Approximability of Cycle Covers and Smoothed Analysis of Binary Search Trees

Bodo Manthey

Dissertation

Universität zu Lübeck
Institut für Theoretische Informatik

Aus dem Institut für Theoretische Informatik
der Universität zu Lübeck
Direktor: Prof. Dr. Rüdiger Reischuk

Approximability of Cycle Covers and Smoothed Analysis of Binary Search Trees

Inauguraldissertation

zur Erlangung der Doktorwürde
der Universität zu Lübeck
aus der Technisch-Naturwissenschaftlichen Fakultät

Vorgelegt von

Bodo Manthey

Lübeck, Dezember 2005

Berichterstatter:	Prof. Dr. Rüdiger Reischuk Prof. Dr. Kurt Mehlhorn Prof. Dr. Heribert Vollmer
Vorsitz des Prüfungsausschusses:	Prof. Dr. Walter Dosch
Tag der mündlichen Prüfung:	7. Dezember 2005

ACKNOWLEDGEMENTS

First of all, I thank my *Doktorvater* Rüdiger Reischuk for his support throughout the years.

I also thank the current and former members of the Institut für Theoretische Informatik at the Universität zu Lübeck for many valuable discussions, not only about computer science. In particular, I am indebted to Markus Bläser from whom I learned a lot about approximation algorithms.

Special thanks go to Jan Arpe for patiently listening to all my ideas and mistakes and to him and Martin Böhme for carefully proofreading this thesis.

Finally, I thank my wife Sandra for her encouragement and my son Falk; a couple of thoughts evolved while I was taking him for a walk in his pram.

CONTENTS

ABSTRACT

In the first part of this thesis, we are concerned with the approximability of restricted cycle covers. A cycle cover of a graph is a set of cycles such that every vertex is part of exactly one cycle. An L-cycle cover is a cycle cover in which the length of every cycle is in the set $L \subseteq \mathbb{N}$. A special case of L-cycle covers are k-cycle covers for $k \in \mathbb{N}$, where the length of each cycle must be at least k. The weight of a cycle cover of an edge-weighted graph is the sum of the weights of its edges.

We come close to settling the complexity and approximability of computing L-cycle covers. On the one hand, we show that for almost all L, computing L-cycle covers of maximum weight in directed and undirected graphs is APX-hard and NP-hard. Most of our hardness results hold even if the edge weights are restricted to zero and one. On the other hand, we show that the problem of computing L-cycle covers of maximum weight can be approximated with factor 2.5 for undirected graphs and with factor 3 in the case of directed graphs. Finally, we show that 4-cycle covers of maximum weight in graphs with edge weights zero and one can be computed in polynomial time.

As a by-product, we prove that the problem of computing minimum vertex covers in λ-regular graphs is APX-complete for every $\lambda \geq 3$.

In the second part of this thesis, we are concerned with binary search trees, one of the most fundamental data structures. While the height of such a tree may be linear in the worst case, the average height with respect to the uniform distribution is only logarithmic. The exact value is one of the best studied problems in average-case complexity.

We investigate what happens in between these two cases by analysing the smoothed height of binary search trees: Randomly perturb a given (adversarial) sequence and then take the expected height of the binary search tree generated by the resulting sequence. As perturbation models, we consider partial permutations, partial alterations, and partial deletions.

On the one hand, we prove tight lower and upper bounds of roughly $\Theta(\sqrt{n})$ for the expected height of binary search trees under partial permutations and partial alterations. This means that worst-case instances are rare and disappear under

slight perturbations. On the other hand, we examine how much a perturbation can increase the height of a binary search tree, i.e. how much worse well-balanced instances can become. We show that under all three perturbation models, the height can increase exponentially.

CHAPTER

ONE

Introduction

1.1 Restricted Cycle Covers

The *travelling salesman problem* (TSP) is one of the most well-known combinatorial optimisation problem; in fact, there are books devoted solely to the TSP [50, 63]. An instance of the TSP is a complete graph with edge weights, and the aim is to find a minimum or maximum weight cycle that visits every vertex exactly once. Such a cycle is called a *Hamiltonian cycle*. The TSP has a variety of applications, ranging from routing problems and computational biology [73], where general edge weights are used, to code optimisation and frequency assignment problems [46, 93], where the TSP is restricted to two different edge weights.

Because the TSP is NP-hard [47, ND22+23], we cannot hope to always find an optimal cycle efficiently. For practical purposes, however, it is often sufficient to obtain a cycle that is close to optimal. In such cases, we require *approximation algorithms*, i.e. polynomial-time algorithms that compute such near-optimal cycles.

The problem of computing *cycle covers* is a relaxation of the TSP: A cycle cover of a graph is a spanning subgraph consisting solely of cycles such that every vertex is part of exactly one cycle. Thus, a solution to the TSP is a cycle cover consisting of a single cycle. In analogy to the TSP, the weight of a cycle cover in an edge weighted graph is the sum of the weights of its edges.

In contrast to the TSP, cycle covers of maximum weight can be computed efficiently. This fact is exploited in approximation algorithms for the TSP; the computation of cycle covers forms the basis for the currently best known approximation algorithms for the maximum TSP [26], maximum asymmetric TSP (ATSP) [58], metric minimum ATSP [58], metric maximum TSP [25], metric maximum ATSP [19], maximum ATSP with weights zero and one, minimum ATSP with weights one and two [14], minimum TSP with strengthened triangle

inequality [23], and minimum ATSP with strengthened triangle inequality [17,24]. Furthermore, the currently best known approximation algorithm for the shortest common superstring problem in computational biology also relies on computing cycle covers [90]. These algorithms usually start by computing an initial cycle cover and then join the cycles to obtain a Hamiltonian cycle. This technique is called *subtour patching* [49].

Short cycles in a cycle cover limit the approximation ratios achieved by such algorithms. In general, the longer the cycles in the initial cover are, the better the approximation ratio. Thus, we are interested in computing cycle covers that do not contain short cycles. Moreover, there are approximation algorithms that behave particularly well if the cycle covers that are computed do not contain cycles of odd length [17]. Finally, some so-called vehicle routing problems (see e.g. Hassin and Rubinstein [54]) require vertices to be covered with cycles of bounded length.

Therefore, we consider *restricted cycle covers*, where cycles of certain lengths are ruled out a priori: Let $L \subseteq \mathbb{N}$, then an *L-cycle cover* is a cycle cover in which the length of each cycle is in L. For directed graphs, we assume $L \subseteq \{2, 3, 4, \ldots\}$, while $L \subseteq \{3, 4, 5, \ldots\}$ in the case of undirected graphs. A special case of L-cycle covers are *k-cycle covers*, which are defined to be $\{k, k+1, k+2, \ldots\}$-cycle covers: the length of every cycle must be at least k.

To fathom the possibility of designing approximation algorithms based on computing cycle covers, we aim to characterise the sets L for which L-cycle covers of maximum weight can efficiently be computed.

If, for a given L, computing L-cycle covers is NP-hard, there may still be good approximation algorithms based on L-cycle covers: Suppose that efficient computability of L-cycle covers of maximum weight provides a factor r approximation algorithm for some optimisation problem. Then a *polynomial-time approximation scheme* (*PTAS*, see Section 2.4) for computing L-cycle covers of maximum weight would usually yield a factor $r + \epsilon$ approximation algorithm for arbitrarily small $\epsilon > 0$ for this maximisation problem, which is only slightly worse than approximation ratio r. Thus, APX-hardness results are of special importance for L-cycle cover problems since they rule out the possibility of designing approximation algorithms that are based on polynomial-time approximation schemes for L-cycle covers of maximum weight.

Beyond being a basic tool for approximation algorithms, cycle covers are interesting in their own right: *Matching theory* and *graph factorisation* are important topics in graph theory. The classical matching problem is the problem of finding *one-factors*, i.e. spanning subgraphs in which every vertex is incident to exactly one edge. Cycle covers of undirected graphs are also known as *two-factors* since every vertex is incident to exactly two edges in a cycle cover.

There are various extensions of matchings and factorisations: An *f-factor* or *degree factor* is a spanning subgraph in which the degree of every vertex v is $f(v) \in \mathbb{N}$. Given a set $\mathcal{H}$ of graphs, an *$\mathcal{H}$-factor* or *component factor* is

a partition of a graph into components each of which is isomorphic to some graph in $\mathcal{H}$. A considerable amount of research has been done on graph factors, both on structural properties of graph factors (cf. Lovász and Plummer [64] and Schrijver [85]) and on the complexity of finding graph factors (cf. Hell [55], Kirkpatrick and Hell [60], and Schrijver [85]). In particular, the complexity of finding restricted two-factors, i.e. L-cycle covers in undirected graphs, has been investigated, and Hell, Kirkpatrick, Kratochvíl, and Kríz [56] showed that finding L-cycle covers in undirected graphs is NP-hard for almost all L. However, almost nothing is known so far about the complexity of finding directed L-cycle covers.

In the first part of this thesis, we are concerned with the complexity of finding restricted cycle covers of maximum weight.

On the one hand, we prove that for almost all L, the problem of computing L-cycle covers of maximum weight is APX-hard and thus cannot be approximated arbitrarily well unless P = NP. More precisely: For undirected graphs, computing L-cycle covers of maximum weight is APX-hard for all L with $L \not\supseteq \{5, 6, 7, \ldots\}$, even if we allow only zero and one as edge weights. If we additionally allow two as an edge weight, the problem becomes APX-hard for all L with $L \not\supseteq \{4, 5, 6, \ldots\}$. For directed graphs, we show a dichotomy: Computing L-cycle covers of maximum weight is APX-hard if $L \neq \{2\}$ and $L \neq \{2, 3, 4, \ldots\}$ and solvable in polynomial time otherwise. This holds even if we only allow zero and one as edge weights.

On the other hand, we devise polynomial-time approximation algorithms for L-cycle covers that achieve approximation ratios of 2.5 and 3 for undirected and directed graphs, respectively. These algorithms work uniformly for all L, although most sets L are not recursive, and hence testing whether a graph possesses an L-cycle cover is not recursive either. Finally, we show that 4-cycle covers of maximum weight in graphs with edge weights zero and one can be computed in polynomial time.

While we have settled the complexity for directed graphs, the complexity of five undirected cycle cover problems remains open: finding 5-cycle covers in graphs and computing 5-cycle covers of maximum weight in complete graphs with edge weights zero and one, the same two problems for $\{3, 5, 6, 7, \ldots\}$-cycle covers, and computing 4-cycle covers of maximum weight in graphs with general edge weights.

1.2 Smoothed Analysis of Binary Search Trees

In the first part of this thesis, we deal with *worst-case* complexity: the complexity of a problem is measured by means of its most difficult instances. Worst-case complexity has two major advantages: it is often easy to analyse, and if the worst-case complexity is low, then the problem considered is easy or the algorithm considered behaves well, no matter which instances actually occur in the

application at hand.

A drawback of worst-case analysis is that it is utterly pessimistic: worst-case instances are often specially constructed to show that some algorithm performs poorly, but they rarely occur in practice. A challenge in algorithmics is the analysis of algorithms that are known to work well in practice but whose worst-case performance is bad.

Average-case analysis was introduced to provide a less pessimistic measure, and indeed many practical algorithms perform much better on random inputs. The results obtained, however, may not match the algorithm's real-world performance: The instances encountered in applications often bear little resemblance to the random inputs that dominate the average-case analysis.

To explain the discrepancy between the average-case and worst-case behaviour of the simplex algorithm, Spielman and Teng introduced the notion of *smoothed analysis* [86,89]. Smoothed analysis interpolates between average-case and worst-case analysis: Instead of taking the worst-case instance or, as in average-case analysis, choosing an instance completely at random, we analyse the complexity of (worst-case) objects subject to slight random perturbations, i.e. the expected complexity in a small neighbourhood of (worst-case) instances. Smoothed analysis takes into account the fact that a *typical* instance is not necessarily a *random* instance and that worst-case instances are usually artificial and rarely occur in practice.

Let C be some complexity measure. The worst-case complexity is $\max_x C(x)$, and the average-case complexity is $\mathbb{E}_{x\sim\Delta}C(x)$, where $\mathbb{E}$ denotes expectation with respect to a probability distribution Δ (typically the uniform distribution). The smoothed complexity is defined as $\max_x \mathbb{E}_{y\sim\Delta(x,p)}C(y)$. Here, x is chosen by an adversary, and y is randomly chosen according to some probability distribution $\Delta(x,p)$ that depends on x and a parameter p. The distribution $\Delta(x,p)$ should favour instances in the vicinity of x. This means that $\Delta(x,p)$ should put almost all of its weight on the neighbourhood of x, where "neighbourhood" has to be defined appropriately depending on the problem considered. The smoothing parameter p denotes how strongly x is perturbed, i.e. we can view it as a parameter for the size of the neighbourhood of x. Intuitively, for $p = 0$, smoothed complexity becomes worst-case complexity, while for large p, the perturbation overwhelms the original instance and smoothed complexity becomes average-case complexity.

