological $K_{3,4}$ ....... 89
3.8 Finding Induced Cycles ............................................ 94
  3.8.1 Degenerated Graphs ........................................... 94
  3.8.2 Minor-Closed Families of Graphs ........................... 97
3.9 Concluding Remarks and Open Problems ...................... 99

4 Prize Collecting Steiner Tree Problems .......................... 101
4.1 Introduction .......................................................... 102
4.2 The Prize Collecting Generalized Steiner Forest Problem ... 104
4.3 The Prize Collecting Steiner Tree Problem ..................... 108
4.4 Concluding Remarks .............................................. 111

Bibliography .................................................................. 113
Introduction

Coping with NP-hardness is the main motivation for the algorithmic results presented in this thesis. NP-hard problems cannot be solved by a polynomial time algorithm, unless $P = NP$, which naturally leads to the exploration of efficient algorithms that search for approximate solutions. The goal of both online and approximation algorithms is to obtain solutions which are as close as possible to the optimal solution. The main difficulty of an online algorithm is that it must make decisions without knowing the entire input, whereas the main limitation of an approximation algorithm is that it must run in polynomial time.

Another interesting approach to cope with NP-hardness is given by the new field of parameterized complexity. The main research goal of this emerging field is in designing exact algorithms, whose running time is polynomial in the size of the input, but possibly exponential in some parameter of the problem. This means that the parameter, which is usually the size of the optimal solution, is the major factor affecting the complexity of the problem.

A different way of analyzing the parameterized complexity of a problem is through the construction of kernels. The notion of a problem kernel refers to a polynomial time algorithm that achieves some provable reduction of the input size. Problem kernels can be used to obtain efficient approximation and exact algorithms, and are also useful in practical settings.

In this thesis we design and analyze online and approximation algorithms for various network optimization problems.

The thesis consists of four chapters. The first chapter addresses the admission control problem in general networks, where the goal is to minimize the number of rejections performed by the online algorithm. In the second
chapter we introduce the new notion of balanced families of hash functions, and demonstrate their applications for approximately counting the number of copies of a small subgraph within a large graph. The third chapter discusses the parameterized complexity of the dominating set problem on degenerated graphs and on graphs with an excluded minor. We study fixed-parameter tractable algorithms for this problem, as well as the construction of polynomial problem kernels. The fourth and last chapter deals with combinatorial approximation algorithms for the prize collecting generalized Steiner forest problem.

The admission control to minimize rejections problem was studied by Blum, Kalai and Kleinberg in [37], where two deterministic algorithms with competitive ratios of $O(\sqrt{m})$ and $c + 1$ are given ($m$ is the number of edges in the graph and $c$ is the maximum capacity). They raised the question of whether an online algorithm with polylogarithmic competitive ratio can be obtained. We settle this question by giving an $O(\log m \log c)$-competitive randomized algorithm. This matches an $\Omega(\log m \log c)$ lower bound of Feige and Korman [74] for the competitive ratio of any randomized polynomial time algorithm for the admission control to minimize rejections problem (assuming $BPP \neq NP$).

Flum and Grohe [52] proved that the problem of counting exactly the number of paths and cycles of length $k$ in both directed and undirected graphs, considered as a problem parameterized by $k$, is $\#W[1]$-complete. This implies that it is unlikely that there is an $f(k) \cdot n^e$-time algorithm for counting the precise number of paths or cycles of length $k$ in a graph of size $n$ for any computable function $f : \mathbb{N} \rightarrow \mathbb{N}$ and constant $c$. However, the problem of approximating these numbers is more tractable. Arvind and Raman [24] obtained a randomized fixed-parameter tractable algorithm to approximately count the number of copies of $k$-paths (or any fixed subgraph with bounded tree-width) within a large graph. In this thesis we consider deterministic approximation counting algorithms for this problem. To this end, we introduce the notion of $(\varepsilon, k)$-balanced families of hash functions and use them to exhibit deterministic approximation algorithms for counting the number of simple $k$-paths in a graph $G = (V, E)$ up to a relative error of $\varepsilon = \frac{1}{\text{poly}(k)}$ in time $2^{O(k)}|E| \log |V|$. Similar results hold for counting approxi-
mately the number of copies of any graph of size $k$ with constant tree-width. Note that this is polynomial for all $k \leq O(\log n)$, and it is unlikely that one can do better, as this would imply the existence of a $2^{o(n)}$-time algorithm for the Hamilton path problem, contradicting the Exponential Time Hypothesis of [65, 66].

The dominating set problem on general graphs is known to be $W[2]$-complete [17]. This suggests the exploration of specific families of graphs for which this problem is fixed-parameter tractable. Fixed-parameter tractable algorithms for the dominating set problem were known for fixed powers of $H$-minor-free graphs and for map graphs. Linear time algorithms were established only for planar graphs. A major result of this thesis is that the dominating set problem is fixed-parameter tractable for degenerated graphs. The running time is $k^{O(dk)}n$ for finding a dominating set of size $k$ in a $d$-degenerated graph with $n$ vertices. This algorithm is linear in the number of vertices of the graph.

By introducing a novel reduction rule, we prove that the dominating set problem on graphs with an excluded minor admits a polynomial problem kernel. This gives an $O(n^{3.5} + 2^{O(\sqrt{k})})$ time algorithm for finding a dominating set of size at most $k$ in an $H$-minor-free graph with $n$ vertices. For graphs that are $K_{3,3}$ $K_{3,\infty}$-minor-free, the reduction rules of Alber, Fellows, and Niedermeier [6] are shown to give a linear problem kernel. The only previous kernel results known for the dominating set problem were the existence of a linear kernel for the planar case as well as for graphs of bounded genus.

Our algorithms for the dominating set problem are based on combinatorial results concerning degenerated graphs and graphs with an excluded minor. One of the consequences of these results is a constant factor approximation algorithm, which together with other techniques gives a faster exact algorithm for the dominating set problem on $H$-minor-free graphs.

The input to the prize collecting generalized Steiner forest problem is an undirected graph $G = (V, E)$, a collection $T = \{T_1, \ldots, T_k\}$, each a subset of $V$ of size at least 2, a weight function $w : E \to \mathbb{R}^+$, and a penalty function $p : T \to \mathbb{R}^+$. The goal is to find a forest $F$ that minimizes the cost of the edges of $F$ plus the penalties paid for subsets $T_i$ whose vertices are not all connected by $F$. Our main result is a $(3 - \frac{4}{n})$-approximation for
the prize collecting generalized Steiner forest problem, where \( n \geq 2 \) is the number of vertices in the graph. The approximation algorithm is obtained by applying the local ratio method, and is much simpler than the best known combinatorial algorithm for this problem.

Converting randomized algorithms to deterministic ones turned out to be a complicated task in some cases. The derandomization process involved several techniques, including the method of conditional probabilities, almost \( k \)-wise independence, error correcting codes, and expanders.

In what follows we describe the contents of each of these chapters.

**Chapter I: Admission Control to Minimize Rejections**

We study the admission control problem in general graphs with edge capacities. An online algorithm can receive a sequence of communications requests on a virtual path, that may be accepted or rejected, while staying within the capacity limitations.

We consider the goal of minimizing the number of rejected requests, which was first studied in [37]. This approach is suitable for applications in which rejections are intended to be rare events. A situation in which a significant fraction of the requests is rejected even by the optimal solution means that the network needs to be upgraded.

In this chapter, we consider preemptive online algorithms for the admission control problem. Each request arrives together with the path it should be routed on. The admission control algorithm decides whether to accept or reject it.

We now formally define the admission control problem. The input consist of the following:

- A directed graph \( G = (V, E) \), where \( |E| = m \). Each edge \( e \) has an integer capacity \( c_e > 0 \). We denote \( c = \max_{e \in E} c_e \).

- A sequence of requests \( r_1, r_2, \ldots \), each of which is a simple path in the graph. Every request \( r_i \) has a cost \( p_i > 0 \) associated with it.
A feasible solution for the problem must assure that for every edge $e$, the number of accepted requests whose paths contain $e$ is at most its capacity $c_e$. The goal is to find a feasible solution of minimum cost of the rejected requests. The online algorithm is given requests one at a time, and must decide whether to accept or reject each request. It is also allowed to preempt a request, i.e. to reject it after already accepting it, but it cannot accept a request after rejecting it.

Let $OPT$ be a feasible solution having minimum cost $C_{OPT}$. An algorithm is $\beta$-competitive if the total cost of the requests rejected by this algorithm is at most $\beta C_{OPT}$.

The admission control to minimize rejections problem was studied by Blum, Kalai and Kleinberg in [37], where two deterministic algorithms with competitive ratios of $O(\sqrt{m})$ and $c + 1$ are given ($m$ is the number of edges in the graph and $c$ is the maximum capacity). They raised the question of whether an online algorithm with polylogarithmic competitive ratio can be obtained.

In this chapter we show that the admission control to minimize rejections problem is a generalization of the online set cover with repetitions problem, defined as follows: Let $X$ be a ground set of $n$ elements, and let $S$ be a family of subsets of $X$, $|S| = m$. Each $S \in S$ has a non-negative cost associated with it. An adversary gives elements to the algorithm from $X$ one by one. Each element of $X$ can be given an arbitrary number of times, not necessarily consecutively. An element should be covered by a number of sets which is equal to the number of times it appeared in the arrival sequence (only one copy of each set is available). We assume that the elements of $X$ and the members of $S$ are known in advance to the algorithm, however, the elements given by the adversary are not known in advance. The objective is to minimize the cost of the sets chosen by the algorithm.

The main result we present in this chapter is an $O(\log^2 (mc))$-competitive randomized algorithm for the admission control to minimize rejections problem. This settles the open question raised by Blum et al. [37]. For the unweighted case, where all costs are equal to 1, we slightly improve this bound and give an $O(\log m \log c)$-competitive randomized algorithm.

We present a simple reduction between online set cover with repetitions
and the admission control to minimize rejections problem. This implies an $O(\log^2(mn))$-competitive randomized algorithm for the online set cover with repetitions problem. For the unweighted case (all costs are equal to 1), we get an $O(\log m \log n)$-competitive randomized algorithm. This matches the lower bound of $\Omega(\log m \log n)$ given by Feige and Korman [74]. Their results also imply a lower bound of $\Omega(\log m \log c)$ for the competitive ratio of any randomized polynomial time algorithm for the admission control to minimize rejections problem (assuming $\text{BPP} \neq \text{NP}$).

References: The results of this chapter appear in:


Chapter 2: Balanced Families of Perfect Hash Functions

Color Coding is an algorithmic technique for deciding efficiently if a given input graph contains a path or a cycle of a given length, or any other prescribed subgraph of bounded tree-width. Focusing, for simplicity, on paths, the method supplies a deterministic algorithm for deciding, in time $2^{O(k)}|E|\log|V|$, whether or not a given input (directed or undirected) graph $G = (V, E)$ contains a (simple) path on $k$ vertices. The basic approach, introduced in [22], is very simple. One first gives a randomized algorithm, and then converts it into a deterministic one. The randomized algorithm works by first coloring the vertices of $G$ randomly by $k$ colors. Call a path on $k$ vertices (a $k$-path, for short) colorful if its vertices get all the distinct $k$ colors. It is not difficult to check in time $O(k2^k|E|)$, using dynamic programming, if there is a colorful path. As the probability of a $k$-path to become colorful in a random coloring is $k!/k^k > e^{-k}$, repeating the above procedure some $Ce^k$ times provides a randomized algorithm in which the probability not to find
a path in case one exists is smaller than $e^{-C}$. The crucial point in the derandomization of this algorithm is the observation that known constructions of families of hash functions given by [79] following [55], supply an explicit family of $2^{O(k) \log |V|}$ colorings of the vertices of $G$ by $k$ colors, so that the members of every set of $k$ vertices get distinct colors in at least one of the colorings. Thus one can simply run the dynamic programming algorithm for each of these colorings, getting a deterministic algorithm for the problem.

The above technique has found several recent applications in computational biology (see [80], [81], [83], [64]). These applications suggest the problem of counting, or approximating the number of $k$-paths (or other graphs of bounded tree-width) in a given graph. As using dynamic programming it is easy to count precisely the number of colorful $k$-paths in a given graph with colored vertices, the existence of efficient randomized approximation algorithms for counting follows quite easily by following the same approach; this is done in [13].

In order to derandomize the randomized counting (or approximate counting) procedures, one needs a strengthening of the usual notion of hash functions. This is given in the following definition.

A family of functions from $[n]$ to $[\ell]$ is an $(\epsilon, k)$-balanced family of hash functions, if for every $S \subset [n], |S| = k$, the number of functions that are one-to-one on $S$ is between $(1 - \epsilon)T$ and $(1 + \epsilon)T$ for some constant $T > 0$. The family is perfectly $k$-balanced if it is $(0, k)$-balanced, that is, it is $(\epsilon, k)$-balanced for $\epsilon = 0$.

Flum and Grohe [52] proved that the problem of counting exactly the number of paths and cycles of length $k$ in both directed and undirected graphs, considered as a problem parameterized by $k$, is $\#W[1]$-complete. This implies that it is unlikely that there is an $f(k) \cdot n^c$-time algorithm for counting the precise number of paths or cycles of length $k$ in a graph of size $n$ for any computable function $f : \mathbb{N} \to \mathbb{N}$ and constant $c$.

However, the problem of approximating these numbers is more tractable. Arvind and Raman [24] obtained a randomized fixed-parameter tractable algorithm to approximately count the number of copies of $k$-paths (or any fixed subgraph with bounded tree-width) within a large graph. A similar approximation appears in [13].
The results of Flum and Grohe mentioned above suggest that there is no perfectly $k$-balanced family of hash functions from $[n]$ to $[k]$ of size $f(k)n^{O(1)}$. We prove a stronger result, showing that every perfectly $k$-balanced family of hash functions from $[n]$ to $[\ell]$ is of size at least $c(k, \ell)n^{\lfloor k/2 \rfloor}$, where $c(k, \ell)$ is a positive constant depending only on $k$ and $\ell$. We also observe that this is not far from being tight, as for every $n > k$ there is a perfectly $k$-balanced family of functions from $[n]$ to $[k]$ of size $(n^k - 1)$. This shows that the Color Coding approach cannot supply an algorithm for counting $k$-paths in an $n$ vertex graph in time $o(n^{k/2})$.

Our main positive result is an explicit construction, for every $\frac{1}{\text{poly}(k)} < \varepsilon \leq 1$, of an $(\varepsilon, k)$-balanced family of hash functions from $[n]$ to $[k]$ of size $e^{k+O(\log^3 k)}\log n$. The running time of the procedure that provides the construction is $e^{k+O(\log^3 k)}n\log n$. Note that the size of the family is optimal up to the error term $O(\log^3 k)$ in the exponent, as there is a known lower bound of $\Omega(e^k \log n/\sqrt{k})$ for the size of any family of hash functions from $[n]$ to $[k]$ (even if it is not balanced and the only requirement is that every set of size $k$ is mapped in a one-to-one fashion at least once).

This supplies deterministic approximation algorithms for counting the number of simple $k$-paths in a graph $G = (V, E)$ up to a relative error of $\varepsilon = \frac{1}{\text{poly}(k)}$ in time $2^{O(k)}|E|\log |V|$. Similar results hold for counting approximately the number of copies of any graph of size $k$ with constant tree-width. Note that this is polynomial for all $k \leq O(\log n)$, and it is unlikely that one can do better, as this would imply the existence of a $2^{o(n)}$-time algorithm for the Hamilton path problem, contradicting the Exponential Time Hypothesis of [65, 66].

Our lower bound for the size of perfectly balanced families are proved by Linear Algebra tools, combining the basic approach of [10] in the proof of the lower bound for the size of sample spaces supporting $k$-wise independent random variables with two additional ideas.

The construction of $(\varepsilon, k)$-balanced families combines several ingredients. Two of them are rather standard and are based on nearly pairwise independent random variables and on the method of conditional expectations. The third one is more challenging, and combines the approach of [77] with an iterative construction based on properties of expanders. It is convenient to
apply here (some version of) the expanders of [19], though other expanders could have been used as well.

References: The results of this chapter appear in:


Chapter 3: The Dominating Set Problem on Graphs with an Excluded Minor

The input to a parameterized problem is a pair \((x, k)\), where \(x\) is the problem instance, \(k\) is the parameter, and \(n := |(x, k)|\) denotes the input size. A parameterized problem is fixed-parameter tractable if it can be solved in time \(f(k) \cdot n^c\), for a computable function \(f : \mathbb{N} \rightarrow \mathbb{N}\) and a constant \(c\).

A kernelization is a polynomial time computable function that given input \((x, k)\) constructs an equivalent input \((x', k')\), such that \(k' \leq k\) and \(|x'| \leq g(k)\) for a computable function \(g : \mathbb{N} \rightarrow \mathbb{N}\). The image \(x'\) is called the problem kernel of \(x\). In this chapter, the notion of a kernel for the dominating set problem refers to a polynomial time algorithm that given a graph \(G\) whose domination number is \(k\), constructs a graph \(G'\) whose size depends only on \(k\), and also has domination number equal to \(k\).

It is easy and known that a parameterized problem is kernelizable if and only if it is fixed-parameter tractable. Thus, a fixed-parameter algorithm for the dominating set problem gives a trivial kernel whose size is some function of \(k\), not necessarily a polynomial. Problem kernels can be used to obtain efficient approximation and exact algorithms for the domination number, and are also useful in practical settings.
The dominating set problem on general graphs is known to be $W[2]$-complete \cite{[17]}. This means that most likely there is no $f(k) \cdot n^c$-algorithm for finding a dominating set of size at most $k$ in a graph of size $n$ for any computable function $f : \mathbb{N} \rightarrow \mathbb{N}$ and constant $c$. This suggests the exploration of specific families of graphs for which this problem is fixed-parameter tractable.

The main result of the chapter is that the dominating set problem is fixed-parameter tractable for degenerated graphs. The running time is $k^{O(dk)} n$ for finding a dominating set of size $k$ in a $d$-degenerated graph with $n$ vertices. The algorithm is linear in the number of vertices of the graph, and we further improve the dependence on $k$ for the following specific families of degenerated graphs. For graphs that do not contain $K_h$ as a topological minor, an improved algorithm for the problem with running time $(O(h))^h k n$ is established. For graphs which are $K_h$-minor-free, the running time obtained is $(O(\log h))^{hk/2} n$. We show that all the algorithms can be generalized to the weighted case in the following sense. A dominating set of size at most $k$ having minimum weight can be found within the same time bounds.

The reduction rules introduced by Alber, Fellows, and Niedermeier were the first to establish a linear problem kernel for planar graphs \cite{[47]}. The kernel obtained was of size $335k$, where $k$ is the domination number of the graph. Fomin and Thilikos proved that the same rules of Alber et al. provide a linear kernel of size $O(k + g)$ for graphs of genus $g$ \cite{[54]}. Chen et al. improved the upper bound for the planar case to $67k$ \cite{[42]}. They also gave the first lower bound, by proving that for any $\varepsilon > 0$, there is no $(2 - \varepsilon)k$ kernel for the planar dominating set problem, unless $P = NP$.

By introducing a novel reduction rule, we prove that the dominating set problem on graphs with an excluded minor admits a polynomial problem kernel. This gives an $O(n^{3.5} + 2^{O(\sqrt{k})})$ time algorithm for finding a dominating set of size at most $k$ in an $H$-minor-free graph with $n$ vertices. For graphs that are $K_{3,k}$-minor-free, the reduction rules of Alber, Fellows, and Niedermeier \cite{[4]} are shown to give a linear problem kernel. All the reduction rules described in this chapter have the property that the only modifications made to an input graph are the removal of vertices and edges. This implies that the graph obtained, as a result of applying the rules, is a subgraph of the input
graph. The advantages of this approach are its simplicity and the fact that it preserves monotone properties, like planarity, being $H$-minor-free, and degeneracy. We show that the rules of Alber et al. can also be described in such a way.

We address two open questions raised by Cai, Chan and Chan in [41] concerning linear time algorithms for finding an induced cycle in degenerated graphs. An $O(n)$ expected time algorithm for finding an induced $k$-cycle in graphs with an excluded minor is presented. The derandomization performed in [41] is improved and we get a deterministic $O(n \log n)$ time algorithm for the problem. As for finding induced cycles in degenerated graphs, we show a deterministic $O(n)$ time algorithm for finding cycles of size at most 5, and also explain why this is unlikely to be possible to achieve for longer cycles.

References: The results of this chapter appear in:


Chapter 4: Prize Collecting Steiner Tree Problems

There is substantial literature dealing with approximation algorithms for prize collecting Steiner tree problems. The purpose of this chapter is to present elegant combinatorial algorithms for these problems. The local ratio technique [30, 31, 32] that we employ enable us to present simple algorithms together with a straightforward analysis.

The main focus of the chapter is on the prize collecting generalized Steiner forest (PCGSF) problem, defined as follows. The input is an undirected
graph $G = (V, E)$, a collection $T = \{T_1, \ldots, T_k\}$, each a subset of $V$ of size at least 2, a weight function $w : E \rightarrow \mathbb{R}^+$, and a penalty function $p : T \rightarrow \mathbb{R}^+$, where $\mathbb{R}^+$ denotes the set of positive real numbers. The objective is to compute a forest $F$ that minimizes the cost of the edges of $F$ and the sum of the penalties of the subsets $T_i$ whose vertices are not all connected by $F$. Thus, all the vertices of a subset $T_i$ must be in the same connected component of $F$ in order to avoid the penalty.

The special case of the PCGSF problem called the prize collecting Steiner forest problem (all subsets $T_i$ are of size 2) has received considerable attention lately. A modification of the LP rounding algorithm in [35] implies a 3-approximation for this problem. This was improved in [60] to give an LP based 2.54-approximation for the problem as well as a primal-dual combinatorial $(3 - \frac{2}{n})$-approximation using Farkas’ Lemma.

A generalized framework of the prize collecting problems with an arbitrary $0 - 1$ connectivity requirement function and a submodular penalty function is studied in [82]. Their model captures both the PCGSF problem defined in this chapter as well as the problems of [62, 60]. The authors give a complicated primal-dual 3-approximation algorithm together with an LP rounding algorithm with a performance ratio of 2.54.

Our main result is a local ratio $(3 - \frac{4}{n})$-approximation for the prize collecting generalized Steiner forest problem, where $n \geq 2$ is the number of vertices in the graph. This obviously implies the same approximation for the special case of the prize collecting Steiner forest problem, which was studied in [60, 55] and [53]. We also present a $(2 - \frac{1}{n-1})$-approximation for the prize collecting Steiner tree problem (all subsets $T_i$ are of size 2 and there is some root vertex $r$ that belongs to all of them). This latter algorithm is in fact the local ratio version of the primal-dual algorithm of Goemans and Williamson [56]. Another special case of our main algorithm is Bar-Yehuda’s local ratio $(2 - \frac{2}{n})$-approximation for the generalized Steiner forest problem (all the penalties are infinity) [31]. Thus, an important contribution of this chapter is in providing a natural generalization of the framework presented by Goemans and Williamson, and later by Bar-Yehuda.

References: The results of this chapter appear in:
Introduction

Chapter 1

Admission Control to Minimize Rejections

*The results of this chapter appear in [9]*

We study the admission control problem in general networks. Communication requests arrive over time, and the online algorithm accepts or rejects each request while maintaining the capacity limitations of the network. The admission control problem has been usually analyzed as a benefit problem, where the goal is to devise an online algorithm that accepts the maximum number of requests possible. The problem with this objective function is that even algorithms with optimal competitive ratios may reject almost all of the requests, when it would have been possible to reject only a few. This could be inappropriate for settings in which rejections are intended to be rare events.

In this chapter, we consider preemptive online algorithms whose goal is to minimize the number of rejected requests. Each request arrives together with the path it should be routed on. We show an $O(\log^2(mc))$-competitive randomized algorithm for the weighted case, where $m$ is the number of edges in the graph and $c$ is the maximum edge capacity. For the unweighted case, we give an $O(\log m \log c)$-competitive randomized algorithm. This settles an open question of Blum, Kalai and Kleinberg raised in [37]. We note that allowing preemption and handling requests with given paths are essential for avoiding trivial lower bounds.
The admission control problem is a generalization of the online set cover with repetitions problem, whose input is a family of \( m \) subsets of a ground set of \( n \) elements. Elements of the ground set are given to the online algorithm one by one, possibly requesting each element a multiple number of times. (If each element arrives at most once, this corresponds to the online set cover problem.) The algorithm must cover each element by different subsets, according to the number of times it has been requested.

We give an \( O(\log m \log n) \)-competitive randomized algorithm for the online set cover with repetitions problem. This matches a recent lower bound of \( \Omega(\log m \log n) \) given by Feige and Korman [74] for the competitive ratio of any randomized polynomial time algorithm, under the \( BPP \neq NP \) assumption. Given any constant \( \varepsilon > 0 \), an \( O(\log m \log n) \)-competitive deterministic bicriteria algorithm is shown that covers each element by at least \( (1 - \varepsilon)k \) sets, where \( k \) is the number of times the element is covered by the optimal solution.

1.1 Introduction

We study the admission control problem in general graphs with edge capacities. An online algorithm can receive a sequence of communications requests on a virtual path, that may be accepted or rejected, while staying within the capacity limitations.

This problem has typically been studied as a benefit problem. This means that the online algorithm has to be competitive with respect to the number of accepted requests. A problem with this objective function is that in some cases an online algorithm with a good competitive ratio may reject the vast majority of the requests, whereas the optimal solution rejects only a small fraction of them.

In this chapter we consider the goal of minimizing the number of rejected requests, which was first studied in [37]. This approach is suitable for applications in which rejections are intended to be rare events. A situation in which a significant fraction of the requests is rejected even by the optimal solution means that the network needs to be upgraded.

We consider preemptive online algorithms for the admission control prob-
1.1 Introduction

Problem. Allowing preemption is necessary for achieving reasonable bounds for the competitive ratio. Each request arrives together with the path it should be routed on. The admission control algorithm decides whether to accept or reject it. An online algorithm for both admission control and routing easily admits a trivial lower bound \[37\].

The admission control to minimize rejections problem. We now formally define the admission control problem. The input consist of the following:

- A directed graph \(G = (V, E)\), where \(|E| = m\). Each edge \(e\) has an integer capacity \(c_e > 0\). We denote \(c = \max_{e \in E} c_e\).
- A sequence of requests \(r_1, r_2, \ldots\), each of which is a simple path in the graph. Every request \(r_i\) has a cost \(p_i > 0\) associated with it.

A feasible solution for the problem must assure that for every edge \(e\), the number of accepted requests whose paths contain \(e\) is at most its capacity \(c_e\). The goal is to find a feasible solution of minimum cost of the rejected requests. The online algorithm is given requests one at a time, and must decide whether to accept or reject each request. It is also allowed to preempt a request, i.e. to reject it after already accepting it, but it cannot accept a request after rejecting it.

Let \(OPT\) be a feasible solution having minimum cost \(C_{OPT}\). An algorithm is \(\beta\)-competitive if the total cost of the requests rejected by this algorithm is at most \(\beta C_{OPT}\).

\(^1\) Consider a path of \(m\) edges with capacity \(c\) on each edge. The first \(c\) requests are for the entire path of \(m\) edges. The online algorithms must accept all the requests, since during this sequence of \(c\) requests, \(OPT\) is equal to zero. These requests will fill all edges exactly to capacity. In the second phase, for each edge, we are given \(c\) requests that consist of this single edge. A non-preemptive online algorithm cannot accept additional requests, so the online algorithm rejects \(mc\) requests, while \(OPT\) rejects only \(c\) requests.

\(^2\) Consider a 4-cycle \(ABCD\) with capacity \(c\) on each edge. We are given \(c\) requests connecting the diagonally opposite nodes \(A\) and \(C\), and then we are given either \(c\) requests connecting \(A\) and \(B\), or else \(c\) requests connecting \(A\) and \(D\), with equal probability. Every online algorithm rejects \(c/2\) requests in expectation, while it was possible to reject none offline. Similarly, with \(n\) separate 4-cycles, the online algorithm rejects \(nc/2\) requests in expectation, while \(OPT\) rejects none.
Previous results for admission control. Tight bounds were achieved for the admission control problem, where the goal is to maximize the number of accepted requests. Awerbuch, Azar and Plotkin [26] provide an $O(\log n)$-competitive algorithm for general graphs. For the admission control problem on a tree, $O(\log d)$-competitive randomized algorithms appear in [27], [28], where $d$ is the diameter of the tree. Adler and Azar presented a constant-competitive preemptive algorithm for admission control on the line [1].

The admission control to minimize rejections problem was studied by Blum, Kalai and Kleinberg in [37], where two deterministic algorithms with competitive ratios of $O(\sqrt{m})$ and $c + 1$ are given ($m$ is the number of edges in the graph and $c$ is the maximum capacity). They raised the question of whether an online algorithm with polylogarithmic competitive ratio can be obtained.

We note that one can combine an algorithm for maximizing throughput of accepted requests and an algorithm for minimizing rejections and get one algorithm which achieves both simultaneously with slightly degrading the competitive ratio [29].

In this chapter we show that the admission control to minimize rejections problem is a generalization of the online set cover with repetitions problem described below:

The online set cover with repetitions problem. The online set cover problem is defined as follows: Let $X$ be a ground set of $n$ elements, and let $\mathcal{S}$ be a family of subsets of $X$, $|\mathcal{S}| = m$. Each $S \in \mathcal{S}$ has a non-negative cost associated with it. An adversary gives elements to the algorithm from $X$ one by one. Each element of $X$ can be given an arbitrary number of times, not necessarily consecutively. An element should be covered by a number of sets which is equal to the number of times it appeared in the arrival sequence (only one copy of each set is available). We assume that the elements of $X$ and the members of $\mathcal{S}$ are known in advance to the algorithm, however, the elements given by the adversary are not known in advance. The objective is to minimize the cost of the sets chosen by the algorithm.

Previous results for online set cover. The offline version of the set cover problem is a classic NP-hard problem that was studied extensively, and the best approximation factor achievable for it in polynomial time (assuming
$P \neq NP$ is $\Theta(n)$ \cite{43,49}. Previous results for online set cover did not consider repetitions. The basic online set cover problem, where repetitions are not allowed, was studied in \cite{7,74}. A different variant of the problem, dealing with maximum benefit, is presented in \cite{25}. An $O(m \log n)$-competitive deterministic algorithm for the online set cover problem was given by \cite{7} where $n$ is the number of elements and $m$ is the number of sets. 

A lower bound of $\Omega(m \log n)$ was also shown for any deterministic online algorithm. A recent result of Feige and Korman \cite{74} establishes a lower bound of $\Omega(m \log n)$ for the competitive ratio of any randomized polynomial time algorithm for the online set cover problem, under the $\text{BPP} \neq \text{NP}$ assumption. They also prove the same lower bound for any deterministic polynomial time algorithm, under the $P \neq NP$ assumption.

Concurrent to the preliminary version of our paper, Berman and Das-Gupta \cite{34} considered the weighted online set $k$-multicover problem. The input to this problem consists of the input to the online set cover problem together with a parameter $k$ called the coverage factor. There are no repetitions, but when an element is presented, it should be covered by at least $k$ sets. Their paper contains some upper and lower bounds for this problem, including an $O(m' \log d)$-competitive randomized algorithm, where $m'$ is the maximum frequency of elements in the ground set and $d$ is the maximum set size.

Buchbinder and Naor defined an online framework together with implications concerning competitive bounds for the basic online set cover problem, when repetitions are not allowed \cite{40}. There are no results in their paper concerning the problems considered in this chapter and although their framework may be adapted to our case it seems it will provide a much worse competitive ratio.

**Our results.** The main result we give in this chapter is an $O(mc)$-competitive randomized algorithm for the admission control to minimize rejections problem. This settles the open question raised by Blum et al. \cite{37}. For the unweighted case, where all costs are equal to 1, we slightly improve this bound and give an $O(m \log c)$-competitive randomized algorithm.