Smoothed complexity can be interpreted as follows: If the smoothed complexity of an algorithm is low, then we must be unlucky to accidentally hit an instance on which our algorithm behaves poorly, even if the worst-case complexity of our algorithm is bad. In this situation, worst-case instances are isolated events.

For continuous problems, Gaussian perturbations seem to be a natural perturbation model: they are concentrated around their mean, and the probability that a perturbed number deviates from its unperturbed counterpart by distance d decreases exponentially in d. Thus, such probability distributions favour instances in the neighbourhood of the adversarial instance. The smoothed complexity of

continuous problems seems to be well understood. There are, however, only few results for the smoothed analysis of discrete problems. For such problems, even the term "neighbourhood" is often not well defined. Thus, special care is needed when defining perturbation models for discrete problems. Perturbation models should reflect "natural" perturbations, and the probability distribution for an instance x should be concentrated around x, particularly for small values of the smoothing parameter p.

In the first part of this thesis, we consider the complexity, and particularly the approximability, of optimisation problems. Analysing approximation properties of optimisation problems yields a finer classification of the worst-case complexity of these problems. Classifying optimisation problems based on approximability reflects the worst-case complexity of optimisation problems realistically since approximate solutions often suffice in practical applications. The smoothed complexity of NP optimisation problems, which classifies optimisation problems based on typical instances instead of worst-case instances, was studied by Beier, Röglin, and Vöcking [12,79]. For a large class of optimization problems, they proved that a problem has polynomial smoothed complexity if and only if it has a randomised pseudo-polynomial-time algorithm.

In the second part of this thesis, we will conduct a smoothed analysis of an ordering problem: We will examine the *smoothed height of binary search trees*.

The binary search tree is one of the most fundamental data structures and is used as a building block for many advanced data structures. The main criterion for the "quality" of a binary search tree is its height, i.e. the length of the longest path from the root to a leaf. A search tree containing n elements is considered *efficient* if its height is $O(\log n)$. If the height of a tree is $\Omega(n^\delta)$, particularly for δ close to 1, then this tree is *inefficient* in the sense that the advantage of search trees over lists vanishes.

Unfortunately, in the worst case, the height is equal to the number of elements, namely for totally unbalanced trees generated by an ordered sequence of elements. On the other hand, if a binary search tree is chosen at random, then the expected height is only logarithmic in the number of elements (more details will be discussed in Section 6.2). Thus, there is a huge discrepancy between the worst-case and the average-case behaviour of binary search trees.

We will analyse what happens in between: An adversarial sequence is perturbed randomly, and then the height of the binary search tree generated by the sequence thus obtained is measured. Thus, our instances are neither adversarial nor completely random. As perturbation models, we consider *partial permutations*, *partial alterations*, and *partial deletions*. For all three, we show tight lower and upper bounds: Under partial permutations and partial alterations, the smoothed height is roughly $\Theta(\sqrt{n})$, while under partial deletions, the smoothed height is $\Theta(n)$ (n is the number of elements in the unperturbed sequence). As a by-product, we also obtain tight bounds for the smoothed number of left-to-right maxima, i.e. the number of new maxima seen when scanning a sequence from left

to right, thus improving a result by Banderier et al. [10].

In smoothed analysis, one analyses how fragile worst-case instances are. We suggest examining also the dual property: Given a good (or best case) instance, how much can the complexity increase if the instance is perturbed slightly? In other words, how stable are best-case instances under perturbations? For binary search trees, we show that there are best-case instances that are indeed not stable, i.e. there are sequences that yield trees of logarithmic height, but slightly perturbing these sequences yields trees of polynomial height.

1.3 Outline

The first part of this thesis deals with cycle covers. Chapter 2 contains general preliminaries that are also used in the second part. It then introduces cycle covers and reviews the complexity theory of combinatorial optimisation problems. We also give a summary of our results in this chapter. We then prove that in general, restricted cycle covers of maximum weight are hard to approximate (Chapter 3). In Chapter 4, we present (approximation) algorithms for restricted cycle covers. The first part ends with a summary and some remarks regarding the problems that remain open (Chapter 5). Some of the APX-hardness results for cycle covers have already been published as joint work with Markus Bläser [15, 16, 20]. All three papers contain inapproximability results and algorithms. In this thesis, I include only the inapproximability results, which were proved mainly by me, while Markus Bläser mainly developed the algorithms. Most of the results of the first part were presented at the 3rd Workshop on Approximation and Online Algorithms [65].

In the second part, we are concerned with smoothed analysis of binary search trees. In Chapter 6, we review previous results on smoothed analysis and binary search trees, introduce some notation, and state our results. Then, we introduce the perturbation models (Chapter 7). We prove tight bounds for the height of binary search trees and for the number of left-to-right maxima under all three models in Chapter 8. Chapter 9 deals with the stability of perturbations. The second part concludes with some conjectures and an outlook on future research (Chapter 10). Most of the results of the second part will be presented at the 16th Annual International Symposium on Algorithms and Computation [66].

Part I

The Approximability of Restricted Cycle Covers

CHAPTER

TWO

Cycle Covers and Combinatorial Optimisation

2.1 General Preliminaries

We denote by $\mathbb{N} = \{0, 1, 2, \ldots\}$, $\mathbb{Z} = \{\ldots, -1, 0, 1, \ldots\}$, and $\mathbb{R}$ the set of natural numbers, integers, and real numbers, respectively. For $n \in \mathbb{N}$, we define $[n] = \{1, 2, \ldots, n\}$. Furthermore, let $[n - \frac{1}{2}] = \{\frac{1}{2}, \frac{3}{2}, \ldots, n - \frac{1}{2}\}$. For $a, b \in \mathbb{R}$, $[a, b]$ denotes the closed interval $\{x \in \mathbb{R} \mid a \leq x \leq b\}$, while $[a, b)$ denotes the half-open interval $\{x \in \mathbb{R} \mid a \leq x < b\}$.

Let M be a set, then $\mathcal{P}(M)$ denotes the power set of M. If M is a finite set, then $|M|$ denotes the cardinality of M.

We denote probabilities by $\mathbb{P}$ and expectations by $\mathbb{E}$.

The logarithms to base e and 2 are ln and log, respectively, while exp denotes the exponential function to base e. The twice iterated logarithm $\log \circ \log$ is abbreviated by llog.

For a function $f : \mathbb{N} \to \mathbb{N}$, we denote by

$$O(f) = \{g : \mathbb{N} \to \mathbb{N} \mid \exists c, n_0 \in \mathbb{N}\, \forall n \geq n_0 :\ g(n) \leq cf(n)\}$$

the set of functions that grow asymptotically at most as fast as f. Conversely, $\Omega(f) = \{g : \mathbb{N} \to \mathbb{N} \mid f \in O(g)\}$ is the set of functions that grow at least as fast as f. The set $\Theta(f) = O(f) \cap \Omega(f)$ contains all functions that grow asymptotically as fast as f. Finally, we denote the set of functions that grow slower than f by $o(f) = O(f) \setminus \Theta(f)$ and the set of functions that grow faster than f by $\omega(f) = \Omega(f) \setminus \Theta(f)$.

We denote by P the class of decision problems that are deterministically decidable in polynomial time. The class NP contains all decision problems that are nondeterministically decidable in polynomial time. For more on complexity theory, we refer to Papadimitriou [69].

2.2 Graphs and Cycles

In this section, we introduce some graph theoretical notations used in the subsequent chapters. For more on graph theory, see for instance Jungnickel [57].

Let V be any finite set. Then $U(V) = \{\{u, v\} \mid u, v \in V, u \neq v\}$ denotes the set of undirected edges that connect elements in V and $D(V) = \{(u, v) \mid u, v \in V, u \neq v\}$ denotes the set of directed edges that connect elements in V. A **graph** $G = (V, E)$ consists of a finite set V of vertices and a set E of edges. If $E \subseteq U(V)$, then we call G an undirected graph, if $E \subseteq D(V)$, we call G a directed graph. Graphs are simple by definition; E is a set, not a multiset, and there are no edges connecting a vertex to itself. Thus, multiple edges or loops are not allowed.

The graphs $(V, U(V))$ and $(V, D(V))$ are called the **undirected** and **directed complete graph** on $|V|$ vertices, respectively.

Undirected Graphs. Let $G = (V, E)$ be an undirected graph. We say that an edge $e = \{u, v\} \in E$ is **incident** to its endpoints u and v. Two vertices u and v are **adjacent** in G if $\{u, v\} \in E$. Two edges e and f are **adjacent** if $e \cap f \neq \emptyset$. The degree $\deg_G(v)$ of a vertex $v \in V$ in G is the number of edges in E that are incident to v. If it is clear from the edge set which graph we are talking about, we write $\deg_E(v)$ instead of $\deg_G(v)$. A graph G is called **λ-regular** if $\deg_G(v) = \lambda$ for all vertices $v \in V$.

We say that $G' = (V', E')$ is a **subgraph** of G if $V' \subseteq V$ and $E' \subseteq E \cap U(V')$. The graph G' is an **induced subgraph** of G if $E' = E \cap U(V')$.

A **matching** of G is a subset $M \subseteq E$ of edges such that no vertex in G is incident to more than one edge in M, i.e. $\deg_M(v) \leq 1$ for all $v \in V$.

A **path** in G is a sequence $P = (x_0, x_1, \ldots, x_k)$ of pairwise distinct vertices such that $e_i = \{x_{i-1}, x_i\} \in E$ for all $i \in [k]$. We also think of P as a sequence of edges $(e_1, \ldots, e_k)$. The length of a path P is the number of edges it consists of. A single vertex is a path of length zero.

A **cycle of length k** in G is sequence $(x_1, x_2, \ldots, x_k, x_1)$ of vertices, where $x_1, \ldots, x_k$ are pairwise distinct vertices and $\{x_1, x_2\}, \{x_2, x_3\}, \ldots, \{x_k, x_1\} \in E$ are pairwise distinct edges. We do not consider a single vertex as a cycle. Since we do not allow multiple edges, the shortest possible cycles in undirected graphs have length three. A **Hamiltonian cycle** of G is a cycle that contains all vertices of G.

A subgraph $C = (V, E')$ of G is called a **cycle cover** of G if C consists solely of cycles and every vertex of V is part of exactly one cycle. Another characterisation is that C is a cycle cover of G if $\deg_C(v) = 2$ for all $v \in V$, i.e. C is a two-regular spanning subgraph of G. Due to this characterisation, cycle covers of undirected graphs are also called *two-factors*. Figure 2.2.1 shows an example of a cycle cover. We usually consider cycle covers as sets of edges, i.e. we consider C to be identical to E'.

(a) An undirected graph. (b) A cycle cover (solid edges) of the graph.

Figure 2.2.1: An example of a cycle cover of an undirected graph.

Let $\mathcal{U} = \{3, 4, 5, \ldots\}$ and $L \subseteq \mathcal{U}$. A cycle cover C is called an ***L*-cycle cover** if the length of every cycle in C belongs to L. A special case of L-cycle covers are k-cycle covers for $k \in \mathcal{U}$: C is a ***k*-cycle cover** if every cycle has a length of at least k. Thus, a k-cycle cover is just a $\{k, k+1, \ldots\}$-cycle cover. For undirected graphs, the terms cycle cover, 3-cycle cover, and $\mathcal{U}$-cycle cover are equivalent; all three denote cycle covers without restrictions on the lengths of the cycles. We set $\overline{L} = \mathcal{U} \setminus L$ if we are concerned with undirected graphs (this will be clear from the context), i.e. the set $\overline{L}$ contains all forbidden cycle lengths.

Directed Graphs. Let $G = (V, E)$ be a directed graph. We say that an edge $e = (u, v)$ is **incident** to vertices u and v, which are also called endpoints of e, and u and v are **adjacent**. Two edges (u, v) and (w, x) are adjacent if $\{u, v\} \cap \{w, x\} \neq \emptyset$. The edge $e = (u, v)$ is an **outgoing edge** of u and an **incoming edge** of v. The **degree** $\deg_G(v)$ of a vertex v in G is the number of edges in E that are incident to v. The **out-degree** $\operatorname{outdeg}_G(v)$ of v is the number of outgoing edges of v, the **in-degree** $\operatorname{indeg}_G(v)$ is the number of incoming edges of v. We sometimes write $\deg_E(v)$, $\operatorname{indeg}_E(v)$, and $\operatorname{outdeg}_E(v)$ instead of $\deg_G(v)$, $\operatorname{indeg}_G(v)$, and $\operatorname{outdeg}_G(v)$ if the graph considered is clear from its edge set.

The terms **subgraph** and **induced subgraph** are defined as for undirected graphs, except that $U(V')$ is replaced by $D(V')$. A **matching** of G is a set $M \subseteq E$ of pairwise non-adjacent edges, i.e. $\deg_M(v) \leq 1$ for all $v \in V$.

A **path** in G is a sequence $P = (x_0, x_1, \ldots, x_k)$ of pairwise distinct vertices such that $e_i = (x_{i-1}, x_i) \in E$ for all $i \in [k]$. We also think of P as a sequence of edges $(e_1, \ldots, e_k)$. The length of a path P is the number of edges it consists of. A single vertex is a path of length zero.

A **cycle of length *k*** in G is sequence $(x_1, x_2, \ldots, x_k, x_1)$ of vertices, where $x_1, \ldots, x_k$ are pairwise distinct and $(x_1, x_2), (x_2, x_3), \ldots, (x_k, x_1) \in E$ are pairwise distinct edges. We do not consider a single vertex as a cycle. Thus, the shortest possible cycles in directed graphs have length two. A **Hamiltonian cycle** of G is a cycle that contains all vertices of G.

A subgraph $C = (V, E')$ of G is called a **cycle cover** of G if C consists solely

(a) A directed graph. (b) A cycle cover (solid edges) of the graph.

Figure 2.2.2: An example of a cycle cover of a directed graph.

of cycles and every vertex is part of exactly one cycle. We can alternatively say that C is a cycle cover if $\text{indeg}(v) = \text{outdeg}(v) = 1$ for all $v \in V$. Again, we will often consider C to be identical to E'. Figure 2.2.2 shows an example of a cycle cover of a directed graph.

Let $\mathcal{D} = \{2, 3, 4, \ldots\}$ and $L \subseteq \mathcal{D}$. A cycle cover C is called an ***L*-cycle cover** if the length of every cycle in C belongs to L. Again, a k-cycle cover is a $\{k, k+1, \ldots\}$-cycle cover. For directed graphs, the three terms cycle cover, 2-cycle cover, and $\mathcal{D}$-cycle cover are equivalent. If we are concerned with cycle covers of directed graphs, we define the set of forbidden cycle lengths as $\overline{L} = \mathcal{D} \setminus L$.

Edge Weighted Graphs. Let $G = (V, E)$ be a (directed or undirected) graph and let $w : E \to \mathbb{N}$ be an **edge weight function**. For any subset $F \subseteq E$ of the edges of G, we define the **weight of *F*** as

$$w(F) = \sum_{e \in F} w(e) \,.$$

We define the following terms for undirected graphs. The definitions can be transferred to directed graphs in a straightforward manner. Let $X \subseteq V$ be a subset of the vertices of G. Then $E \cap U(X)$ is the set of **internal edges of *X***, i.e. internal edges of X have both endpoints in X. We denote by $w_X(F)$ the sum of the weights of all internal edges of X in F, i.e. $w_X(F) = w(F \cap U(X))$. The **external edges at *X*** are all edges with exactly one endpoint in X.

Let $G' = (V', E')$ be a subgraph of G. We define $w(G') = w(E')$ and $w_X(G') = w_X(F)$.

2.3 Problem Definitions

In this section, we formally define the optimisation and decision problems that we will consider in the subsequent chapters.

For any $L \subseteq \mathcal{U}$, L-UCC is the following decision problem:

L-UCC:
Instance: An undirected graph $G = (V, E)$.
Question: Does G contain an L-cycle cover?

L-DCC is analogously defined for directed graphs, except that we have $L \subseteq \mathcal{D}$:

L-DCC:
Instance: A directed graph $G = (V, E)$.
Question: Does G contain an L-cycle cover?

In addition to the decision problems defined above, we consider the optimisation problems Max-L-UCC for $L \subseteq \mathcal{U}$ and Max-L-DCC for $L \subseteq \mathcal{D}$:

Max-L-UCC:
Instance: An undirected complete graph $G = (V, U(V))$ with edge weight function $w : U(V) \to \{0, 1\}$.
Solution: An L-cycle cover C of G.
Goal: Maximise $w(C)$.

Max-L-DCC:
Instance: A directed complete graph $G = (V, D(V))$ with edge weight function $w : D(V) \to \{0, 1\}$.
Solution: An L-cycle cover C of G.
Goal: Maximise $w(C)$.

Max-L-UCC can be viewed as a generalisation of L-UCC: For an undirected graph $G = (V, E)$, let $G' = (V, U(V))$ and set $w(e) = 1$ for $e \in E$ and $w(e) = 0$ for $e \notin E$. Then G contains an L-cycle cover if and only of G' contains an L-cycle cover of weight $|V|$. In the same sense, Max-L-DCC generalises L-DCC.

More generally, we can allow arbitrary natural numbers as edge weights instead of only zero and one. Max-W-L-UCC and Max-W-L-DCC are defined like Max-L-UCC and Max-L-DCC, respectively, except that the edge weight function maps to $\mathbb{N}$ instead of $\{0, 1\}$:

Max-W-L-UCC:
Instance: $G = (V, U(V))$, $w : U(V) \to \mathbb{N}$.
Solution, Goal: As for Max-L-UCC.

Max-W-L-DCC:
Instance: $G = (V, D(V))$, $w : D(V) \to \mathbb{N}$.
Solution, Goal: As for Max-L-DCC.

Let us now define some additional problems, namely the vertex cover and λ-dimensional matching problem, which will be needed for the hardness results presented in Chapter 3.

Let $H = (X, F)$ be an undirected graph. A **vertex cover** of H is a subset $\tilde{X} \subseteq X$ of the vertices of H such that every edge in F is incident to at least one vertex in $\tilde{X}$, i.e. $a \cap \tilde{X} \neq \emptyset$ for all edges $a \in F$.

Min-Vertex-Cover is the following optimisation problem:

Min-Vertex-Cover:
Instance: An undirected graph $H = (X, F)$.
Solution: A vertex cover $\tilde{X} \subseteq X$ of H.
Goal: Minimise $|\tilde{X}|$.

For $\lambda \in \mathbb{N}$, Min-Vertex-Cover(λ) is defined like Min-Vertex-Cover, except that only λ-regular graphs are instances of Min-Vertex-Cover(λ):

Min-Vertex-Cover($\boldsymbol{\lambda}$):
Instance: A λ-regular graph $H = (X, F)$.
Solution, Goal: As for Min-Vertex-Cover.

For $\lambda \in \mathbb{N}$, the λ-dimensional matching problem is the following decision problem:

$\boldsymbol{\lambda}$-DM:
Instance: A finite set X and a collection F of subsets of X with $|a| = \lambda$ for all $a \in F$.
Question: Does a subset $\tilde{F} \subseteq F$ exist such that for every $x \in X$ there is exactly one $a \in \tilde{F}$ with $x \in a$?

In the next section, we introduce the complexity of optimisation problems. As an accompanying example, we consider the travelling salesman problem (TSP). An instance of the TSP is a complete graph $G = (V, E)$ together with an edge weight function $w : E \to \mathbb{N}$. The aim is to find a Hamiltonian cycle that minimises the sum of the edge weights in the cycle. If $E = U(V)$, we speak of the symmetric TSP, denoted by **Min-TSP**; if $E = D(V)$, then we speak of the asymmetric TSP, denoted by **Min-ATSP**.

If we demand that the edge weight function to fulfil the triangle inequality, i.e. $w(\{u, v\}) \leq w(\{u, x\}) + w(\{x, v\})$ and $w(u, v) \leq w(u, x) + w(x, v)$ for all $u, x, v \in V$ in the case of Min-TSP and Min-ATSP, respectively, we obtain **Min-$\boldsymbol{\Delta}$TSP** and **Min-$\boldsymbol{\Delta}$ATSP**. An even more restricted version of Min-ΔTSP is **Min-Euc-TSP**, where the vertices are points in the Euclidean plane and the edge weight of $\{u, v\}$ is the Euclidean distance between u and v.

2.4 Complexity of Optimisation Problems

2.4.1 Optimisation Problems

In this part of the thesis, we are concerned with optimisation problems. For a more thorough treatment of the complexity of optimisation problems, we refer to

Ausiello et al. [9].

Definition 2.4.1. *An **optimisation problem** Π is characterised by a four-tuple $(I, \mathrm{sol}, m, \mathrm{goal})$:*

1. *I is a set of instances of Π.*
2. *For every instance $x \in I$, $\mathrm{sol}(x)$ denotes the set of feasible solutions for x.*
3. *Let $x \in I$ and $y \in \mathrm{sol}(x)$, then $m(x, y)$ is the measure of y with respect to x. (We assume that $m(x, y)$ is always a non-negative rational number.)*
4. *Π is either a maximisation or a minimisation problem, as indicated by $\mathrm{goal} \in \{\min, \max\}$.*

*We denote by $m^\star(x) = \mathrm{goal}_{y \in \mathrm{sol}(x)} m(x, y)$ the value of an optimum solution of x. An **optimum solution of x** is a solution $y \in \mathrm{sol}(x)$ with $m(x, y) = m^\star(x)$.*

An important class of optimisation problems is the class of NP optimisation problems. In the following definition, we identify instances x and solutions y with an appropriate encoding of x and y, respectively. The lengths of these encodings are denoted by $|x|$ and $|y|$. The exact manner in which this encoding is carried out (for instance an adjacency matrix or adjacency lists in the case of x being a graph) is not important.

Definition 2.4.2. *An optimisation problem $\Pi = (I, \mathrm{sol}, m, \mathrm{goal})$ is an **NP optimisation problem** if*

1. *the set of instances is deterministically recognisable in polynomial time, i.e. $I \in \mathsf{P}$,*
2. *there exists a polynomial p such that for all $x \in I$ and $y \in \mathrm{sol}(x)$, $|y| \leq p(|x|)$ and the question whether $y \in \mathrm{sol}(x)$ can be decided deterministically in time polynomial in $|x|$, and*
3. *for all $x \in I$ and $y \in \mathrm{sol}(x)$, $m(x, y)$ can be evaluated deterministically in time polynomial in $|x|$.*

*An optimisation problem is a **P optimisation problem** if, on input x, an optimum solution can be computed in time polynomial in $|x|$.*

NPO *denotes the class of all* NP *optimisation problems.* **PO** *denotes the class of all* P *optimisation problems.*

The **budget problem Π_B** associated with an optimisation problem Π is given as follows: Let Π be a minimisation problem. An instance of Π_B is an instance $x \in I$ and number k. The question is whether there is a solution y whose measure $m(x, y)$ does not exceed k. In case of Π being a maximisation problem,

the question is whether there is a solution y whose measure is at least k. To be more formally: For minimisation problems, $\Pi_{\mathrm{B}} = \{(x,k) \mid x \in I \wedge m^\star(x) \leq k\}$, and for maximisation problems, $\Pi_{\mathrm{B}} = \{(x,k) \mid x \in I \wedge m^\star(x) \geq k\}$. From the definition, we immediately obtain that $\Pi_{\mathrm{B}} \in \mathsf{NP}$ for all $\Pi \in \mathsf{NPO}$.

Strictly speaking, NP-hardness is only defined for decision problems and not for optimisation problems. However, we call an optimisation problem Π NP-hard if the corresponding budget problem Π_{B} is NP-hard.

Example 2.4.3. *Min-Vertex-Cover, Min-Vertex-Cover(λ), and all variants of the TSP introduced in the previous section are examples of* NP *optimisation problems.*

Max-L-UCC, Max-L-DCC, Max-W-L-UCC, and Max-W-L-DCC are NP *optimisation problems if $\{1^\lambda \mid \lambda \in L\} \in \mathsf{P}$, i.e. if L allows efficient membership testing. If $\{1^\lambda \mid \lambda \in L\} \notin \mathsf{P}$, then it is impossible to decide in polynomial time if a cycle cover is an L-cycle cover, thus Item 2 of Definition 2.4.2 is violated.*

2.4.2 Approximation Algorithms

If an optimisation problem Π is hard, for instance because the budget problem Π_{B} is NP-hard, but we want to obtain an acceptable solution, we require approximate solutions. For detailed information about approximation algorithms, we refer to Vazirani [92].

Definition 2.4.4. *Let $\Pi = (I, \mathrm{sol}, m, \mathrm{goal})$ be an optimisation problem. Let $x \in I$ be any instance of Π and $y \in \mathrm{sol}(x)$ be a feasible solution of x. The* ***performance ratio*** *of y is defined as*

$$R(x,y) = \max\left\{\frac{m(x,y)}{m^\star(x)}, \frac{m^\star(x)}{m(x,y)}\right\} .$$

For maximisation problems, we have $R(x,y) = m^\star(x)/m(x,y)$, while for minimisation problems, $R(x,y) = m(x,y)/m^\star(x)$. Thus, $R(x,y) \geq 1$ with $R(x,y) = 1$ if and only if y is an optimum solution of x. The notion of performance ratio leads immediately to approximation algorithms.