We present a simple reduction between online set cover with repetitions and the admission control to minimize rejections problem. This implies an
$O(\log^2 (mn))$-competitive randomized algorithm for the online set cover with repetitions problem. For the unweighted case (all costs are equal to 1), we get an $O(\log m \log n)$-competitive randomized algorithm. This matches the lower bound of $\Omega(\log m \log n)$ given by Feige and Korman [74]. Their results also imply a lower bound of $\Omega(\log m \log c)$ for the competitive ratio of any randomized polynomial time algorithm for the admission control to minimize rejections problem (assuming $\text{BPP} \neq \text{NP}$).

The derandomization techniques used in [7] for the online set cover problem do not seem to apply here. This is why we also consider the bicriteria version of the online set cover with repetition problem. For a given constant $\varepsilon > 0$, the online algorithm is required to cover each element by a fraction of $1 - \varepsilon$ times the number of its appearances. Specifically, at any point of time, if an element has been requested $k$ times so far, then the optimal solution covers it by $k$ different sets, whereas the online algorithm covers it by $(1 - \varepsilon)k$ different sets. We give an $O(\log m \log n)$-competitive deterministic bicriteria algorithm for this problem.

**Techniques.** The techniques we use follow those of [7,8] together with some new ideas. We start with an online fractional solution which is monotone increasing during the algorithm. Then, the fractional solution is converted into a randomized algorithm. Interestingly, to get a deterministic bicriteria algorithm we use a different fractional algorithm than the one used for the randomized algorithm.

### 1.2 A Fractional Algorithm for the Admission Control Problem

In this section we describe a fractional algorithm for the problem. A fractional algorithm is allowed to reject a fraction of a request $r_i$. We use a weight $f_i$ for this fraction. Specifically, if $0 \leq f_i < 1$, we reject with percentage of precisely $f_i$. If $f_i \geq 1$, then the request is completely rejected. At any stage of the fractional algorithm we will use the following notation:

- $REQ_e$ will denote the set of requests that arrived so far whose paths contain the edge $e$. 
1.2 A Fractional Algorithm for the Admission Control Problem

- \( \text{REQ} \) will denote \( \bigcup_{e \in E} \text{REQ}_e \).
- \( \text{ALIVE}_e \) will denote the requests from \( \text{REQ}_e \) that have not been fully rejected (requests \( r_i \) for which \( f_i < 1 \)).
- \( n_e \) will denote the excess of edge \( e \) caused by the requests in \( \text{ALIVE}_e \).

\[
n_e = |\text{ALIVE}_e| - c_e
\]

The requirement from a fractional algorithm is that for every edge \( e \),

\[
\sum_{i \in \text{ALIVE}_e} f_i \geq n_e
\]

The cost associated with a fractional solution to the problem is defined to be \( \sum_{i \in \text{REQ}} \min\{f_i, 1\} p_i \).

We will now describe an \( O(\log(mc)) \)-competitive algorithm for the problem, even versus a fractional optimum. The cost of the optimal fractional solution, \( C_{OPT} \) is denoted by \( \alpha \).

We may assume, by doubling, that the value of \( \alpha \) is known up to a factor of 2. To determine the initial value of \( \alpha \) we look for the first time in which we must reject a request from an edge \( e \). We can start guessing \( \alpha = \min_{i \in \text{REQ}} p_i \), and then run the algorithm with this bound on the optimal solution. If it turns out that the value of the optimal solution is larger than our current guess for it, (that is, the cost exceeds \( \Theta(\alpha \log(mc)) \)), then we ”forget” about all the request fractions rejected so far, update the value of \( \alpha \) by doubling it, and continue. This means that we initialize all the variables and start the algorithms again from the beginning of the arrival sequence. Specifically, all the request fractions \( f_i \) are set to 0. We note that the cost of the request fractions that we have ”forgotten” about can increase the cost of our solution by at most a factor of 2, since the value of \( \alpha \) was doubled in each step.

We thus assume that \( \alpha \) is known. Denote by \( R_{\text{big}} \) the requests with cost exceeding \( 2\alpha \). The optimal fractional solution can reject a total fraction of at most 1/2 out of the requests of \( R_{\text{big}} \). Hence, when an edge is requested more than its capacity, the fractional optimum must reject a total fraction of at least 1/2 out of the requests not in \( R_{\text{big}} \) whose paths contain the edge. By doubling the fraction of rejection for all the requests not in \( R_{\text{big}} \) (keeping
fractions to be at most 1) and completely accepting all the requests in $R_{\text{big}}$, we get a feasible fractional solution whose cost is at most twice the optimum. Hence, the online algorithm can always completely accept requests of cost exceeding $2\alpha$ (and adjust the edge capacities $c_e$ by subtracting the number of requests of cost exceeding $2\alpha$ whose paths contain the edge $e$).

Denote by $R_{\text{small}}$ the requests with cost at most $\alpha/(mc)$. We claim that we can completely reject all the requests from $R_{\text{small}}$. For each edge $e$, the optimal solution can accept a total fraction of at most $c_e$ out of the requests whose paths contain the edge $e$, and therefore it can accept a total fraction of at most $mc$ requests. The fractions of requests accepted out of $R_{\text{small}}$ have total cost at most $mc \cdot \alpha/(mc) = \alpha$. It follows that the optimal solution pays at least $\text{cost}(R_{\text{small}}) - \alpha$ for the fractions of requests out of $R_{\text{small}}$ that it rejected. Therefore, the online algorithm can reject all the requests in $R_{\text{small}}$ and pay $\text{cost}(R_{\text{small}})$. If $\text{cost}(R_{\text{small}}) < 2\alpha$, then this adds only $O(\alpha)$ to the cost of the online algorithm. If $\text{cost}(R_{\text{small}}) \geq 2\alpha$, then $\text{cost}(R_{\text{small}}) \leq 2(\text{cost}(R_{\text{small}}) - \alpha)$, so the online algorithm is 2-competitive with respect to the requests in $R_{\text{small}}$.

By the above arguments, all the requests of cost smaller than $\alpha/(mc)$ or greater than $2\alpha$ are rejected immediately or accepted permanently (edge capacities are decreased in this case), respectively. An algorithm needs to handle only requests of cost between $\alpha/(mc)$ and $2\alpha$. We normalize the costs so that the minimum cost is 1 and the maximum cost is $g \leq 2mc$, and fix $\alpha$ appropriately.

The algorithm maintains a weight $f_i$ for each request $r_i$. The weights can only increase during the run of the algorithm. Initially $f_i = 0$ for all the requests. Assume now that the algorithm receives a request $r_i$ for a path of cost $p_i$. For each edge $e$, we update $\text{REQ}_e$, $\text{ALIVE}_e$ and $n_e$ according to the definitions given above. The following is performed for all the edges $e$ of the path of $r_i$, in an arbitrary order.

1. If $\sum_{i \in \text{ALIVE}_e} f_i \geq n_e$, then do nothing.

2. Else, while $\sum_{i \in \text{ALIVE}_e} f_i < n_e$, perform a weight augmentation:

   (a) For each $i \in \text{ALIVE}_e$, if $f_i = 0$, then set $f_i = 1/(gc)$.
1.2 A Fractional Algorithm for the Admission Control Problem

(b) For each $i \in ALIVE_e$, $f_i \leftarrow f_i(1 + \frac{1}{n_e p_i})$.

(c) Update $ALIVE_e$ and $n_e$.

Note that the fractional algorithm starts with all weights equal to zero. This is necessary, since the online algorithm must reject 0 requests in case the optimal solution rejects 0 requests. Hence, the algorithm is competitive for $\alpha = 0$, and from now on we assume without loss of generality that $\alpha > 0$. In the following we analyze the performance of the algorithm.

**Lemma 1.2.1.** *The total number of weight augmentations performed during the algorithm is at most $O(\alpha \log(gc))$.*

*Proof.* Consider the following potential function:

$$
\Phi = \prod_{i \in REQ} \max\{f_i, 1/(gc)\}^{f_i^{*} p_i}
$$

where $f_i^{*}$ is the weight of the request $r_i$ in the optimal fractional solution. We now show three properties of $\Phi$:

- The initial value of the potential function is: $(gc)^{-\alpha}$.
- The potential function never exceeds $2^\alpha$.
- In each weight augmentation step, the potential function is multiplied by at least 2.

The first two properties follow directly from the initial value and from the fact that no request gets a weight of more than $1 + 1/p_i \leq 2$. Consider an iteration in which the adversary gives a request $r_i$ with cost $p_i$. Now suppose that a weight augmentation is performed for an edge $e$. We must have $\sum_{i \in ALIVE_e} f_i^{*} \geq n_e$ since the optimal solution must satisfy the capacity constraint. Thus, the potential function is multiplied by at least:

$$
\prod_{i \in ALIVE_e} \left(1 + \frac{1}{n_e p_i}\right)^{f_i^{*} p_i} \geq \prod_{i \in ALIVE_e} \left(1 + \frac{1}{n_e}\right)^{f_i^{*}} \geq 2
$$
The first inequality follows since for all $x \geq 1$ and $z \geq 0$, $(1 + z/x)^x \geq 1 + z$ and the last inequality follows since $\sum_{i \in ALIVE_{e}} f_i^x \geq n_e$. It follows that the total number of weight augmentation steps is at most:

$$\log_2(2gc)^x = O(\alpha \log gc)$$

\[\square\]

**Theorem 1.2.2.** For the weighted case, the fractional algorithm for admission control is $O(\log(mc))$-competitive. In case all costs are equal to 1, the algorithm is $O(\log c)$-competitive.

*Proof.* The cost associated with the online algorithm is $\sum_{i \in REQ} \min\{f_i, 1\}p_i$, which we will denote by $C_{ON}$. Consider a weight augmentation step performed for an edge $e$. In step 2a of the algorithm, the weights of at most $c + 1$ requests change from 0 to $1/(gc)$. This is because before the current request arrived, there could have been at most $c$ requests containing the edge $e$ and having $f_i = 0$ (the maximum capacity is $c$). Since the maximum cost is $g$, the total increase of $C_{ON}$ in step 2a of the algorithm is at most $(c+1) \frac{1}{gc}g = 1 + 1/c$. If follows that in step 2a the quantity $\sum_{i \in ALIVE_{e}} f_i$ can increase by at most $1 + 1/c$. A weight augmentation is performed as long as $\sum_{i \in ALIVE_{e}} f_i < n_e$. Before step 2b we have that $\sum_{i \in ALIVE_{e}} f_i < n_e + 1 + 1/c$. Thus, the total increase of $C_{ON}$ in step 2b of the algorithm does not exceed

$$\sum_{i \in ALIVE_{e}} f_i p_i \frac{1}{n_e p_i} = \sum_{i \in ALIVE_{e}} \frac{f_i}{n_e} < 2 + 1/c$$

It follows that the total increase of $C_{ON}$ in a weight augmentation step is at most $3 + 2/c$. Using lemma 1.2.1 which bounds the number of augmentation steps, we conclude that the algorithm is $O(\log(gc))$-competitive.

For the weighted case, we saw that the input can be transformed so that $g \leq 2mc$, which implies that the algorithm is $O(\log(mc))$-competitive. In case all the costs are equal to 1, $g$ is also equal to 1 and the algorithm is $O(\log c)$-competitive.

\[\square\]
1.3 A Randomized Algorithm for the Admission Control Problem

We describe in this section an $O(\log^2(mc))$-competitive randomized algorithm for the weighted case and a slightly better $O(\log m \log c)$-competitive randomized algorithm for the unweighted case.

The algorithm maintains a weight $f_i$ for each request $r_i$, exactly like the fractional algorithm. Assume now that the algorithm receives a request $r_i$ with cost $p_i$. The following is performed in this case.

1. Perform all the weight augmentations according to the fractional algorithm.
2. Reject all requests whose weight is at least $\frac{1}{12 \log(mc)}$.
3. For every request $r$, if its weight $f$ increased by $\delta$, then reject the request $r$ with probability $12\delta \log(mc)$.
4. If the current request $r_i$ cannot be accepted (some edge would be over capacity), then reject the request. Else, accept the request $r_i$.

We can assume that $|REQ_e|$, the total number of requests whose paths contain a specific edge $e$, is less than $4mc^2$. To see this, note that the fractional algorithm normalizes the costs so that the minimum cost is 1 and the maximum cost is at most $2mc$. If $|REQ_e| \geq 4mc^2$, then since the optimal solution can accept at most $c$ requests from $REQ_e$, it must pay a cost of at least $t - 2mc^2$ for requests rejected out of $REQ_e$, where $t$ is the total cost of these requests. The online algorithm can reject all the requests in $REQ_e$, pay $t$ and it will still be 2-competitive with respect to the requests in $REQ_e$, since $t \geq 4mc^2$.

**Theorem 1.3.1.** For the weighted case, the randomized algorithm for admission control is $O(\log^2(mc))$-competitive.

**Proof.** Denote by $C_{frac}$ the cost of the fractional algorithm. The expected cost of requests rejected in step 3 of the algorithm is at most $12C_{frac} \log(mc)$. The cost of requests rejected in step 2 has the same upper bound.
We now calculate the probability for a request $r$ to be rejected in step 4. This can happen only if the path of request $r$ contains an edge $e$ for which $\sum_{i \in ALIVE_e} f_i \geq n_e$ but the randomized algorithm rejected less than $n_e$ requests whose paths contain the edge $e$. All the requests with weight at least $\frac{1}{12 \log(mc)}$ are rejected for sure, so we can assume that $f_i < \frac{1}{12 \log(mc)}$ for all $i \in ALIVE_e$.

Suppose that $i \in ALIVE_e$ and that during all runs of step 3 of the algorithm the request $r_i$ has been rejected with probabilities $q_1, \ldots, q_n$, where $\sum_{k=1}^{n} q_k = 12f_i\log(mc)$. The probability that $r_i$ will be rejected is at least

$$1 - \prod_{k=1}^{n} (1 - q_k) \geq 1 - e^{-\sum_{k=1}^{n} q_k} = 1 - e^{-12f_i\log mc} \geq 6f_i\log mc$$

The last inequality follows since for all $0 \leq x \leq 1$, $1 - e^{-x} \geq x/2$.

The number of requests in $ALIVE_e$ which were rejected by the algorithm is a random variable whose value is the sum of mutually independent $\{0, 1\}$-valued random variables and its expectation is at least $\mu = 6n_e\log mc$. By Chernoff bound (c.f., e.g., [21]), the probability for this random variable to get a value less than $(1 - a)\mu$ is at most $e^{-a^2\mu/2}$ for every $a > 0$. Therefore, the probability to be less than $n_e$ is at most

$$e^{-\left(1 - \frac{1}{6 \log mc}\right)^2(6n_e\log mc)/2} \leq \frac{3}{m^2c^3}$$

The request costs were normalized, so that the maximum cost is at most $2mc$. Each edge is contained in the paths of at most $4mc^2$ requests. Therefore, the expected cost of requests which are rejected in step 4 because of this edge is at most $(4mc^2)(2mc)\sqrt{3/(m^2c^3)} = 24/m$. Thus, the total expected cost of requests rejected in step 4 is 24. The result now follows from Theorem 1.2.2.

For the unweighted case we slightly change the algorithm as follows. In step 3 of the algorithm we reject a request with probability $4\delta \log m$, and in step 2 we reject all the requests whose weight is at least $1/(4 \log m)$.

**Theorem 1.3.2.** For the unweighted case (all costs are equal to 1), the randomized algorithm is $O(\log m \log c)$-competitive.
Proof. Following the proof of Theorem 1.3.1, we get that the probability for an edge to cause a specific request to be rejected in step 4 of the randomized algorithm is at most
\[ e^{-\left(1-\frac{1}{2\log m}\right)^2\left(2n_e \log m\right)/2} \leq \frac{3}{m} \]

Denote by \( Q \) the quantity \( \max_{e \in E}(|REQ_e| - c_e) \). Hence, \( Q \) is the maximum excess capacity in the network. The total expected cost of requests rejected in step 4 is at most \( Q(3/m)m = 3Q \). It is obvious that the optimal solution must reject at least \( Q \) requests. The result now follows from Theorem 1.2.2.

1.4 Reducing Online Set Cover to Admission Control

We now describe the reduction between online set cover and admission control. Suppose we are given the following input to the online set cover with repetitions problem: \( X \) is a ground set of \( n \) elements and \( S \) is a family of \( m \) subsets of \( X \), with a positive cost \( c_S \) associated with each \( S \in S \). The instance of the admission control to minimize rejections problem is constructed as follows: The graph \( G = (V, E) \) has an edge \( e_j \) for each element \( j \in X \). The capacity of the edge \( e_j \) is defined to be the number of sets that contain the element \( j \). The maximum capacity is therefore at most \( m \).

The requests are given to the admission control algorithm in two phases. In the first phase, before any element is given to the online set cover algorithm, we generate \( m \) requests to the admission control online algorithm. For every \( S \in S \), the request consists of all the edges \( e_j \) such that \( j \in S \). The online algorithm can accept all the requests and this will cause the edges to reach their full capacity.

In the second phase, each time the adversary gives an element \( j \) to the online set cover algorithm, we generate a request which consists of the one edge \( e_j \) and give it to the admission control algorithm. In case the request
caused the edge $e_j$ to be over capacity, the algorithm will have to reject one request in order to keep the capacity constraint.

In case there is a feasible cover for the input given to the online set cover problem, there is no reason for the admission control algorithm to reject requests that were given in the second phase. This is because requests in the second phase consist of only one edge. Thus, we can assume that the admission control algorithm rejects only requests given in the first phase, which correspond to subsets of $X$.

It is easy to see that the requests rejected by the admission control algorithm correspond to a legal set cover. We reduced an online set cover problem with $n$ elements and $m$ sets to an admission control problem with $n$ edges and maximum capacity at most $m$. The fact that the requests we generated are not simple paths in the graph can be easily fixed by adding extra edges.

### 1.5 A Deterministic Bicriteria Algorithm for the Online Set Cover Problem

In this section we describe, given any constant $\varepsilon > 0$, an $O(\log m \log n)$-competitive deterministic bicriteria algorithm that covers each element by at least $(1 - \varepsilon)k$ sets, where $k$ is the number of times the element has been requested, whereas the optimum covers it $k$ times. We assume for simplicity that all the sets have cost equal to 1. The result can be easily generalized for the weighted case using techniques from [7].

The algorithm maintains a weight $w_S > 0$ for each $S \in \mathcal{S}$. Initially $w_S = 1/(2m)$ for each $S \in \mathcal{S}$. The weight of each element $j \in X$ is defined as $w_j = \sum_{S \in \mathcal{S}_j} w_S$, where $\mathcal{S}_j$ denotes the collection of sets containing element $j$. Initially, the algorithm starts with the empty cover $\mathcal{C} = \emptyset$. For each $j \in X$, we define $\text{cover}_j = |\mathcal{S}_j \cap \mathcal{C}|$, which is the number of times element $j$ is covered so far. The following potential function is used throughout the algorithm:

$$\Phi = \sum_{j \in X} n^{2(w_j - \text{cover}_j)}$$

We give a high level description of a single iteration of the algorithm in
which the adversary gives an element $j$ and the algorithm chooses sets that cover it. We denote by $k$ the number of times that the element $j$ has been requested so far.

1. If $\text{cover}_j \geq (1 - \varepsilon)k$, then do nothing.

2. Else, while $\text{cover}_j < (1 - \varepsilon)k$, perform a weight augmentation:
   
   (a) For each $S \in \mathcal{S}_j - \mathcal{C}$, $w_S \leftarrow w_S(1 + \frac{1}{\varepsilon})$.
   
   (b) Add to $\mathcal{C}$ all the subsets for which $w_S \geq 1$.
   
   (c) Choose from $\mathcal{S}_j$ at most $2 \log n$ sets to $\mathcal{C}$ so that the value of the potential function $\Phi$ does not exceed its value before the weight augmentation.

In the following we analyze the performance of the algorithm and explain which sets to add to the cover $\mathcal{C}$ in step 2c of the algorithm. The cost of the optimal solution $\mathcal{C}_{OPT}$ is denoted by $\alpha$.

**Lemma 1.5.1.** The total number of weight augmentations performed during the algorithm is at most $O(\alpha \log m)$.

**Proof.** Consider the following potential function:

$$\Psi = \prod_{S \in \mathcal{C}_{OPT}} w_S$$

We now show three properties of $\Psi$:

- The initial value of the potential function is: $(2m)^{-\alpha}$.
- The potential function never exceeds $1.5^\alpha$.
- In each weight augmentation step, the potential function is multiplied by at least $2^{\varepsilon/2}$.

The first two properties follow directly from the initial value and from the fact that no request gets a weight of more than 1.5. Consider an iteration in which the adversary gives an element $j$ for the $k$th time. Now suppose
that a weight augmentation is performed for element \( j \). We must have that \( \text{cover}_j < (1-\varepsilon)k \), which means that the online algorithm has covered element \( j \) less than \((1-\varepsilon)k\) times. The optimal solution \( OPT \) covers element \( j \) at least \( k \) times, which means that there are at least \( \varepsilon k \) subsets of \( OPT \) containing \( j \) which were not chosen yet. Thus, in step 2a of the algorithm the potential function is multiplied by at least:

\[
(1 + \frac{1}{2k})^{\varepsilon k} \geq 2^{\varepsilon/2}
\]

It follows that for fixed \( \varepsilon > 0 \) the total number of weight augmentation steps is at most:

\[
\frac{\log(3m)^\alpha}{\log 2^{\varepsilon/2}} = O(\alpha \log m)
\]

**Lemma 1.5.2.** Consider an iteration in which a weight augmentation is performed. Let \( \Phi_s \) and \( \Phi_e \) be the values of the potential function \( \Phi \) before and after the iteration, respectively. Then, there exist at most \( 2 \log n \) sets that can be added to \( C \) during the iteration such that \( \Phi_e \leq \Phi_s \). Furthermore, the value of the potential function never exceeds \( n^2 \).

**Proof.** The proof is by induction on the iterations of the algorithm. Initially, the value of the potential function \( \Phi \) is less than \( n \cdot n = n^2 \). Suppose that in the iteration the adversary gives element \( j \) for the \( k \)th time. For each set \( S \in S_j \), let \( w_S \) and \( w_S + \delta_S \) denote the weight of \( S \) before and after the iteration, respectively. Define \( \delta_j = \sum_{S \in S_j} \delta_S \). By the induction hypothesis, we know that \( 2(w_j - \text{cover}_j) < 2 \), because otherwise \( \Phi_s \) would have been greater than \( n^2 \). Thus, \( w_j < \text{cover}_j + 1 \leq (1-\varepsilon)k + 1 = k \). This means that \( \delta_j \leq k \cdot 1/(2k) = 1/2 \).

We now explain which sets from \( S_j \) are added to \( C \).

Repeat \( 2 \log n \) times: choose at most one set from \( S_j \) such that each set \( S \in S_j \) is chosen with probability \( 2\delta_S \). (This can be implemented by choosing a number uniformly at random in \([0,1]\), since \( 2\delta_j \leq 1 \).)

Consider an element \( j' \in X \). Let the weight of \( j' \) before the iteration be \( w_{j'} \) and let the weight after the iteration be \( w_{j'} + \delta_{j'} \). Element \( j' \) contributes
before the iteration to the potential function the value $n^{2w_j'}$. In each random choice, the probability that we do not choose a set containing element $j'$ is $1 - 2\delta_j'$. The probability that this happens in all the $2 \log n$ random choices is therefore $(1 - 2\delta_j')^{2 \log n} \leq n^{-4\delta_j'}$.

Note that $\delta_j' \leq 1/2$. In case we choose a set containing element $j'$, then $\text{cover}_{j'}$ will increase by at least 1 and the contribution of element $j'$ to the potential function will be at most $n^{2w_j' + \delta_j' - 1} \leq n^{2w_j' - 1}$. Therefore, the expected contribution of element $j'$ to the potential function after the iteration is at most

$$n^{-4\delta_j'} n^{2(w_j' + \delta_j')} + (1 - n^{-4\delta_j'}) n^{2w_j' - 1} = n^{2w_j'} (n^{-2\delta_j'} + n^{-1} - n^{-4\delta_j' - 1}) \leq n^{2w_j'}$$

where to justify the last inequality, we prove that $f(x) = n^x + n^{-1} - n^{2x - 1} \leq 1$ for every $x \leq 0$. To show this we note that $f(0) = 1$ and $f'(x) = n^x \log n(1 - 2n^x - 1)$. This implies that $f'(x) \geq 0$ for every $x \leq 0$. We can conclude that $f(x) \leq 1$ for every $x \leq 0$, as needed.

By linearity of expectation it follows that $\mathbf{Exp}[\Phi_e] \leq \Phi_s$. Hence, there exists a choice of at most $2 \log n$ sets such that $\Phi_e \leq \Phi_s$. The choices of the various sets $S$ to be added to $C$ can be done deterministically and efficiently, by the method of conditional probabilities, c.f., e.g., [21], chapter 15. After each weight augmentation, we can greedily add sets to $C$ one by one, making sure that the potential function will decrease as much as possible after each such choice.

**Theorem 1.5.3.** The deterministic bicriteria algorithm for online set cover is $O(\log m \log n)$-competitive.

**Proof.** It follows from Lemma 1.5.1 that the number of iterations is at most $O(\alpha \log m)$. By Lemma 1.5.2 in each iteration we choose at most $2 \log n$ sets to $C$ in step [24] of the algorithm. The sets chosen is step [24] of the algorithm are those which have weight at least 1. The sum of weights of all the sets is initially 1/2 and it increases by at most 1/2 in each weight augmentation. This means that at the end of the algorithm, there can be only $O(\alpha \log m)$ sets whose weight is at least 1. Therefore, the total number of sets chosen by the algorithm is as claimed.
1.6 Concluding Remarks

- An interesting open problem is to decide if the algorithm presented here for the admission control problem can be derandomized.

- Recently, Feige and Korman established a lower bound of $\Omega(\log m \log n)$ for the competitive ratio of any randomized polynomial time algorithm for the online set cover problem, under the $BPP \neq NP$ assumption \[74]. It is interesting to decide whether this lower bound applies for superpolynomial time algorithms as well.

- The algorithms we gave for the admission control problem did not use the fact that the requests are simple paths in the graph. All the algorithms treated a request as an arbitrary subset of edges.

- In this chapter, the rejection costs are arbitrary, but when the bandwidth requirements are also arbitrary, our results do not apply. It may be possible to extend the results to this more general case, but for simplicity and clarity of exposition this is not done here.
Chapter 2

Balanced Families of Perfect Hash Functions

The results of this chapter appear in [16, 15].

Color Coding is an algorithmic technique for deciding efficiently if a given input graph contains a path of a given length (or another small subgraph of constant tree-width). Applications of the method in computational biology motivate the study of similar algorithms for counting the number of copies of a given subgraph. While it is unlikely that exact counting of this type can be performed efficiently, as the problem is \#W[1]-complete even for paths, approximate counting is possible, and leads to the investigation of an intriguing variant of families of perfect hash functions. A family of functions from \([n]\) to \([k]\) is an \((\varepsilon, k)\)-balanced family of hash functions, if there exists a positive \(T\) so that for every \(K \subseteq [n]\) of size \(|K| = k\), the number of functions in the family that are one-to-one on \(K\) is between \((1 - \varepsilon)T\) and \((1 + \varepsilon)T\). The family is perfectly \(k\)-balanced if it is \((0, k)\)-balanced.

We show that every such perfectly \(k\)-balanced family is of size at least \(c(k)n^{k/2}\), and that for every \(\varepsilon > \frac{1}{\text{poly}(k)}\) there are explicit constructions of \((\varepsilon, k)\)-balanced families of hash functions from \([n]\) to \([k]\) of size \(e^{(1+o(1))k \log n}\). This is tight up to the \(o(1)\)-term in the exponent, and supplies deterministic polynomial time algorithms for approximately counting the number of paths or cycles of a specified length \(k\) (or copies of any graph \(H\) with \(k\) vertices and
2.1 Introduction

2.1.1 Motivation and Background

Color Coding is an algorithmic technique for deciding efficiently if a given input graph contains a path or a cycle of a given length, or any other prescribed subgraph of bounded tree-width. Focusing, for simplicity, on paths, the method supplies a deterministic algorithm for deciding, in time $2^{O(k)}|E| \log |V|$, whether or not a given input (directed or undirected) graph $G = (V, E)$ contains a (simple) path on $k$ vertices. The basic approach, introduced in [22], is very simple. One first gives a randomized algorithm, and then converts it into a deterministic one. The randomized algorithm works by first coloring the vertices of $G$ randomly by $k$ colors. Call a path on $k$ vertices (a $k$-path, for short) colorful if its vertices get all the distinct $k$ colors. It is not difficult to check in time $O(k^2|E|)$, using dynamic programming, if there is a colorful path. As the probability of a $k$-path to become colorful in a random coloring is $k!/k^k > e^{-k}$, repeating the above procedure some $Ce^k$ times provides a randomized algorithm in which the probability not to find a path in case one exists is smaller than $e^{-C}$. The crucial point in the de-randomization of this algorithm is the observation that known constructions of families of hash functions given by [79] following [55], supply an explicit family of $2^{O(k)} \log |V|$ colorings of the vertices of $G$ by $k$ colors, so that the members of every set of $k$ vertices get distinct colors in at least one of the colorings. Thus one can simply run the dynamic programming algorithm for each of these colorings, getting a deterministic algorithm for the problem.

The above technique has found several recent applications in computational biology (see [80], [81], [83], [64]), where it has been applied for detecting signaling pathways in protein interaction networks. These applications suggest the problem of counting, or approximating the number of $k$-paths (or other graphs of bounded tree-width) in a given graph. As using dynamic programming it is easy to count precisely the number of colorful $k$-paths.
2.1 Introduction

in a given graph with colored vertices, the existence of efficient randomized approximation algorithms for counting follows quite easily by following the same approach; this is done in [13].

In order to derandomize the randomized counting (or approximate counting) procedures, one needs a strengthening of the usual notion of hash functions. This is given in the following definition.

A family of functions from \([n] \) to \([\ell]\) is an \((\varepsilon, k)\)-balanced family of hash functions, if for every \(S \subseteq [n], \ |S| = k\), the number of functions that are one-to-one on \(S\) is between \((1 - \varepsilon)T\) and \((1 + \varepsilon)T\) for some constant \(T > 0\). The family is perfectly \(k\)-balanced if it is \((0, k)\)-balanced, that is, it is \((\varepsilon, k)\)-balanced for \(\varepsilon = 0\).

Note that with a perfectly \(k\)-balanced family one can count precisely the number of \(k\)-paths in a graph on \(n\) vertices: we simply count, by dynamic programming, the number of colorful \(k\)-paths for each of the functions (considered as a coloring of the vertices), sum the results and divide by \(T\). Similarly, an \((\varepsilon, k)\)-balanced family will enable us to approximate the number of paths up to a relative error of \(\varepsilon\). This suggests the study of the smallest possible size of such families, and the problem of constructing explicitly such families.

2.1.2 Related Work

The problem of counting paths and cycles in graphs has been considered by various researchers. In [23] the authors describe an \(O(|V|^\omega)\) algorithm for counting the number of cycles of size at most 7, where \(\omega < 2.38\) is the exponent in fast matrix multiplication. The method of this paper does not extend to longer paths, and indeed Flum and Grohe [32] proved that the problem of counting exactly the number of paths and cycles of length \(k\) in both directed and undirected graphs, considered as a problem parameterized by \(k\), is \#W[1]-complete. This implies that it is unlikely that there is an \(f(k) \cdot n^c\)-time algorithm for counting the precise number of paths or cycles of length \(k\) in a graph of size \(n\) for any computable function \(f : \mathbb{N} \rightarrow \mathbb{N}\) and constant \(c\). The best known algorithms for computing exactly the number of \(k\)-paths in an \(n\) vertex graph run in time \(n^{k/2+O(1)}\), see [36], [86].
However, the problem of approximating these numbers is more tractable. Arvind and Raman [24] obtained a randomized fixed-parameter tractable algorithm to approximately count the number of copies of $k$-paths (or any fixed subgraph with bounded tree-width) within a large graph. A similar approximation appears in [13].