Definition 2.4.5. *Let $\Pi(I, \mathrm{sol}, m, \mathrm{goal})$ be an optimisation problem and $\alpha \geq 1$. A polynomial-time algorithm $\mathcal{A}$ is an* ***approximation algorithm with approximation ratio α*** *for Π if, for every instance $x \in I$ with $\mathrm{sol}(x) \neq \emptyset$, $\mathcal{A}$ computes a solution $\mathcal{A}(x) \in \mathrm{sol}(x)$ such that $R(x, \mathcal{A}(x)) \leq \alpha$.*

More generally, we can consider functions $f : I \to [1, \infty)$ instead of α: $\mathcal{A}$ is an approximation algorithm that achieves approximation ratio f if $R(x, \mathcal{A}(x)) \leq f(x)$ for all $x \in I$ with $\mathrm{sol}(x) \neq \emptyset$. Usually, f depends on the size of the instance x.

A **polynomial-time approximation scheme** (or **PTAS** for short) is a family of approximation algorithms such that for every $\epsilon > 0$, there is an algorithm that achieves approximation ratio $1 + \epsilon$.

The optimisation problems in NPO are divided into several subclasses according to their approximation properties. Let us define the two most important classes.

Definition 2.4.6. *The class* **APX** $\subseteq$ NPO *contains all* NP *optimisation problems that admit a factor* α *approximation for some constant* α*.*

The class **PTAS** $\subseteq$ APX *contains all* NP *optimisation problems that admit a polynomial-time approximation scheme.*

From the definitions, we immediately obtain

$$\mathsf{PO} \subseteq \mathsf{PTAS} \subseteq \mathsf{APX} \subseteq \mathsf{NPO}$$

with PO = NPO if and only if P = NP. Provided that P $\neq$ NP, all three inclusions are strict (cf. Ausiello et al. [9]).

Example 2.4.7. *Min-TSP and Min-ATSP cannot be approximated at all [81].*

There exists a factor $0.842 \cdot \log n$ *approximation for Min-*Δ*ATSP where* n *is the number of vertices [58]. It is unknown whether Min-*Δ*ATSP* $\in$ APX*.*

Christofides' algorithm is a factor $3/2$ *approximation for Min-*Δ*TSP [27] (cf. Vazirani [92, Sect. 3.2.2]), thus Min-*Δ*TSP* $\in$ APX*.*

Min-Euc-TSP is in PTAS*, i.e. it can be approximated arbitrarily well [6].*

All variants of the TSP mentioned above are NP-complete due to the NP-completeness of the Euclidean TSP [68] (cf. Garey and Johnson [47, ND22+23]). Thus, although the budget problems share the same complexity, their approximation properties differ greatly.

So far, we are able to prove that a certain optimisation problem is, say, in APX by presenting an approximation algorithm. But we cannot achieve good inapproximability results: Proving an optimisation problem to be NP-hard does not rule out the possibility of good approximation algorithms. One exception is Min-TSP, where the inapproximability can be proved by reduction from the *Hamiltonian circuit problem* [81] (cf. Vazirani [92, Theorem 3.6]), which is NP-complete [59] (cf. Garey and Johnson [47, GT37]).

2.4.3 Reductions and Inapproximability

Strong inapproximability results were made possible by the *PCP theorem* (PCP stands for *probabilistically checkable proofs*), which is an alternative characterisation of NP proved by Arora et al. [7, 8]. Using the PCP theorem, Arora et al. showed that several optimisation problems do not belong to PTAS unless P = NP. The inapproximability of several other optimisation problems followed via reductions. For instance, Min-ΔTSP is not in PTAS unless P= NP [71].

To date, several different kinds of reductions between optimisation problems have been proposed for showing inapproximability [31]. We present only AP- and L-reductions, which we need for our hardness results.

Definition 2.4.8 (Crescenzi et al. [32]). *Let* $\Pi = (I, \mathrm{sol}, m, \mathrm{goal})$ *and* $\Pi' = (I', \mathrm{sol}', m', \mathrm{goal}')$ *be two optimisation problems. Then* Π ***AP-reduces*** *to* Π'*, denoted by* $\Pi \leq_{\mathrm{AP}} \Pi'$*, if there exist two functions* f_{AP} *and* g_{AP} *and a constant* $\alpha_{\mathrm{AP}} \geq 1$ *such that the following properties hold for every fixed rational number* $r > 1$*:*

1. *For every instance* $x \in I$ *of* Π*,* $f_{\mathrm{AP}}(x, r) = x' \in I'$ *is an instance of* Π'*. If* $\mathrm{sol}(x) \neq \emptyset$*, then* $\mathrm{sol}'(x') \neq \emptyset$*.*
2. *For all* $y' \in \mathrm{sol}'(x')$*, we have* $y = g_{\mathrm{AP}}(x, y', r) \in \mathrm{sol}(x)$*.*
3. *The functions* f_{AP} *and* g_{AP} *are computable in time polynomial in* $|x|$*.*
4. *For every* $x \in I$ *and* $y' \in \mathrm{sol}(x')$*,*

$$R'(x', y') \leq r \;\text{ implies }\; R(x, y) \leq 1 + \alpha_{\mathrm{AP}} \cdot (r - 1)\,.$$

(R' *denotes the approximation ratio with respect to* Π'*.)*

The classes APX and PTAS are closed under AP-reductions: Assume that Π AP-reduces to Π'. Then $\Pi' \in \mathsf{APX}$ implies $\Pi \in \mathsf{APX}$, and $\Pi' \in \mathsf{PTAS}$ implies $\Pi \in \mathsf{PTAS}$ [9, Lemma 8.1].

Definition 2.4.9 (Papadimitriou and Yannakakis [70]). *Let* Π *and* Π' *be two optimisation problems. Then* Π ***L-reduces*** *to* Π'*, denoted by* $\Pi \leq_{\mathrm{L}} \Pi'$*, if there exist functions* f_{L} *and* g_{L} *and constants* $\alpha_{\mathrm{L}}, \beta_{\mathrm{L}} > 0$ *such that the following conditions hold for every instance* $x \in I$ *of* Π*:*

1. *The function* f_{L} *produces an instance* $x' = f_{\mathrm{L}}(x) \in I'$ *of* Π' *with*

$$m'^{\star}(x') \leq \alpha_{\mathrm{L}} \cdot m^{\star}(x)\,.$$

If $\mathrm{sol}(x) \neq \emptyset$*, then* $\mathrm{sol}'(x') \neq \emptyset$*.*

2. *For all solutions* $y' \in \mathrm{sol}'(x')$*, the function* g_{L} *produces a solution* $y = g_{\mathrm{L}}(x, y') \in \mathrm{sol}(x)$ *of* Π *with*

$$|m(x, y) - m^{\star}(x)| \leq \beta_{\mathrm{L}} \cdot |m'(x', y') - m'^{\star}(x')|\,.$$

Furthermore, $|y'| \leq p(|x|)$ *for some polynomial* p*.*

3. *The functions* f_{L} *and* g_{L} *are computable in polynomial time.*

The class PTAS is closed under L-reductions, but it is open whether APX is closed under L-reductions. Crescenzi et al. [32] conjectured that this is not the case.

Completeness in APX is defined via AP-reductions [9, Sect. 8.4]: An optimisation problem Π is **APX-hard** if $\Pi' \leq_{\mathrm{AP}} \Pi$ for all $\Pi' \in \mathsf{APX}$. If additionally $\Pi \in \mathsf{APX}$, then Π is **APX-complete**. Finding a PTAS for any APX-hard problem would immediately prove P= NP.

Example 2.4.10. *Min-ΔTSP is* APX*-complete, even if the edge weights are restricted to be one or two [71] (cf. Ausiello et al. [9, ND33]).*

L-reductions are useful for proving the existence of AP-reductions: Let Π and Π' be two NP optimisation problems with $\Pi \in$ APX. Then $\Pi \leq_{\mathrm{L}} \Pi'$ implies $\Pi \leq_{\mathrm{AP}} \Pi'$ [9, Lemma 8.2]. All APX-hardness results of this work are obtained via reduction from Min-Vertex-Cover(λ), which actually is APX-complete (cf. Ausiello et al. [9, GT1]). We reduce also to problems not in NPO. Therefore, for the sake of completeness, we show in Appendix A.1 that the requirement $\Pi' \in$ NPO is not needed.

2.5 Existing Results for Cycle Covers

Before taking a closer look at existing results for cycle covers, let us briefly mention some known results for λ-DM and Min-Vertex-Cover. 3-DM and the budget problem Min-Vertex-Cover$_{\mathrm{B}}$ are NP-complete [59] (cf. Garey and Johnson [47, GT1+SP1]). Generalising the NP-completeness of 3-DM to λ-DM for $\lambda \geq 3$ is straightforward. Min-Vertex-Cover(λ) is also NP-hard for $\lambda \geq 3$ [48]. Min-Vertex-Cover and Min-Vertex-Cover(3) are APX-complete [4,70] (cf. Ausiello et al. [9, GT1]).

2.5.1 Cycle Covers in Undirected Graphs

$\mathcal{U}$-UCC and Max-$\mathcal{U}$-UCC can be solved in polynomial time by Tutte's reduction to the classical matching problem [91], which in turn can be solved in polynomial time by Edmond's algorithm [43]. Max-W-$\mathcal{U}$-UCC can also be solved in polynomial time by Tutte's reduction [42]. The currently best algorithms for $\mathcal{U}$-UCC and Max-$\mathcal{U}$-UCC achieve a running time of $O(n^{2.5})$, where n is the number of vertices [3, Chap. 12]. This result was recently improved by Mucha and Sankowski [67], who presented a randomised algorithm with a running time of $O(n^{\omega})$, where $\omega < 2.38$ is the matrix multiplication exponent [28]. Max-W-$\mathcal{U}$-UCC can be solved in time $O(n^3)$. There are several deterministic algorithms that achieve this time bounds. We refer to Ahuja et al. [3, Chap. 12] for a survey of matching algorithms.

Hartvigsen presented a polynomial-time algorithm for computing a *maximum-cardinality triangle-free two-matching* [51]. His algorithm can be used to decide 4-UCC in polynomial time. Furthermore, it can be used to approximate Max-4-UCC within an additive error of one according to Bläser [13]. As far as we are aware, it has not been proved that Max-4-UCC is exactly solvable in polynomial time, even though there are approximation algorithms that require an efficient algorithm for Max-4-UCC [18, 20, 71]. We prove that Max-4-UCC is indeed solvable in polynomial time by exploiting Hartvigsen's algorithm (Section 4.2).

Max-W-k-UCC admits an easy factor 3/2 approximation for all k: Compute a cycle cover of maximum weight, break the lightest edge of each cycle (thus, at least two thirds of the weight remain), and join the paths obtained into a Hamiltonian cycle, which is sufficiently long provided that the graph contains at least k vertices (otherwise, the graph does not contain any k-cycle cover at all). Unfortunately, such a simple algorithm does not work for Max-W-L-UCC with general L. For the problem of computing k-cycle covers of minimum weight in graphs with edge weights one and two, there exists a factor 7/6 approximation algorithm for all k [20]. Hassin and Rubinstein [53] devised a randomised approximation algorithm for Max-W-$\{3\}$-UCC that achieves approximation ratio $169/89 + \epsilon$ for every fixed $\epsilon > 0$.

Cornuéjols and Pulleyblank presented a proof due to Papadimitriou that k-UCC is NP-complete for $k \geq 6$ [30]. Vornberger proved that Max-W-5-UCC and Max-W-$\overline{\{4\}}$-UCC are NP-complete [94]. Hell et al. [56] proved that L-UCC is NP-hard for $\overline{L} \not\subseteq \{3, 4\}$.

Although the complexity of finding restricted cycle covers in undirected graphs is well understood, almost nothing is known about their approximability.

For most L, L-UCC, Max-L-UCC, and Max-W-L-UCC are not even recursive since there are uncountably many L but only countably many recursive functions. Consequently, for most L, L-UCC is not in NP and Max-L-UCC and Max-W-L-UCC are not in NPO. This does not matter for hardness results but may cause problems if one wants to design approximation algorithms that are based on computing L-cycle covers. However, it turns out that this does not affect our approximation algorithms, as we show in Section 4.1.

2.5.2 Cycle Covers in Directed Graphs

$\mathcal{D}$-DCC and Max-$\mathcal{D}$-DCC can be solved in polynomial time by reduction to the matching problem in bipartite graphs, which can be solved deterministically in time $O(n^{2.5})$ [3, Chap. 12] or with a randomised algorithm in time $O(n^{\omega})$ [67]. Max-W-$\mathcal{D}$-DCC, also known as the *assignment problem*, can be solved in polynomial time using the Hungarian method [62] for computing perfect matchings of maximum weight in bipartite graphs. The fastest algorithms to date need time $O(n^3)$ [3, Chap. 12].