In an earlier paper [16] we consider deterministic approximation counting algorithms for this problem. To this end, we introduced the notion of $(\varepsilon, k)$-balanced families of hash functions and used them to exhibit a deterministic polynomial time algorithm for approximating the number of paths of length $k$ up to any $k \leq O(\log n \log \log n)$ in a graph with $n$ vertices. This was done by constructing explicitly $(\varepsilon, k)$-balanced families from $[n]$ to $[k]$, where the size of the family is $2^{O(k \log \log k)} \log n$ and the time for construction is $2^{O(k \log \log k)} n \log n$. The main open problem raised in [16] is to find such a construction of size $2^{O(k \log \log k)} n$ (in time $2^{O(k \log \log k)} n^{O(1)}$), which is optimal, even for standard (non-balanced) families of hash functions, and will supply polynomial time deterministic approximation algorithms for counting the number of paths of length $k$ in a given graph of size $n$, for all $k \leq O(\log n)$. This problem is settled in this chapter.

### 2.1.3 The New Results

The results of Flum and Grohe mentioned above suggest that there is no perfectly $k$-balanced family of hash functions from $[n]$ to $[k]$ of size $f(k)n^{O(1)}$. We prove a stronger result, showing that every perfectly $k$-balanced family of hash functions from $[n]$ to $[\ell]$ is of size at least $c(k, \ell)n^{k/2}$, where $c(k, \ell)$ is a positive constant depending only on $k$ and $\ell$. We also observe that this is not far from being tight, as for every $n > k$ there is a perfectly $k$-balanced family of functions from $[n]$ to $[k]$ of size $\binom{n}{k}$. This shows that the Color Coding approach cannot supply an algorithm for counting $k$-paths in an $n$ vertex graph in time $o(n^{k/2})$.

Our main positive result is an explicit construction, for every $\frac{1}{\text{poly}(k)} < \varepsilon \leq 1$, of an $(\varepsilon, k)$-balanced family of hash functions from $[n]$ to $[k]$ of size $e^{k+O(\log^3 k)} \log n$. The running time of the procedure that provides the construction is $e^{k+O(\log^3 k)} n \log n$. Note that the size of the family is optimal up
to the error term $O(\log^3 k)$ in the exponent, as there is a known lower bound of $\Omega(e^k \log n/\sqrt{k})$ for the size of any family of hash functions from $[n]$ to $[k]$, (even if it is not balanced and the only requirement is that every set of size $k$ is mapped in a one-to-one fashion at least once).

This supplies deterministic approximation algorithms for counting the number of simple $k$-paths in a graph $G = (V, E)$ up to a relative error of $\varepsilon = \frac{1}{\text{poly}(k)}$ in time $2^{O(k)}|E|\log|V|$. Similar results hold for counting approximately the number of copies of any graph of size $k$ with constant tree-width. Note that this is polynomial for all $k \leq O(\log n)$, and it is unlikely that one can do better, as this would imply the existence of a $2^{o(n)}$-time algorithm for the Hamilton path problem, contradicting the Exponential Time Hypothesis of [65, 66].

### 2.1.4 Methods and Organization

Our lower bound for the size of perfectly balanced families are proved by Linear Algebra tools, combining the basic approach of [10] in the proof of the lower bound for the size of sample spaces supporting $k$-wise independent random variables with two additional ideas.

The construction of $(\varepsilon, k)$-balanced families combines several ingredients. Two of them are rather standard and are based on nearly pairwise independent random variables and on the method of conditional expectations. The third one is more challenging, and combines the approach of [77] with an iterative construction based on properties of expanders. It is convenient to apply here (some version of) the expanders of [19], though other expanders could have been used as well.

Since our main motivation is the application for the subgraph approximate counting problem using Color Coding, there is no reason to provide explicit constructions of $(\varepsilon, k)$-balanced families of functions which are more efficient than the time of writing these functions down, as anyway our counting algorithm will have to go through these functions. We thus describe the constructions in this way, without trying to describe separately which parts of them admit more efficient descriptions. It is worth noting, however, that the part of our construction which applies the method of conditional
expectations indeed requires the time stated in its description.

The rest of this chapter is organized as follows. In section 2.2 we describe the main ingredients of the construction: balanced families of hash functions and balanced splitters, a modified version of a notion introduced in [77]. Section 2.3 contains the results concerning perfectly balanced families of hash functions. In Section 2.4 we discuss probabilistic arguments for the existence of balanced splitters. The composition lemmas presented in section 2.5 enable us to obtain an almost optimal explicit construction of balanced families of hash functions. This construction, given in section 2.6, is achieved by combining three types of balanced splitters. Section 2.7 demonstrates how the constructions, together with the color coding technique, are used for designing algorithms for approximately counting the number of copies of subgraphs of bounded tree-width in given graphs.

The explicit construction of expanders presented in section 2.8 is used in section 2.9 for constructing small sample spaces supporting a certain relaxed version of nearly $k$-wise independent random variables. This is used to obtain a construction of what we call balanced $(n, k, \ell)$-splitters, which is later applied in section 2.10 as a crucial ingredient in the construction of optimal balanced families of hash functions. We conclude with some remarks and open problems.

2.2 The Ingredients of the Construction

In this section we formally define the notions of balanced families of hash functions and balanced splitters. For a positive integer $n$, denote by $[n]$ the set $\{1, \ldots, n\}$. For any $k$, $1 \leq k \leq n$, the family of $k$-sized subsets of $[n]$ is denoted by $\binom{[n]}{k}$. As usual, $k \mod \ell$ denotes the unique integer $0 \leq r < \ell$ so that $k = q\ell + r$, for some integer $q$.

**Definition 2.2.1.** Suppose that $1 \leq k \leq \ell \leq n$ and $\varepsilon \geq 0$. A family of functions from $[n]$ to $[\ell]$ is an $(\varepsilon, k)$-balanced family of hash functions if there exists a constant $T > 0$, such that for every $S \in \binom{[n]}{k}$, the number of functions that are one-to-one on $S$ is between $(1 - \varepsilon)T$ and $(1 + \varepsilon)T$. The family is perfectly $k$-balanced if it is $(0, k)$-balanced.
2.3 Perfectly Balanced Families

The following definition is motivated by a related notion defined and used in [77].

Definition 2.2.2. Suppose that \( 1 \leq \ell < k \leq n \) and \( \varepsilon \geq 0 \), and let \( H \) be a family of functions from \([n]\) to \([\ell]\). For a set \( S \in \binom{[n]}{k} \), let \( split_H(S) \) denote the number of functions \( h \in H \) so that for every \( j, 1 \leq j \leq k \mod \ell \), \( |h^{-1}(j) \cap S| = \lfloor k/\ell \rfloor \), and for all \( k \mod \ell < j \leq \ell \), \( |h^{-1}(j) \cap S| = \lceil k/\ell \rceil \). The family \( H \) is an \( \varepsilon \)-balanced \((n,k,\ell)\)-splitter if there exists a constant \( T > 0 \), such that for every \( S \in \binom{[n]}{k} \), \((1 - \varepsilon)T \leq split_H(S) \leq (1 + \varepsilon)T\).

Note that if \( \ell \) divides \( k \), then in the above definition \( split_H(S) \) is the number of functions that split \( S \) into equal parts. The splitters of [77] differ from the ones defined here, just as usual families of hash functions differ from balanced families; in [77] it is only required that for every set \( S \) there will be some function in \( H \) splitting it evenly, while in our splitters each \( S \) should be divided evenly by roughly the same number of functions. The construction of balanced splitters is thus much harder than the one of splitters in [77], and is in fact the most challenging part in the explicit description of balanced families of hash functions.

Each function \( f \) in our explicit construction of balanced families of hash functions is the composition of members from three families. The first one is an \((\varepsilon_1, k)\)-balanced family of hash functions from \([n]\) to \([q]\), where \( q = \Theta(k^{\frac{2}{\varepsilon}}} \). The second one is an \( \varepsilon_2 \)-balanced \((q,k,\ell)\)-splitter from \([q]\) to \([\ell]\), where \( \ell = \Theta(\log q) \), and the last one is an \((\varepsilon_3, k/\ell)\)-balanced family of hash functions from \([q]\) to \([k/\ell]\) (for simplicity assume for now that \( \ell \) divides \( k \)). In order to define \( f \) we actually need \( \ell \) members of the third family, with each of them being applied to the elements mapped by the members of the second family to a single \( j \in [\ell] \).

2.3 Perfectly Balanced Families

Let \( n > \ell \geq k > 0 \) be positive integers. Recall that a family \( \mathcal{F} \) of functions from \([n]\) to \([\ell]\) is perfectly \( k \)-balanced, if there exists a number \( T > 0 \) so that for every set \( K \subset [n] \) of size \( |K| = k \), \(|\{f \in \mathcal{F} : |f(K)| = k\}| = T\).
In this section we show that the size of each such family must be at least $c(k, \ell)n^{|k/2|}$, where $c(k, \ell)$ is a positive constant depending only on $k$ and $\ell$.

**Theorem 2.3.1.** Let $\mathcal{F}$ be a perfectly $k$-balanced family of functions from $[n]$ to $[\ell]$, where $n > \ell \geq k$.

(i) If $k = 2r$ is even then

$$|\mathcal{F}| \geq \binom{n}{r} \binom{\ell}{r}.$$

(ii) If $k = 2r + 1$ is odd then

$$|\mathcal{F}| \geq \binom{n-1}{r} \binom{\ell-1}{r-1}.$$

(iii) If $\ell = k = 2$ then $|\mathcal{F}| \geq n - 1$, and equality can hold if and only if there is a Hadamard matrix of order $n$. Otherwise, the smallest possible size of $\mathcal{F}$ is precisely $n$.

**Proof.** (i) Let $\mathcal{F}$ be a perfectly $2r$-balanced family of functions from $[n]$ to $[\ell]$.

For each $R \subset [n]$ of size $|R| = r$, define two vectors $u_R$ and $w_R$, each of length $|\mathcal{F}| \binom{r}{(\ell - r)}$, whose coordinates are indexed by the set of all ordered triples $(f, S_1, S_2)$, with

$$f \in \mathcal{F}, \ S_1, S_2 \subset [\ell], \ |S_1| = |S_2| = r, \ \text{and} \ S_1 \cap S_2 = \emptyset.$$

These vectors are defined as follows:

$$u_R(f, S_1, S_2) = 1 \ \text{if} \ f(R) = S_1, \ \text{and} \ u_R(f, S_1, S_2) = 0 \ \text{otherwise}.$$

$$w_R(f, S_1, S_2) = 1 \ \text{if} \ f(R) = S_2, \ \text{and} \ w_R(f, S_1, S_2) = 0 \ \text{otherwise}.$$

Note that the inner product of two such vectors $u_{R_1}$ and $w_{R_2}$ is zero if $R_1 \cap R_2 \neq \emptyset$. Indeed, in this case $f(R_1)$ must have a nonempty intersection with $f(R_2)$ for all $f \in \mathcal{F}$, and thus there is no coordinate $(f, S_1, S_2)$ as above in which both $u_{R_1}$ and $w_{R_2}$ do not vanish. Similarly, if $R_1 \cap R_2 = \emptyset$, the inner product of $u_{R_1}$ and $w_{R_2}$ is precisely the number of functions $f \in \mathcal{F}$ which are one-to-one on $R_1 \cup R_2$. Indeed, for each such $f$ there is a unique pair of
disjoint sets $S_1, S_2$, each of size $r$, so that $f(R_1) = S_1$ and $f(R_2) = S_2$, while if $f$ maps two elements of $R_1 \cup R_2$ to the same image, there is no such pair. Since $F$ is a perfectly balanced $2r$-family, there exists a positive $T$ so that for every disjoint $R_1, R_2$ as above, the inner product of $u_{R_1}$ with $w_{R_2}$ is $T$.

Let $U$ be the \binom{\ell}{r} by $|F| \binom{\ell - r}{r}$ matrix whose rows are all vectors $u_R$ with $R \subset [n], |R| = r$, and let $W$ be the matrix whose rows are all vectors $w_R$. By the above discussion, the product $U \cdot W^t = T \cdot DIS_{n,r}$, where $DIS_{n,r}$ is the disjointness matrix whose rows and columns are indexed by the $r$-subsets of $[n]$, defined by $DIS_{n,r}(R_1, R_2) = 1$ if $R_1 \cap R_2 = \emptyset$ and $DIS_{n,r}(R_1, R_2) = 0$ otherwise. It is well known (see, e.g., [70]) that the matrix $DIS_{n,r}$ is nonsingular (over the reals) for all $n \geq 2r$, and as this is the case here and $T$ is nonzero, it follows that the rank of $U$ is at least that of $U \cdot W^t$ which is \binom{n}{r}$. As this rank is at most the number of columns of $U$, we conclude that
\[ |F| \binom{\ell - r}{r} \geq \binom{n}{r}, \]
completing the proof of part (i).

(ii) The proof is similar to that of part (i), with a few modifications. Here are the details. Let $F$ be a perfectly $2r + 1$-balanced family of functions from $[n]$ to $[\ell]$.

For each $R \subset [n - 1]$ of size $|R| = r$ define two vectors $u_R$ and $w_R$, each of length $|F| \binom{\ell - 1 - r}{r}$, whose coordinates are indexed by the set of all ordered triples $(f, S_1, S_2)$, satisfying
\[ f \in F, \ S_1, S_2 \subset [\ell] - \{f(n)\}, \ |S_1| = |S_2| = r, \ \text{and} \ S_1 \cap S_2 = \emptyset. \]
These vectors are defined as before:
\[ u_R(f, S_1, S_2) = 1 \ \text{if} \ f(R) = S_1, \ \text{and} \ u_R(f, S_1, S_2) = 0 \ \text{otherwise.} \]
\[ w_R(f, S_1, S_2) = 1 \ \text{if} \ f(R) = S_2, \ \text{and} \ w_R(f, S_1, S_2) = 0 \ \text{otherwise.} \]
It is clear that just as before, the inner product of two such vectors $u_{R_1}$ and $w_{R_2}$ is zero if $R_1 \cap R_2 \neq \emptyset$. Similarly, if $R_1 \cap R_2 = \emptyset$, the inner product of $u_{R_1}$ and $w_{R_2}$ is precisely the number of functions $f \in F$ which are one-to-one on $R_1 \cup R_2 \cup \{n\}$. Indeed, for each such $f$ there is a unique pair of
disjoint subsets $S_1, S_2$ of $[\ell] - \{f(n)\}$, each of size $r$, so that $f(R_1) = S_1$ and $f(R_2) = S_2$, while if $f$ does not map $R_1 \cup R_2 \cup \{n\}$ in a one-to-one manner, there is no such pair. As before, since $\mathcal{F}$ is a perfectly balanced $2r+1$-family, there exists a positive $T$ so that for the matrices $U$ and $W$ whose rows are all vectors $u_R$ and $w_R$, respectively, with $R \subset [n-1]$, $|R| = r$, the product $U \cdot W^t = T \cdot DIS_{n-1,r}$. The desired result follows as before, since $DIS_{n-1,r}$ is nonsingular and yet its rank cannot exceed the number of columns of $U$. This completes the proof of part (ii).

(iii) Let $\mathcal{F}$ be a perfectly 2-balanced family of functions from $[n]$ to $[2]$. Note that by part (i), $|\mathcal{F}| \geq n/2$, but here one can improve the constant factor and obtain a tight bound. To do so, define, for each $i \in [n]$, a vector $u_i$ of length $|\mathcal{F}|$, whose coordinates are indexed by the elements of $\mathcal{F}$, where here $u_i(f) = (-1)^{f(i)-1}$. It is easy to check that the inner product of $u_i$ and $u_j$ is $|\mathcal{F}|$ if $i = j$, and is $|\mathcal{F}| - 2T$ if $i \neq j$, where here $T > 0$ is the number of functions $f \in \mathcal{F}$ that map $i$ and $j$ to distinct elements. (This number is the same for all $i \neq j$, as $\mathcal{F}$ is perfectly 2-balanced.) We conclude that all diagonal elements of the gram matrix of the $n$ vectors $u_i$ are $|\mathcal{F}|$, while all other elements are $|\mathcal{F}|-2T$. It is easy to check that this matrix is nonsingular unless the sum of its elements in each row is zero, in which case it has rank $n-1$. In fact, all eigenvalues of this matrix are $2T$, with multiplicity $n-1$, and the sum of all entries in a row, with multiplicity 1. (In case this sum is also $2T$, then the matrix is $2T$ times the identity matrix, and all eigenvalues are equal). We conclude that the length of the vectors, $|\mathcal{F}|$ is always at least $n-1$. Equality can hold only if the sum of elements in a row of the gram matrix is 0. In this case, $|\mathcal{F}| = n-1$ and $n-1-2T = -1$, that is, the inner product of each two of our $n$ vectors is $-1$. For each $1 \leq i \leq n$, let $\overline{u}_i$ denote the vector obtained from $u_i$ by adding to it a coordinate in which its value is 1. Then the vectors $\overline{u}_i$ are $n$ pairwise orthogonal vectors of length $n$ with $\{-1,1\}$ entries, that is, they form the rows of a Hadamard matrix of order $n$. Thus, if there is no Hadamard matrix of order $n$ then any family of perfectly 2-balanced functions from $[n]$ to $[2]$ has at least $n$ functions. The family $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$ in which $f_i(i) = 1$ and $f_i(j) = 2$ for all $j \neq i$ shows that this is tight, completing the proof of the theorem. ■
Remarks:
(i) A well known conjecture (c.f., e.g., [61]) asserts that for \( n > 2 \) there is a Hadamard matrix of order \( n \) if and only if \( n \) is divisible by 4. It is easy to see that if there is such a matrix that \( n \) is indeed divisible by 4. The converse is not known, but there are many infinite families of known Hadamard matrices, showing that the \( (n - 1) \)-bound in part (iii) of the theorem is tight in many cases.
(ii) It is easy to see that for every \( n > k \) there is a perfectly \( k \)-balanced family \( F \) of functions from \([n]\) to \([k]\) of size \( |F| = \binom{n}{k-1} \). Indeed, for each subset \( R = \{r_1, r_2, \ldots, r_{k-1}\} \) of \([n]\), with \( r_1 < r_2 < \ldots < r_{k-1} \) let \( f_R \) denote the function defined by \( f_R(i) = i \) for all \( 1 \leq i \leq k-1 \), and \( f_R(j) = k \) for all \( j \in [n] - R \). It is not difficult to check that the family of all these functions \( f_R \) is perfectly \( k \)-balanced (with \( T = k \)).
(iii) The lower bounds in Theorem 2.3.1 hold for weighted families as well, even if the weight \( weight(f) \) of some of the functions \( f \) is negative, as long as there is a real \( T \neq 0 \) so that for every \( K \subset [n], |K| = k \), the total weight of functions which are one-to-one on \( K \) is exactly \( T \). To see this, repeat the proof above, modifying the definition of the vectors \( u_R \) to be

\[
u_R(f, S_1, S_2) = weight(f) \text{ if } f(R) = S_1, \text{ and } u_R(f, S_1, S_2) = 0 \text{ otherwise,}
\]
keeping the definition of the vectors \( w_R \) as before.

2.4 Probabilistic Constructions

We will use the following two claims: a variant of the Chernoff bound (c.f., e.g., [21]) and Robbins’ formula [50] (a tight version of Stirling’s formula).

Claim 2.4.1. Let \( Y \) be the sum of mutually independent indicator random variables, \( \mu = E[Y] \). For all \( 0 \leq \delta \leq 1 \),

\[
Pr[(1 - \delta)\mu \leq Y \leq (1 + \delta)\mu] > 1 - 2e^{-\delta^2\mu/4}.
\]

Claim 2.4.2. For every integer \( n \geq 1 \),

\[
\sqrt{2\pi}n^{n+1/2}e^{-n+1/(12n+1)} < n! < \sqrt{2\pi}n^{n+1/2}e^{-n+1/(12n)}.
\]
Now we state the results for \( \delta \)-balanced \((n, k, \ell)\)-splitters of the three types: \( k = \ell \), \( k < \ell \) and \( k > \ell \).

**Theorem 2.4.3.** For any \( 0 < \delta \leq 1 \), there exists a \((\delta, k)\)-balanced family of hash functions from \([n]\) to \([k]\) of size \( O\left(\frac{e^k \sqrt{k \log n}}{\delta^2}\right) \).

**Proof.** Set \( p = k! / k^k \) and \( M = \lceil \frac{4(k \ln n + 1)}{p \delta^2} \rceil \). We choose \( M \) independent random functions. For a specific set \( S \in \binom{[n]}{k} \), the expected number of functions that are one-to-one on \( S \) is exactly \( pM \). By the Chernoff bound, the probability that for at least one set \( S \in \binom{[n]}{k} \), the number of functions that are one-to-one on \( S \) will not be as needed is at most

\[
\binom{n}{k} 2 e^{-\delta^2 pM/4} \leq 2 \binom{n}{k} e^{-(k \ln n + 1)} < 1.
\]

\[\blacksquare\]

**Theorem 2.4.4.** For any \( k < \ell \) and \( 0 < \delta \leq 1 \), there exists a \( \delta \)-balanced \((n, k, \ell)\)-splitter of size \( O\left(\frac{e^{k^2/\ell_k \log n}}{\delta^2}\right) \).

**Proof.** We set \( p = \frac{\ell!}{(\ell - k)! k^\ell} \) and \( M = \lceil \frac{4(k \ln n + 1)}{p \delta^2} \rceil \). Using Robbins’ formula, we get

\[
\frac{1}{p} \leq e^{k+1/12}(1 - \frac{k}{\ell})^{\ell-k+1/2} \leq e^{k+1/12} e^{-\frac{k^2}{2(\ell-k+1/2)}} = e^{\frac{k^2-k/2+1/12}{2}}.
\]

We choose \( M \) independent random functions and proceed as in the proof of Theorem 2.4.3. \[\blacksquare\]

For the case \( k > \ell \), the probabilistic arguments from [77] can be generalized to prove existence of balanced \((n, k, \ell)\)-splitters. Here we focus on the special case of balanced \((n, k, 2)\)-splitters, which will be of interest later.

**Theorem 2.4.5.** For any \( k \geq 2 \) and \( 0 < \delta \leq 1 \), there exists a \( \delta \)-balanced \((n, k, 2)\)-splitter of size \( O\left(\frac{e^{k^2/\ell_k \log n}}{\delta^2}\right) \).

**Proof.** Set \( M = \lceil \frac{4(k \ln n + 1)}{p \delta^2} \rceil \), where \( p \) denotes the probability to get the needed split in a random function. It follows easily from Robbins’ formula that \( 1/p = O(\sqrt{k}) \). We choose \( M \) independent random functions and proceed as in the proof of Theorem 2.4.3. \[\blacksquare\]
2.5 Composition Lemmas

The definitions of balanced families of perfect hash functions and balanced splitters given above enable us to state the following easy composition lemmas.

**Lemma 2.5.1.** For any \( k \leq \ell, \delta > 0, \) and \( 0 < \gamma \leq 1, \) let \( H \) be an explicit \((\delta, k)\)-balanced family of hash functions from \([n]\) to \([\ell]\) of size \( N \) and let \( G \) be an explicit \((\gamma, k)\)-balanced family of hash functions from \([\ell]\) to \([k]\) of size \( M \). We can use \( H \) and \( G \) to get an explicit \((2\delta + \gamma, k)\)-balanced family of hash functions from \([n]\) to \([k]\) of size \( NM \).

**Proof.** We compose every function of \( H \) with every function of \( G \) and get the needed result. This gives a family of hash functions from \([n]\) to \([k]\). There is a constant \( T > 0 \), such that for every \( S \in \binom{[n]}{k} \), the number functions that are of one-to-one on \( S \) functions is between \((1−\delta)(1−\gamma)T \) and \((1+\delta)(1+\gamma)T \). Note that \((1−\delta)(1−\gamma) \geq 1−(\delta+\gamma) \). Since \( \gamma \leq 1 \), it follows that \((1+\delta)(1+\gamma) \leq 1 + 2\delta + \gamma \). ■

**Lemma 2.5.2.** For any \( k > \ell, \delta, \gamma_1, \ldots, \gamma_\ell > 0, \) such that \( \sum_{j=1}^\ell \gamma_j \leq 1/2 \), let \( H \) be an explicit \( \delta \)-balanced \((n, k, \ell)\)-splitter of size \( N \). For every \( j, j = 1, \ldots, \ell \), let \( G_j \) be an explicit \((\gamma_j, k_j)\)-balanced family of hash functions from \([n]\) to \([k_j]\) of size \( M_j \), where \( k_j = [k/\ell]\) for every \( j \leq k \mod \ell \) and \( k_j = [k/\ell] \) otherwise. We can use these constructions to get an explicit \((2(\delta + \sum_{j=1}^\ell \gamma_j), k)\)-balanced family of hash functions from \([n]\) to \([k]\) of size \( N \prod_{j=1}^\ell M_j \).

**Proof.** We divide the set \([k]\) into \( \ell \) disjoint intervals \( I_1, \ldots, I_\ell \), where the size of \( I_j \) is \( k_j \) for every \( j = 1, \ldots, \ell \). We think of \( G_j \) as a family of functions from \([n]\) to \( I_j \). For every combination of \( h \in H \) and \( g_j \in G_j, j = 1, \ldots, \ell \), we create a new function that maps an element \( x \in [n] \) to \( g_{h(x)}(x) \).

This gives a family of hash functions from \([n]\) to \([k]\). There is a constant \( T > 0 \), such that for every \( S \in \binom{[n]}{k} \), the number functions that are of one-to-one on \( S \) functions is between \((1−\delta) \prod_{j=1}^\ell (1−\gamma_j)T \) and \((1+\delta) \prod_{j=1}^\ell (1+\gamma_j)T \). Note that \((1−\delta) \prod_{j=1}^\ell (1−\gamma_j) \geq 1−(\delta+\sum_{j=1}^\ell \gamma_j) \).

It is easy to prove by induction that if \( \sum_{j=1}^\ell \gamma_j \leq 1/2 \), then \( \prod_{j=1}^\ell (1+\gamma_j) \leq 1 + 2\sum_{j=1}^\ell \gamma_j \). It follows that \((1+\delta) \prod_{j=1}^\ell (1+\gamma_j) \leq 1 + 2(\delta + \sum_{j=1}^\ell \gamma_j) \). ■
2.6 Explicit Constructions

In this chapter, we use the term explicit construction for an algorithm that lists all the elements of the required family of functions in time which is polynomial in the total size of the functions. For a discussion on other definitions for this term, the reader is referred to [77]. We state our results for \( \delta \)-balanced \((n, k, \ell)\)-splitters of the three types: \( k = \ell \), \( k < \ell \) and \( k > \ell \).

**Theorem 2.6.1.** For any fixed \( 0 < \delta \leq 1 \), a \((\delta, k)\)-balanced family of hash functions from \([n]\) to \([k]\) of size \( O\left( \frac{e^k \sqrt{k} \log n}{\delta^2} \right) \) can be constructed deterministically within time \( \left( \frac{n^k}{k} \right) e^{kO(1)} n^{\log_2 n} \).

**Proof.** We set \( p = \frac{k!}{k^k} \) and \( M = \left\lfloor \frac{4(k \ln n + 1)}{p \delta^2} \right\rfloor \). Denote \( \lambda = \delta/2 \), so obviously \( 0 < \lambda \leq 1/2 \). Consider a choice of \( M \) independent random functions from \([n]\) to \([k]\). This choice will be derandomized in the course of the algorithm. For every \( S \in \binom{[n]}{k} \), we define \( X_S = \sum_{i=1}^{M} X_{S,i} \), where \( X_{S,i} \) is the indicator random variable that is equal to 1 iff the \( i \)th function is one-to-one on \( S \).

Consider the following potential function:

\[
\Phi = \sum_{S \in \binom{[n]}{k}} e^{\lambda (X_S - pM)} + e^{\lambda (pM - X_S)}.
\]

Its expectation can be calculated as follows:

\[
E[\Phi] = \binom{n}{k} (e^{-\lambda pM} \prod_{i=1}^{M} E[e^{\lambda X_{S,i}}] + e^{\lambda pM} \prod_{i=1}^{M} E[e^{-\lambda X_{S,i}}]) =
\]

\[
= \binom{n}{k} (e^{-\lambda pM} [pe^\lambda + (1 - p)]^M + e^{\lambda pM} [pe^{-\lambda} + (1 - p)]^M).
\]

We now give an upper bound for \( E[\Phi] \). Since \( 1 + u \leq e^u \) for all \( u \) and \( e^{-u} \leq 1 - u + u^2/2 \) for all \( u \geq 0 \), we get that \( pe^{-\lambda} + (1 - p) \leq e^{p(e^{-\lambda} - 1)} \leq e^{p(-\lambda + \lambda^2/2)} \).

Define \( \varepsilon = e^\lambda - 1 \), that is \( \lambda = \ln(1 + \varepsilon) \). Thus \( pe^\lambda + (1 - p) = 1 + \varepsilon p \leq e^{p} \).

This implies that

\[
E[\Phi] \leq n^k ((\frac{e^\varepsilon}{1 + \varepsilon}) pM + e^{\lambda^2 pM/2}).
\]
Since \( e^u \leq 1 + u + u^2 \) for all \( 0 \leq u \leq 1 \), we have that \( \frac{e^u}{1+e} = e^{e^{1-u}-1} \leq e^{\lambda^2} \).

We conclude that
\[
E[\Phi] \leq 2n^k e^{\lambda^2 pM} \leq e^{2(k\ln n + 1)}.
\]

We now describe a deterministic algorithm for finding \( M \) functions, so that \( E[\Phi] \) will still obey the last upper bound. This is performed using the method of conditional probabilities (c.f., e.g., \( [21] \), chapter 15). The algorithm will have \( M \) phases, where each phase will consist of \( n \) steps. In step \( i \) of phase \( j \) the algorithm will determine the \( i \)th value of the \( j \)th function.

Out of the \( k \) possible values, we greedily choose the value that will decrease \( E[\Phi] \) as much as possible. We note that at any specific step of the algorithm, the exact value of the conditional expectation of the potential function can be easily computed in time \( (n^k)^O(1) \).

After all the \( M \) functions have been determined, every set \( S \in \binom{[n]}{k} \) satisfies the following:
\[
e^{\lambda(X_S-pM)} + e^{\lambda(pM-X_S)} \leq e^{2(k\ln n + 1)}.
\]

This implies that
\[
-2(k\ln n + 1) \leq \lambda(X_S - pM) \leq 2(k\ln n + 1).
\]

Recall that \( \lambda = \frac{\delta}{2} \), and therefore
\[
(1 - \frac{4(k\ln n + 1)}{\delta pM})pM \leq X_S \leq (1 + \frac{4(k\ln n + 1)}{\delta pM})pM.
\]

Plugging in the values of \( M \) and \( p \) we get the desired result
\[
(1 - \delta)pM \leq X_S \leq (1 + \delta)pM.
\]

The \( \binom{n}{k} \) term in the time bound of the last Theorem is not good enough for our applications. Our goal is to improve the running time so that it will be polynomial in \( n \) for \( k \leq O(\frac{\log n}{\log \log \log n}) \) and later also for \( k \leq O(\log n) \). Still, the last Theorem will be part of the main construction.
Theorem 2.6.2. For any $0 < \delta \leq 1$, a $(\delta, k)$-balanced family of hash functions from $[n]$ to $[q]$, where $q = \lceil \frac{k^2}{\delta} \rceil$, of size $\frac{k^{O(1)} \log n}{\delta^{O(1)}}$ can be constructed in time $\frac{k^{O(1)} \log n}{\delta^{O(1)}}$.

Proof. Denote $q = \lceil \frac{k^2}{\delta} \rceil$. Consider an explicit construction of an error correcting code with $n$ codewords over alphabet $[q]$ whose normalized Hamming distance is at least $1 - \frac{2}{q}$. Such explicit codes of length $O(q^2 \log n)$ exist [11].

Now let every index of the code corresponds to a function from $[n]$ to $[q]$. If we denote by $M$ the length of the code, which is in fact the size of the family of functions, then for every $S \in \binom{[n]}{k}$, the number of one-to-one functions is at least

$$(1 - \left(\frac{k}{2}\right)^2 \frac{2}{q}) M \geq (1 - \delta)M.$$

For our next construction we use small probability spaces that support a sequence of almost $k$-wise independent random variables. A sequence $X_1, \ldots, X_n$ of random Boolean variables is $(\varepsilon, k)$-independent if for any $k$ positions $i_1 < \cdots < i_k$ and any $k$ bits $\alpha_1, \ldots, \alpha_k$ we have

$$|Pr[X_{i_1} = \alpha_1, \ldots, X_{i_k} = \alpha_k] - 2^{-k}| < \varepsilon.$$ 

It is known ([76], [14], [11]) that sample spaces of size $2^{O(k + \log \frac{1}{\varepsilon})} \log n$ that support $n$ random variables that are $(\varepsilon, k)$-independent can be constructed in time $2^{O(k + \log \frac{1}{\varepsilon})} n \log n$.