The reduction of Max-$\mathcal{D}$-DCC or Max-W-$\mathcal{D}$-DCC to matching in bipartite graphs is as follows: Consider a directed complete graph $G = (V, D(V))$ with edge weights $w : D(V) \to \mathbb{N}$. Let $V' = \{v' \mid v \in V\}$. We construct a bipartite graph G' with vertex set $V \cup V'$. For any $u, v \in V$ with $u \neq v$, we add an edge (u, v') of weight $w(u, v)$ to G'. Now, we have a one-to-one correspondence between cycle covers C of G and perfect matchings M of G': For all $u, v \in V$, $(u, v) \in C$ if and only if $(u, v') \in M$. The weight of C is equal to the weight of M, and C is a cycle cover if and only if M is a perfect matching.

For all $k \geq 3$, k-DCC is NP-complete [47, GT13]. (This follows also from the results of Section 3.3.4.)

Similar to the factor 3/2 approximation algorithm for undirected k-cycle covers, Max-W-k-DCC has an easy factor 2 approximation algorithm for all k: Compute a cycle cover of maximum weight, break the lightest edge of every cycle (thus, at least half of the weight remains), and join the paths obtained into a Hamiltonian cycle. Again, this simple algorithm does not work for Max-W-L-DCC with general L. There is a factor 4/3 approximation algorithm for Max-W-3-DCC [19] and a factor 3/2 approximation algorithm for Max-k-DCC for $k \geq 3$ [16].

As in the case of cycle covers in undirected graphs, for most L, L-DCC, Max-L-DCC, and Max-W-L-DCC are not recursive.

While the complexity of finding k-cycle covers in directed graphs is settled, almost nothing, neither positive nor negative, is known about the approximability of k-cycle covers and the complexity and approximability of L-cycle covers in general.

2.6 New Results

We almost settle the complexity and approximability of restricted cycle covers for both undirected and directed graphs. Table 2.6.1 summarises the results for the complexity of L-cycle covers. A more detailed description of our results is given in the following two sections.

Only the complexity of the following five problems remains open: 5-UCC, $\overline{\{4\}}$-UCC, Max-5-UCC, Max-$\overline{\{4\}}$-UCC, and Max-W-4-UCC. In Chapter 5, we will take a closer look at these five problems.

2.6.1 Hardness Results

We prove that Max-L-UCC is APX-hard for all L with $\overline{L} \not\subseteq \{3, 4\}$ (Theorem 3.3.11), i.e. if at least one length greater than or equal to five is forbidden. We further extend this hardness result for general edge weights: Max-W-L-UCC is APX-hard for all L with $\overline{L} \not\subseteq \{3\}$, even if we allow only zero, one, and two as edge weights (Theorems 3.2.7, 3.2.9, and 3.3.11).

We show a dichotomy for cycle covers of directed graphs: For all L with $L \neq \{2\}$ and $L \neq \mathcal{D}$, L-DCC is NP-hard (Theorem 3.3.17), and Max-L-DCC and Max-W-L-DCC are APX-hard (Theorem 3.3.16), while it is known that all three problems are solvable in polynomial time if $L = \{2\}$ or $L = \mathcal{D}$.

If computing L-cycle covers of maximum weight in (directed or undirected) graphs with edge weights zero and one is APX-hard, then this carries over to arbitrary non-trivial weight functions, i.e. weight functions the range of which contains at least two different values a and b: Assume $a < b$, then we map weight zero to weight a and weight one to weight b. Moreover, the hardness

	L-UCC	Max-L-UCC	Max-W-L-UCC
$\overline{L} = \emptyset$	in P	in PO	in PO
$\overline{L} = \{3\}$	in P	in PO	
$\overline{L} = \{4\}$			APX-complete
$\overline{L} = \{3, 4\}$			APX-complete
$\overline{L} \not\subseteq \{3, 4\}$	NP-hard	APX-hard	APX-hard

(a) Undirected cycle covers.

	L-DCC	Max-L-DCC	Max-W-L-DCC
$L \in \{\{2\}, \mathcal{D}\}$	in P	in PO	in PO
$L \notin \{\{2\}, \mathcal{D}\}$	NP-hard	APX-hard	APX-hard

(b) Directed cycle covers.

Table 2.6.1: The complexity of computing L-cycle covers.

holds also for the problem of computing L-cycle covers of minimum weight. This particularly includes computing L-cycle covers of minimum weight in graphs with edge weights one and two.

To show the hardness results for directed cycle covers, we prove that certain kinds of graphs, so-called *L-clamps*, exist for non-empty $L \subseteq \mathcal{D}$ if and only if $L \neq \mathcal{D}$ (Theorem 3.3.13). This graph-theoretical result might be of independent interest.

As a by-product, we prove that Min-Vertex-Cover(λ) is APX-complete for all $\lambda \geq 3$ (Theorem 3.4.1). We need this result for the APX-hardness proofs in Section 3.3, and as far as we are aware, it is unproved so far. ?

2.6.2 Algorithms

We present a polynomial-time approximation algorithm for Max-W-L-UCC that works for all $L \subseteq \mathcal{U}$ and achieves an approximation ratio of 2.5 (Section 4.1.1). For Max-W-L-DCC, we devise a polynomial-time factor 3 approximation algorithm that works for all $L \subseteq \mathcal{D}$ (Section 4.1.2). Both algorithms work for arbitrary L, even if L is not a recursively enumerable set.

Although most of our hardness results for undirected cycle covers are for Max-L-UCC, where only zero and one are allowed as edge weights, these approximation algorithms work for arbitrary edge weights.

Finally, we show that Max-4-UCC is indeed solvable in polynomial time by exploiting Hartvigsen's algorithm [51] for finding maximum-cardinality triangle-free two-matchings (Section 4.2).

CHAPTER

THREE

Approximation Hardness

In this chapter, we prove that Max-L-UCC, Max-W-L-UCC, and Max-L-DCC are APX-hard for almost all L. Furthermore, we prove the NP-hardness of L-DCC for almost all L. Finally, we show that Min-Vertex-Cover(λ) is APX-complete for all $\lambda \geq 3$.

While the NP-hardness is proved via a standard many-one reduction [47], the APX-hardness results are obtained by constructing L-reductions. For the APX-hardness of the cycle cover problems, we L-reduce from Min-Vertex-Cover(λ). In the next section, we outline such an L-reduction; to actually construct an L-reduction, it then essentially suffices to instantiate three lemmas, stated as Generic Lemmas 3.1.1 to 3.1.3 below.

3.1 Outline of an L-Reduction

Starting from a λ-regular graph $H = (X, F)$ on n vertices as an instance of Min-Vertex-Cover(λ), we have to construct a complete graph G (directed or undirected) together with an appropriate edge weight function w in polynomial time as an instance of Π.

To construct an L-reduction, we need to be able to instantiate the following three generic lemmas for a certain constant γ and a certain definition of the term *legal cycle cover*.

Generic Lemma 3.1.1. *Given a vertex cover $\tilde{X}$ of H, there is an L-cycle cover $\tilde{C}$ of G with weight $w(\tilde{C}) = \gamma n - |\tilde{X}|$.*

Generic Lemma 3.1.2. *Given an arbitrary L-cycle cover C, we can construct a legal L-cycle cover $\tilde{C}$ with $w(\tilde{C}) \geq w(C)$ in polynomial time.*

Generic Lemma 3.1.3. *Given a legal L-cycle cover $\tilde{C}$ of weight $w(\tilde{C}) = \gamma n - \tilde{n}$, we can construct a vertex cover $\tilde{X}$ of H with $|\tilde{X}| = \tilde{n}$.*

Instantiating and proving these three generic lemmas yields an L-reduction.

Lemma 3.1.4. *If the three generic lemmas above hold, Min-Vertex-Cover(λ) L-reduces to* Π.

Proof. Let $\mathrm{opt}(H)$ denote the size of a minimum vertex cover of H and let $\mathrm{opt}(G)$ denote the weight of a maximum-weight L-cycle cover of G. Since H is λ-regular, every vertex can cover at most λ edges. Thus, $\mathrm{opt}(H) \geq n/\lambda$. Any L-cycle cover of maximum weight can be transformed into a legal L-cycle cover without losing weight (Generic Lemma 3.1.2). From such a legal L-cycle cover of weight $\gamma n - \tilde{n}$, we can obtain a vertex cover of size $\tilde{n}$ (Generic Lemma 3.1.3). Thus, $\mathrm{opt}(G) \leq \gamma n$ and we obtain

$$\mathrm{opt}(G) \leq \gamma n \leq \lambda\gamma \cdot \mathrm{opt}(H) \,.$$

Now let C be an arbitrary L-cycle cover of G, $\tilde{C}$ be a legal L-cycle obtained from C, and $\tilde{X} \subseteq X$ be the vertex cover obtained from $\tilde{C}$. Then

$$\big||\tilde{X}| - \mathrm{opt}(H)\big| = \big|\underbrace{w(\tilde{C})}_{\gamma n - |\tilde{X}|} - \underbrace{\mathrm{opt}(G)}_{\gamma n - \mathrm{opt}(H)}\big| \leq \big|w(C) - \mathrm{opt}(G)\big| \,.$$

Thus, we obtain an L-reduction with $\alpha_{\mathrm{L}} = \lambda\gamma$ and $\beta_{\mathrm{L}} = 1$: The function f_{L} maps H to G and w, while the function g_{L} maps an arbitrary L-cycle cover C to a vertex cover $\tilde{X}$ according to Generic Lemmas 3.1.2 and 3.1.3. □

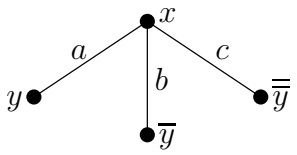
3.2 A Generic Reduction for *L*-Cycle Covers

In this section, we present a generic reduction from Min-Vertex-Cover(3) to Max-L-UCC or Max-W-L-UCC. To instantiate the reduction for a certain L, we use a small graph, which we call a *gadget*, the specific structure of which depends on L. Such a gadget together with the generic reduction is an L-reduction from Min-Vertex-Cover(3) to Max-L-UCC or Max-W-L-UCC. The aim is to prove the APX-hardness of Max-W-$\overline{\{4\}}$-UCC and Max-W-5-UCC. We also briefly show how to prove the APX-completeness of Max-k-UCC for all $k \geq 6$ using the generic reduction, although this also follows from the results of Section 3.3.2.

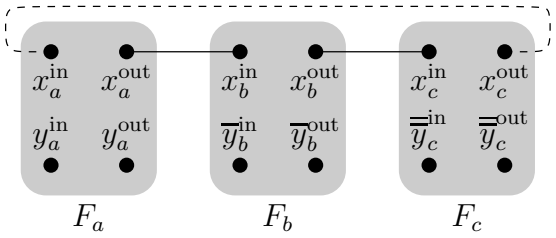
3.2.1 The Generic Reduction

Let $H = (X, F)$ be a cubic graph with vertex set X and edge set F as an instance of Min-Vertex-Cover(3). Let $n = |X|$ and $m = |F| = 3n/2$. We construct an undirected complete graph G with edge weight function w as a generic instance of Max-L-UCC or Max-W-L-UCC.

For each edge $a = \{x, y\} \in F$, we construct a subgraph F_a of G called the **gadget of** $\boldsymbol{a}$. We consider F_a as set of vertices, thus $w_{F_a}(C)$ for a subset C of the edges of G is well defined. This gadget contains four distinguished vertices



(a) Vertex x and its edges.



(b) The gadgets F_a, F_b, and F_c and their connections via the three junctions of x. The dashed edge has weight zero. Other weight zero edges and the junctions of y, $\overline{y}$, and $\overline{\overline{y}}$ are not shown.

Figure 3.2.1: The construction for a vertex $x \in X$ incident to $a, b, c \in F$.

x_a^{in}, x_a^{out}, y_a^{in}, and y_a^{out}. These four vertices are used to connect F_a to the rest of the graph. What such a gadget looks like depends on L.

If all edges in such a gadget have weight zero or one, we obtain an instance of Max-L-UCC since all edges between different gadgets will have weight zero or one. Otherwise, we have an instance of Max-W-L-UCC. Examples of gadgets will be given in Sections 3.2.2 and 3.2.3.

Let $a, b, c \in F$ be the three edges incident to vertex $x \in X$ (the order is arbitrary). Then we assign weight one to the edges connecting x_a^{out} to x_b^{in} and x_b^{out} to x_c^{in} and weight zero to the edge connecting x_c^{out} to x_a^{in}. We call the three edges $\{x_a^{\text{out}}, x_b^{\text{in}}\}$, $\{x_b^{\text{out}}, x_c^{\text{in}}\}$, and $\{x_c^{\text{out}}, x_a^{\text{in}}\}$ the **junctions of x**. We say that $\{x_a^{\text{out}}, x_b^{\text{in}}\}$ and $\{x_c^{\text{out}}, x_a^{\text{in}}\}$ are the junctions of x **at F_a**. An example is shown in Figure 3.2.1.