Theorem 2.6.3. For any $k \geq \ell$ and $0 < \delta \leq 1$, a $\delta$-balanced $(n, k, \ell)$-splitter of size $2^{O(k \log \ell - \log \delta)} \log n$ can be constructed in time $2^{O(k \log \ell - \log \delta)} n \log n$.

Proof. We use an explicit probability space of size $2^{O(k \log \ell - \log \delta)} \log n$ that supports $n[\log_2 \ell]$ random variables that are $(\varepsilon, k[\log_2 \ell])$-independent where $\varepsilon = 2^{-k[\log_2 \ell] \delta}$. We attach $[\log_2 \ell]$ random variables to each element of $[n]$, thereby assigning it a value from $[2^{[\log_2 \ell]}]$. In case $\ell$ is not a power of 2, all elements of $[2^{[\log_2 \ell]}] - [\ell]$ can be mapped to $[\ell]$ by some arbitrary fixed function. If follows from the construction that there exists a constant $T > 0$ so that for every $S \in \binom{[n]}{k}$, the number of good splits satisfies

$$(1 - \delta)T \leq \text{split}(S) \leq (1 + \delta)T.$$
Note that the constant $T$ is independent of the chosen set $S$. ■

**Corollary 2.6.4.** For any fixed $c > 0$, a $(c^{-k})$-balanced $(n, k, 2)$-splitter of size $2^{O(k)} \log n$ can be constructed in time $2^{O(k)} n \log n$.

Setting $\ell = k$ in Theorem 2.6.3, we get that a $(\delta, k)$-balanced family of hash functions from $[n]$ to $[k]$ of size $2^{O(k \log k - \log \delta)} \log n$ can be constructed within time $2^{O(k \log k - \log \delta)} n \log n$. Note that if $k$ is small enough with respect to $n$, say $k = O(\log n / \log \log n)$, then for any fixed $0 < \delta \leq 1$, this already gives a family of functions of size polynomial in $n$. We improve upon this last result in the following Theorem, which is the main construction of this section. This construction is achieved by composing three types of splitters. Further details concerning the way these splitters can be combined is presented in section 2.10 in which an optimal construction is given.

**Theorem 2.6.5.** For any $0 < \delta \leq 1$, a $(\delta, k)$-balanced family of hash functions from $[n]$ to $[k]$ of size $2^{O(k \log k \log log k)} \log n$ can be constructed in time $2^{O(k \log k \log log k)} n \log n + \delta^{-O(k / \log k)}$. In particular, for any fixed $0 < \delta \leq 1$, the size is $2^{O(k \log k \log log k)} \log n$ and the time is $2^{O(k \log k \log log k)} n \log n$.

**Proof.** Denote $\ell = \lceil \log_2 k \rceil, \delta' = \delta / 6, \delta'' = \delta / (6\ell)$, and $q = \lceil k^{2} / m \rceil$. Let $H$ be a $\delta'$-balanced $(q, k, \ell)$-splitter of size $2^{O(k \log k \log k)} \delta'^{-O(1)}$ constructed using Theorem 2.6.3. For every $j, j = 1, \ldots, \ell$, let $B_j$ be a $(\delta'', k_j)$-balanced family of hash functions from $[q]$ to $[k_j]$ of size $O(\ell^{k_j} \log k_j) \delta''^{-O(1)}$ constructed using Theorem 2.6.4, where $k_j = \lceil k / \ell \rceil$ for every $j \leq k \mod \ell$ and $k_j = \lceil k / \ell \rceil$ otherwise. Using Lemma 2.5.2 for composing $H$ and $\{B_j\}_{j=1}^\ell$, we get a $(4\delta', k)$-balanced family $D'$ of hash functions from $[q]$ to $[k]$.

Now let $D''$ be a $(\delta', k)$-balanced family of hash functions from $[n]$ to $[q]$ of size $k^{O(1)} \delta''^{-O(1)} \log n$ constructed using Theorem 2.6.2. Using Lemma 2.5.1 for composing $D''$ and $D'$, we get a $(\delta, k)$-balanced family of hash functions from $[n]$ to $[k]$, as needed. Note that for calculating the size of each $B_j$, we use the fact that $\delta'' = \delta / (6\ell)$. The time needed to construct each $B_j$ is $2^{O(k)} \delta^{-O(k / \log k)}$. The $2^{O(k)}$ term is omitted in the final result, as it is negligible in respect to the other terms. ■
2.7 Approximate Counting of Paths and Cycles

We now state what it means for an algorithm to approximate a counting problem.

Definition 2.7.1. We say that an algorithm approximates a counting problem up to relative error $\delta \geq 0$ if for every input $x$, the output $\text{ALG}(x)$ of the algorithm satisfies $(1 - \delta)N(x) \leq \text{ALG}(x) \leq (1 + \delta)N(x)$, where $N(x)$ is the exact output of the counting problem for input $x$.

The technique of color-coding is used for approximate counting of paths and cycles. Let $G = (V, E)$ be a directed or undirected graph. In our algorithms we will use constructions of balanced families of hash functions from $V$ to $[k]$. Each such function defines a coloring of the vertices of the graph. A path is said to be colorful if each vertex on it is colored by a distinct color. Our goal is to count the exact number of colorful paths in each of these colorings.

Theorem 2.7.2. For any $0 < \delta \leq 1$, the number of simple (directed or undirected) paths of length $k-1$ in a (directed or undirected) graph $G = (V, E)$ can be approximated up to relative error $\delta$ in time $\frac{2^{O(k \log \log k)}}{\delta^{O(\log k)}} |E| \log |V| + \delta^{-O(k/\log k)}$.

Proof. We use the $(\delta, k)$-balanced family of hash functions from $V$ to $[k]$ constructed using Theorem 2.6.5. Each function of the family defines a coloring of the vertices in $k$ colors. We know that there exists a constant $T > 0$, so that for each set $S \subset V$ of $k$ vertices, the number of functions that are one-to-one on $S$ is between $(1 - \delta)T$ and $(1 + \delta)T$. The exact value of $T$ can be easily calculated in all of our explicit constructions. In all these constructions, the value $T$ associated with a $\delta$-balanced $(n, k, \ell)$-splitter depends only on the parameters $\delta$, $n$, $k$, and $\ell$. Thus, there is no need to calculate $T$ from the explicit family of functions.

For each coloring, we use a dynamic programming approach in order to calculate the exact number of colorful paths. We do this in $k$ phases. In the $i$th phase, for each vertex $v \in V$ and for each subset $C \subset \{1, \ldots, k\}$
of $i$ colors, we calculate the number of colorful paths of length $i - 1$ that end at $v$ and use the colors of $C$. To do so, for every edge $(u, v) \in E$, we check whether it can be the last edge of a colorful path of length $i - 1$ ending at either $u$ or $v$. Its contribution to the number of paths of length $i - 1$ is calculated using our knowledge on the number of paths of length $i - 2$. The initialization of phase 1 is easy and after performing phase $k$ we know the exact number of paths of length $k - 1$ that end at each vertex $v \in V$. The time to process each coloring is therefore $2^{O(k)} |E|$.

We sum the results over all colorings and all ending vertices $v \in V$. The result is divided by $T$. In case the graph is undirected, we further divide by 2. This is guaranteed to be the needed approximation.

Theorem 2.7.3. For any $0 < \delta \leq 1$, the number of simple (directed or undirected) cycles of size $k$ in a (directed or undirected) graph $G = (V, E)$ can be approximated up to relative error $\delta$ in time $2^{O(k \log \log k)} |E||V| \log |V| + \delta^{-O(k/ \log k)}$.

Proof. We use the $(\delta, k)$-balanced family of hash functions from $V$ to $[k]$ constructed using Theorem 2.6.5. For every set $S$ of $k$ vertices, the number of functions that are one-to-one on $S$ is between $(1 - \delta)T$ and $(1 + \delta)T$. Every function defines a coloring and for each such coloring we proceed as follows. For every vertex $s \in V$ we run the algorithm described in the proof of Theorem 2.7.2 in order to calculate for each vertex $v \in V$ the exact number of colorful paths of length $k - 1$ from $s$ to $v$. In case there is an edge $(v, s)$ that completes a cycle, we add the result to our count.

We sum the results over all the colorings and all pairs of vertices $s$ and $v$ as described above. The result is divided by $kT$. In case the graph is undirected, we further divide by 2. The needed approximation is achieved.

Corollary 2.7.4. For any constant $c > 0$, there is a deterministic polynomial time algorithm for approximating both the number of simple paths of length $k$ and the number of simple cycles of size $k$ for every $k \leq O(\frac{\log n}{\log \log \log n})$ in a graph with $n$ vertices, where the approximation is up to relative error of $(\ln \ln n)^{-c \ln \ln n}$.

The last corollary is improved in section 2.10 with a deterministic polynomial time algorithm for approximately counting the number of paths or
cycles of a specified length \( k \leq O(\log n) \), where the approximation is up to any relative error \( \varepsilon \).

## 2.8 Expanders

In this section we describe a special case of the Cayley expanders of \[19\] that suffices for our purposes. Note that these are not bounded-degree graphs, and their degrees grow with the number of vertices, but they suffice for our purpose. This is a special case of a construction suggested in \[19\], which is based on one of the codes described in \[14\].

The following are standard definitions and observations concerning eigenvalues and expanders (c.f., e.g., \[21\], \[63\]).

Let \( G = (V, E) \) be a \( d \)-regular graph and let \( A = A_G = (a_{uv})_{u,v \in V} \) be its adjacency matrix. Since \( G \) is \( d \)-regular, the largest eigenvalue of \( A \) is \( d \), corresponding to the all 1 eigenvector. Let \( \lambda = \lambda(G) \) denote the largest absolute value of an eigenvalue other than the first one. For two (not necessarily disjoint) subsets \( B \) and \( C \) of \( V \), let \( e(B, C) \) denote the number of ordered pairs \((u, v)\), where \( u \in B, v \in C \) and \( uv \) is an edge of \( G \). The following useful bound is the Expander Mixing Lemma (c.f., e.g., \[21\], page 146).

**Proposition 2.8.1.** Let \( G \) be a \( d \)-regular graph with \( n \) vertices and set \( \lambda = \lambda(G) \). For every two sets of vertices \( B \) and \( C \) of \( G \), where \( |B| = bn \) and \( |C| = cn \), we have
\[
|e(B, C) - bcdn| \leq \lambda \sqrt{bcn}.
\]

We need the following explicit expanders, described, for example, in \[20\], following \[19\]. Let \( \text{bin} : \text{GF}(2^k) \rightarrow \{0, 1\}^k \) be a one-to-one mapping satisfying \( \text{bin}(0) = 0^k \) and \( \text{bin}(x+y) = \text{bin}(x) \oplus \text{bin}(y) \), where \( \alpha \oplus \beta \) means the bit-by-bit xor of the binary strings \( \alpha \) and \( \beta \). (The standard representation of \( \text{GF}(2^k) \) as a vector space satisfies the above conditions.) Given \( x, y \in \text{GF}(2^k) \), let \( < x, y > \) denote the bit \( (\text{bin}(x), \text{bin}(y))_2 \), where \( (\alpha, \beta)_2 \) is the inner-product mod 2 of the binary vectors \( \alpha \) and \( \beta \). For a fixed \( d \) and \( x, y \in \text{GF}(2^k) \), the binary vector \( u_{xy} \) is defined as \( < x, y > < x^2, y > \cdots < x^d, y > \). For every \( d, k \geq 1 \), we define a \( 4^k \)-regular graph \( G_{d,k} \) with \( 2^d \) vertices, as follows.
The vertex set is \(\{0,1\}^d\) and every vertex \(v\) is adjacent to \(v \oplus u_{xy}\) for all \(x, y \in GF(2^k)\).

**Theorem 2.8.2.** For every two positive integers \(d\) and \(k\) satisfying \(4^k < 2^d\) there is an explicit construction of a \(4^k\)-regular graph \(G_{d,k}\) on \(2^d\) vertices so that \(\lambda(G_{d,k}) \leq d \cdot 2^k\).

**Proof.** Denote \(F = GF(2^k), D = \{0,1\}^d\), and let \(A\) be the \(2^d \times 2^d\) adjacency matrix of \(G_{d,k}\). For every \(a = a_1 a_2 \cdots a_d \in D\), let \(v_a\) be the vector whose \(b\)th entry, where \(b \in D\), satisfies \(v_a(b) = (-1)^{(a,b)_2}\). Let \(p_a(x)\) be the polynomial \(\sum_{i=1}^d a_i x^i\) and denote \(\lambda_a = \sum_{x,y \in F} (-1)^{<p_a(x),y>}\). We now prove that \(v_a\) is an eigenvector of \(A\) over \(\mathbb{R}\) with eigenvalue \(\lambda_a\).

\[
(Av_a)(b) = \sum_{c \in D} A_{bc} v_a(c) = \sum_{x,y \in F} v_a(b \oplus u_{xy}) = v_a(b) \sum_{x,y \in F} v_a(u_{xy}) = v_a(b) \sum_{x,y \in F} (-1)^{(a,u_{xy})_2} = \lambda_a v_a(b).
\]

It is easy to verify that the \(2^d\) vectors \(\{v_a\}_{a \in D}\) are orthogonal, and therefore we found all the eigenvalues of \(A\). It remains to bound the absolute value of \(\lambda_a\). For a fixed \(x \in F\), the term \(\sum_{y \in F} (-1)^{<p_a(x),y>}\) is equal to \(2^k\) if \(p_a(x) = 0\), and to zero in case \(p_a(x) \neq 0\). If \(a \neq 0^d\), then \(p_a(x)\) is a non-zero polynomial of degree at most \(d\), and therefore has at most \(d\) roots. Thus, \(|\lambda_a| \leq d \cdot 2^k\), as needed.

Note that this construction is applicable for a wide range of parameters, that is, the number of vertices of the expander can be any power of 2, whereas the degree can be any power of 4.

### 2.9 Partially Independent Variables

In this section we introduce a certain relaxation of almost \(k\)-wise independence and describe an appropriate explicit construction, which will give the main building block required in the construction of balanced families of hash functions of optimal size. For notational convenience, we give the following definitions related to the probabilities implied by a multinomial distribution.
Definition 2.9.1. Suppose that $1 \leq \ell \leq k$ and $k = k_1 + k_2 + \cdots + k_\ell$, where $k_i \geq 0$ for every $i$. Define $m(k_1, \ldots, k_\ell)$ to be the following probability:

$$\frac{k!}{k_1!k_2! \cdots k_\ell!\ell^k}.$$

For random variables $X_1, \ldots, X_k$, let $Y_i$ denote the number of variables $X_j$ that are equal to $i$. Define $M(X_1, \ldots X_k; k_1, \ldots k_\ell)$ to be the event that $Y_i = k_i$ for every $i$, $1 \leq i \leq \ell$.

We now construct probability distributions which are uniform over a set of strings of length $q$ in the alphabet $[\ell]$. In the standard notion of almost $k$-wise independence, it is required that in any $k$ positions, each substring of length $k$ appears with probability close to $\ell^{-k}$. Here we are interested in a weaker condition. Our objective is to construct small probability spaces of the following type.

Definition 2.9.2. A sequence $X_1, \ldots X_q$ of random variables that take values from $[\ell]$ is $(\varepsilon, k)$-partially-independent if for any $p \leq k$ positions $i_1 < \cdots < i_p$ and any $\ell$ values $k_1, \ldots, k_\ell$ such that $k_1 + \cdots + k_\ell = p$, we have

$$|\Pr[M(X_{i_1}, \ldots X_{i_p}; k_1, \ldots k_\ell)] - m(k_1, \ldots, k_\ell)| < \varepsilon.$$

Observe that we require the property to hold for any $p$ variables, where $1 \leq p \leq k$. This is needed since the fact that the property is satisfied for a value $p$ does not imply that it holds for $p' < p$. Furthermore, requiring that it applies for every value $p \leq k$ is crucial for the correctness of our recursive construction. To demonstrate the definition, here is what it means for $\ell = 2$.

A sequence $X_1, \ldots X_q$ of random Boolean variables (taking values from $\{0,1\}$) is $(\varepsilon, k)$-partially-independent if for any $p \leq k$ positions $i_1 < \cdots < i_p$ and any $r$, $0 \leq r \leq p$, we have $|\Pr[X_{i_1} + \cdots + X_{i_p} = r] - \binom{p}{r}2^{-p}| < \varepsilon$.

Theorem 2.9.3. For any $\ell \leq k \leq q$ and $0 < \varepsilon \leq 1$, a sample space of size $\left(\frac{q\ell}{\varepsilon}\right)^{O(\log q)}$ that supports $q$ variables that take values from $[\ell]$ and are $(\varepsilon, k)$-partially-independent can be constructed in time $\left(\frac{q\ell}{\varepsilon}\right)^{O(\log q)}$. 
2.9 Partially Independent Variables

Proof. Assume, without loss of generality, that $q$ is a power 2. Otherwise, $q$ can be simply rounded to the next power of 2. Assume also that $\varepsilon \leq \frac{1}{D^2}$. If this is not the case, then $\varepsilon$ can be replaced by $\frac{\varepsilon}{D}$. We recursively construct sample spaces that support an increasing number of variables. For every $t = 0, 1, \ldots, \log_2 q$, we shall construct a sample space $C_t$ that supports $2^t$ variables that take values from $[\ell]$ and are $(\frac{4 \varepsilon q^2}{\ell^t}, k)$-partially-independent. The sample space $C_t$ will consists of strings of length $2^t$ over the alphabet $[\ell]$.

We start with $t = 0$. To support one variable, it is possible to simply define a sample space that consists of the $\ell$ strings of length 1, and there will be no error at all in this case. For our purpose, the size of each sample space should be a power of 2, so let $N_0$ be the result of rounding the value $4(\frac{20q^2k^t}{\varepsilon})^4$ to the next higher power of 2. The sample space consists of $N_0$ strings, where each string of length 1 appears either $\lfloor \frac{N_0}{\ell} \rfloor$ or $\lceil \frac{N_0}{\ell} \rceil$ times. Obviously $N_0 \leq 8(\frac{20q^2k^t}{\varepsilon})^4$ and we have one variable which is certainly $(1, \frac{\varepsilon}{\ell^t})$-partially-independent.

Let $D$ be the result of rounding the value $(\frac{20q^2k^t}{\varepsilon})^4$ to the next higher power of 4. Suppose that in step $t$, a sample space of size $N_t \leq 8D^{t+1}$ that supports $2^t$ variables that are $(\frac{4 \varepsilon q^2}{\ell^t}, k)$-partially-independent has been constructed. We now describe step $t + 1$. Let $G$ be the $D$-regular expander with $N_t$ vertices described in section 2.8 (note that $D < N_t$). It follows from Theorem 2.8.2 that

$$\frac{\lambda(G)}{D} \leq \frac{\log_2 N_t}{\sqrt{D}} \leq 3 + (t + 1) \frac{\log_2 D}{\sqrt{D}} \leq \frac{(\log_2 D)^2}{\sqrt{D}} \leq \frac{20}{D^{1/4}} \leq \frac{\varepsilon}{q^2 k^t}.$$ 

To every vertex of the graph $G$ we assign one of the $N_t$ strings of length $2^t$ from $C_t$ that were constructed in step $t$. For every ordered pair $(u, v)$ such that $uv$ is an edge of $G$, the concatenation of the string assigned to $u$ followed by the string assigned to $v$ is added to the sample space $C_{t+1}$. The resulting sample space is of size $N_{t+1} = DN_t$.

Suppose that in step $t$, a sample space $C_t$ of size $N_t$ that supports $2^t$ variables that are $(\gamma, k)$-partially-independent has been constructed, where $\gamma = \frac{\varepsilon q^2}{\ell^t}$. We now prove that the approximation error is increased in step $t + 1$ by a multiplicative factor of at most 4, that is, the sample space $C_{t+1}$
supports $2^{t+1}$ variables that are $(4\gamma, k)$-partially-independent. Suppose that $p \leq k$ and take any $p$ positions $1 \leq i_1 < \cdots < i_p \leq 2^{t+1}$ and any $\ell$ values $k_1, \ldots, k_\ell$ such that $k_1 + \cdots + k_\ell = p$. We further assume that among the $p$ positions selected, exactly $p'$ positions are in the first half of the string. Therefore $Pr[M(X_{i_1}, \ldots, X_{i_p}; k_1, \ldots, k_\ell)]$ is equal to

$$
\sum_{k_1' + \cdots + k_\ell' = p'} Pr[M(X_{i_1}, \ldots, X_{i_p}; k_1', \ldots, k_\ell') \cap M(X_{i_p'+1}, \ldots, X_{i_p}; k_1-k_1', \ldots, k_\ell-k_\ell')].
$$

We would like $Pr[M(X_{i_1}, \ldots, X_{i_p}; k_1, \ldots, k_\ell)]$ to be close to:

$$m(k_1, \ldots k_\ell) = \sum_{k_1' + \cdots + k_\ell' = p'} m(k_1', \ldots k_\ell') m(k_1-k_1', \ldots, k_\ell-k_\ell').$$

Note that the number of terms in the two summations above is at most $k^\ell$ and that obviously $\sum_{k_1' + \cdots + k_\ell' = p'} m(k_1', \ldots, k_\ell') \leq 1$. Since $C_t$ is $(\gamma, k)$-partially-independent, it follows from Proposition 2.8.1 that the estimation error is as follows:

$$|Pr[M(X_{i_1}, \ldots, X_{i_p}; k_1, \ldots, k_\ell)] - m(k_1, \ldots k_\ell)| \leq$$

$$\sum_{k_1' + \cdots + k_\ell' = p'} \left[ (m(k_1', \ldots, k_\ell') + \gamma)(m(k_1-k_1', \ldots, k_\ell-k_\ell') + \gamma) + \frac{\lambda(G)}{D} \right] -$$

$$\sum_{k_1' + \cdots + k_\ell' = p'} m(k_1', \ldots, k_\ell') m(k_1-k_1', \ldots, k_\ell-k_\ell') =$$

$$\sum_{k_1' + \cdots + k_\ell' = p'} \gamma [m(k_1', \ldots, k_\ell') + m(k_1-k_1', \ldots, k_\ell-k_\ell')] + \gamma^2 + \frac{\lambda(G)}{D} \leq$$

$$2\gamma + k^\ell \left( \gamma^2 + \frac{\lambda(G)}{D} \right) \leq 4\gamma,$$

where the last inequality follows from the inequalities $\gamma \leq \varepsilon \leq \frac{1}{k^2}$ and $\frac{\lambda(G)}{D} \leq \frac{\varepsilon}{q\ell'^2} \leq \frac{\gamma}{k^2}$. After step $\log_2 q$, the sample space constructed is $(\varepsilon, k)$-partially-independent, as needed.

\[\square\]
2.10 Optimal Balanced Families

Robbins’ formula (see Claim 2.4.2) supplies the following simple lower bound for the multinomial distribution (recall Definition 2.9.1).

**Lemma 2.10.1.** If \( k \geq \ell > 0 \), then
\[
m\left(\left\lceil \frac{k}{\ell} \right\rceil, \ldots, \left\lceil \frac{k}{\ell} \right\rceil, \ell - (k \mod \ell), \left\lfloor \frac{k}{\ell} \right\rfloor, \ldots, \left\lfloor \frac{k}{\ell} \right\rfloor \right) > (15k/\ell)^{-\ell/2}.
\]

**Proof.** Assume first that \( \ell \) divides \( k \). Using Robbins’ formula, we get:
\[
m(\ell, \ldots, \ell) = \frac{k!}{(k/\ell)^{\ell} \ell^k} > (2\pi k/\ell)^{-\ell/2} e^{-\ell^2/12k} \geq (2\pi e^{1/6} k/\ell)^{-\ell/2} > (7.5k/\ell)^{-\ell/2}.
\]

Note that if \( k = k_1 + \cdots + k_\ell \) and \( k_\ell + 1 \geq (k + 1)/\ell \), then \( m(k_1, \ldots, k_\ell) \geq m(k_1, \ldots, k_{\ell-1}, k_\ell + 1) \). In case \( \ell \) does not divide \( k \), then the result follows from this monotonicity property. ■

The previous Lemma shows that the events we would like to estimate have a relatively high probability, enabling us to give the following construction.

**Theorem 2.10.2.** For any \( k \geq \ell \) and \( 0 < \varepsilon \leq 1 \), an \( \varepsilon \)-balanced \((q, k, \ell)\)-splitter of size \( \left(\frac{qk^\ell}{\varepsilon}\right)^{O(\log q)} \) can be constructed in time \( \left(\frac{qk^\ell}{\varepsilon}\right)^{O(\log q)} \).

**Proof.** As implied by Theorem 2.9.3, we use an explicit probability space of size \( \left(\frac{qk^\ell}{\gamma}\right)^{O(\log q)} \) that supports \( q \) random variables that take values from \([\ell]\) and are \((\gamma, k)\)-partially-independent, where \( \gamma = (15k/\ell)^{-\ell/2}\varepsilon \). We attach one of the random variables to each element of \([q]\). If follows from Lemma 2.10.1 that the splitter achieves the required approximation. ■

We can now describe our main construction of balanced families of hash functions, using the ingredients mentioned at the end of section 2.2. Recall that there are three ingredients in this construction. Two of them were given in Theorems 2.6.1 and 2.6.2.
The main part of the construction is the balanced \((q, k, \ell)\)-splitter described in Theorem 2.10.2. The three ingredients are combined as follows. Each function \(f\) of our final family is described by a member \(f_1\) of an \((\varepsilon/6, k)\)-balanced family of Theorem 2.6.2, a member \(f_2\) of the \(\varepsilon_2\)-balanced splitter of Theorem 2.10.2 with \(\varepsilon_2 = \frac{\varepsilon}{6}\), \(q = \lceil \frac{k^3}{\varepsilon_2^2} \rceil\) and \(\ell = \lceil \log q \rceil\), and \(\ell\) members \(\phi_1, \ldots, \phi_\ell\) of the \((\frac{\varepsilon}{6\ell}, k/\ell)\)-balanced family of hash functions from \([q]\) to \([k/\ell]\).

(For simplicity we assume here that \(\ell\) divides \(k\).) To compute the value of \(f\) on some \(x \in [n]\), we first apply \(f_1\) to \(x\), getting a value \(y\) in \([q]\), then we apply \(f_2\) to \(y\), getting as a result some \(i \in [\ell]\), and finally we apply \(\phi_i\) to \(y\), where the final result is \((i - 1)k/\ell + \phi_i(y)\). A \(k\)-set \(S \subset [n]\) can be mapped in a one-to-one manner by such an \(f\) only if it is mapped in a one-to-one manner by \(f_1\), and then only if it is split evenly into \(\ell\) parts by \(f_2\), and then only if its elements mapped to each of the \(\ell\) parts are mapped in a one-to-one manner by each of the functions \(\phi_i\). Since all the ingredients in the construction are sufficiently balanced, this gives the required balanced family.

**Theorem 2.10.3.** For any \(0 < \varepsilon \leq 1\), an \((\varepsilon, k)\)-balanced family of hash functions from \([n]\) to \([k]\) of size \(e^{k + O(\log^{3/2} k)} \log n\) can be constructed in time \(e^{k + O(\log^{3/2} k)} \log n\).

**Proof.** Denote \(\varepsilon' = \varepsilon/6\), \(\varepsilon'' = \varepsilon/(6\ell)\), \(q = \lceil \frac{k^3}{\varepsilon_2^2} \rceil\), and \(\ell = \lceil \log q \rceil\). Let \(H\) be an \(\varepsilon'\)-balanced \((q, k, \ell)\)-splitter of size \(\frac{q^{\ell^2}}{\ell^2} = e^{O(\log^{3/2} k)}\) constructed using Theorem 2.10.2. For every \(j, j = 1, \ldots, \ell\), let \(B_j\) be an \((\varepsilon'', k_j)\)-balanced family of hash functions from \([q]\) to \([k_j]\) of size \(O(\frac{e^{k\ell^3}k_j\log q}{\varepsilon''^{j^2}}) = O(e^{k\ell^3}k\varepsilon^{-O(1)})\) constructed using Theorem 2.6.1, where \(k_j = \lceil k/\ell \rceil\) for every \(j \leq k \mod \ell\) and \(k_j = \lceil k/\ell \rceil\) otherwise. Using Lemma 2.5.2 for composing \(H\) and \(\{B_j\}_{j=1}^\ell\), we get a \((4\varepsilon', k)\)-balanced family \(D'\) of hash functions from \([q]\) to \([k]\).

Now let \(D''\) be an \((\varepsilon', \ell)\)-balanced family of hash functions from \([n]\) to \([q]\) of size \(k^{O(1)}\varepsilon'^{-O(1)} \log n\) constructed using Theorem 2.6.2. Using Lemma 2.5.1 for composing \(D''\) and \(D'\), we get an \((\varepsilon, k)\)-balanced family of hash functions from \([n]\) to \([k]\), as needed. Note that for calculating the size of each \(B_j\), we use the fact that \(\varepsilon'' = \varepsilon/(6\ell)\). The time needed to construct each \(B_j\) is

\[
\left( \frac{q}{k_j} \right)^{\frac{k_j}{\ell^2} O(1)} q \log q = \frac{e^{k}}{\varepsilon'^{O(1)}}.
\]
where the last equality follows from the fact that \( \binom{n}{k} \leq \left( \frac{en}{k} \right)^k \).

Using Color-Coding we can now approximate the number of paths and cycles (or other fixed graphs of bounded tree-width) in a given input graph. Let \( G = (V, E) \) be a directed or undirected graph. The algorithms use the construction of \((\varepsilon, k)\)-balanced families of hash functions from \( V \) to \([k]\). Each such function defines a coloring of the vertices of the graph. Recall that a path is colorful if each vertex on it is colored by a distinct color. Using dynamic programming one can count efficiently the exact number of colorful paths in each of these colorings. The properties of the balanced family of hash functions then provide the following deterministic polynomial time algorithms for approximately counting the number of paths or cycles of size \( k \) in a given input graph of size \( n \) for all \( k \leq \log n \). Similar results apply for approximate counting of prescribed subgraphs of size \( k \) and bounded tree-width.

**Theorem 2.10.4.** For any \( \frac{1}{\text{poly}(k)} < \varepsilon \leq 1 \), the number of simple (directed or undirected) paths of \( k \) vertices in a (directed or undirected) graph \( G = (V, E) \) can be approximated deterministically up to relative error \( \varepsilon \) in time \( 2^{O(k)}|E|\log|V| \).

**Theorem 2.10.5.** For any \( \frac{1}{\text{poly}(k)} < \varepsilon \leq 1 \), the number of simple (directed or undirected) cycles of size \( k \) in a (directed or undirected) graph \( G = (V, E) \) can be approximated deterministically up to relative error \( \varepsilon \) in time \( 2^{O(k)}|E||V|\log|V| \).

## 2.11 Concluding Remarks

- The notion of balanced families of hash functions seems natural and useful, and it will be interesting to find additional applications of it.

- An easy combination of Theorem 2.6.2 and Theorem 2.10.2 supplies, for any \( \varepsilon \geq \frac{1}{k^2} \), explicit \( \varepsilon \)-balanced \((n, k, \ell)\)-splitters of size at most \( e^{O(\ell^2 \log^2 k)} \log n \). In particular, for \( \ell = 2 \) the size is \( e^{O(\log^2 k)} \log n \). A simple probabilistic argument (see Theorem 2.4.5) shows, however,
that for any fixed $\varepsilon > 0$ there are $\varepsilon$-balanced $(n, k, 2)$-splitters of size $O(k\sqrt{k}\log n)$, and although this is not crucial for our application here, it will be interesting to find an explicit construction of such splitters of size polynomial in $k$ and $\log n$.

- Our results settle the problem of approximately counting the number of paths and cycles of length $k = \Theta(\log n)$ in an $n$-vertex graph in deterministic polynomial time. As mentioned in the introduction, it is probably impossible to extend the result for larger values of $k$, since even a polynomial time algorithm for deciding whether there exists one simple path of length $k$ where $\log n = o(k)$ would imply a subexponential time algorithm for the Hamiltonian cycle problem. This follows easily by padding a graph on $k$ vertices by $n - k = 2^{o(k)}$ isolated ones, thus converting the above decision algorithm to one that decides in time $2^{o(k)}$ whether a graph on $k$ vertices is Hamiltonian, contradicting the Exponential Time Hypothesis (ETH) [65, 66].

- Our method here, combined with the Color Coding technique, easily yields results for additional approximate counting problems for graphs. In particular, given a weighted graph $G$ on $n$ vertices, we can approximate deterministically, in polynomial time, the number of minimum (or maximum) weight paths or cycles (or copies of any prescribed subgraph of bounded tree width) on $k$ vertices in $G$ up to any fixed desired relative accuracy, for all $k \leq O(\log n)$.

- In the definition of a balanced family of hash functions from $[n]$ to $[k]$, there is some constant $T > 0$, such that for every $S \subset [n]$, $|S| = k$, the number of functions that are one-to-one on $S$ is close to $T$. We note that the value of $T$ need not be equal to the expected number of one-to-one functions on a set of size $k$, for the case that the functions were chosen independently according to a uniform distribution. For example, the value of $T$ in the construction of Theorem 2.10.3 is not even asymptotically equal to what one would expect in a uniform distribution. This is due to the fact that balanced splitters can be composed in different ways and our main construction, described in Theorem 2.10.3, is achieved by composing three types of splitters.
The exact value of $T$ can be easily calculated in all of our explicit constructions. In all these constructions, the value $T$ associated with a $\delta$-balanced $(n, k, \ell)$-splitter depends only on the parameters $\delta$, $n$, $k$, and $\ell$. Thus, there is no need to calculate $T$ from the explicit family of functions.
Chapter 3

The Dominating Set Problem on Graphs with an Excluded Minor

The results of this chapter appear in [17, 18]

The domination number of a graph $G = (V, E)$ is the minimum size of a dominating set $U \subset V$, which satisfies that every vertex in $V \setminus U$ is adjacent to at least one vertex in $U$. There is substantial literature dealing with fixed parameter algorithms for the dominating set problem on various families of graphs. In this chapter, we give a $k^{O(dk)}n$ time algorithm for finding a dominating set of size at most $k$ in a $d$-degenerated graph with $n$ vertices. This proves that the dominating set problem is fixed-parameter tractable for degenerated graphs. For graphs that do not contain $K_h$ as a topological minor, we give an improved algorithm for the problem with running time $(O(h))^h n$. For graphs which are $K_h$-minor-free, the running time is further reduced to $(O(\log h))^{h^2/2} n$. Fixed-parameter tractable algorithms that are linear in the number of vertices of the graph were previously known only for planar graphs.

The notion of a problem kernel refers to a polynomial time algorithm that achieves some provable reduction of the input size. Given a graph $G$ whose domination number is $k$, the objective is to design a polynomial time algorithm that produces a graph $G'$ whose size depends only on $k$, and also
has domination number equal to $k$. Note that the graph $G'$ is constructed without knowing the value of $k$. Problem kernels can be used to obtain efficient approximation and exact algorithms for the domination number, and are also useful in practical settings.

In this chapter, we present the first nontrivial result for the general case of graphs with an excluded minor, as follows. For every fixed $h$, given a graph $G$ with $n$ vertices that does not contain $K_{h}$ as a topological minor, our $O(n^{3.5} + k^{O(1)})$ time algorithm constructs a subgraph $G'$ of $G$, such that if the domination number of $G$ is $k$, then the domination number of $G'$ is also $k$ and $G'$ has at most $k^{c}$ vertices, where $c$ is a constant that depends only on $h$. This result is improved for graphs that do not contain $K_{3,h}$ as a topological minor, using a simpler algorithm that constructs a subgraph with at most $ck$ vertices, where $c$ is a constant that depends only on $h$.

Our results imply that there is a problem kernel of polynomial size for graphs with an excluded minor and a linear kernel for graphs that are $K_{3,h}$-minor-free. The only previous kernel results known for the dominating set problem are the existence of a linear kernel for the planar case as well as for graphs of bounded genus. Using the polynomial kernel construction, we give an $O(n^{3.5} + 2^{O(\sqrt{k})})$ time algorithm for finding a dominating set of size at most $k$ in an $H$-minor-free graph with $n$ vertices. This improves the running time of the previously best known algorithm.

For the families of graphs discussed above, the problem of finding an induced cycle of a given length is also addressed. For every fixed $H$ and $k$, we show that if an $H$-minor-free graph $G$ with $n$ vertices contains an induced cycle of size $k$, then such a cycle can be found in $O(n)$ expected time as well as in $O(n \log n)$ worst-case time. Some results are stated concerning the (im)possibility of establishing linear time algorithms for the more general family of degenerated graphs.

### 3.1 Introduction

The input to a parameterized problem is a pair $(x, k)$, where $x$ is the problem instance, $k$ is the parameter, and $n := |(x, k)|$ denotes the input size. A parameterized problem is fixed-parameter tractable if it can be solved in
time $f(k) \cdot n^c$, for a computable function $f : \mathbb{N} \rightarrow \mathbb{N}$ and a constant $c$. A kernelization is a polynomial time computable function that given input $(x, k)$ constructs an equivalent input $(x', k')$, such that $k' \leq k$ and $|x'| \leq g(k)$ for a computable function $g : \mathbb{N} \rightarrow \mathbb{N}$. The image $x'$ is called the problem kernel of $x$. In this chapter, the notion of a kernel for the dominating set problem refers to a polynomial time algorithm that given a graph $G$ whose domination number is $k$, constructs a graph $G'$ whose size depends only on $k$, and also has domination number equal to $k$.

It is easy and known that a parameterized problem is kernelizable if and only if it is fixed-parameter tractable. Thus, a fixed-parameter algorithm for the dominating set problem gives a trivial kernel whose size is some function of $k$, not necessarily a polynomial. Problem kernels can be used to obtain efficient approximation and exact algorithms for the domination number, and are also useful in practical settings.

This chapter deals with algorithms for degenerated graphs. The degeneracy $d(G)$ of an undirected graph $G = (V, E)$ is the smallest number $d$ for which there exists an acyclic orientation of $G$ in which all the outdegrees are at most $d$. Many interesting families of graphs are degenerated (have bounded degeneracy). For example, graphs embeddable on some fixed surface, degree-bounded graphs, graphs of bounded tree-width, and non-trivial minor-closed families of graphs.

There is an extensive literature dealing with fixed-parameter algorithms for the dominating set problem on various families of graphs. Our first result is a linear time algorithm for finding a dominating set of fixed size in degenerated graphs. This is the most general class of graphs for which fixed-parameter tractability for this problem has been established. To the best of our knowledge, linear time algorithms for the dominating set problem were previously known only for planar graphs. Our algorithms both generalize and simplify the classical bounded search tree algorithms for this problem (see, e.g., [5, 18]).

Our second result is a polynomial problem kernel for the case of graphs with an excluded minor. This is the most general class of graphs for which a polynomial problem kernel has been established. To the best of our knowledge, the only previous results are a linear kernel for the planar case as well
The problem of finding induced cycles in degenerated graphs has been studied by Cai, Chan and Chan [41]. Our third result in this chapter is a randomized algorithm for finding an induced cycle of fixed size in graphs with an excluded minor. The algorithm’s expected running time is linear, and its derandomization is done in an efficient way, answering an open question from [41]. The problem of finding induced cycles in degenerated graphs is also addressed.

Fixed-Parameter Algorithms for the Dominating Set Problem. The dominating set problem on general graphs is known to be \( W[2]\)-complete [47]. This means that most likely there is no \( f(k) \cdot n^c \)-algorithm for finding a dominating set of size at most \( k \) in a graph of size \( n \) for any computable function \( f : \mathbb{N} \to \mathbb{N} \) and constant \( c \). This suggests the exploration of specific families of graphs for which this problem is fixed-parameter tractable.

The method of bounded search trees has been used to give an \( O(8^k n) \) time algorithm for the dominating set problem in planar graphs [5] and an \( O((4g + 40)^k n^2) \) time algorithm for the problem in graphs of bounded genus \( g \geq 1 \) [48]. The algorithms for planar graph were improved to \( O(4^{\sqrt[3]{34k}} n) \) [3], then to \( O(2^{27\sqrt{k}} n) \) [71], and finally to \( O(2^{15\sqrt{k}} k + n^3 + k^4) \) [53]. Fixed-parameter algorithms are now known also for map graphs [44] and for constant powers of \( H \)-minor-free graphs [45]. The running time given in [45] for finding a dominating set of size \( k \) in an \( H \)-minor-free graph \( G \) with \( n \) vertices is \( 2^{O(\sqrt{k})} n^c \), where \( c \) is a constant depending only on \( H \). To summarize these results, fixed-parameter tractable algorithms for the dominating set problem were known for fixed powers of \( H \)-minor-free graphs and for map graphs. Linear time algorithms were established only for planar graphs.

Kernels for the Dominating Set Problem. The reduction rules introduced by Alber, Fellows, and Niedermeier were the first to establish a linear problem kernel for planar graphs [6]. The kernel obtained was of size \( 335k \), where \( k \) is the domination number of the graph. Fomin and Thilikos proved that the same rules of Alber et al. provide a linear kernel of size \( O(k + g) \) for graphs of genus \( g \) [54]. Chen et al. improved the upper bound for the planar case to \( 67k \) [42]. They also gave the first lower bound, by
proving that for any \( \varepsilon > 0 \), there is no \( (2 - \varepsilon)k \) kernel for the planar dominating set problem, unless \( P = NP \). It is interesting to note that Alber, Dorn, and Niedermeier introduced a reduction rule that explores the joint neighborhood of \( l \) distinct vertices [4], but this general rule has been applied only for \( l = 1 \) and \( l = 2 \), in order to prove that the directed dominating set problem on planar graphs has a linear size kernel. Their reduction rule generates a constraint, which is encoded by a corresponding gadget in the graph. Thus, the kernel constructed is not necessarily a subgraph of the input graph.

**Finding Paths and Cycles.** The foundations for the algorithms for finding cycles, presented in this chapter, have been laid in [22], where the authors introduce the color-coding technique. Two main randomized algorithms are presented there, as follows. A simple directed or undirected path of length \( k - 1 \) in a graph \( G = (V, E) \) that contains such a path can be found in \( 2^{O(k)} |E| \) expected time in the directed case and in \( 2^{O(k)} |V| \) expected time in the undirected case. A simple directed or undirected cycle of size \( k \) in a graph \( G = (V, E) \) that contains such a cycle can be found in either \( 2^{O(k)} |V| |E| \) or \( 2^{O(k)} |V|^\omega \) expected time, where \( \omega < 2.376 \) is the exponent of matrix multiplication. These algorithms can be derandomized at a cost of an extra \( \log |V| \) factor. As for the case of even cycles, it is shown in [87] that for every fixed \( k \geq 2 \), there is an \( O(|V|^2) \) algorithm for finding a simple cycle of size \( 2k \) in an undirected graph (that contains such a cycle). Improved algorithms for detecting given length cycles have been presented in [23] and [88]. The authors of [23] describe fast algorithms for finding short cycles in \( d \)-degenerated graphs. In particular, \( C_3 \)'s and \( C_4 \)'s can be found in \( O(|E| \cdot d(G)) \) time and \( C_5 \)'s in \( O(|E| \cdot d(G)^2) \) time.

**Finding Induced Paths and Cycles.** Cai, Chan and Chan have recently introduced a new interesting technique they call random separation for solving fixed-cardinality optimization problems on graphs [41]. They combine this technique together with color-coding to give the following algorithms for finding an induced graph within a large graph. For fixed constants \( k \) and \( d \), if a \( d \)-degenerated graph \( G \) with \( n \) vertices contains some fixed induced tree \( T \) on \( k \) vertices, then it can be found in \( O(n) \) expected time and \( O(n \log^2 n) \) worst-case time. If such a graph \( G \) contains an induced \( k \)-cycle, then it can be
found in $O(n^2)$ expected time and $O(n^2 \log^2 n)$ worst-case time. Two open problems are raised by the authors of the paper. First, they ask whether the $\log^2 n$ factor incurred in the derandomization can be reduced to $\log n$. A second question is whether there is an $O(n)$ expected time algorithm for finding an induced $k$-cycle in a $d$-degenerated graph with $n$ vertices. In this chapter, we show that when combining the techniques of random separation and color-coding, an improved derandomization with a loss of only $\log n$ is indeed possible. An $O(n)$ expected time algorithm finding an induced $k$-cycle in graphs with an excluded minor is presented. We give evidence that establishing such an algorithm even for 2-degenerated graphs has far-reaching consequences.

Our Results. The main result of the chapter is that the dominating set problem is fixed-parameter tractable for degenerated graphs. The running time is $k^{O(dk)} n$ for finding a dominating set of size $k$ in a $d$-degenerated graph with $n$ vertices. The algorithm is linear in the number of vertices of the graph, and we further improve the dependence on $k$ for the following specific families of degenerated graphs. For graphs that do not contain $K_h$ as a topological minor, an improved algorithm for the problem with running time $(O(h))^{hk} n$ is established. For graphs which are $K_h$-minor-free, the running time obtained is $(O(\log h))^{hk/2} n$. We show that all the algorithms can be generalized to the weighted case in the following sense. A dominating set of size at most $k$ having minimum weight can be found within the same time bounds.

By introducing a novel reduction rule, we prove that the dominating set problem on graphs with an excluded minor admits a polynomial problem kernel. This gives an $O(n^{3.5} + 2^{O(\sqrt{k})})$ time algorithm for finding a dominating set of size at most $k$ in an $H$-minor-free graph with $n$ vertices. For graphs that are $K_{3,h}$-minor-free, the reduction rules of Alber, Fellows, and Niedermeier [6] are shown to give a linear problem kernel. All the reduction rules described in this chapter have the property that the only modifications made to an input graph are the removal of vertices and edges. This implies that the graph obtained, as a result of applying the rules, is a subgraph of the input graph. The advantages of this approach are its simplicity and the fact that it preserves monotone properties, like planarity, being $H$-minor-free, and degeneracy. We show that the rules of Alber et al. can also be described in
such a way.

We address two open questions raised by Cai, Chan and Chan in [41] concerning linear time algorithms for finding an induced cycle in degenerated graphs. An $O(n)$ expected time algorithm for finding an induced $k$-cycle in graphs with an excluded minor is presented. The derandomization performed in [41] is improved and we get a deterministic $O(n \log n)$ time algorithm for the problem. As for finding induced cycles in degenerated graphs, we show a deterministic $O(n)$ time algorithm for finding cycles of size at most 5, and also explain why this is unlikely to be possible to achieve for longer cycles.

**Techniques.** We generalize the known search tree algorithms for the dominating set problem. This is enabled by proving some combinatorial lemmas, which are interesting in their own right. For degenerated graphs, we bound the number of vertices that dominate many elements of a given set, whereas for graphs with an excluded minor, our interest is in vertices that still need to be dominated and have a small degree.

Our new reduction rule uses a succinct representation of all subsets of some bounded size that dominate a given set of vertices. Interestingly, this is done by applying a fixed-parameter algorithm for finding dominating sets in degenerated graphs. A challenging part of the combinatorial proofs is to show that given a graph with an excluded minor and a dominating set $D$ of size $k$, there exists a subset of vertices $U$ whose size is linear in $k$, such that all vertices not in $D \cup U$ belong to the "inner neighborhood" of a constant number of vertices from $D \cup U$.

The algorithm for finding an induced cycle in non-trivial minor-closed families is based on random separation and color-coding. Its derandomization is performed using known explicit constructions of families of (generalized) perfect hash functions.

### 3.2 Preliminaries

This chapter deals with undirected and simple graphs, unless stated otherwise. Generally speaking, we will follow the notation used in [39] and [46]. For an undirected graph $G = (V, E)$ and a vertex $v \in V$, $N(v)$ denotes the set of all vertices adjacent to $v$ (not including $v$ itself), whereas $N[v]$ de-
notes \( N(v) \cup \{v\} \). This is generalized to the neighborhood of arbitrary sets by defining \( N(A) := \left( \bigcup_{v \in A} N(v) \right) \setminus A \) and \( N[A] := \bigcup_{v \in A} N[v] \). The graph obtained from \( G \) by deleting a vertex \( v \) is denoted \( G - v \). The subgraph of \( G \) induced by some set \( V' \subset V \) is denoted by \( G[V'] \).

A dominating set of a graph \( G = (V, E) \) is a subset of vertices \( U \subset V \), such that every vertex in \( V \setminus U \) is adjacent to at least one vertex in \( U \). The domination number of a graph \( G \), denoted by \( \gamma(G) \), is the minimum size of a dominating set. We say that vertex \( v \) dominates the vertices of \( N(v) \cup \{v\} \).

For a set of vertices \( A \), if \( U \subset N[A] \), then we say that \( A \) dominates \( U \).

A graph \( G \) is \( d \)-degenerated if every induced subgraph of \( G \) has a vertex of degree at most \( d \). It is easy and known that every \( d \)-degenerated graph \( G = (V, E) \) admits an acyclic orientation such that the outdegree of each vertex is at most \( d \). Such an orientation can be found in \( O(|E|) \) time. A \( d \)-degenerated graph with \( n \) vertices has less than \( dn \) edges and therefore its average degree is less than \( 2d \).

For a directed graph \( D = (V, A) \) and a vertex \( v \in V \), the set of out-neighbors of \( v \) is denoted by \( N^+(v) \). For a set \( V' \subset V \), the notation \( N^+(V') \) stands for the set of all vertices that are out-neighbors of at least one vertex of \( V' \). For a directed graph \( D = (V, A) \) and a vertex \( v \in V \), we define \( N^+_i(v) = N^+(v) \) and \( N^+_i(v) = N^+(N^+_i(v)) \) for \( i \geq 2 \).

An edge is said to be subdivided when it is deleted and replaced by a path of length two connecting its ends, the internal vertex of this path being a new vertex. A subdivision of a graph \( G \) is a graph that can be obtained from \( G \) by a sequence of edge subdivisions. If a subdivision of a graph \( H \) is the subgraph of another graph \( G \), then \( H \) is a topological minor of \( G \). A graph \( H \) is called a minor of a graph \( G \) if it can be obtained from a subgraph of \( G \) by a series of edge contractions.

In the parameterized dominating set problem, we are given an undirected graph \( G = (V, E) \), a parameter \( k \), and need to find a set of at most \( k \) vertices that dominate all the other vertices. Following the terminology of [3], the following generalization of the problem is considered. The input is a black and white graph, which simply means that the vertex set \( V \) of the graph \( G \) has been partitioned into two disjoint sets \( B \) and \( W \) of black and white vertices, respectively, i.e., \( V = B \uplus W \), where \( \uplus \) denotes disjoint set union.
Given a black and white graph $G = (B \sqcup W, E)$ and an integer $k$, the problem is to find a set of at most $k$ vertices that dominate the black vertices. More formally, we ask whether there is a subset $U \subset B \sqcup W$, such that $|U| \leq k$ and every vertex $v \in B - U$ satisfies $N(v) \cap U \neq \emptyset$. Finally, we give a new definition, specific to this thesis, for what it means to be a reduced black and white graph.

Definition 3.2.1. A black and white graph $G = (B \sqcup W, E)$ is called reduced if it satisfies the following conditions:

- $W$ is an independent set.
- All the vertices of $W$ have degree at least 2.
- $N(w_1) \neq N(w_2)$ for every two distinct vertices $w_1, w_2 \in W$.

In this chapter, we consider only simple paths, that is, paths of the form $x_0 - x_1 - \cdots - x_k$, where the $x_i$ are all distinct. The vertices $x_1, \ldots, x_{k-1}$ are the inner vertices of the path. The number of edges of a path is its length. Suppose that $G = (V, E)$ is a graph, $U \subset V$, and $r$ and $l$ are two integers. We denote by $\hat{U}_{r,l}$ the set of all vertices $v \in V \setminus U$ for which there are $r$ vertex disjoint paths of length at most $l$ from $v$ to $r$ different vertices of $U$. To avoid confusion, we stress the fact that $v$ is the starting vertex of all the paths, but any other vertex belongs to at most one of the paths. The vertices of $\hat{U}_{r,l}$ are called central vertices, and when the values of $r$ and $l$ are clear from the context, the simpler notation $\hat{U}$ will be used.

### 3.3 Algorithms for the Dominating Set Problem

#### 3.3.1 Degenerated Graphs

The algorithm for degenerated graphs is based on the following combinatorial lemma.
Lemma 3.3.1. Let $G = (B \uplus W, E)$ be a $d$-degenerated black and white graph. If $|B| > (4d + 2)k$, then there are at most $(4d + 2)k$ vertices in $G$ that dominate at least $|B|/k$ vertices of $B$.

Proof. Denote $R = \{v \in B \cup W \mid |(N_G(v) \cup \{v\}) \cap B| \geq |B|/k\}$. By contradiction, assume that $|R| > (4d + 2)k$. The induced subgraph $G[R \cup B]$ has at most $|R| + |B|$ vertices and at least $|R|/2 \cdot (|B|/k - 1)$ edges. The average degree of $G[R \cup B]$ is thus at least

$$\frac{|R|(|B| - k)}{k(|R| + |B|)} \geq \frac{\min\{|R|, |B|\}}{2k} - 1 > 2d.$$ 

This contradicts the fact that $G[R \cup B]$ is $d$-degenerated. \hfill \blacksquare

Theorem 3.3.2. There is a $k^{O(dk)}n$ time algorithm for finding a dominating set of size at most $k$ in a $d$-degenerated black and white graph with $n$ vertices that contains such a set.

Proof. The pseudocode of algorithm $DominatingSetDegenerated(G, k)$ that solves this problem appears below. If there is indeed a dominating set of size at most $k$, then this means that we can split $B$ into $k$ disjoint pieces (some of them can be empty), so that each piece has a vertex that dominates it. If $|B| \leq (4d + 2)k$, then there are at most $k^{(4d+2)k}$ ways to divide the set $B$ into $k$ disjoint pieces. For each such split, we can check in $O(kdn)$ time whether every piece is dominated by a vertex. If $|B| > (4d + 2)k$, then it follows from Lemma 3.3.1 that $|R| \leq (4d + 2)k$. This means that the search tree can grow to be of size at most $(4d + 2)^k k!$ before possibly reaching the previous case. This gives the needed time bound. \hfill \blacksquare

3.3.2 Graphs with an Excluded Minor

Graphs with either an excluded minor or with no topological minor are known to be degenerated. We will apply the following useful propositions.

Proposition 3.3.3. \cite{38, 72} There exists a constant $c$ such that, for every $h$, every graph that does not contain $K_h$ as a topological minor is $ch^2$-degenerated.
Algorithm 1: \textit{DominatingSetDegenerated}(G, k)

\textbf{Input:} Black and white $d$-degenerated graph $G = (B \cup W, E)$, integers $k, d$

\textbf{Output:} A set dominating all vertices of $B$ of size at most $k$ or \textit{NONE} if no such set exists

\begin{itemize}
  \item [if] $B = \emptyset$ then
    \hspace{1em} return $\emptyset$
  \item [else if] $k = 0$ then
    \hspace{1em} return \textit{NONE}
  \item [else if] $|B| \leq (4d + 2)k$ then
    \hspace{1em} forall possible ways of splitting $B$ into $k$ (possibly empty) disjoint pieces $B_1, \ldots, B_k$ do
    \hspace{2em} if each piece $B_i$ has a vertex $v_i$ that dominates it then
    \hspace{3em} return \{ $v_1, \ldots, v_k$ \}
    \hspace{1em} return \textit{NONE}
  \item [else]
    \hspace{1em} $R \leftarrow \{ v \in B \cup W \mid |(N_G(v) \cup \{v\}) \cap B| \geq |B|/k \}$
    \hspace{1em} forall $v \in R$ do
    \hspace{2em} Create a new graph $G'$ from $G$ by marking all the elements of $N_G(v)$ as white and removing $v$ from the graph
    \hspace{2em} $D \leftarrow \textit{DominatingSetDegenerated}(G', k - 1)$
    \hspace{2em} if $D \neq \textit{NONE}$ then
    \hspace{3em} return $D \cup \{v\}$
    \hspace{1em} return \textit{NONE}
\end{itemize}
Proposition 3.3.4. \([\text{[75]} \text{[84]} \text{[85]}]\) There exists a constant \(c\) such that, for every \(h\), every graph with no \(K_h\) minor is \(ch\sqrt{\log h}\)-degenerated.

The following lemma gives an upper bound on the number of cliques of a prescribed fixed size in a degenerated graph.

Lemma 3.3.5. If a graph \(G\) with \(n\) vertices is \(d\)-degenerated, then for every \(k \geq 1\), \(G\) contains at most \(\binom{d}{k-1}n\) copies of \(K_k\).

Proof. By induction on \(n\). For \(n = 1\) this is obviously true. In the general case, let \(v\) be a vertex of degree at most \(d\). The number of copies of \(K_k\) that contain \(v\) is at most \(\binom{d}{k-1}\). By the induction hypothesis, the number of copies of \(K_k\) in \(G - v\) is at most \(\binom{d}{k-1}(n - 1)\). ■

We can now prove our main combinatorial results.

Theorem 3.3.6. Let \(s\) be the constant from Proposition 3.3.3. For every reduced black and white graph \(G = (B \sqcup W, E)\), if \(G\) does not contain \(K_h\) as a topological minor, then there exists a vertex \(b \in B\) of degree at most \((3sh)^h\).

Proof. Denote \(|B| = n > 0\) and \(d = sh^2\) where \(s\) is the constant from Proposition 3.3.3. Consider the vertices of \(W\) in some arbitrary order. For each vertex \(w \in W\), if there exist two vertices \(b_1, b_2 \in N(w)\), such that \(b_1\) and \(b_2\) are not connected, add the edge \(\{b_1, b_2\}\) and remove the vertex \(w\) from the graph. Denote the resulting graph \(G' = (B \sqcup W', E')\). Obviously, \(G'[B]\) does not contain \(K_h\) as a topological minor and therefore has at most \(dn\) edges. The number of edges in the induced subgraph \(G'[B]\) is at least the number of white vertices that were deleted from the graph, which means that at most \(dn\) were deleted so far.

We now bound \(|W'|\), the number of white vertices in \(G'\). It follows from the definition of a reduced black and white graph that there are no white vertices in \(G'\) of degree smaller than 2. The graph \(G'\) cannot contain a white vertex of degree \(h - 1\) or more, since this would mean that the original graph \(G\) contained a subdivision of \(K_h\). Now let \(w\) be a white vertex of \(G'\) of degree \(k\), where \(2 \leq k \leq h - 2\). The reason why \(w\) was not deleted during the process of generating \(G'\) is because \(N(w)\) is a clique of size \(k\) in \(G'[B]\). The graph \(G'\) is a reduced black and white graph, and
N(w_1) \neq N(w_2) for every two different white vertices w_1 and w_2. This means that the neighbors of each white vertex induce a different clique in G'[B]. By applying Lemma 3.3.5 to G'[B], we get that the number of white vertices of degree k in G' is at most \((\frac{d}{k-1}) n\). This means that \(|W'| \leq \left( \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{d} \right) n\). We know that \(|W| \leq |W'| + dn\) and therefore \(|E| \leq d(|B| + |W|) \leq d \left[ 3d + \binom{d}{2} + \cdots + \binom{d}{h-3} \right] n\). Obviously, there exists a black vertex of degree at most \(2|E|/n\). The result now follows by plugging the value of \(d\) and using the fact that \(\binom{n}{k} \leq \left(\frac{en}{k}\right)^k\).

\[ \square \]

**Theorem 3.3.7.** There exists a constant \(c > 0\), such that for every reduced black and white graph \(G = (B \uplus W, E)\), if \(G\) is \(K_h\)-minor-free, then there exists a vertex \(b \in B\) of degree at most \((c \log h)^{h/2}\).

**Proof.** We proceed as in the proof of Theorem 3.3.6 using Proposition 3.3.4 instead of Proposition 3.3.3.

**Theorem 3.3.8.** There is an \((O(h))^{hk} n\) time algorithm for finding a dominating set of size at most \(k\) in a black and white graph with \(n\) vertices and no \(K_h\) as a topological minor.

**Proof.** The pseudocode of algorithm DominatingSetNoMinor\((G, k)\) that solves this problem appears below. Let the input be a black and white graph \(G = (B \uplus W, E)\). It is important to notice that the algorithm removes vertices and edges in order to get a (nearly) reduced black and white graph. Edges between white vertices as well as white vertices of degree at most 1 can be removed in time \(O(|E|)\). Now we can eliminate all duplicates, that is, ensure that \(N(w_1) \neq N(w_2)\) for every two different vertices \(w_1, w_2 \in W\). It follows from the proof of Theorem 3.3.6 that it is enough to do so for vertices with at most \(h - 2\) neighbors.

Duplicates can be removed as follows. Given the adjacency-list representation of the graph, it is easy to create in time \(O(|E|)\) a new representation in which all the adjacency lists are sorted. This is done by starting from empty lists and by scanning all vertices in order. For each vertex \(i\) in its turn, one updates all lists of its neighbors by appending \(i\) to their ends. We can now perform radix sorting on all adjacency lists of size at most \(h - 2\). Two vertices with the same adjacency list will appear consecutively in this
sorted order. Radix sorting and duplicates removal takes time $O(hn)$. The time bound for the algorithm now follows from Theorem 3.3.6. ■

**Algorithm 2:** DominatingSetNoMinor($G, k$)

**Input:** Black and white ($K_h$-minor-free) graph $G = (B \oplus W, E)$, integer $k$

**Output:** A set dominating all vertices of $B$ of size at most $k$ or $\text{NONE}$ if no such set exists

if $B = \emptyset$ then
  \text{return} \emptyset
else if $k = 0$ then
  \text{return} \text{NONE}
else
  Remove all edges of $G$ whose two endpoints are in $W$
  Remove all white vertices of $G$ of degree 0 or 1
  As long as there are two different vertices $w_1, w_2 \in W$ with $N(w_1) = N(w_2)$, $|N(w_1)| < h - 1$, remove one of them from the graph
  Let $b \in B$ be a vertex of minimum degree among all vertices in $B$
  forall $v \in N_G(b) \cup \{b\}$ do
    Create a new graph $G'$ from $G$ by marking all the elements of $N_G(v)$ as white and removing $v$ from the graph
    $D \leftarrow \text{DominatingSetNoMinor}(G', k - 1)$
    if $D \neq \text{NONE}$ then
      \text{return} $D \cup \{v\}$
  \text{return} \text{NONE}

**Theorem 3.3.9.** There is an $(O(\log h))^{h^{k/2} n}$ time algorithm for finding a dominating set of size at most $k$ in a black and white graph with $n$ vertices which is $K_h$-minor-free.

**Proof.** The proof is analogues to that of Theorem 3.3.8 using Theorem 3.3.7 instead of Theorem 3.3.6. ■
3.3.3 The Weighted Case

In the weighted dominating set problem, each vertex of the graph has some positive real weight. The goal is to find a dominating set of size at most \( k \), such that the sum of the weights of all the vertices of the dominating set is as small as possible. The algorithms we presented can be generalized to deal with the weighted case without changing the time bounds. In this case, the whole search tree needs to be scanned and one cannot settle for the first valid solution found.

Let \( G = (B \uplus W, E) \) be the input graph to the algorithm. In Algorithm 1 for degenerated graphs, we need to address the case where \( |B| \leq (4d+2)k \). In this case, the algorithm scans all possible ways of splitting \( B \) into \( k \) disjoint pieces \( B_1, \ldots, B_k \), and it has to be modified, so that it will always choose a vertex with minimum weight that dominates each piece. In Algorithm 2 for graphs with an excluded minor, the criterion for removing white vertices from the graph is modified so that whenever two vertices \( w_1, w_2 \in W \) satisfy \( N(w_1) = N(w_2) \), the vertex with the bigger weight is removed.

3.4 Dominating Sets in Degenerated Graphs

Graphs with either an excluded minor or with no topological minor are known to be degenerated. Some of our kernel results for graphs with no topological \( K_h \) use the constant from Proposition 3.3.3. The results can be improved for graphs that are \( K_h \)-minor-free using Proposition 3.3.4.