We call an edge **illegal** if it connects two different gadgets but is not a junction. Thus, an illegal edge is an external edge at two different gadgets. All illegal edges have weight zero, i.e. there are no edges of weight one that connect two different gadgets except for the junctions. The weights of the internal edges of the gadgets depend on the gadget, which in turn depends on L.

The following terms are defined for arbitrary subsets C of the edges of G and so in particular for L-cycle covers. We say that **C legally connects F_a** if

- C contains no illegal edges incident to F_a,
- C contains exactly two or four junctions at F_a, and
- if C contains exactly two junctions at F_a, then these belong to the same vertex $x \in a$.

We call C **legal** if C legally connects all gadgets.

Lemma 3.2.1. *Let $\tilde{C}$ be an arbitrary legal subset of the edges of G. Then for all $x \in X$, either all junctions of x are in $\tilde{C}$ or no junction of x is in $\tilde{C}$.*

Proof. Assume that there is an $x \in X$ such that neither none nor all junctions of x are in $\tilde{C}$. Then there is an edge $a \in F$ with $x \in a$ such that only one junction of x at F_a is in $\tilde{C}$. Thus, $\tilde{C}$ does not legally connect F_a. □

From a legal subset $\tilde{C}$ of the edges of G, we obtain a subset $\tilde{X} \subseteq X$ of the vertices of H as follows: If all junctions of x are in $\tilde{C}$, then $x \in \tilde{X}$, otherwise $x \notin \tilde{X}$. The set $\tilde{X}$ turns out to be a vertex cover of H.

Lemma 3.2.2. *Let $\tilde{C}$ be a legal subset of the edges of G. Then the set*

$$\tilde{X} = \{x \mid \textit{the junctions of } x \textit{ are in } \tilde{C}\}$$

obtained from $\tilde{C}$ is a vertex cover of H.

Proof. Consider an arbitrary edge $a = \{x, y\} \in F$. Either two or four junctions at F_a are in $\tilde{C}$. Assume without loss of generality that $\tilde{C}$ contains x's junctions at F_a. Then all of x's junctions are in $\tilde{C}$ by Lemma 3.2.1, which implies $x \in \tilde{X}$. □

Let us now define the requirements the gadgets must fulfil. In the following, let C be an arbitrary L-cycle cover of G and $a = \{x, y\} \in F$ be an arbitrary edge of H.

R0: There exists a fixed number $s \in \mathbb{N}$, which we call the **gadget parameter**, that depends only on the gadget. The role of the gadget parameter will become clear in the subsequent requirements.

R1: $w_{F_a}(C) \leq s - 1$.

R2: If C contains 2α external edges at F_a, then $w_{F_a}(C) \leq s - \alpha$.

R3: If C contains exactly one junction of x at F_a and exactly one junction of y at F_a, then $w_{F_a}(C) \leq s - 2$. (In this case, C does not legally connect F_a.)

R4: Let C' be an arbitrary subset of the edges of G that legally connects F_a. Assume that there are 2α junctions ($\alpha \in \{1, 2\}$) at F_a in C'.

Then there exists a C'' with the following properties:

- C'' differs from C' only in F_a's internal edges and
- $w_{F_a}(C'') = s - \alpha$.

Thus, given C', C'' can be obtained by locally modifying C' within F_a. We call the process of obtaining C'' from C' **rearranging C' in F_a**.

R5: Let C' be a legal subset of the edges of G. Then there exists a subset $\tilde{C}$ of edges obtained by rearranging all gadgets as described in R4 such that $\tilde{C}$ is an L-cycle cover.

We show the existence of such gadgets in the subsequent sections when instantiating the generic reduction of this section to actually prove APX-hardness results. In the following, let us assume that a gadget for Max-L-UCC or Max-W-L-UCC exists. A consequence of the requirements above is the following lemma, which instantiates Generic Lemma 3.1.1 with $\gamma = (3/2) \cdot s$.

Lemma 3.2.3. *Let $\tilde{X} \subseteq X$ be a vertex cover of size $\tilde{n}$ of H. Then there exists a legal L-cycle cover $\tilde{C}$ with $w(\tilde{C}) = ms - \tilde{n}$.*

Proof. We construct $\tilde{C}$ as follows: If $x \in \tilde{X}$, then we add all junctions of x to $\tilde{C}$. Otherwise, no junction of x is added to $\tilde{C}$. $\tilde{C}$ does not contain any other external edges at any gadget. So far, $\tilde{C}$ is legal. The internal edges of the gadgets are chosen according to R5. Thus, $\tilde{C}$ is an L-cycle cover.

To calculate $w(\tilde{C})$, assume for the moment that all junctions have weight one. The weight of a junction at F_a and at F_b is split among F_a and F_b. If $w_{F_a}(\tilde{C}) = s - 1$, then there are two junctions in $\tilde{C}$ at F_a according to R4. Thus, F_a gets additional weight one from these two junctions, i.e. weight $1/2$ from each junction. If $w_{F_a}(\tilde{C}) = s - 2$, then there are four junctions in $\tilde{C}$ at F_a according to R4. Thus, F_a gets additional weight two from these four junctions. Overall, the weight of $\tilde{C}$ with the weight of all junctions set to one is ms. We have to subtract the number of weight zero junctions in $\tilde{C}$ from this weight. There are $\tilde{n}$ junctions of weight zero in $\tilde{C}$, which proves the lemma. □

Moreover, the requirements assert that connecting the gadgets legally is never worse than connecting them illegally. To put it another way, given an arbitrary L-cycle cover we can compute a legal L-cycle cover without losing any weight. This will be proved in the next lemma, which instantiates Generic Lemma 3.1.2.

Lemma 3.2.4. *Given an arbitrary L-cycle cover C, we can compute a legal L-cycle cover $\tilde{C}$ with $w(\tilde{C}) \geq w(C)$ in polynomial time.*

Proof. We proceed as follows to construct a legal L-cycle cover $\tilde{C}$ from an arbitrary L-cycle cover C:

1. Let C' be C with all illegal edges removed.

2. For all $x \in X$ in arbitrary order: If at least one junction of x is in C, then put all junctions of x into C'.

3. For all $a \in F$ in arbitrary order: If there is no junction at F_a in C', then choose one vertex $x \in a$ arbitrarily and add all junctions of x to C'.

4. Rearrange all gadgets in C' according to R5. Call the subset of G's edges obtained in this way $\tilde{C}$.

The running time of the algorithm is obviously polynomial. We have to prove the following: $\tilde{C}$ is a legal L-cycle cover, and $w(\tilde{C}) \geq w(C)$.

Let us start by proving that $\tilde{C}$ is indeed a legal L-cycle cover. $\tilde{C}$ does not contain any illegal edge due to Step 1. If one junction of x is in $\tilde{C}$, then all junctions of x are in $\tilde{C}$ due to Step 2. There is no gadget that is not incident with any junction due to Step 3. Finally, $\tilde{C}$ is an L-cycle cover due to R5 since all gadgets are rearranged in Step 4.

Now we turn to proving $w(\tilde{C}) \geq w(C)$. All illegal edges have weight zero, and we do not remove any junction. Thus, no weight is lost by removing external edges at any gadget. The internal edges of the gadgets remain to be considered.

Let $a = \{x, y\} \in F$ be an arbitrary edge of H. If $w_{F_a}(C) \leq w_{F_a}(\tilde{C})$, then nothing has to be shown. What remains to be considered are gadgets F_a with $w_{F_a}(C) > w_{F_a}(\tilde{C})$. We have $w_{F_a}(\tilde{C}) \geq s-2$ according to R4 and $w_{F_a}(C) \leq s-1$ according to R1. Thus, $w_{F_a}(C) = w_{F_a}(\tilde{C}) + 1$ for all F_a with $w_{F_a}(C) > w_{F_a}(\tilde{C})$. We will now prove that for all such gadgets, there is a junction of weight one in $\tilde{C}$ that is not in C and can thus compensate for the loss of weight.

If $w_{F_a}(C) > w_{F_a}(\tilde{C})$, then according to R3, the junctions at F_a in C belong to the same vertex (there are zero, one, or two junctions at F_a in C), and all four junctions at F_a are in $\tilde{C}$ according to R2 and R4. Thus, during the execution of the algorithm there is a moment at which at least one of, say, y's junctions at F_a is in C', and the junctions of x are added in the next step. We say that a vertex ***x* compensates F_a** if

1. $\tilde{C}$ contains x's junctions,
2. no junction of x at F_a is in C, and
3. at the moment at which x's junctions are added, C' already contains at least one junction of y at F_a.

Thus, every gadget F_a with $w_{F_a}(C) > w_{F_a}(\tilde{C})$ is compensated by some vertex x.

It remains to be shown that the number of gadgets that are compensated by some vertex is at most the number of weight one junctions added to C'. To prove this, let $a, b, c \in F$ be the three edges incident to $x \in X$. We distinguish three cases:

Case 1: C contains two or three junctions of x. Then x does not compensate any gadget since at all three gadgets F_a, F_b, and F_c there is at least one junction of x in C.

Case 2: C contains exactly one junction of x. Assume that this junction connects F_b to F_c. Then x does not compensate F_b and F_c. Thus, at most one gadget is compensated by x.

Since two junctions of x are added to C', at least one of them has weight one.

Case 3: C does not contain any junction of x. Then the junctions of x are added during Step 3. Thus, there is at least one gadget of F_a, F_b, F_c, say F_a, such that there is no junction at all in C' at F_a before adding x's junctions. According to the third condition for compensation given above, x does not compensate F_a. This implies that at most two gadgets are compensated by x.

All three junctions of x are added to C' by the algorithm, and two of them have weight one each.

The lemma is proved since we have proved that $\tilde{C}$ is a legal L-cycle cover and $w(\tilde{C}) \geq w(C)$. □

As the last ingredient, we need the following lemma, which instantiates Generic Lemma 3.1.3.

Lemma 3.2.5. *Let $\tilde{C}$ be the L-cycle cover constructed according to Lemma 3.2.4. Choose $\tilde{n}$ such that $w(\tilde{C}) = ms - \tilde{n}$. Let $\tilde{X} \subseteq X$ be the subset of vertices obtained from $\tilde{C}$. Then $|\tilde{X}| = \tilde{n}$.*

Proof. The proof is similar to the proof of Lemma 3.2.3. We set the weight of all junctions to one. With respect to the modified edge weights, the weight of $\tilde{C}$ is ms according to the requirements. Thus, $\tilde{n}$ is the number of weight zero junctions in $\tilde{C}$, which is just $|\tilde{X}|$. □

Since all three generic lemmas have been instantiated, we obtain the following lemma via Lemma 3.1.4 as the main result of this section.

Lemma 3.2.6. *Assume that a gadget as described exists for $L \subseteq \mathcal{U}$.*

Then the reduction presented is an L-reduction from Min-Vertex-Cover(3) to Max-W-L-UCC. If the gadget contains only edges of weight zero or one, then the reduction is an L-reduction from Min-Vertex-Cover(3) to Max-L-UCC.

3.2.2 Max-W-5-UCC and Max-W-$\overline{\{4\}}$-UCC

The gadget for Max-W-5-UCC is shown in Figure 3.2.2. Let G be the graph constructed via the reduction presented in Section 3.2.1 with the gadget of this section. Let C be an arbitrary L-cycle cover of G and $a = \{x, y\} \in F$. We have to prove that all requirements are fulfilled.

R0: The gadget parameter of the gadget for Max-W-5-UCC is $s = 6$.

R1: Since the gadget consists of only four vertices, every 5-cycle cover contains at most three of its internal edges. Otherwise, we would have a cycle of length four, which is forbidden. With three internal edges, we can achieve at most weight $5 = s - 1$ by taking the two edges of weight two and one edge of weight one.

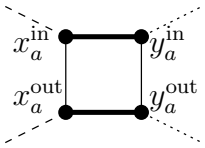
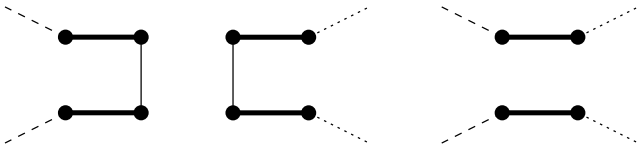


Figure 3.2.2: The edge gadget F_a for an edge $a = \{x, y\}$ that is used to prove the APX-completeness of Max-W-5-UCC. Bold edges are internal edges of weight two, solid edges are internal edges of weight one, internal edges of weight zero are not shown. The dashed and dotted edges are the junctions of x and y, respectively, at F_a.



(a) $x \in \tilde{X}$. (b) $y \in \tilde{X}$. (c) $x, y \in \tilde{X}$.

Figure 3.2.3: Traversals of the gadget for Max-W-5-UCC that achieve maximum weight.

R2: If C contains 2α external edges at F_a, then it contains $4 - \alpha$ internal edges of F_a. At most two of them have weight two, which implies the upper bound of $6 - \alpha = s - \alpha$.