A major part of Rule 2, described in section 3.6, involves getting a succinct representation of all sets of some bounded size that dominate a specific set of vertices in a degenerated graph. This useful representation is achieved by applying the \( k^{O(dk)}n \) time algorithm from Theorem 3.3.2 for finding a dominating set of size at most \( k \) in a \( d \)-degenerated graph with \( n \) vertices. The combinatorial Lemma 3.3.1 gives the following useful characterization of dominating sets in degenerated graphs.

**Theorem 3.4.1.** Suppose that \( G = (V, E) \) is a \( d \)-degenerated graph with \( n \) vertices, \( B \subset V \), and \( k \geq 1 \). There is an \( k^{O(dk)}n \) time algorithm for finding a family \( \mathcal{F} \) of \( t \leq (4d+2)^k k! \) pairs \( (D_i, B_i) \) of subsets of \( V \), such that
The Dominating Set Problem on Graphs with an Excluded Minor

$|D_i| \leq k$ and $|B_i| \leq (4d + 2)k$ for every $1 \leq i \leq t$, for which the following holds. If $D \subset V$ is a subset of size at most $k$ that dominates $B$, then some $i, 1 \leq i \leq t$, satisfies that $D_i \subset D$ and $B_i = B \setminus N[D_i]$.

Proof. The algorithm uses the method of bounded search trees. In each step of the algorithm, $B$ denotes the vertices that still need to be dominated. If $|B| > (4d + 2)k$, then denote by $R$ the set of all vertices that dominate at least $|B|/k$ vertices of $B$. Every set of size at most $k$ that dominates $B$ must contain a vertex from $R$. It follows from Lemma 3.3.1 that $|R| \leq (4d + 2)k$, so we can build our search tree, by creating $|R|$ branches and checking all possible options of adding one of the vertices of $R$ to the dominating set. For each such vertex $v \in R$, we add $v$ to the dominating set, assign $B := B \setminus N(v)$, and remove $v$ from the graph. We continue until $|B| \leq (4d + 2)k$ in all the leaves of the search tree. The search tree can grow to be of size at most $(4d + 2)^k k!$, and each subset $D \subset V$ of size at most $k$ that dominates the original input set $B$ will correspond to one of the leaves of this search tree, as needed.

Though the dominating set problem has a polynomial time approximation scheme when restricted to a class of graphs with an excluded minor \cite{57}, for our purposes, a fast algorithm that achieves a constant approximation is required. The combinatorial Theorem 3.3.6 gives the following constant factor approximation algorithm.

Theorem 3.4.2. Let $s$ be the constant from Proposition 3.3.3. Suppose that the graph $G = (B \uplus W, E)$ does not contain $K_h$ as a topological minor, and there is a set of size $k$ that dominates $B$. There is an $O(nk)$ time algorithm that finds a set of size at most $(3sh)^h k$ that dominates $B$.

Proof. Start with a solution $D := \emptyset$. Given a graph $G = (B \uplus W, E)$, remove all edges whose two endpoints are in $W$ and all vertices of $W$ of degree 0 or 1. As long as there are two different vertices $w_1, w_2 \in W$ with $N(w_1) = N(w_2)$, $|N(w_i)| < h - 1$, remove one of them from the graph. As proved in Theorem 3.3.8 these modifications can be performed in time $O(|E|)$ and they obviously do not affect the minimum size of a set that dominates $B$. It follows from Theorem 3.3.6 that there is a vertex $b \in B$ of degree at most $(3sh)^h$. We
assign $D := D \cup N[b]$, move the vertices of $N(N[b]) \cap B$ from $B$ to $W$, and remove the vertices of $N[b]$ from the graph. The size of the optimal solution decreased by at least one, since every set that dominates $b$ must contain at least one vertex from $N[b]$. We continue as before in the resulting graph, and after at most $k$ steps, the algorithm will compute a dominating set as needed.

### 3.5 Bounds on the Number of Central Vertices

For graphs with no topological $K_h$, the following bound applies.

**Lemma 3.5.1.** Let $s$ be the constant from Proposition 3.3.3. If the graph $G = (V, E)$ does not contain $K_h$ as a topological minor, and $U \subset V$ is of size $k$, then for every $l$, $|\hat{U}_{h-1,l}| \leq (2sh^2l)^{hl}k$.

**Proof.** Denote $d = sh^2$. To bound the size of $\hat{U}$, we initially define the set $B$ to be equal to $U$, and then in a series of $1 + (h - 1)(l - 1)$ phases, vertices will be added to $B$, until eventually $\hat{U} \subset B$. As proved later, after every phase $i$, $1 \leq i \leq 1 + (h - 1)(l - 1)$, the set $B$ will be of size at most $(1 + sh^2(2l - 1))^i k$. This gives the needed bound for $\hat{U}$, by setting $i = 1 + (h - 1)(l - 1)$.

The following is the description of a phase. At the beginning of phase $i$, the set $B$ is of size at most $(1 + sh^2(2l - 1))^{i-1} k$. Consider the vertices of $V \setminus B$ in some arbitrary order. For each vertex $w \notin B$, if there exist two vertex disjoint paths of length at most $l$ from $w$ to two vertices $b_1, b_2 \in B$, such that $b_1$ and $b_2$ are not connected, and all the inner vertices of the two paths are not in $B$, then add the edge $\{b_1, b_2\}$ to $G$ and remove the vertex $w$ from the graph together with the two paths (the vertices $b_1$ and $b_2$ remain in the graph). Denote the resulting graph by $G'$. Obviously, $G'[B]$ does not contain $K_h$ as a topological minor and therefore has at most $d|B| = sh^2|B|$ edges. The number of edges in the induced subgraph $G'[B]$ is at least the number of deleted vertices divided by $(2l - 1)$, which means that at most $sh^2(2l - 1)|B|$ vertices were deleted so far. All the vertices that were removed from the graph during this phase are added to the set $B$, and now we start...
the next phase with the original graph \( G \) and a new set \( B \) of size at most 
\((1 + sh^2(2l - 1))^k\).

Consider a vertex \( v \in \hat{U} \) at the beginning of a phase. There are \( h-1 \) vertex disjoint paths of length at most \( l \) from \( v \) to a set \( H \) of \( h-1 \) different vertices of \( U \). Assume that when \( v \) is considered in the arbitrary order, all the vertices of these \( h-1 \) paths are still in the graph. We claim that the \( h-1 \) vertices of \( H \) cannot all be adjacent to each other, since otherwise they form a topological \( K_h \) together with \( v \). Thus, if \( v \) was not removed from the graph during the phase, then this can only happen in case there exists a vertex \( u \notin B \) on one of the \( h-1 \) vertex disjoint paths, which was removed from the graph before \( v \) was considered. This vertex \( u \) was later added to \( B \) at the end of the phase. There are \( h-1 \) vertex disjoint paths of length at most \( l \) from \( v \) to \( H \), and these paths contain at most \((h-1)(l-1)\) inner vertices. Thus, after at most \( 1 + (h-1)(l-1) \) phases, the vertex \( v \) will be added to \( B \).

Given a graph with no topological \( K_{m,h} \) and a specific set \( M \) of \( m \) vertices, we would like to bound the number of vertices that have \( m \) vertex disjoint paths of total length at most \( l \) to the \( m \) vertices of \( M \).

**Lemma 3.5.2.** Suppose that \( G = (V, E) \) does not contain \( K_{m,h} \) as a topological minor and \( M \subset V \) is a set of \( m > 1 \) vertices. The number of vertices \( v \notin M \) that have \( m \) vertex disjoint paths of total length at most \( l \) to the \( m \) vertices of \( M \) is less than \( h(m(h-1))^{l-m}(l-m+1)! \).

**Proof.** Define \( f(m, h, l) := h(m(h-1))^{l-m}(l-m+1)! \). The lemma is proved by induction on \( l \). For \( l = m \), the lemma bounds the number of vertices that are adjacent to all the vertices of \( M \), and the claim follows from the fact that \( G \) does not have a subgraph isomorphic to \( K_{m,h} \). Suppose that the lemma is true for \( l \geq m \). We prove it for \( l+1 \), as follows. Given a set \( M \) of \( m \) vertices, denote by \( Q \) the set of vertices \( v \notin M \) that have \( m \) vertex disjoint paths of total length at most \( l+1 \) to the \( m \) vertices of \( M \), and denote by \( P_v \) the set of all vertices on these \( m \) paths, except for the vertices of \( M \). Thus, \( |P_v| \leq l + 2 - m \) and \( v \in P_v \).

We first prove that each vertex \( w \notin M \) belongs to at most \( m \cdot f(m, h, l) - m + 1 \) of the sets \( \{P_v\}_{v \in Q} \). Suppose, by contradiction, that there is a vertex
that belongs to more than \( m \cdot f(m, h, l) - m + 1 \) of the sets \( P_v \). Since possibly \( w \in P_w \), there are more than \( m(f(m, h, l) - 1) \) vertices \( v \) for which \( w \in P_v \) and \( w \) is an inner vertex of one of the \( m \) paths from \( v \) to \( M \). Thus, there is some vertex \( u \in M \), such that there are at least \( f(m, h, l) \) vertices \( v \) for which \( w \in P_v \) and \( w \) appears in \( P_v \) on the path from \( v \) to \( u \). This means that there are \( f(m, h, l) \) vertices \( v \) for which there are \( m \) vertex disjoint paths of total length at most \( l \) to the \( m \) vertices \((M \setminus \{u\}) \cup \{w\}\). This contradicts the induction hypothesis.

We showed that every vertex \( w \) belongs to at most \( m \cdot f(m, h, l) - m + 1 \) of the sets \( P_v \). To prove the lemma for \( l + 1 \), assume, by contradiction, that \( |Q| \geq f(m, h, l + 1) \). Assign \( W := Q \). We do the following \( h - 1 \) times, for each \( i, 1 \leq i \leq h - 1 \). Let \( w_i \) be an arbitrary element of \( W \). Note that \( |P_{w_i}| \leq l + 2 - m \). Remove from \( W \) all the vertices \( v \in W \) such that \( P_v \cap P_{w_i} \neq \emptyset \), and continue to choose an arbitrary \( w_{i+1} \in W \). At each step, at most \( (l+2-m)(m \cdot f(m, h, l) - m + 1) \) vertices are removed from \( W \), so after \( h - 1 \) steps, at most \( (h - 1)(l+2-m)(m \cdot f(m, h, l) - m + 1) < f(m, h, l + 1) \) vertices will be removed from \( W \). Thus, the \( h - 1 \) iterations can indeed be performed, and at the end, the set \( W \) is not empty, so we can choose the last arbitrary element \( w_h \in W \). The vertices of \( M \) together with the \( h \) vertices \( w_1, \ldots, w_h \) form a topological \( K_{m,h} \). This is a contradiction and the lemma is proved.

The following result bounds the number of central vertices in graphs with no topological \( K_{m,h} \).

**Lemma 3.5.3.** Let \( s \) be the constant from Proposition 3.3.3. If \( G = (V, E) \) does not contain \( K_{m,h} \) as a topological minor, and \( U \subset V \) is of size \( k \), then for every \( l \), \( |\hat{U}_{m,l}| \leq (smlh)^{2mhl^2}k \).

**Proof.** Denote \( d = s(m + h)^2 \). The proof is similar to that of Lemma 3.5.1 so we highlight only the modifications needed. Initially we set \( B \) to be equal to \( U \). During a phase, if there is still a vertex \( w \notin B \) for which there are two vertex disjoint paths of length at most \( l \) from \( w \) to two vertices \( b_1, b_2 \in B \), such that \( b_1 \) and \( b_2 \) are not connected, and all the inner vertices of the two paths are not in \( B \), then add the edge \( \{b_1, b_2\} \) and remove the vertex \( w \) from the graph together with the two paths. Denote the resulting graph by \( G' \). It
follows from the analysis of Lemma 3.5.1 that at most \( sh^2(2l - 1)|B| \) vertices are removed during the phase. All the removed vertices are later added to \( B \).

In addition to the vertices that were added to \( B \) in the way described, we would like to add more vertices to \( B \), as follows. Consider a vertex \( v \in \hat{U} \) at the beginning of the phase. There are \( m \) vertex disjoint paths of length at most \( l \) from \( v \) to a set \( M \) of \( m \) different vertices. Assume that none of the vertices on these \( m \) paths were removed during the phase. This means that if \( v \) was not removed either, then this can only happen in case \( G'[M] \) is a clique of size \( m \). In this case we also add \( v \) to \( B \). We now count the number of vertices \( v \) of this type. Since \( G' \) does not contain \( K_{m,h} \) as a topological minor, we get from Lemma 3.5.2 that there can be at most \( h(m(h - 1))^{lm - m}(lm - m + 1)! \) vertices \( v \) with \( m \) vertex disjoint paths of length at most \( l \) from \( v \) to the \( m \) vertices of \( M \). It follows from Lemma 3.3.5 that there are at most \( (d_{m-1})|B| \leq (s(m + h)^2)^{m-1}|B| \) cliques of size \( m \) in \( G'[B] \), which means that at most \( (s(m + h)^2)^{m-1}h(m(h - 1))^{lm - m}(lm - m + 1)!|B| \) vertices of \( \hat{U} \) were not accounted for. The total number of vertices that are added to \( B \) during a phase is therefore less than \( (smlh)2ml|B| \), whereas the number of phases is at most \( m(l - 1) + 1 \). This gives the needed bound for \(|\hat{U}|\).}

Itai, Perl, and Shiloach [67] proved that given a graph \( G \) with two distinct vertices \( s \) and \( t \), the problem of deciding whether there exist \( m \) vertex disjoint paths of length at most \( K \) from \( s \) to \( t \) is \( NP \)-complete for \( K \geq 5 \) and polynomially solvable for \( K \leq 4 \). Thus, \( \hat{U}_{r,3} \) can be efficiently computed as follows.

**Lemma 3.5.4.** There is an \( O(|V|^{1.5}|E|) \) time algorithm for computing \( \hat{U}_{r,3} \) for a graph \( G = (V, E) \), a subset \( U \subset V \), and an integer \( r \).

**Proof.** Suppose that \( v \in V \setminus U \), and let \( w \) be a new vertex that is connected to all the vertices of \( U \). By definition, \( v \in \hat{U}_{r,3} \) if and only if there are \( r \) vertex disjoint paths of length at most 4 from \( v \) to \( w \). To determine this, apply the \( O(|V|^{0.5}|E|) \) time algorithm of Itai et al. [67] for finding the maximum number of vertex disjoint paths of length at most 4 from \( v \) to \( w \).
3.6 Polynomial Kernel for Graphs with an Excluded Minor

The reduction rules described in [6] examine the neighborhood of either a single vertex or a pair of vertices. In this section we generalize these definitions to a neighborhood of a set of arbitrary size.

**Definition 3.6.1.** Consider a subset of vertices $A \subset V$ of a given graph $G = (V, E)$. The neighborhood of $A$ is partitioned into four disjoint sets $N_1(A), N_2(A), N_3(A)$, and $N_4(A)$.

- $N_1(A) := \{u \in N(A) : N(u) \setminus N[A] \neq \emptyset\}$
- $N_2(A) := \{u \in N(A) \setminus N_1(A) : N(u) \cap N_1(A) \neq \emptyset\}$
- $N_3(A) := \{u \in N(A) \setminus (N_1(A) \cup N_2(A)) : N(u) \cap N_2(A) \neq \emptyset\}$
- $N_4(A) := N(A) \setminus (N_1(A) \cup N_2(A) \cup N_3(A))$

In the original definitions from [6], which are described in section 3.7, the neighborhood is partitioned into only three parts. Here, the definition of $N_3(A)$ is modified and $N_4(A)$ takes the role of the “inner neighborhood” of $A$.

**Proposition 3.6.2.** Let $D$ be a dominating set of a graph $G$. If $v \notin N_4(A) \cup A$, then there is a path of length at most 4 from $v$ to a vertex of $D$, and the path does not contain any vertices of $A$.

*Proof.* Since $v \notin N_4(A) \cup A$, there is a path of length at most 3 from $v$ to a vertex $w \notin N[A]$, and the path does not contain any vertices of $A$. Since $D$ is a dominating set, this vertex $w$ is adjacent to some vertex $d \in D$. Since $w \notin N[A]$, then obviously $d \notin A$ (it could be that $d \in N(A)$). This gives a path of length at most 4 from $v$ to $d$, as needed. ■

We now define our two reduction rules. Rule 2 applies Rule 1 as a subroutine. Rule 1 removes a vertex $u$ from the graph in case there are two other vertices $v$ and $w$ such that $\{u, v, w\}$ is an independent set and $N(u) = N(v) = N(w) \neq \emptyset$. Rule 2 examines the “inner neighborhood” $N_4(A)$ of a
The Dominating Set Problem on Graphs with an Excluded Minor

subset $A$ of size $k$. By applying a fixed-parameter algorithm for finding dominating sets in degenerated graphs, it calculates a small set $W$ that contains all the vertices that dominate many vertices of $N_3(A) \cup N_4(A)$. More formally, for every set $D$ of size at most $k$ that dominates $N_3(A) \cup N_4(A)$, there is a subset $D' \subset D$, such that $D' \subset W$ and $(N_3(A) \cup N_4(A)) \setminus N[D'] \subset W$. In case $N_4(A)$ is large, many of the vertices of $N_4(A) \setminus W$ can be removed from the graph. The main goal of this section will be to analyze graphs for which Rule 2 cannot be applied anymore.

**Rule 1:** Let $A \subset V$ be an independent set of the graph $G = (V, E)$ and assume that $N(v) \neq \emptyset$ for every $v \in A$.

- Partition the set $A$ into disjoint subsets $A_1, A_2, \ldots, A_t$ according to the neighborhoods of vertices of $A$. That is, every two vertices $v, w \in A_i$ satisfy $N(v) = N(w)$, whereas every two vertices $v \in A_i$ and $w \in A_j$ for $i \neq j$ satisfy $N(v) \neq N(w)$.

- For every $1 \leq i \leq t$ for which $|A_i| > 2$, let $v, w \in A_i$ be two arbitrary distinct vertices. Remove all the vertices of $A_i \setminus \{v, w\}$ from the graph.

**Rule 2:** Suppose that $G = (V, E)$ is $d$-degenerated and $A \subset V$ is a subset of $k$ vertices. If $|N_4(A)| > 2^{(4dk+3k)^k+1}$, do the following.

- Let $F$ be a family of $t \leq (4d+2)^kk!$ pairs $(D_i, B_i)$ of subsets of $V$, such that $|D_i| \leq k$ and $|B_i| \leq (4d+2)k$ for every $1 \leq i \leq t$ for which the following holds. If $D \subset V$ is a subset of size at most $k$ that dominates $N_3(A) \cup N_4(A)$, then some $i, 1 \leq i \leq t$, satisfies that $D_i \subset D$ and $B_i = (N_3(A) \cup N_4(A)) \setminus N[D_i]$.

- Denote $W := A \cup \bigcup_{i=1}^t (D_i \cup B_i)$. Remove all edges between vertices of $(N_3(A) \cup N_4(A)) \setminus W$.

- Apply Rule 1 to the resulting graph and the independent set $N_4(A) \setminus W$.

The next two Lemmas prove the correctness of these rules.

**Lemma 3.6.3.** Let $A \subset V$ be an independent set of the graph $G = (V, E)$. Applying Rule 1 to $G$ and $A$ does not change the domination number.
Proof. It is enough to prove that if \( \{x, y, z\} \) is an independent set, such that \( N(x) = N(y) = N(z) \neq \emptyset \), then \( \gamma(G - z) = \gamma(G) \). To prove that \( \gamma(G) \leq \gamma(G - z) \), let \( D \) be a dominating set of \( G - z \). If \( D \cap N(x) = \emptyset \), then \( \{x, y\} \subset D \), and therefore \( (D \setminus \{y\}) \cup \{u\} \) is a dominating set of \( G \), for any \( u \in N(x) \).

To prove that \( \gamma(G - z) \leq \gamma(G) \), let \( D \) be a minimum dominating set of \( G \). It cannot be the case that \( \{x, y, z\} \subset D \), since adding one of the vertices of \( N(x) \) to \( D \setminus \{y, z\} \) results in a smaller dominating set. We can assume, without loss of generality, that \( z \notin D \), and therefore \( D \) is a dominating set of \( G - z \).

**Lemma 3.6.4.** Suppose that \( G = (V, E) \) is \( d \)-degenerated and \( A \subset V \) is a subset of \( k \) vertices. In case Rule 2 is applied to \( G \) and \( A \), then at least one vertex is removed from the graph, whereas the domination number does not change.

Proof. Using the notations of Rule 2, denote by \( G' \) the graph obtained from \( G \) by removing all edges between vertices of \( (N_3(A) \cup N_4(A)) \setminus W \), just before Rule 1 is applied. It follows from Lemma 3.6.3 that in order to verify that Rule 2 does not change the domination number, it is enough to prove that \( \gamma(G') = \gamma(G) \). It is obvious that \( \gamma(G') \geq \gamma(G) \), since removing edges cannot decrease the domination number. We now prove that \( \gamma(G') \leq \gamma(G) \). Let \( D \) be a minimum dominating set of \( G \), and let \( D' \subset D \) be a subset of minimum size that dominates \( N_3(A) \cup N_4(A) \). This implies that \( D' \subset A \cup N_2(A) \cup N_4(A) \) and \( N[D'] \subset N[A] \). Obviously \( |D'| \leq k \), since otherwise \( (D \setminus D') \cup A \) would be a smaller dominating set of \( G \). Thus, from Theorem 3.4.1, some \( i \), \( 1 \leq i \leq t \), satisfies that \( D_i \subset D' \) and \( B_i = (N_3(A) \cup N_4(A)) \setminus N[D_i] \). To prove that \( D \) is also a dominating set of \( G' \), we need to show that the vertices of \( (N_3(A) \cup N_4(A)) \setminus W \) are dominated by \( D \) in \( G' \), since the neighborhood of all other vertices remained the same. Assume that \( v \in (N_3(A) \cup N_4(A)) \setminus W \). Since \( B_i \subset W \), it follows that \( v \notin B_i \), and therefore \( v \) is dominated in \( G \) by some vertex \( d \in D_i \). This means that \( v \) is still dominated by \( d \) in \( G' \), since \( D_i \subset W \). This completes the proof that Rule 2 does not change the domination number.

We now prove that when Rule 2 is applied, at least one vertex of \( N_4(A) \setminus W \) is removed from the graph \( G' \). First, note that \( (N_3(A) \cup N_4(A)) \setminus W \) is an
independent set, and therefore \( N_4(A) \setminus W \) is also independent. Given a vertex \( v \in N_4(A) \setminus W \), obviously \( N(v) \subseteq A \cup N_3(A) \cup N_4(A) \) and \( N(v) \neq \emptyset \), since it is adjacent to at least one vertex of \( A \). The important property of \( v \) is that it is adjacent in \( G' \) only to vertices of \( W \), since all other edges incident at \( v \) were removed. Since \( W = A \cup \bigcup_{i=1}^t (D_i \cup B_i) \), it follows that \( |W| \leq k + (4d + 2)^k \cdot k!(k + (4d + 2)k) = (4d + 3)k(4d + 2)^k k! + k \). It is easy to verify that \( 2 \cdot 2^{|W|} + |W| \leq 2^{|W|+2} \leq 2(4dk+3k)^{k+1} < N_4(A) \). Thus, \( |N_4(A) \setminus W| \geq |N_4(A)| - |W| > 2 \cdot 2^{|W|} \). By the pigeonhole principle, we conclude that there are three distinct vertices \( x, y, z \in N_4(A) \setminus W \), such that \( N(x) = N(y) = N(z) \neq \emptyset \). One of these three vertices will be removed by Rule 1.

The following Lemma is useful for showing that given a graph with an excluded minor and a dominating set \( D \) of size \( k \), there exists a subset of vertices \( U \) whose size is linear in \( k \), such that all vertices not in \( D \cup U \) belong to the "inner neighborhood" \( N_4(A) \) of a subset \( A \subseteq D \cup U \) of constant size.

**Lemma 3.6.5.** Let \( D \) be a dominating set of the graph \( G = (V, E) \). If \( r \geq 1 \) and \( v \notin D \cup \hat{D}_{r+1,4} \), then there exists a subset \( A \subseteq D \cup \hat{D}_{r+1,3} \) of size at most 40r^5, such that \( v \in N_4(A) \).

**Proof.** To simplify the notation, the symbol \( \hat{D} \) will refer to \( \hat{D}_{r+1,3} \). Let \( q \) be the maximum number of disjoint paths of length 4 from \( v \) to \( q \) different vertices of \( D \). Since \( v \notin D \cup \hat{D}_{r+1,4} \), it follows from the definition of \( \hat{D}_{r+1,4} \) that \( q \leq r \). Construct \( q \) such paths, whose total length is the minimum possible. Denote by \( B \) the set of all vertices that appear in these \( q \) paths and call the inner vertices of these paths \( B' := B \setminus (D \cup \{v\}) \). Assign \( t := 3r(r + r^2 + r^4) + 1 \), and assume, by contradiction, that \( v \notin N_4(A) \) for all subsets \( A \subseteq D \cup \hat{D} \) of size at most 4(r + t - 1). Note that \( 4(r + t - 1) \leq 40r^5 \).

We will now construct \( t \) paths of length at most 4 and a series of \( t \) subsets \( A_1 \subset A_2 \subset \cdots \subset A_t \) of size at most 4(r + t - 1). Let \( A_1 := B \cap (D \cup \hat{D}) \). For each \( i \) from 1 to \( t \), do the following. According to our assumption \( v \notin N_4(A_i) \cup A_i \), which means by Proposition 3.6.2 that there is a path of length at most 4 from \( v \) to a vertex of \( D \), and this path does not contain any vertices from \( A_i \). Denote by \( P_i \) the vertices of a minimum length path, which satisfies
these properties. Define \( A_{i+1} := A_i \cup (P_i \cap (D \cup \hat{D})) \) and proceed to the next iteration to construct \( P_{i+1} \).

Note that \( |A_1| \leq 4r \) and \( |A_{i+1}| \leq |A_i| + 4 \). Thus, all the sets \( A_i \) are of size at most \( 4r + 4(t - 1) = 4(r + t - 1) \). After completing this process, we get \( t \) paths of length at most 4 that start at \( v \). Note that a vertex \( u \in D \cup \hat{D} \) can participate in at most one of these paths, since once it appears in a path \( P_i \), it is immediately added to \( A_{i+1} \). Because of the maximality of \( q \), each path \( P_i \) must contain a vertex of \( B' \). From now on, we will consider the last appearance of a vertex from \( B' \) in a path \( P_i \) as the starting point of the path. This means that all the paths \( P_i \) start at a vertex of \( B' \) and are of length at most 3. Since \( |B'| \leq 3q \leq 3r \) and the number of paths is \( t = 3r(r + r^2 + r^4) + 1 \), by the pigeonhole principle there must be a vertex \( b \in B' \) that is a starting point of \( r + r^2 + r^4 + 1 \) paths of length at most 3. We now prove that \( b \in \hat{D} \). There are three possible cases.

**Case 1:** The vertex \( b \) starts at least \( r + 1 \) paths of length 1. This means that \( b \) is adjacent to \( r + 1 \) vertices of \( D \) and therefore \( b \in \hat{D} \).

**Case 2:** The vertex \( b \) starts at least \( r^2 + 1 \) paths of length 2. It follows from the construction that all these paths are from \( b \) to a different vertex of \( D \). A vertex \( u \) cannot be the middle vertex of more than \( r \) of these paths, since this would imply that \( u \in \hat{D} \), but as mentioned before, vertices of \( \hat{D} \) can appear in at most one path. Thus, there are at least \( r + 1 \) middle vertices that are part of \( r + 1 \) vertex disjoint paths of length 2 from \( b \) to \( D \), which implies that \( b \in \hat{D} \).

**Case 3:** The vertex \( b \) starts at least \( r^4 + 1 \) paths of length 3. The vertex \( b \) is the first vertex of these paths, whereas the fourth vertex is always a different vertex from \( D \). Denote by \( U_2 \) and \( U_3 \) the vertices that appear as a second and third vertex on one of these paths, respectively. Recall that when creating the paths \( P_i \), we always chose a path of minimum length that leads to a vertex of \( D \). This implies that \( U_2 \cap U_3 = \emptyset \). As before, vertices of \( U_2 \) and \( U_3 \) can belong to at most \( r^2 \) and \( r \) paths, respectively. The total number of paths is \( r^4 + 1 \), and therefore \( |U_2| \geq r^2 + 1 \). Since a vertex of \( U_3 \) belongs to at most \( r \) paths, we can find \( r + 1 \) vertices of \( U_2 \) that can be matched to \( r + 1 \) different vertices of \( U_3 \) in a way which would give \( r + 1 \) vertex disjoint paths of length 3 from \( b \) to \( r + 1 \) different vertices of \( D \). This implies that
$b \in \hat{D}$.

In all three cases $b \in \hat{D}$, which means that $b \in A_1$. Thus, $b$ cannot belong to any path $P_i$, and we get a contradiction.

The following is the main result of the chapter.

**Theorem 3.6.6.** For every fixed $h$, given a graph $G$ with $n$ vertices that does not contain $K_h$ as a topological minor, there is an $O(n^{3.5} + kO(1))$ time algorithm that constructs a subgraph $G'$ of $G$, such that if $\gamma(G) = k$, then $\gamma(G') = k$ and $G'$ has at most $k^c$ vertices, where $c$ is a constant that depends only on $h$.

**Proof.** Let $s$ be the constant from Proposition 3.3.3. Suppose that the graph $G$ contains no $K_h$ as a topological minor and $\gamma(G) = k > 1$. To construct the kernel, we perform at most $n$ iterations, as follows. The iteration starts by applying the $O(nk)$ time approximation algorithm described in Theorem 3.4.2 in order to compute a dominating set $D$ of size at most $(3sh)^h k$. It followed from Lemmas 3.5.1 and 3.5.4 that the set $\hat{D}_{h-1,3}$ is of size at most $(6sh^2)^{3h}|D|$, and can be computed in time $O(n^{2.5})$. In case there is a sub-
set $A \subset D \cup \hat{D}_{h-1,3}$ of size $40(h-2)^5$, for which the conditions of Rule 2 are satisfied, then the rule is applied. It follows from Lemma 3.6.4 that at least one vertex is removed from the graph and the domination number does not change. We continue to the next iteration with the resulting graph. Upon termination, this process computes a kernel $G'$ with $\gamma(G') = k$, and a dominating set $D$ of size at most $(3sh)^h k$.

As for the kernel size, Lemma 3.5.1 implies that $|\hat{D}_{h-1,4}| = O(k)$, whereas from Lemma 3.6.5 we know that if $v \notin D \cup \hat{D}_{h-1,4}$, then there exists a subset $A \subset D \cup \hat{D}_{h-1,3}$ of size at most $40(h-2)^5$, such that $v \in N_4(A)$. The number of such subsets $A$ is $kO(1)$ and it follows from Lemma 3.6.4 that each subset $A$ satisfied that $N_4(A) = O(1)$, since Rule 2 cannot be applied anymore. We conclude that the number of vertices not in $D \cup \hat{D}_{h-1,4}$ is $kO(1)$, and the theorem is proved.

**Theorem 3.6.7.** There is an $O(n^{3.5} + 2^{O(\sqrt{k})})$ time algorithm for finding a dominating set of size at most $k$ in a $K_h$-minor-free graph with $n$ vertices that contains such a set.
Proof. Construct a problem kernel $G'$ using Theorem 3.6.6 and apply the $2^{O(\sqrt{k})}n^c$ time algorithm of Demaine et al. [45] on the graph $G'$. ■

3.7 Linear Kernel for Graphs with no Topological $K_{3,h}$

All graphs considered in this section contain no $K_{3,h}$ as a topological minor, for some fixed $h$. Whenever using the big Oh notation, the hidden constant depends only on $h$. We use the following definitions from [6] concerning the neighborhood of a single vertex and the neighborhood of a pair of vertices.

Definition 3.7.1. Consider a vertex $v \in V$ of a given graph $G = (V, E)$. The neighborhood of $v$ is partitioned into three disjoint sets $N_1(v)$, $N_2(v)$, and $N_3(v)$.

- $N_1(v) := \{ u \in N(v) : N(u) \setminus N[v] \neq \emptyset \}$
- $N_2(v) := \{ u \in N(v) \setminus N_1(v) : N(u) \cap N_1(v) \neq \emptyset \}$
- $N_3(v) := N(v) \setminus (N_1(v) \cup N_2(v))$

Definition 3.7.2. Consider two distinct vertices $v, w \in V$ of a given graph $G = (V, E)$. The neighborhood of the two vertices is partitioned into three disjoint sets $N_1(v, w)$, $N_2(v, w)$, and $N_3(v, w)$.

- $N_1(v, w) := \{ u \in N(v, w) : N(u) \setminus N[v, w] \neq \emptyset \}$
- $N_2(v, w) := \{ u \in N(v, w) \setminus N_1(v, w) : N(u) \cap N_1(v, w) \neq \emptyset \}$
- $N_3(v, w) := N(v, w) \setminus (N_1(v, w) \cup N_2(v, w))$

Here is a simple observation that follows immediately from the previous definitions.

Proposition 3.7.3. Let $D$ be a dominating set of a graph $G$. If $u \notin N_3(v) \cup \{v\}$, then there is a path of length at most 3 from $u$ to a vertex of $D$, and the path does not contain $v$. If $u \notin N_3(v, w) \cup \{v, w\}$, then there is a path of length at most 3 from $u$ to a vertex of $D$, and the path contains neither $v$ nor $w$. 
The following is a simplified presentation of the two reduction rules of Alber et al. [6]. As proved there, these reduction rules do not change the domination number of the graph. Unlike the original rules, in which new vertices can be added to the graph, in our formulation the only modifications made to the graph are the removal of vertices and edges. Another useful property of the following formulation is that in case a rule is applied, at least one vertex is removed from the graph. Unlike the original rules, vertices that belong to $N_2(v)$ or $N_2(v, w)$ are not removed in our rules. We note that Alber et al. specifically proved that their rules preserve planarity, and this might imply that the authors did not notice the fact that the reduction rules actually construct a subgraph of the input graph, and therefore all monotone properties are preserved.

**Rule 3:** Given a graph $G = (V, E)$ and a vertex $v \in V$, if $|N_3(v)| > 1$, then do the following. Let $v'$ be some arbitrary vertex of $N_3(v)$. Remove all the vertices of $N_3(v) \setminus \{v'\}$ and all the edges incident at $v'$, except for $\{v, v'\}$.

**Rule 4:** Let $v$ and $w$ be two distinct vertices of the graph $G = (V, E)$. If $|N_3(v, w)| > 2$ and $N_3(v, w)$ cannot be dominated by a single vertex from $N_2(v, w) \cup N_3(v, w)$, then do the following.

- If both $v$ and $w$ dominate $N_3(v, w)$, then let $z$ and $z'$ be two arbitrary distinct vertices of $N_3(v, w)$. Remove all the vertices of $N_3(v, w) \setminus \{z, z'\}$ and all the edges incident at $z$ and $z'$, except for the edges $\{v, z\}, \{w, z\}, \{v, z'\}, \{w, z'\}$.

- If $v$ dominates $N_3(v, w)$ but $w$ does not dominate it, then let $v'$ be some arbitrary vertex of $N_3(v, w)$. Remove all the vertices of $N_3(v, w) \setminus \{v'\}$ and all the edges incident at $v'$, except for the edge $\{v, v'\}$. The case that only $w$ dominates $N_3(v, w)$ is handled in a symmetric manner.

- If neither $v$ nor $w$ dominate $N_3(v, w)$, then let $v'$ and $w'$ be two arbitrary distinct vertices of $N_3(v, w)$ such that $v'$ is adjacent to $v$ and $w'$ is adjacent to $w$. Remove all the vertices of $N_3(v, w) \setminus \{v', w'\}$ and all the edges incident at $v'$ and $w'$, except for the edges $\{v, v'\}, \{w, w'\}$.

A graph is called **reduced** in case Rules 3 and 4 cannot be applied to it anymore. The following definitions are specific to this section.
Definition 3.7.4. Let $D$ be a dominating set of the graph $G = (V, E)$.

- Denote by $\tilde{D}$ the set of vertices in $V \setminus D$ that have at least two neighbors from $D$.

- Let $d_1, d_2 \in D$ be two distinct vertices. Denote by $\text{Inner}(d_1, d_2)$ the set of all inner vertices of paths of length 3 of the type $d_1 - x - y - d_2$, such that $x, y \in N_3(d_1, d_2) \setminus (D \cup \tilde{D} \cup \hat{D})$. Denote $\text{Inner}(D) := \bigcup_{d_1, d_2 \in D, d_1 \neq d_2} \text{Inner}(d_1, d_2)$

Lemma 3.7.5. For a fixed $h \geq 2$, suppose that $G = (V, E)$ is a reduced graph that contains no $K_{3,h}$ as a topological minor. If $D$ is a dominating set of size $k$, then $|\tilde{D}| = O(k)$.

Proof. Assume that $v \in \tilde{D}$. This means that $v$ is adjacent to at least 2 vertices of $D$, so we distinguish between three cases.

Case 1: The vertex $v$ is adjacent to at least 3 vertices of $D$. Thus, by definition $v \in \hat{D}_{3,3}$, and it follows from Lemma 3.5.3 that $|\hat{D}_{3,3}| = O(k)$.

Case 2: The vertex $v$ is adjacent to exactly 2 vertices $d_1, d_2 \in D$ and $v \notin N_3(d_1, d_2)$. It follows from proposition 3.7.3 that there is a path of a length at most 3 from $v$ to a vertex of $D$, and the path does not use the vertices $d_1$ and $d_2$. This implies that $v \in \hat{D}_{3,3}$ and we proceed as in the previous case.

Case 3: The vertex $v$ is adjacent to exactly 2 vertices $d_1, d_2 \in D$ and $v \in N_3(d_1, d_2)$. The number of pairs $d_1, d_2 \in D$ for which there is a vertex $v \notin D$ such that $N(v) \cap D = \{d_1, d_2\}$ is $O(k)$. To see this, just connect each such pair $d_1, d_2$, in case they were not connected before. Denote the resulting graph by $G'$. The number of edges in $G'[D]$ is at least the number of pairs we are counting. Since $G'[D]$ does not contain $K_{3,h}$ as a topological minor, it has $O(k)$ edges.

For two distinct vertices $d_1, d_2 \in D$, denote by $Q$ the set of vertices $v \in N_3(d_1, d_2)$ that are adjacent to both $d_1$ and $d_2$. It is now enough to prove that $|Q| \leq h$. By contradiction, assume that $|Q| > h \geq 2$. Since the graph is reduced, there is a vertex $w \in N_3(d_1, d_2) \cup N_3(d_1, d_2)$ that dominates $N_3(d_1, d_2)$. Note that $w$ can possibly belong to $N_3(d_1, d_2)$. This implies that $d_1, d_2$, and $w$ together with $Q \setminus \{w\}$ form a $K_{3,h}$. This is a contradiction, and the claim is proved.
Corollary 3.7.6. For a fixed $h \geq 2$, let $D$ be a dominating set of size $k$ of a reduced graph $G = (V, E)$ that contains no $K_{3,h}$ as a topological minor. If a subset $U \subseteq V$ of size $m$ satisfies that $D \cap U = \emptyset$, then $|N[U]| = O(k + m)$.

Proof. The set $D \cup U$ is obviously a dominating set. A vertex $v \in N[U] \setminus (D \cup U)$ is adjacent to a vertex of $U$ and also to a vertex of $D$, since $D$ is a dominating set. This means that $v$ is adjacent to at least two vertices of $D \cup U$. The result now follows from Lemma 3.7.3. ■

Lemma 3.7.7. Suppose that $G = (V, E)$ is a reduced graph that contains no $K_{3,h}$ as a topological minor. If $D$ is a dominating set of size $k$, then there are $O(k)$ pairs $d_1, d_2 \in D$ for which $\text{Inner}(d_1, d_2) \neq \emptyset$.

Proof. Consider the pairs $d_1, d_2 \in D$ for which $\text{Inner}(d_1, d_2) \neq \emptyset$ in some arbitrary order. For each such pair $d_1, d_2$, there are two vertices $x, y \in N_3(d_1, d_2) \setminus (D \cup \bar{D})$ that appear on the path $d_1 - x - y - d_2$. We claim that both $x$ and $y$ do not belong to any other pair $\text{Inner}(d'_1, d'_2)$. To see this, suppose by contradiction that $x \in \text{Inner}(d_1, d_2) \cap \text{Inner}(d'_1, d'_2)$ for $\{d'_1, d'_2\} \neq \{d_1, d_2\}$. Since $x \notin \bar{D}$, it has only one neighbor in $D$, so assume, without loss of generality, that $x$ is adjacent to $d_1 = d'_1$ and $x$ appears on the two paths $d_1 - x - y - d_2$ and $d_1 - x - z - d'_2$. This implies that $x \in \bar{D}_{3,3}$, a contradiction, and the claim is proved.

In each case as above, we delete the vertices $x$ and $y$, and add an edge between $d_1$ and $d_2$, assuming this edge does not exist. Denote the resulting graph by $G'$. Obviously, $G'[D]$ does not contain $K_{3,h}$ as a topological minor and therefore has at most $O(k)$ edges. The number of edges in the induced subgraph $G'[D]$ is at least the number of pairs for which $\text{Inner}(d_1, d_2) \neq \emptyset$, as claimed. ■

Lemma 3.7.8. Let $D$ be a dominating set of a reduced graph $G = (V, E)$ that contains no $K_{3,h}$ as a topological minor. Every two distinct vertices $d_1, d_2 \in D$ satisfy $|\text{Inner}(d_1, d_2)| \leq 2h^2$.

Proof. By contradiction, assume that $|\text{Inner}(d_1, d_2)| \geq 2h^2 + 1$. This implies that $|N_3(d_1, d_2)| > 2$, and since the graph is reduced, there is a vertex $v \in N_2(d_1, d_2) \cup N_3(d_1, d_2)$ that dominates $N_3(d_1, d_2)$. Let $q$ be the maximum number of vertex disjoint paths of the type $d_1 - x - y - d_2$, such that $x, y \in
Follows immediately from Lemmas 3.7.5 and 3.7.8. We examine the inner vertices of paths of the form \( v, x \in N(d_1, d_2) \), and denote by \( W \) the \( 2q \) inner vertices of these paths. Note that \( v \) can possibly belong to \( W \). We must have that \( q \leq h \), since otherwise \( d_1, d_2 \), and \( v \) would be part of a topological \( K_{3,h} \). Since \( |W| = 2q \leq 2h \), there are at least \( 2h(h-1)+1 \) vertices of \( \text{Inner}(d_1, d_2) \setminus W \) that appear on a path of the type \( d_1 - x - y - d_2 \) together with one of the vertices of \( W \). Thus, there is a vertex \( w \in W \) that belongs to at least \( h \) of these paths. Assuming, without loss of generality, that \( w \) is adjacent to \( d_1 \), there are \( h+1 \) different paths of length 2 from \( w \) to \( d_2 \), and the inner vertices of these paths are from \( \text{Inner}(d_1, d_2) \). Thus, \( w, d_2 \), and \( v \) are part of a \( K_{3,h} \). This is a contradiction, and the claim is proved. \( \qed \)

**Lemma 3.7.9.** Suppose that the reduced graph \( G = (V, E) \) contains no \( K_{3,h} \) as a topological minor. If \( D \) is a dominating set of size \( k \), then \( |\text{Inner}(D)| = O(k) \).

**Proof.** Follows immediately from Lemmas 3.7.7 and 3.7.8. \( \qed \)

**Lemma 3.7.10.** Suppose that the reduced graph \( G = (V, E) \) contains no \( K_{3,h} \) as a topological minor. If \( D \) is a dominating set of size \( k \), then the number of vertices that appear on a path of length 3 between two vertices of \( D \) is \( O(k) \).

**Proof.** We examine the inner vertices of paths of the form \( d_1 - v - x - d_2 \), such that \( d_1, d_2 \in D \). It follows from Lemmas 3.5.3 and 3.7.3 that \( |\hat{D}_{3,3} \cup \overline{D}| = O(k) \), which means that it remains to count the number of vertices not in \( D \cup \hat{D}_{3,3} \cup \overline{D} \). Assume that \( v \notin D \cup \hat{D}_{3,3} \cup \overline{D} \). Since \( v \notin \overline{D} \), it is adjacent to exactly one vertex of \( D \), and therefore \( x \notin D \). If \( x \in \hat{D}_{3,3} \cup \overline{D} \), then \( v \in N[\hat{D}_{3,3} \cup \overline{D}] \), but it follows from Corollary 3.7.6 that \( |N[\hat{D}_{3,3} \cup \overline{D}]| = O(k) \). If either \( v \) or \( x \) do not belong to \( N_3(d_1, d_2) \), then this implies that \( x \in \hat{D}_{3,3} \), but this case has already been addressed. The only remaining case is that \( v, x \in N_3(d_1, d_2) \setminus (D \cup \hat{D}_{3,3} \cup \overline{D}) \), which means that \( v \in \text{Inner}(D) \), and we know from Lemma 3.7.9 that \( |\text{Inner}(D)| = O(k) \). \( \qed \)

We can now state the main result of this section.

**Theorem 3.7.11.** For every fixed \( h \), given a graph \( G \) that does not contain \( K_{3,h} \) as a topological minor, there is a polynomial time algorithm that constructs a subgraph \( G' \) of \( G \), such that if \( \gamma(G) = k \), then \( \gamma(G') = k \) and \( G' \) has at most \( ck \) vertices, where \( c \) is a constant that depends only on \( h \).
Proof. Suppose that $G$ contains no $K_{3,h}$ as a topological minor and $\gamma(G) = k$. As long as the conditions of Rules 3 and 4 are satisfied, apply these rules to get a reduced subgraph $G'$. Alber et al. [6] proved that $\gamma(G') = k$, so let $D$ be a dominating set of $G'$ of size $k$. It follows from Lemma 3.7.3 that $|\tilde{D}| = O(k)$, so we need to count the number of vertices not in $D \cup \tilde{D}$. Assume $v \notin D \cup \tilde{D}$ is adjacent to $d_1 \in D$. If $v \in N_3(d_1)$, then in a reduced graph $|N_3(d_1)| \leq 1$, which means that there could be at most $k$ vertices of this type. Assume now that $v \notin N_3(d_1)$, so by Proposition 3.7.3 there is a path of length at most 3 from $v$ to a vertex $d_2 \in D$, and $d_1$ is not part of this path. We examine a shortest path $p$ from $d_1$ to $d_2$, in which $v$ is the second vertex of the path. Since $v \notin \tilde{D}$, it is adjacent to only one vertex of $D$, so the path $p$ can be of length either 3 or 4.

In case $p$ is of length 3, then it follows from Lemma 3.7.10 that there are at most $O(k)$ vertices of this type. If $p$ is of length 4, denote it by $d_1 - v - x - y - d_2$, where $x, y \notin D$. The vertex $x$ is adjacent to some vertex of $D$. It cannot be adjacent to $d_2$, since a path $p$ on minimum length was chosen. If $x$ is adjacent to a vertex of $D \setminus \{d_1, d_2\}$, then $x \in \hat{D}_{3,3}$ and $v \in N[\hat{D}_{3,3}]$, but it follows from Corollary 3.7.6 that $|N[\hat{D}_{3,3}]| = O(k)$. The remaining case is that $d_1$ is the only vertex in $D$ that is adjacent to $x$. Since $x \notin D$ is on a path of length 3 from $d_1$ to $d_2$, it follows from Lemma 3.7.10 and Corollary 3.7.6 that the number of vertices $v$ of this type is also $O(k)$. ■

3.8 Finding Induced Cycles

3.8.1 Degenerated Graphs

Recall that $N_i^+(v)$ is the set of all vertices that can be reached from $v$ by a directed path of length exactly $i$. If the outdegree of every vertex in a directed graph $D = (V, A)$ is at most $d$, then obviously $|N_i^+(v)| \leq d^i$ for every $v \in V$ and $i \geq 1$.

Theorem 3.8.1. For every fixed $d \geq 1$ and $k \leq 5$, there is a deterministic $O(n)$ time algorithm for finding an induced cycle of length $k$ in a $d$-degenerated graph on $n$ vertices.
3.8 Finding Induced Cycles

Proof. Given a \( d \)-degenerated graph \( G = (V, E) \) with \( n \) vertices, we orient the edges so that the outdegree of all vertices is at most \( d \). This can be done in time \( O(|E|) \). Denote the resulting directed graph \( D = (V, A) \). We can further assume that \( V = \{1, 2, \ldots, n\} \) and that every directed edge \( \{u, v\} \in A \) satisfies \( u < v \). This means that an out-neighbor of a vertex \( u \) will always have an index which is bigger than that of \( u \). We now describe how to find cycles of size at most 5.

To find cycles of size 3 we simply check for each vertex \( v \) whether \( N^+(v) \cap N^+_2(v) \neq \emptyset \). Suppose now that we want to find an induced cycle \( v_1 - v_2 - v_3 - v_4 - v_1 \) of size 4. Without loss of generality, assume that \( v_1 < v_2 < v_4 \). We distinguish between two possible cases.

- \( v_1 < v_3 < v_2 < v_4 \): Keep two counters \( C_1 \) and \( C_2 \) for each pair of vertices. For every vertex \( v \in V \) and every unordered pair of distinct vertices \( u, w \in N^+(v) \), such that \( u \) and \( w \) are not connected, we raise the counter \( C_1(\{u, w\}) \) by one. In addition to that, for every vertex \( x \in N^+(v) \) such that \( u, w \in N^+(x) \), the counter \( C_2(\{u, w\}) \) is incremented. After completing this process, we check whether there are two vertices for which \( (C_1(\{u, w\}))^2 - C_2(\{u, w\}) > 0 \). This would imply that an induced 4-cycle was found.

To see this, note that if \( u \) and \( w \) are not connected and \( u, w \in N^+(v) \cap N^+(x) \), then \( v - u - x - w - v \) is a cycle, but it is not necessarily induced, since there could be an edge between \( v \) and \( x \).

Let \( u \) and \( w \) be two specific vertices that are not adjacent. The number of cycles (not necessarily induced) of the form \( v - u - x - w - v \) in which \( v \) and \( x \) are adjacent is \( C_2(\{u, w\}) \). The total number of cycles of this form, where \( v \) and \( x \) can be either adjacent or not adjacent, is \( (C_1(\{u, w\}))^2 \). This means that the number of induced cycles of the form \( v - u - x - w - v \), where \( u, w \in N^+(v) \cap N^+(x) \), is exactly \( (C_1(\{u, w\}))^2 - C_2(\{u, w\}) \).

- \( v_1 < v_2 < v_3 < v_4 \) or \( v_1 < v_2 < v_4 < v_3 \): This case is easier, since the induced cycle is contained in the set \( \{v_1\} \cup N^+(v_1) \cup N^+_2(v_1) \). The size of this set is at most \( 1 + d + d^2 \), which is a constant (\( d \) is fixed). Thus, it takes constant time to find an induced cycle that is contained in this
To conclude, in order to find induced cycles of this type, check for each vertex $v$ whether the set $\{v\} \cup N^+(v) \cup N_2^+(v)$ contains an induced cycle.

To find an induced cycle of size 5, a more detailed case analysis is needed. It is easy to verify that such a cycle has one of the following two types.

- There is a vertex $v$ such that $\{v\} \cup N^+(v) \cup N_2^+(v) \cup N_3^+(v)$ contains the induced cycle: The size of the set $\{v\} \cup N^+(v) \cup N_2^+(v) \cup N_3^+(v)$ is at most $1 + d + d^2 + d^3$, which is a constant, and we continue as before.

- The cycle is of the form $v - x - u - y - w - v$, where $x \in N^+(v)$, $u \in N^+(x) \cap N^+(y)$, and $w \in N^+(v) \cap N^+(y)$: Keep two counters $C_1$ and $C_2$ for each ordered pair of vertices. Let $u$ and $w$ be two specific vertices that are not adjacent. For every vertex $y \in V$, if $u, w \in N^+(y)$, we raise the counter $C_1(u, w)$ by one. For every vertex $v \in V$ and every $x \in N^+(v)$ such that $u \in N^+(x)$, $w \in N^+(v)$ and the two pairs $\{v, u\}, \{x, w\}$ are not adjacent, the counter $C_2(u, w)$ is incremented.

Let $u$ and $w$ be two specific vertices that are not adjacent. The number of cycles (not necessarily induced) of the form $v - x - u - y - w - v$, where $x \in N^+(v)$, $u \in N^+(x) \cap N^+(y)$, and $w \in N^+(v) \cap N^+(y)$ and the two pairs $\{v, u\}, \{x, w\}$ are not adjacent, is exactly $C_1(u, w)C_2(u, w)$. We now need to subtract the number of cycles that are not induced, by defining counters in a similar way to what was done before.

The following simple lemma shows that a linear time algorithm for finding an induced $C_6$ in a 2-degenerated graph would imply that a triangle (a $C_3$) can be found in a general graph in $O(|V| + |E|)$ time. It is a long standing open question to improve the natural $O(|V|^\omega)$ time algorithm for this problem [68].

**Lemma 3.8.2.** Given a linear time algorithm for finding an induced $C_6$ in a 2-degenerated graph, it is possible to find triangles in general graphs in $O(|V| + |E|)$ time.
3.8 Finding Induced Cycles

Proof. Given a graph \( G = (V, E) \), subdivide all the edges. The new graph obtained \( G' \) is 2-degenerated and has \(|V| + |E|\) vertices. A linear time algorithm for finding an induced \( C_6 \) in \( G' \) actually finds a triangle in \( G \). By assumption, the running time is \( O(|V| + |E|) \leq O(|V|^2) \).

3.8.2 Minor-Closed Families of Graphs

Theorem 3.8.3. Suppose that \( G \) is a graph with \( n \) vertices taken from some non-trivial minor-closed family of graphs. For every fixed \( k \), if \( G \) contains an induced cycle of size \( k \), then it can be found in \( O(n) \) expected time.

Proof. There is some absolute constant \( d \), so that \( G \) is \( d \)-degenerated. Orient the edges so that the maximum outdegree is at most \( d \) and denote the resulting graph \( D = (V, E) \). We now use the technique of random separation. Each vertex \( v \in V \) of the graph is independently removed with probability \( 1/2 \), to get some new directed graph \( D' \). Now examine some (undirected) induced cycle of size \( k \) in the original directed graph \( D \), and denote its vertices by \( U \). The probability that all the vertices in \( U \) remained in the graph and all vertices in \( N^+(U) - U \) were removed from the graph is at least \( 2^{-k(d+1)} \).

We employ the color-coding method to the graph \( D' \). Choose a random coloring of the vertices of \( D' \) with the \( k \) colors \( \{1, 2, \ldots, k\} \). For each vertex \( v \) colored \( i \), if \( N^+(v) \) contains a vertex with a color which is neither \( i - 1 \) nor \( i + 1 \) (mod \( k \)), then it is removed from the graph. For each induced cycle of size \( k \), its vertices will receive distinct colors and it will remain in the graph with probability at least \( 2k^{1-k} \).

We now use the \( O(n) \) time algorithm from [22] to find a multicolored cycle of length \( k \) in the resulting graph. If such a cycle exists, then it must be an induced cycle. Since \( k \) and \( d \) are constants, the algorithm succeeds with some small constant probability and the expected running time is as needed.

The next theorem shows how to derandomize this algorithm while incurring a loss of only \( O(\log n) \).

Theorem 3.8.4. Suppose that \( G \) is a graph with \( n \) vertices taken from some non-trivial minor-closed family of graphs. For every fixed \( k \), there is an
$O(n \log n)$ time deterministic algorithm for finding an \textit{induced} cycle of size $k$ in $G$.

\textit{Proof.} Denote $G = (V, E)$ and assume that $G$ is $d$-degenerated. We derandomize the algorithm in Theorem 3.8.3 using an $(n, dk + k)$-family of perfect hash functions. This is a family of functions from $[n]$ to $[dk + k]$ such that for every $S \subset [n], |S| = dk + k$, there exists a function in the family that is one-to-one on $S$. Such a family of size $e^{dk+k}(dk+k)^{O(\log (dk+k))} \log n$ can be efficiently constructed \cite{77}. We think of each function as a coloring of the vertices with the $dk+k$ colors $C = \{1, 2, \ldots, dk+k\}$. For every combination of a coloring, a subset $L \subset C$ of $k$ colors and a bijection $f : L \rightarrow \{1, 2, \ldots, k\}$ the following is performed. All the vertices that got a color from $c \in L$ now get the color $f(c)$. The other vertices are removed from the graph.

All the vertices of the resulting graph are now colored with the $k$ colors $\{1, 2, \ldots, k\}$. Examine some induced cycle of size $k$ in the original graph, and denote its vertices by $U$. There exists some coloring $c$ in the family of perfect hash functions for which all the vertices in $U \cup N^+(U)$ received different colors. Now let $L$ be the $k$ colors of the vertices in the cycle $U$ and let $f : L \rightarrow \{1, 2, \ldots, k\}$ be the bijection that gives consecutive colors to vertices along the cycle. This means that for this choice of $c$, $L$, and $f$, the induced cycle $U$ will remain in the graph as a multicolored cycle, whereas all the vertices in $N^+(U) \setminus U$ will be removed from the graph. We proceed as in the previous algorithm.

Better dependence on the parameters $d$ and $k$ can be obtained using the results in \cite{12} concerning $(t, u)$-hashing families. We are interested in a family of functions from $[n]$ to $[k+1]$ such that for every $T \subset U \subset [n], |T| = k, |U| = dk + k$, there exists a function in the family that is one-to-one on $T$, satisfying $f(x) \neq f(y)$ for any $x \in T, y \in U \setminus T$. Such a family of size $(dk+k)^{O(k) \log n}$ can be efficiently constructed using an explicit construction of a $(k, dk+k)$-hashing code over an alphabet of size $k+1$ \cite{12}. We think of each function as a coloring of the vertices with the $k+1$ colors $C = \{1, 2, \ldots, k+1\}$. For every combination of a coloring and a bijection $f : C \rightarrow C$ the following is performed. All the vertices that got a color $c$ get the color $f(c)$. The vertices whose new color is $k+1$ are removed from the graph. We continue as before.
3.9 Concluding Remarks and Open Problems

- The algorithm for finding a dominating set in graphs with an excluded minor, presented in this chapter, generalizes and improves known algorithms for planar graphs and graphs with bounded genus. We believe that similar techniques may be useful in improving and simplifying other known fixed-parameter algorithms for graphs with an excluded minor.

- The dominating set problem is fixed-parameter tractable for degenerated graphs. An interesting open problem is to decide whether there is a polynomial size kernel in this case.

- Another challenging question is to characterize the families of graphs for which the dominating set problem admits a linear kernel. We cannot rule out the possibility that a linear kernel can be obtained for graphs with any fixed excluded minor.
Chapter 4

Prize Collecting Steiner Tree Problems

The results of this chapter appear in [59].

This chapter deals with approximation algorithms for the prize collecting generalized Steiner forest problem, defined as follows. The input is an undirected graph $G = (V, E)$, a collection $T = \{T_1, \ldots, T_k\}$, each a subset of $V$ of size at least 2, a weight function $w : E \to \mathbb{R}^+$, and a penalty function $p : T \to \mathbb{R}^+$. The goal is to find a forest $F$ that minimizes the cost of the edges of $F$ plus the penalties paid for subsets $T_i$ whose vertices are not all connected by $F$. Our main result is a $(3 - \frac{4}{n})$-approximation for the prize collecting generalized Steiner forest problem, where $n \geq 2$ is the number of vertices in the graph. This obviously implies the same approximation for the special case called the prize collecting Steiner forest problem (all subsets $T_i$ are of size 2). The approximation algorithm is obtained by applying the local ratio method, and is much simpler than the best known combinatorial algorithm for this problem.

Our approach gives a $(2 - \frac{1}{n-1})$-approximation for the prize collecting Steiner tree problem (all subsets $T_i$ are of size 2 and there is some root vertex $r$ that belongs to all of them). This latter algorithm is in fact the local ratio version of the primal-dual algorithm of Goemans and Williamson [59]. Another special case of our main algorithm is Bar-Yehuda’s local ratio $(2 - \frac{2}{n})$-approximation for the generalized Steiner forest problem (all the penalties
are infinity) \cite{31}. Thus, an important contribution of the new algorithm is in providing a natural generalization of the framework presented by Goemans and Williamson, and later by Bar-Yehuda.

4.1 Introduction

There is substantial literature dealing with approximation algorithms for prize collecting Steiner tree problems. The purpose of this chapter is to present elegant combinatorial algorithms for these problems. The local ratio technique \cite{30,31,32} that we employ enable us to present simple algorithms together with a straightforward analysis.

The main focus of the chapter is on the prize collecting generalized Steiner forest (PCGSF) problem, defined as follows. The input is an undirected graph $G = (V, E)$, a collection $T = \{T_1, \ldots, T_k\}$, each a subset of $V$ of size at least 2, a weight function $w : E \rightarrow \mathbb{R}^+$, and a penalty function $p : T \rightarrow \mathbb{R}^+$, where $\mathbb{R}^+$ denotes the set of positive real numbers. The objective is to compute a forest $F$ that minimizes the cost of the edges of $F$ and the sum of the penalties of the subsets $T_i$ whose vertices are not all connected by $F$. Thus, all the vertices of a subset $T_i$ must be in the same connected component of $F$ in order to avoid the penalty. Note that we intentionally define costs and penalties to be positive, as this will turn out to be convenient later. During intermediate stages of the algorithm, zero cost edges are contracted, whereas zero penalties can be ignored.

Previous Results. The special case of the PCGSF problem called the prize collecting Steiner forest problem (all subsets $T_i$ are of size 2) has received considerable attention lately. A modification of the LP rounding algorithm in \cite{35} implies a 3-approximation for this problem. This was improved in \cite{60} to give an LP based 2.54-approximation for the problem as well as a primal-dual combinatorial $(3 - \frac{2}{n})$-approximation using Farkas’ Lemma. The authors of \cite{58} give a 3-budget-balanced and group-strategyproof mechanism for the game-theoretic version of the prize collecting Steiner forest problem, which is an extension of the method presented in \cite{73}. Their result also provides a primal-dual 3-approximation algorithm for this problem.

A generalized framework of the prize collecting problems with an arbi-
trary 0−1 connectivity requirement function and a submodular penalty function is studied in [82]. Their model captures both the PCGSF problem defined in this chapter as well as the problems of [62, 60]. The authors give a complicated primal-dual 3-approximation algorithm together with an LP rounding algorithm with a performance ratio of 2.54.

Two classical primal-dual algorithms, relevant to this chapter, are due to Goemans and Williamson [56]. They give a \((2 - \frac{1}{n})\)-approximation for the prize collecting Steiner tree problem (all subsets \(T_i\) are of size 2 and there is some root vertex \(r\) that belongs to all of them) as well as a \((2 - \frac{2}{n})\)-approximation for the generalized Steiner forest problem (all the penalties are infinity) that simulates an algorithm of Agrawal, Klein, and Ravi [2]. For the latter problem, a simple \((2 - \frac{2}{n})\)-approximation based on the local ratio technique is presented by Bar-Yehuda in [31].

**Our Results.** The main result is a local ratio \((3 - \frac{4}{n})\)-approximation for the prize collecting generalized Steiner forest problem, where \(n \geq 2\) is the number of vertices in the graph. This obviously implies the same approximation for the special case of the prize collecting Steiner forest problem, which was studied in [60, 58]. We also present a \((2 - \frac{1}{n})\)-approximation for the prize collecting Steiner tree problem (all subsets \(T_i\) are of size 2 and there is some root vertex \(r\) that belongs to all of them). This latter algorithm is in fact the local ratio version of the primal-dual algorithm of Goemans and Williamson [56]. Another special case of our main algorithm is Bar-Yehuda’s local ratio \((2 - \frac{2}{n})\)-approximation for the generalized Steiner forest problem (all the penalties are infinity) [31]. Thus, an important contribution of this chapter is in providing a natural generalization of the framework presented by Goemans and Williamson, and later by Bar-Yehuda. The algorithm we present in this chapter for the PCGSF problem is not the local ratio version of the primal-dual 3-approximation algorithm from [82] and cannot be obtained from it using the equivalence between the primal-dual schema and the local ratio technique [33].
4.2 The Prize Collecting Generalized Steiner Forest Problem

In this section we present the algorithm for the PCGSF problem. The following are some definitions that are needed for presenting the algorithm.