R3: In every 5-cycle cover that contains exactly one junction of x and one junction of y at F_a, there can be at most two edges of weight two or one edge of weight two and two edges of weight one, which implies $w_{F_a}(C) \leq 4$.

R4: We can traverse the gadget as shown in Figure 3.2.3.

R5: Assume that all gadgets are traversed in $\tilde{C}$ in one of the ways shown in Figure 3.2.3. We have to show that all cycles in $\tilde{C}$ have a length of at least five. Obviously, no cycle traverses only one gadget. Assume that a cycle traverses only two gadgets F_a and F_b. Then these two gadgets are connected via two junctions. These two junctions cannot belong to the same vertex x since H is cubic: For all $a, b \in F$ and $x \in X$, there is at most one junction of x connecting F_a to F_b by construction. If they belong to different vertices $x, y \in X$, then $a = b = \{x, y\}$. This cannot happen since H is assumed to be simple. Thus, every cycle runs through at least three gadgets.

If a cycle traverses a gadget, then it contains at least two vertices of this gadget. Hence, every cycle has length at least six, which is long enough.

The gadget together with Lemma 3.2.6 yields the following result.

Theorem 3.2.7. *Max-W-5-UCC is* APX*-hard, even if the edge weights are restricted to be zero, one, or two.* □

Although the status of Max-5-UCC is still open, allowing only one additional edge weight of two already yields an APX-complete problem.

Vornberger [94] used edge weights one, two, and infinity to prove the NP-hardness of computing 5-cycle covers of minimum weight. When seeking minimum weight cycle covers, edges of infinite weight can be considered as nonexistent. To convert his proof into a proof for the NP-hardness of Max-W-5-UCC, there are two possibilities: Either we replace edges of weight infinity by weight zero, weight two by weight $n+1$, and weight one by weight $n+2$, where n is the number of vertices in the graph. Or we consider the graph as being not complete, replace weight two by weight zero, weight one by weight one, and omit edges of weight infinity. Thus, the following corollary is slightly stronger than Vornberger's result since it holds for complete graphs with edge weights from a fixed set.

Corollary 3.2.8. *The budget problem Max-W-5-UCC*$_{\mathrm{B}}$ *is* NP*-hard, even if the edge weights are restricted to be zero, one, or two.* □

The generic reduction together with the gadget used for Max-W-5-UCC works also for Max-W-$\overline{\{4\}}$-UCC. The gadget only requires that cycles of length four are forbidden since otherwise R1 is not satisfied. Thus, all requirements are fulfilled for Max-W-$\overline{\{4\}}$-UCC in exactly the same way as for Max-W-5-UCC. In addition to the APX-hardness of Max-W-$\overline{\{4\}}$-UCC, the reduction also slightly strengthens Vornberger's NP-hardness result for Max-W-$\overline{\{4\}}$-UCC (more precisely, the NP-completeness of its budget problem Max-W-$\overline{\{4\}}$-UCC$_B$) in the same sense as Corollary 3.2.8.

Theorem 3.2.9. *Max-W-*$\overline{\{4\}}$*-UCC is* APX*-hard, even if the edge weights are restricted to be zero, one, or two.* □

Corollary 3.2.10. *The budget problem Max-W-*$\overline{\{4\}}$*-UCC*$_{\mathrm{B}}$ *is* NP*-hard, even if the edge weights are restricted to be zero, one, or two.* □

3.2.3 Max-k-UCC for $k \geq 6$

For the sake of completeness, we describe gadgets that can be used for proving the APX-hardness of Max-k-UCC for $k \geq 6$. Since these hardness results can also be obtained via the uniform reduction presented in the next section, we omit the proofs. Figure 3.2.4 depicts the gadget used for Max-6-UCC while Figure 3.2.5 shows how to traverse it. Its gadget parameter is $s = 7$.

For all $k \geq 7$, the gadget used to prove the APX-hardness of Max-k-UCC is shown in Figure 3.2.6. The gadget merely consists of a cycle of length $k - 1 = 4 + \lfloor \frac{k-5}{2} \rfloor + \lceil \frac{k-5}{2} \rceil$, its gadget parameter is $s = k - 1$. Since $k \geq 7$, $\lfloor \frac{k-5}{2} \rfloor \geq 1$, i.e.

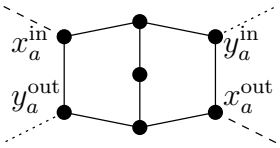


Figure 3.2.4: The edge gadget F_a for an edge $a = \{x, y\} \in F$ that is used to prove the APX-completeness of Max-6-UCC. The solid edges are the internal edges of the gadget that have weight one, internal edges of weight zero are not shown. The dashed and dotted edges are the junctions of x and y, respectively, at F_a.

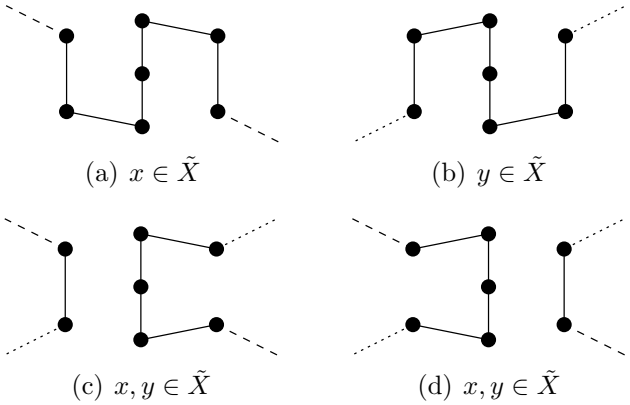


Figure 3.2.5: Traversals of the gadget for Max-6-UCC that achieve maximum weight.

there is at least one vertex between x_a^{in} and y_a^{in} and at least one vertex between x_a^{out} and y_a^{out}. Figure 3.2.7 shows how to traverse such a gadget. Overall, we obtain the following result.

Theorem 3.2.11. *Max-k-UCC is* APX*-complete for all* $k \geq 6$.

3.3 A Uniform Reduction for *L*-Cycle Covers

3.3.1 Clamps

To begin this section, we define so-called *clamps*, which were introduced by Hell et al. [56]. Clamps are crucial for the uniform hardness proof presented later on in this section.

Let $K = (V, E)$ be an undirected graph, let $u, v \in V$ be two vertices of K, and let $L \subseteq \mathcal{U}$. We denote by K_{-u} and K_{-v} the subgraphs of K induced by $V \setminus \{u\}$ and $V \setminus \{v\}$. Moreover, K_{-u-v} denotes the subgraph of G induced by $V \setminus \{u, v\}$. Finally, for $k \in \mathbb{N}$, K^k is the following graph: Let $y_1, \ldots, y_k$ be vertices with $y_i \notin V$, add edges $\{u, y_1\}$, $\{y_i, y_{i+1}\}$ for $1 \leq i \leq k-1$, and $\{y_k, v\}$. For $k = 0$, we directly connect u to v.

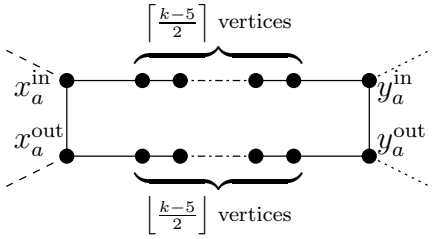


Figure 3.2.6: The edge gadget for Max-k-UCC, $k \geq 7$.

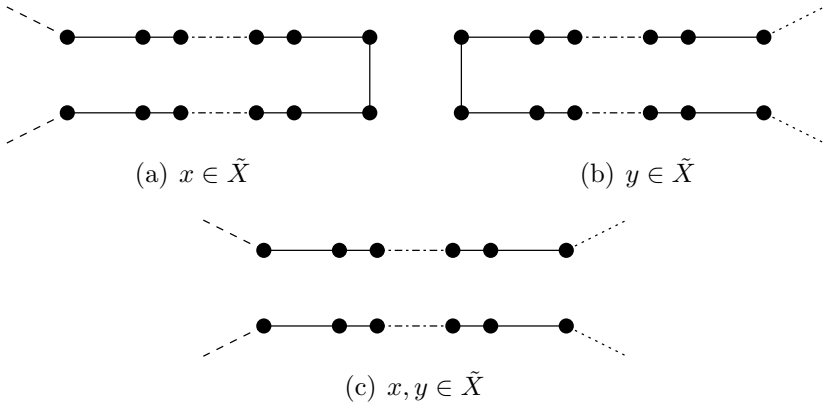


Figure 3.2.7: Traversals of the gadget for Max-k-UCC that achieve maximum weight.

Definition 3.3.1 (Hell et al. [56]). *Let $K = (V, E)$ be an undirected graph, $u, v \in V$, and $L \subseteq \mathcal{U}$. We call K an **L-clamp** with **connectors** u and v if the following properties hold:*

1. *Both K_{-u} and K_{-v} contain an L-cycle cover.*
2. *Neither K nor K_{-u-v} nor K^k for any $k \in \mathbb{N}$ contains an L-cycle cover.*

Hell et al. [56] proved that L-UCC is NP-hard for all L with $\overline{L} \not\subseteq \{3, 4\}$. L-clamps are crucial to their proof. They proved the following result which we will exploit for our reduction.

Lemma 3.3.2 (Hell et al. [56]). *Let $L \subseteq \mathcal{U}$ be non-empty. Then there exists an L-clamp if and only if $\overline{L} \not\subseteq \{3, 4\}$.*

In this and the following section, we are concerned with undirected graphs. In Section 3.3.3, we will extend the notion of L-clamps to directed graphs and prove that directed L-clamps exist for all non-empty sets $L \subseteq \mathcal{D}$ with $L \neq \mathcal{D}$.

Figure 3.3.1 shows an example of an L-clamp for finite L. For other L-clamps, we refer to Hell et al. [56].

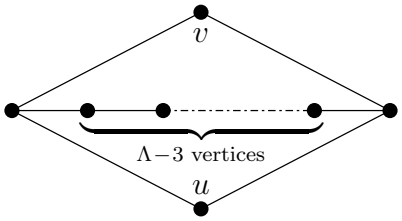


Figure 3.3.1: An L-clamp for finite L with $\max(L) = \Lambda$.

If there exists an L-clamp for some L, then we can assume that the connectors u and v both have degree two since we can remove all edges that are not used in the L-cycle covers of K_{-v} and K_{-u}.

For our purpose, consider any non-empty set $L \subseteq \{3, 4, 5, \ldots\}$ with $\overline{L} \not\subseteq \{3, 4\}$. We fix one L-clamp K with connectors $u, v \in V$ arbitrarily and refer to it in the following as the L-clamp, although there exists more than one L-clamp. Let σ be the number of vertices of K.

We are concerned with edge-weighted graphs. Therefore, we transfer the notion of clamps to graphs with edge weights zero and one in the obvious way: Let G be an undirected complete graph with vertex set V and edge weights zero and one and let K be an L-clamp. Let $U \subseteq V$. We say that U is an L-clamp with connectors $u, v \in U$ if the subgraph of G induced by U restricted to the edges of weight one is isomorphic to K with u and v mapped to connectors of K.

Let C be a cycle cover of G. For any $V' \subseteq V$, we say that $\boldsymbol{V'}$ **is isolated in** $\boldsymbol{C}$ if there is no edge in C connecting V' to $V \setminus V'$.

Let U be a clamp with connectors u and v in G. We say that $\boldsymbol{U}$ **absorbs** $\boldsymbol{u}$ and $\boldsymbol{U}$ **expels** $\boldsymbol{v}$ if $U \setminus \{v\}$ is isolated in C. We call U **healthy in** $\boldsymbol{C}$ if U either absorbs u and expels v or absorbs v and expels u and $w_U(C) = \sigma - 1$.

Let us prove some properties of L-clamps in edge-weighted graphs. In particular, we will prove that $\sigma - 1$ is the maximum weight that an L-cycle cover can achieve within an L-clamp. Thus, healthy clamps achieve maximum weight.

Lemma 3.3.3. *Let G be an undirected graph with vertex set V and edge weights zero and one, and let $U \subseteq V$ be an L-clamp with connectors u and v in G. Let C be an arbitrary L-cycle cover of G and $|U| = \sigma$. Then the following properties hold:*

1. *$w_U(C) \leq \sigma - 1$.*

2. *If there are 2α external edges at U in C, then $w_U(C) \leq \sigma - \alpha$.*

3. *Assume that U absorbs u. Then there exists an L-cycle cover $\tilde{C}$ that differs from C only in the internal edges of U and has $w_U(\tilde{C}) = \sigma - 1$.*

   The same holds if U absorbs v.

4. Assume that there is one external edge at U in C that is incident to u and one external edge at U in C that is incident to v. Then $w_U(C) \leq \sigma - 2$.

Proof. If $w_U(C) = \sigma$ was true, then U would contain an L-cycle cover consisting solely of weight one edges since $|U| = \sigma$. This would contradict U being an L-clamp.

The second claim follows immediately from $|U| = \sigma$.