Definition 4.2.1. For an instance \((G, T = \{T_1, \ldots, T_k\}, w, p)\) of the PCGSF problem, a vertex is said to be an **active vertex** if it belongs to at least one of the subsets \(T_i\).

Definition 4.2.2. A solution \(F\) to the PCGSF problem is a **minimal solution** if every leaf (a vertex of degree 1) of the forest \(F\) is an active vertex.

Definition 4.2.3. Suppose \(G = (V, E)\) is an undirected graph and \(T = \{T_1, \ldots, T_k\}\) is a collection of subsets of \(V\) of size at least 2. For every active vertex \(v\), let \(t(v)\) be some arbitrary subset \(T_i\) for which \(v \in T_i\). The **degree-weighted instance** corresponding to \(G, T, t\) is the quadruple \((G, T, w, p)\), defined as follows.

- For each \(e \in E\), \(w(e)\) is the number of its endpoints that are active vertices.
- For each \(T_i \in T\), \(p(T_i) = |\{v \in T_i| t(v) = T_i\}|\).

Note that in the previous definition, the weight of an edge \(w(e)\) can take a value of 0, 1, or 2, whereas the penalty of a subset \(T_i\) satisfies \(0 \leq p(T_i) \leq |T_i|\). The following definition describes the important and natural operation of an edge contraction.

Definition 4.2.4. Let \((G, T, w, p)\) denote an instance of the PCGSF problem. The **contraction** of an edge \(\{u, v\}\) into a new vertex \(x\) results in a new instance \((G', T', w', p')\), defined as follows.

- For a vertex \(y\), if \(y\) is adjacent in \(G\) to both \(u\) and \(v\), then it is adjacent to \(x\) in \(G'\) and \(w'(\{y, x\}) = \min\{w(y, u), w(y, v)\}\). We say that the edge \(\{y, x\}\) of the graph \(G'\) **corresponds** to the original edge in the graph \(G\) for which the minimum is attained.
• For any subset $T_i \in T$, if $T_i \cap \{u, v\} \neq \emptyset$, then define $T_i' = (T_i \cup \{x\}) - \{u, v\}$. In case $T_i \neq \{u, v\}$, it follows that $|T_i'| \geq 2$ and the new subset $T_i'$ is added to $T'$. During this process, there could be two subsets $T_i$ and $T_j$ for which $T_i' = T_j'$. Obviously, the two subsets can be joined and their new penalty in $p'$ is defined to be $p(T_i) + p(T_j)$.

The following two lemmas establish the important property of minimal solutions to degree-weighted instances.

**Lemma 4.2.5.** Every solution to a degree-weighted instance of the PCGSF problem has a total cost of at least $n$, where $n \geq 2$ is the number of active vertices.

*Proof.* Consider some specific active vertex $v$. There is some subset $T_i \in T$ for which $t(v) = T_i$. In case the vertex $v$ is not connected in the solution to all the other vertices of $T_i$, then we pay a penalty of 1 because of $v$. Otherwise, the solution must contain at least one edge incident with $v$. This means that the solution pays a cost of 1 for this edge due to vertex $v$. Every active vertex incurs a cost or penalty of at least 1, and therefore the total cost of the solution is at least $n$. ■

**Lemma 4.2.6.** Every minimal solution to a degree-weighted instance of the PCGSF problem has a total cost of at most $3n - 4$, where $n \geq 2$ is the number of active vertices.

*Proof.* Examine some specific connected component of the solution that contains $q$ active vertices. Since this is a minimal solution, all of its leafs are active vertices. We prove by induction on $q$ that the cost of the edges in this connected component is at most $2(q - 1)$. For $q = 1$, the connected component is a vertex with no edges, so this is obviously true. For $q = 2$, the connected component must be a path, whose two endpoints are active vertices, and the claim holds.

Suppose that $q > 2$. Take some leaf $v$, which by our assumption must be an active vertex. Now examine the path that starts from $v$ and continues until the first time that either a vertex of degree at least 3 or another active vertex is reached. The cost of the edges in this path is at most 2. After removing this path from the solution, we are left with a connected component with $q - 1$
active vertices whose leafs are all active vertices. The result now follows from the induction hypothesis.

For proving the lemma, we distinguish between two cases. If all the active vertices are in one connected component of the solution, then the solution is actually a tree and no penalties are paid. The cost of the edges in the solution is at most $2(n - 1)$. The total cost, including the penalties, is also $2(n - 1) \leq 3n - 4$, since $n \geq 2$. Otherwise, there are at least two connected components in the solution. The cost of the edges in all the components can be at most $2(n - 2)$. Note that the sum of all penalties in the instance is exactly $n$. The solution pays at most $n$ for penalties, so the total cost is at most $2(n - 2) + n = 3n - 4$.

The last two lemmas determine the approximation ratio of the algorithm. The purpose of the next lemma is to show that the analysis is indeed tight.

**Lemma 4.2.7.** For every $n \geq 2$, there exists a degree-weighted instance of the PCGSF problem on a graph with $n$ vertices for which the optimal solution has total cost $n$ whereas some minimal solution has a total cost of $3n - 4$.

**Proof.** Consider the following instance with vertices $v_1, \ldots, v_n$, which are all active. Between every two vertices there is an edge of cost 2. Define $p(v_1, v_2) = 2$ and $p(v_1, v_i) = 1$ for every $3 \leq i \leq n$. All other penalties are zero.

The optimal solution has no edges. The cost of the edges is zero and the sum of penalties paid is $n$ for a total cost of $n$. A possible minimal solution is the path $v_2, v_3, \ldots, v_n$. The cost of the edges is $2(n - 2)$ and the payment of penalties is still $n$ for a total cost of $3n - 4$.

We now present the main algorithm of the chapter.

**Theorem 4.2.8.** There is a local ratio $(3 - \frac{4}{n})$-approximation algorithm for the PCGSF problem, where $n \geq 2$ is the number of vertices in the graph.

**Proof.** The pseudocode of algorithm $PCGSF(G, T, w, p)$ that solves this problem appears below. The proof is by induction on $|E| + k$. Given an instance $(G, T, w, p)$ of the problem, the functions $w$ and $p$ are decomposed by the algorithm, so that $w = w' + \delta$ and $p = p' + \gamma$, where the quadruple
Algorithm 3: \textit{PCGSF}(G, T, w, p)

\textbf{Input:} Graph $G = (V, E)$, collection $T = \{T_1, \ldots, T_k\}$, each a subset of $V$ of size at least 2, weight function $w : E \to \mathbb{R}^+$, penalty function $p : T \to \mathbb{R}^+$

\textbf{Output:} A forest $F \subset E$

if $T = \emptyset$ then
  return $\emptyset$

else

  The set of active vertices is defined as $Active \leftarrow \{v \in V|\exists i \; v \in T_i\}$

  For every edge $e \in E$, let $d(e)$ be the number of its active endpoints

  For every active vertex $v$, let $t(v)$ be an arbitrary subset $T_i$ for which $v \in T_i$

  For every $T_i \in T$, define $d(T_i) = |\{v \in T_i|t(v) = T_i\}|

  $\varepsilon \leftarrow \min(\{w(e)/d(e)|e \in E, d(e) \neq 0\} \cup \{p(T_i)/d(T_i)|T_i \in T, d(T_i) \neq 0\})$

  Define a weight function $w'$ as follows: $w'(e) = w(e) - d(e) \cdot \varepsilon$ for all $e \in E$

  Define a penalty function $p'$ as follows: $p'(T_i) = p(T_i) - d(T_i) \cdot \varepsilon$ for all $T_i \in T$

  Let $Z$ be the set of all edges $e \in E$ for which $w'(e) = 0$ and let $Z'$ be a spanning forest of $(V, Z)$

  Let $(G', \{T'_1, \ldots, T'_k\}, w'', p'')$ be obtained from $(G, T, w', p')$ by contracting the edges in $Z'$

  Let $T'$ be the collection of subsets $T'_i$ of size at least 2 for which $p''(T'_i) > 0$

  $F' \leftarrow \text{PCGSF}(G', T', w'', p'')$

  Let $F''$ be the forest obtained from the edges in $G$ corresponding to those in $F'$ together with $Z'$

  while there is a leaf in $F''$ which is not an active vertex do
    Remove from $F''$ the edge incident with that leaf
  return $F''$
The Prize Collecting Steiner Tree Problem

This section introduces the algorithm for the prize collecting Steiner tree (PCST) problem. An instance of this problem consists of a graph \( G = (V, E) \), a root vertex \( r \in V \), a subset \( U \subset V - \{r\} \) of active vertices, a weight function \( w : E \to \mathbb{R}^+ \), and a penalty function \( p : U \to \mathbb{R}^+ \). Given an instance \((G, r, U, w, p)\), the goal is to compute a tree rooted at \( r \) that minimizes the cost of the edges of the tree plus the penalties paid for vertices not in the tree. An active vertex is simply a vertex with positive penalty (the root vertex is not active). The set of active vertices is denoted by \( U \) in the problem instance. A minimal solution is a tree rooted at \( r \) whose leaves are active vertices (and possibly also \( r \)). In a degree-weighted instance corresponding to a graph \( G = (V, E) \), a root \( r \) and a subset \( U \subset V - \{r\} \) of active vertices, the weight function \( w(e) \) is equal to the number of active endpoints of the edge \( e \), whereas the penalty function satisfies \( p(v) = 1 \) for every \( v \in U \) and \( p(v) = 0 \) otherwise. When the edge \( \{u, v\} \) is contracted, the penalty of the
new vertex created is $p(u) + p(v)$. This is except for when an edge \{\(r, v\)\} incident with the root is contracted. In this case, the new vertex created is also called \(r\) and it still has zero penalty.

**Lemma 4.3.1.** Every solution to a degree-weighted instance of the PCST problem has a total cost of at least \(n\), where \(n\) is the number of active vertices.

*Proof.* Let \(q\) be the number of active vertices connected to \(r\) in the solution. Each such vertex must have some edge incident with it in the solution and therefore a cost of at least 1 is paid for this vertex. For each of the \(n - q\) active vertices that are not connected to \(r\), a penalty of 1 is paid. \(\blacksquare\)

**Lemma 4.3.2.** Every minimal solution to a degree-weighted instance of the PCST problem has a total cost of at most \(2n - 1\), where \(n\) is the number of active vertices.

*Proof.* Let \(q\) be the number of active vertices connected to \(r\) in the solution. An argument similar to the one used in the proof of Lemma 4.2.6 gives that the cost of the edges in a minimal solution is at most \(2q - 1\). The penalty paid is \(n - q\) for a total cost of \(2q - 1 + n - q = n + q - 1 \leq 2n - 1\), since \(q \leq n\). \(\blacksquare\)

The last two lemmas determine the approximation ratio of the algorithm. The purpose of the next lemma is to show that the analysis is indeed tight.

**Lemma 4.3.3.** For every \(n \geq 1\), there exists a degree-weighted instance of the PCST problem on a graph with \(n\) active vertices for which the optimal solution has total cost \(n\) whereas some minimal solution has a total cost of \(2n - 1\).

*Proof.* Consider the following instance with a root vertex \(r\) together with the vertices \(v_1, \ldots, v_n\). For every \(1 \leq i \leq n\), the vertex \(v_i\) is active and has a penalty of 1. Between every two vertices there is an edge of cost 2, except for edges between the root \(r\) and a vertex \(v_i\) that have a cost of 1.

The optimal solution has no edges. The cost of the edges is zero and the sum of penalties paid is \(n\) for a total cost of \(n\). A possible minimal solution is the path \(r, v_1, v_2, \ldots, v_n\). The cost of the edges is \(1 + 2(n - 1)\) and the payment of penalties is zero for a total cost of \(2n - 1\). \(\blacksquare\)
Theorem 4.3.4. There is a local ratio \(2 - \frac{1}{n-1}\)-approximation algorithm for the PCST problem, where \(n \geq 2\) is the number of vertices in the graph.

Proof. The pseudocode of algorithm PCST\((G, r, U, w, p)\) that solves this problem appears below. The proof is analogous to that of Theorem 4.2.8 using Lemmas 4.3.1 and 4.3.2 instead of Lemmas 4.2.5 and 4.2.6. Note that since the root is not an active vertex, the maximum number of active vertices is \(n - 1\). □

Algorithm 4: PCST\((G, r, U, w, p)\)

**Input:** Graph \(G = (V, E)\), root vertex \(r \in V\), subset \(U \subset V - \{r\}\) of active vertices, weight function \(w : E \rightarrow \mathbb{R}^+\), penalty function \(p : U \rightarrow \mathbb{R}^+\)

**Output:** A tree \(T \subset E\) rooted at \(r\)

if \(U = \emptyset\) then
  return \(\emptyset\)
else
  For every edge \(e \in E\), let \(d(e)\) be the number of its active endpoints
  \(\varepsilon \leftarrow \min\{\{w(e)/d(e)|e \in E, d(e) \neq 0\} \cup \{p(v)|v \in U\}\}\)
  Define a weight function \(w'\) as follows: \(w'(e) = w(e) - d(e) \cdot \varepsilon\) for every \(e \in E\)
  Define a penalty function \(p'\) as follows: \(p'(v) = p(v) - \varepsilon\) for every \(v \in U\)
  Let \(Z\) be the set of all edges \(e \in E\) for which \(w'(e) = 0\) and let \(Z'\) be a spanning forest of \((V, Z)\)
  Let \((G', r, U', w'', p'')\) be obtained from \((G, r, U, w, p)\) by contracting the edges in \(Z'\)
  \(T' \leftarrow PCST(G', r, U', w'', p'')\)
  Let the tree \(T''\) be the connected component of \(r\) in the union of the edges in \(G\) corresponding to those in \(T'\) together with \(Z'\)
  while there is a leaf in \(T''\) which is not an active vertex (or the root \(r\)) do
    Remove from \(T''\) the edge incident with that leaf
  return \(T''\)
4.4 Concluding Remarks

- The integral solution computed by our algorithm for the prize collecting generalized Steiner forest problem can be compared with the optimal solution of the natural LP for this problem. It can be verified that the algorithm is still a \((3 - \frac{1}{n})\)-approximation, even with respect to this optimal fractional solution. This is also implied by the equivalence between the local ratio method and the primal-dual schema.

- The algorithms presented in this chapter use the local ratio method which seems like the most natural, simple and time-efficient framework for addressing prize collecting Steiner tree problems. This should be further explored to determine whether this approach has applications for facility location problems and for the multicommodity rent-or-buy (MRoB) problem. The techniques presented in [69] might be helpful in enhancing the time performance of our algorithms.

- An interesting open problem is to decide whether there is a combinatorial algorithm for the prize collecting Steiner forest problem with an approximation factor better than 3. Improving upon the performance guarantee of the LP rounding 2.54-approximation algorithm is another intriguing challenge.
Bibliography


אלגוריתמים להעשות אופטימיזציה
בעשות וגרפים

חיבור Prints קבלת תואר "דוקטור לפילוסופיה"
מאת
שי גוטנר

בניהיתו של
פרופסור נoga אלון ופרופסור יוסי עזר

הוגש לתFFFFFFר של אוניברסיטת תל-אביב
ינואר 2009
In this paper, we address problems in networks. Specifically, we consider
online algorithms for admission control, graphs—close variants of
subproblems of the Steiner minimal tree and the set cover problem in
cuttings of fixed dimension. We prove

\[ T(n) = O\left(\log^2 m \log c\right) \]

for the case of a random algorithm - for the case of a non-random algorithm, an
algorithm of Feige and Korman [74] gives an \( \Omega\left(\log^2 m \log n\right) \)

and an \( \Omega\left(\log^2 n\right) \) lower bound. In both cases, the algorithm
do not use randomness and its time complexity is polynomial.

The main result is a new type of hash

\[ (\varepsilon,k) \]

families, which are for any \( k \) and \( \varepsilon > 0 \) polynomially large in

\[ (1+o(1))k \log n \]

and \( e^{1+o(1)} \log n \)

and, for any \( n \), \( [k] \)-subsets of \( [n] \) are polynomially large in

\[ \varepsilon=1/poly(k) \]

for any \( k \) and \( m \) large enough in terms of \( k \).

We also prove that the problem of

\[ \text{admission control} \]

is a generalization of the problem of covering with sets with

\[ \text{circuits} \]

of \( k \) vertices in a graph of size \( n \).
מספר השליטה של גרף \( G=(V,E) \) הוא הגודל המינימאלי של קבוצה שלטת \( U \subset V \), המקיימה שכל \( G-U \)-צומת ב \( V-U \) הוא שכן של לפחות צומת אחד מ \( U \). בפרק 3 אנו נותנים אלגוריתם בעל ז \( k=O(d_k) \) למציאת קבוצה שלטת בגודל לכל היותר \( k \) בגרף \( d \)-degenerated עם \( n \) צמתי. תוצאה זו מוכיחה שהקבוצה השלטת היא \( fixed-parameter tractable \) לגרפים \( d \)-degenerated. לגרפים שונים \( K_h \) אנו נותנים אלגוריתם מהיר יותר עם זמן \( O(h \log h) \) \( n \) הפעילות \( C_k \) קיים קבוצת שלטת \( k \)-כמעטarians פולינומי \( n \) מינור \( (O(h))^{3.5} k=O(n) \) והenever \( C_k \) הינו קבוע התלוי רק \( h \).

הveal עם \( T \) המוכללת עם שטיינר \( \delta \) הוא גרף \( (V,E) \), אוסף \( T=\{T_1,\ldots,T_k\} \) קבוצת \( T_i \)-שלטת \( V \) בגודל \( k \), פונקצית משקל \( w:E\rightarrow \mathbb{R}^{+} \), פונקצית קנס \( p:T\rightarrow \mathbb{R}^{+} \). המטרה היא למצוא יער \( F \) המוביל \( \delta \) הקשתות של \( F \) קבוצות \( r \) ארבעה יריבות \( (3-4/n) \) \( n \geq 2 \) יישום של שיטת \( local ratio \) אלגוריתם הקירוב \[ [56] \] הפרטי_ADD additional details here.}

[31] (כל הקונוטופים והאנספורים).
תכנית

המתמודדות עם נקשייה-מקוון, עלינו להתמקד באלגוריתמים יעילים עבור התוצאות האלגוריתמיות המוצגות. בעיות 
P=NP, והדבר מוביל באופן טבעי לחקור את פתרונות הקשיות ก็ hoạchים, של ו安くפוס או דה פולינומיאליים. הקושי 
ה]<=ריצת זמן פולינומיאלי ומקובלים סכום הבודד הידוע לבעיה קבוצה של בעיות י镣 הווה פולטימסיי עילית המקובלים המשועטים 
בעיה קבוצת בעיה היותursion קוברים כל הבודד להגלה של פתרונות המאובנים. הקושי העיקר של פתרונות המקובלים 
הונות העבירות שני היברקלילות המקולקלת בודק את כל הקוכל, בסעירה המבוך העיקרית של 
אלגוריתמים קוברים השבה ייחיב לירדן בום פולינומיים. 

גישת מעניינת נוספת להתמודדות עם נקשייה-מקוון, התחום החדש של -קשיות של בעיות ניתנת על 
יעד המחקר העיקרי של תחום זה הוא תכנון אלגוריתמים יעילים, אך옿ו בפרמטר מסוים, שזמן הב.ids ייעל 
המושג של parameterized complexity. המושג שלalgoיה שלפינות Больדוים,決定 הבודד שיעיותו בפרמטר מסוימ, מהו 
.exports המושファン העיקר ייעל סיבוכיות הביעה. 

דרק הארת לחנה את-ה. parameterized complexity-=size של בעיה, היא באמצעה בניה של 
kernels. המושפע של pszate שלalgoיה שלפינות Больדוים,決定 הבודד שיעיותו בפרараметרים של 
problem kernels בודד. נ sonra לחש HEAP בודד. ayud להעליות של algoיה שלפינות Больדוים,決定 הבודד שיעיותו בפרараметרים של 
אלגוריתמים קוברים ומקובלים, והם שיעיותו גוסับ עליות משונות. 

בהתואמה עם המחקרים היום, קיימים מביתים ומקובלים של עליות ואפספוריות.
פרק ראשון: בעיית ה-

** Admission Control

** בעיות מCoreApplication

הפרק מתאר בעיית ה-

** Admission Control

ברשסתיה, ניתן למצוא את הפרק ב巀יעם אלגוריתם מתאמה לשמשת הצהרת עד-

** admission control

باشرחתו, אך המטרה היא להיביא למינום את מסFTER הדוחית שמתאמה לيري

** admission control

בפרק שני, אנו מציגים את השם החדש של משפחות מהוזלות של פונקציות

** hash

ןברמסים בגרף, ומודגמים את היישומים שלן. הפרק מכיל דוגים

** parameterized complexity

כותר, וברכתי גם בגרף עניין, אך ניתן למצוא בו מדריך של פער

** fixed-parameter tractable

בפרק שלישי, נדון בקואזיקון של גרפים המוגדרים

** parameterized complexity

בגרפים הם

** degenerated

שכפ стали באמצעות מכרי אלגוריתמים, ואלו

** problem kernels

וחזרי אלגוריתמים ומידיקים שונים, אך בגרפים עם מינור

** fixed-parameter tractable

כברцикл, זו כותרת בגרף עניין, אך ניתן למצוא בו מדריך של פער

** fixed-parameter tractable

בפרק הרביעי, נדון במחקר של алгорיתמים קרוב

** admission control

הколо, בכיסוי לשוני לא רכיבי והמסירים

** admission control

לעומת על עם ק市の

** admission control

הופך טיליר

** admission control

הקסט

** admission control

בהפר

** admission control

לעומת על עם ק市の

** admission control

נראה את התחום של כל פרק הפרק.
The problem of admission control, denoted by OPT, is to decide whether and how to accept requests on a given path, without deferring requests that have already been accepted or deferred previously.

We define the following problem:

- A directed graph $G = (V, E)$, where $|E| = m$.
- Each edge $e$ has a capacity $c_e > 0$.
- The capacity is denoted $c = \max_{e \in E} c_e$.
- A request arrives in the order $r_1, r_2, \ldots$.
- Each request $r_i$ is a simple path in the graph $r_i$ with a cost $p_i > 0$.
- A feasible solution is a set of paths such that each path contains at most $c$ requests and the total cost is minimized.

The problem is to find a feasible solution with the minimum total cost. The algorithm maintains a feasible solution at all times, and if an edge is accepted, it is deferred if the path contains the edge.

The admission control problem is known to be NP-hard, and the best known approximation algorithm for it is $O(\sqrt{m})$.

In a recent paper by Blum, Kalai, and Kleinberg [37], they present two deterministic algorithms with competitive ratios of $O(\sqrt{m})$ and $c + 1$. The first algorithm is based on the Santha-Vazirani mechanism, and the second is based on the optimal algorithm for the problem of covering sets with a minimum number of sets.

The algorithm maintains a feasible solution at all times, and if an edge is accepted, it is deferred if the path contains the edge. This ensures that the total cost is minimized.

In conclusion, the problem of admission control is a fundamental problem in network design, and its solution has implications for the design of efficient and scalable systems.
The main result we present in this section is a randomized algorithm $O(\log^2 (mc))$-competitive for the admission control problem, under the assumption that $BPP \neq NP$.

We also give a randomized algorithm $O(\log m \log n)$-competitive for the admission control problem with repetitions.

N. Alon, Y. Azar and S. Gutner, Admission control to minimize rejections and online set cover with repetitions, Proc. of the 17th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2005), 238-244.

Hash: Persistent Matroids and Applications to Color Coding

The technique of Color Coding allows us to determine in an efficient manner whether a given input graph contains a path of length $k$ or a cycle in an oriented or not oriented graph. The first time this approach was presented in a paper, the basic idea was simple. The algorithm is randomized at first, and then it is transformed into a deterministic randomized algorithm, where the vertices of $G=(V,E)$ are colored randomly in $k$ colors. If the colors of all vertices are distinct, the algorithm terminates. If not, the algorithm is repeated for $k^2$ colors, and the process is repeated until the coloring is complete. The probability of finding a path of length $k$ in a random coloring is $e^{-k}/k^k$, and repeating the process $Ce^k$ times provides a deterministic algorithm whose probability of failure is less than $e^{-C}$. The crucial aspect of the randomization of this algorithm is the use of known families of hash functions presented in a paper [79], [55], which provide a formal family of $O(k) \log |V|$ colorings of the vertices of $G$-simplest colorings where the elements of each color class are different. This can be done simply by running the dynamic programming algorithm for each coloring, and then obtaining a deterministic algorithm for the problem. There have recently been applications of this technique in biology [80], [81], [82], [64], [83], [81]. This is related to the usual use of hash functions.

A family of hash functions $[n]-k$-self-colorings of $[n]$ is defined as follows. A function $f: [n] \rightarrow [m]$ is a $k$-self-coloring if for any $S \subseteq [n]$ with $|S|=k$, the number of $k$-colorings of $S$ such that for any $i \in S$, $f(i)$ is different from $f(j)$ for any $j \in S, j \neq i$ is at least $(1-\epsilon)T$.
In the family $k$-M (for $k$-independent), it is perfectly balanced if it is $(0, k)$-balanced, meaning it is $(\epsilon, k)$-balanced for some $\epsilon > 0$.

By comparing [52] Mokhičha-Flum and Grohe-Flum shows that the family is perfectly balanced if it is $(0, k)$-balanced, meaning it is $(\epsilon, k)$-balanced for some $\epsilon > 0$. In the case of two or more different $\epsilon$ values, it becomes complicated to define a family of perfectly balanced families.

In the article [52], Flum and Grohe proved that the problem of counting exactly the number of $k$-cycles or paths of length $k$ in a graph is complete in $\text{W}[1]$ for any $k$. Therefore, it is not expected to have an algorithm with running time $f(k) \cdot n^c$ for counting exactly the number of $k$-cycles or paths of length $k$ in a graph of size $n$ for any function $f: \mathbb{N} \to \mathbb{N}$.

However, Arvind and Raman [24] have developed a randomized fixed-parameter tractable algorithm for counting the number of $k$-trees or $k$-graphs that are complete in $\text{W}[1]$. This means that there is no algorithm with running time $f(k) \cdot n^c$ for counting exactly the number of $k$-trees or $k$-graphs in a graph of size $n$ for any function $f: \mathbb{N} \to \mathbb{N}$.

The results of Flum and Grohe mentioned above indicate that there is no family of perfectly balanced families of hash functions from $\mathbb{N}$ to $\mathbb{N}$ that is complete in $\text{W}[1]$. Therefore, it is not expected to have an algorithm with running time $f(k) \cdot n^c$ for counting exactly the number of $k$-cycles or paths of length $k$ in a graph of size $n$ for any function $f: \mathbb{N} \to \mathbb{N}$.

The main result of our work is a constructive algorithm that provably counts exactly the number of $k$-cycles or paths of length $k$ in a graph of size $n$ for any function $f: \mathbb{N} \to \mathbb{N}$.

In the article [13], we noted that the problem of counting exactly the number of $k$-cycles or paths of length $k$ in a graph is complete in $\text{W}[1]$ for any $k$. Therefore, it is not expected to have an algorithm with running time $f(k) \cdot n^c$ for counting exactly the number of $k$-cycles or paths of length $k$ in a graph of size $n$ for any function $f: \mathbb{N} \to \mathbb{N}$.

In the article [24], Arvind and Raman have developed a randomized fixed-parameter tractable algorithm for counting the number of $k$-trees or $k$-graphs that are complete in $\text{W}[1]$. This means that there is no algorithm with running time $f(k) \cdot n^c$ for counting exactly the number of $k$-trees or $k$-graphs in a graph of size $n$ for any function $f: \mathbb{N} \to \mathbb{N}$.
Theorem:

For this problem, the following references are provided:

- N. Alon and S. Gutner,
  Balanced families of perfect hash functions and their applications,
  Proc. of the 34th International Colloquium on Automata, Languages and Programming (ICALP 2007), 435-446.

- N. Alon and S. Gutner,
  Balanced hashing, color coding and approximate counting, submitted.

Chapter 3: Identifying Control Sets on Graphs

Definition:

In this chapter, the control set on graphs with minors forbidden is the input to a parameterized problem, given as a pair $(x,k)$, where $x$ is the instance, and $k$ is the parameter.

A parameterized problem is fixed-parameter tractable if there exists an algorithm that can solve it in time $f(k) \cdot n^c$, for a computable function $f: \mathbb{N} \to \mathbb{N}$ and constant $c$.

A kernel algorithm is a function that can be computed in polynomial time on $(x,k)$ and returns an equivalent instance $(x',k')$ such that $k' \leq k$ and $|x'| \leq g(k)$ for a computable function $g: \mathbb{N} \to \mathbb{N}$.

The problem kernel of a fixed-parameter problem is the set of equivalent instances that can be obtained from the original problem.

A fixed-parameter problem is fixed-parameter tractable if it has a non-trivial kernel.

For the problem of identifying control sets on graphs, the kernel algorithm provides a way to reduce the problem to a smaller instance, which can be solved more efficiently.

The problem of identifying control sets on graphs is complete for W[2], which means that it is as hard as the hardest problems in W[2].
For each function \( f: N \rightarrow N \) and constant \( c \), the problem of finding a solution of size \( k \) in a \( k \)-degenerated graph with \( n \) vertices is fixed-parameter tractable.

The above approach is useful for problems that are fixed-parameter tractable for graphs with a certain property, such as degeneracy. The problem kernel technique can be used to reduce the size of the problem to a constant factor, resulting in an algorithm with a runtime of \( O(n \log h) \) for graphs without \( K_h \) minors.

A generalization of this approach, due to Alber et al., shows that for graphs with a certain property (such as being planar), it is possible to achieve a problem kernel of size \( O(k^2) \) for the problem of finding a solution of size \( k \) in a graph with \( n \) vertices.

In the above, we have assumed that the graph is simple (i.e., it has no loops or multiple edges). This assumption is important because it allows us to use certain properties of the graph, such as its connectivity and planarity, to simplify the problem.

In conclusion, the approach described above provides a powerful tool for solving problems on graphs, especially those that are fixed-parameter tractable. By using problem kernels and algorithmic reductions, we can achieve efficient algorithms for a wide range of problems on graphs.
Akra-Bazzi inequality gives an upper bound of $O(n)$ for the problem of finding a Hamiltonian cycle in a graph with a fixed number of vertices.

We improve the randomized algorithm presented in [41] by using a deterministic algorithm with $O(n \log n)$ time complexity for finding Hamiltonian cycles in graphs with no forbidden minors. We also show a deterministic algorithm with $O(n)$ runtime for finding cycles of length 5, and explain why it is not possible to achieve this for longer cycles.


Chapter 4: Cutting Across Steiner Trees

The goal of this chapter is to present local ratio techniques for designing simple algorithms whose analysis is easy and that can be used to design algorithms for other problems. The chapter presents a systematic approach to solving Steiner tree problems. The PCGSF model of the problem is defined as follows:

PCGSF: Let $G=(V,E)$ be an undirected graph, $T_i\subseteq V$ for each $i$, and $w:E\to \mathbb{R}^+$ be a weight function on the edges. The goal is to find a minimum-weight subgraph $F$ that connects all the $T_i$ vertices. The local ratio technique guarantees that the solution is within a factor of $\log |V|$ of the optimal solution.

References:


Key references:

The paper focuses on the prize-collecting Steiner tree problem (PCGSF) and related problems, specifically with a private case. The authors discuss the connections between these problems and the Steiner tree problem. They present an algorithm based on LP relaxation that achieves a 2.54-approximation ratio. Additionally, they introduce a primal-dual algorithm with a 3-approximation ratio using Farkas' Lemma.

The paper also highlights the connection between these problems and the local ratio algorithm, particularly by Bar-Yehuda, which achieves a 2-approximation ratio for the Steiner tree problem. The authors present new results that improve upon existing algorithms and their approximations.

**References:**


Also: Information Processing Letters 107 (2008), 39-44.