Since U is an L-clamp, $U \setminus \{u\}$ and $U \setminus \{v\}$ both contain an L-cycle cover consisting solely of weight one edges. Since there are $\sigma - 1$ edges in $U \setminus \{u\}$ and $U \setminus \{v\}$, the third claim follows.

The fourth claim remains to be proved. Both u and v are incident to an external edge in C. Thus, if there is any further external edge at U in C, we have at least four external edges and thus $w_U(C) \leq \sigma - 2$. So assume that there are only two external edges at U in C, one incident to u and the other incident to v. Thus, u and v are on the same cycle in C. Let k be the number of vertices of this cycle that are not in U. We have $\sigma - 1$ internal edges of U in C. If all of them weigh one, then this contradicts the fact that K^k does not contain an L-cycle cover, where K is the L-clamp. □

3.3.2 *L*-Cycle Covers in Undirected Graphs

Let $L \subseteq \mathcal{U}$ be non-empty with $\overline{L} \not\subseteq \{3, 4\}$. Thus, L-clamps exist and we fix one as in the previous section. Let σ be the number of vertices in the L-clamp. Let $\lambda = \min(L)$. (This choice is arbitrary. We could choose any number in L.) We will reduce Min-Vertex-Cover(λ) to Max-L-UCC. Min-Vertex-Cover(λ) is APX-complete since $\lambda \geq 3$ (see Section 3.4).

Let $H = (X, F)$ be an instance of Min-Vertex-Cover(λ) with $n = |X|$ vertices and $m = \lambda n/2 = |F|$ edges. Our instance G for Max-L-UCC consists of λ subgraphs $G_1, \ldots, G_\lambda$, each containing $2\sigma m$ vertices. We start by describing G_1. Then we state the differences between G_1 and $G_2, \ldots, G_\lambda$ and say to which edges between these graphs we assign weight one.

Let $a = \{x, y\} \in F$ be any edge of H. We construct an edge gadget F_a for a that consists of two L-clamps X_a^1 and Y_a^1 and one additional vertex t_a^1 as shown in Figure 3.3.2. The connectors of X_a^1 are x_a^1 and z_a^1 while the connectors of Y_a^1 are y_a^1 and z_a^1, i.e. X_a^1 and Y_a^1 share the connector z_a^1. Let p_a^1 and q_a^1 be the two unique vertices in Y_a^1 that share a weight one edge with z_a^1. (The choice of Y_a^1 is arbitrary, we could choose the corresponding vertices in X_a^1 as well.) We assign weight one to both $\{p_a^1, t_a^1\}$ and $\{q_a^1, t_a^1\}$. Thus, the vertex t_a^1 can also serve as a connector for Y_a^1. We extend the notions of absorbing and expelling appropriately: Y_a^1 absorbs t_a^1 and expels both y_a^1 and z_a^1 if $(Y_a^1 \cup \{t_a^1\}) \setminus \{y_a^1, z_a^1\}$ is isolated. If Y_a^1 absorbs y_a^1 or z_a^1, then Y_a^1 expels t_a^1.

Now let $x \in X$ be any vertex of H and let $a_1, \ldots, a_\lambda \in F$ be the λ edges that are incident to x. We connect the vertices $x_{a_1}^1, \ldots, x_{a_\lambda}^1$ to form a path

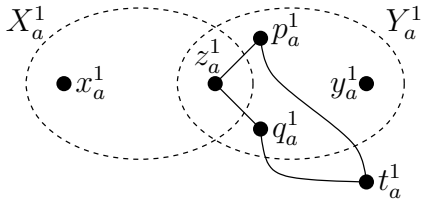


Figure 3.3.2: The edge gadget for $a = \{x, y\}$ that consists of two L-clamps. The vertex z_a^1 is the only vertex that belongs to both clamps X_a^1 and Y_a^1.

by assigning weight one to the edges $\{x_{a_\eta}^1, x_{a_{\eta+1}}^1\}$ for $\eta \in [\lambda - 1]$. Together with edge $\{x_{a_\lambda}^1, x_{a_1}^1\}$, these edges form a cycle of length $\lambda \in L$, but note that $w(\{x_{a_\lambda}^1, x_{a_1}^1\}) = 0$. These λ edges are called the **junctions of x**. The **junctions at F_a** for some $a = \{x, y\} \in F$ are the junctions of x and y that are incident to F_a. Overall, the graph G_1 consists of $2\sigma m$ vertices since every edge gadget consists of 2σ vertices.

The graphs $G_2, \ldots, G_\lambda$ are almost exact copies of G_1. The graph G_ξ, $\xi \in \{2, \ldots, \lambda\}$ has clamps X_a^ξ and Y_a^ξ and vertices $x_a^\xi, y_a^\xi, z_a^\xi, t_a^\xi, p_a^\xi, q_a^\xi$ for each edge $a = \{x, y\} \in F$, just as above. The edge weights are also identical with the single exception that the edge $\{x_{a_\lambda}^\xi, x_{a_1}^\xi\}$ also has weight one. Note that we only use the term "gadget" for the subgraphs of G_1 defined above although almost the same subgraphs occur in $G_2, \ldots, G_\lambda$ as well. Similarly, the term "junction" refers only to an edge in G_1 as defined above. The copies in $G_2, \ldots, G_\lambda$ of a junction in G_1 are not called junctions.

Finally, we describe how to connect $G_1, \ldots, G_\lambda$ with each other. For every edge $a \in F$, there are λ vertices $t_a^1, \ldots, t_a^\lambda$. These are connected to form a cycle consisting solely of weight one edges, i.e. we assign weight one to all edges $\{t_a^\xi, t_a^{\xi+1}\}$ for $\xi \in [\lambda - 1]$ and to $\{t_a^\lambda, t_a^1\}$. Figure 3.3.3 shows an example of the whole construction from the viewpoint of a single vertex.

As in the previous section, we call edges that are not junctions but connect two different gadgets **illegal**. Edges with both vertices in the same gadget are again called internal edges. In addition to junctions, illegal edges, and internal edges, we have a fourth kind of edges: The ***t*-edges** of F_a for $a \in F$ are the two edges $\{t_a^1, t_a^2\}$ and $\{t_a^1, t_a^\lambda\}$. The t-edges are not illegal. All other edges connecting G_1 to G_ξ for $\xi \neq 1$ are illegal.

Let C be any subset of the edges of the graph G thus constructed, and let $a = \{x, y\} \in F$ be an arbitrary edge of H. We say that **C legally connects F_a** if the following properties are fulfilled:

- C contains no illegal edges incident to F_a and exactly two or four junctions at F_a.
- If C contains exactly two junctions at F_a, then these belong to the same vertex and there are two t-edges at F_a in C.

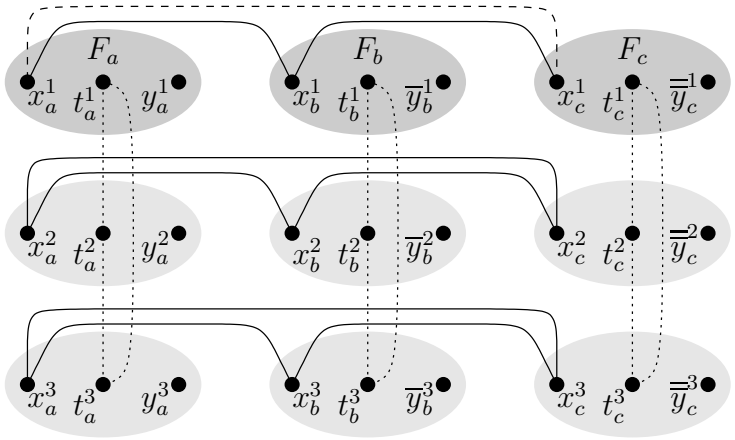


Figure 3.3.3: The construction for a vertex $x \in X$ incident to edges $a, b, c \in F$ for $\lambda = 3$ (Figure 3.2.1(a) on page 25 shows the corresponding graph). The dark grey areas are the edge gadgets F_a, F_b, and F_c. Their copies in G_2 and G_3 are light grey. The cycles connecting the t-vertices are dotted. The cycles connecting the x-vertices are solid, except for the edge $\{x_c^1, x_a^1\}$, which has weight zero and is dashed. The vertices $z_a^1, \ldots, z_c^3$ are not shown for legibility.

- If C contains four junctions at F_a, then these are the only external edges in C incident to F_a. In particular, C does not contain t-edges at F_a.

We call C **legal** if C legally connects all gadgets.

In analogy to Lemma 3.2.1 in the previous section, we have the following lemma. The proof is identical to the proof of Lemma 3.2.1 and is therefore omitted.

Lemma 3.3.4. *Let $\tilde{C}$ be an arbitrary legal subset of the edges of G. Then for all $x \in X$, either all junctions of x are in $\tilde{C}$ or no junction at all of x is in $\tilde{C}$.*

Thus, from a legal L-cycle cover $\tilde{C}$, we obtain the subset $\tilde{X}$ containing all vertices whose junctions are in $\tilde{C}$. Again, the set $\tilde{X}$ obtained from a legal subset $\tilde{C}$ is a vertex cover of H. We omit the proof of the following lemma as well and refer to the proof of Lemma 3.2.2.

Lemma 3.3.5. *Let $\tilde{C}$ be a legal subset of the edges of G. Then the set*

$$\tilde{X} = \{x \mid \textit{the junctions of } x \textit{ are in } \tilde{C}\}$$

obtained from $\tilde{C}$ is a vertex cover of H.

We only considered G_1 when defining the terms "legally connected" and "legal". This is because in G_1, we lose weight one for putting x into the vertex cover since the junction $\{x_{a_\lambda}^1, x_{a_1}^1\}$ weighs zero. The other $\lambda - 1$ copies of the construction are only needed for the following reason: If, for some edge a, the

vertex t_a^1 is not absorbed by Y_a^1, it has to be part of some other cycle. Since we want a reduction that works for all L with $\overline{L} \not\subseteq \{3,4\}$, we do not know much about L except that $\lambda \in L$.

The next lemma is an analogue of Lemma 3.2.3 and instantiates Generic Lemma 3.1.1 with $\gamma = \sigma\lambda^2$.

Lemma 3.3.6. *Let $\tilde{X}$ be a vertex cover of size $\tilde{n}$ of H. Then G contains an L-cycle cover $\tilde{C}$ with $w(\tilde{C}) = 2\sigma\lambda m - \tilde{n}$.*

Proof. We start by describing $\tilde{C}$ in G_1. For every vertex $x \in \tilde{X}$, the cycle consisting of all λ junctions is in $\tilde{C}$. Let $a = \{x, y\} \in F$ be any edge. Then either x or y or both are in $\tilde{X}$. If only x is in $\tilde{X}$, we let X_a^1 absorb z_a^1, Y_a^1 absorb y_a^1, and t_a^1 is free. If only y is in $\tilde{X}$, we let X_a^1 absorb x_a^1, Y_a^1 absorb z_a^1 and t_a^1 is again free. If both x and y are in $\tilde{X}$, then we let X_a^1 absorb z_a^1 and Y_a^1 absorb t_a^1, thus t_a^1 is not free.

We perform the same construction for G_1 for all copies $G_2, \ldots, G_\lambda$. If t_a^1 is free, then $t_a^2, \ldots, t_a^\lambda$ are also not part of any cycle yet and we let them form a cycle of length λ in $\tilde{C}$.

Clearly, $\tilde{C}$ is legal. Furthermore, $\tilde{C}$ is an L-cycle cover: Every cycle either has length $\lambda \in L$ or lies totally inside a single L-clamp. Since all L-clamps are healthy in $\tilde{C}$, $\tilde{C}$ is an L-cycle cover.

Let us calculate the weight of $\tilde{C}$. All edges used within $G_2, \ldots, G_\lambda$ have weight one. The only edges that connect different copies G_ξ and $G_{\xi'}$ are edges $\{t_a^\xi, t_a^{\xi+1}\}$ with $\xi' = \xi + 1$ (with interpreting $n + 1$ as 1), which have weight one as well. Almost all edges used in G_1 also have weight one; the only exception is one junction of weight zero for each $x \in \tilde{X}$.

Since $|\tilde{X}| = \tilde{n}$, there are $\tilde{n}$ edges of weight zero in $\tilde{C}$. The graph G contains $2\sigma\lambda m$ vertices, thus $\tilde{C}$ contains $2\sigma\lambda m$ edges, $2\sigma\lambda m - \tilde{n}$ of which have weight one. □

Let C be an L-cycle cover of G and let $a \in F$. We define $W_{F_a}(C)$ as the sum of the weights of all internal edges of F_a plus half the number of t-edges in C at F_a. Analogously, $W_{G_\xi}(C)$ is the number of weight one edges with both vertices in G_ξ plus half the number of weight one edges with exactly one vertex in G_ξ.

Lemma 3.3.7. *Let C be an L-cycle cover and let j be the number of weight one junctions in C. Then*

$$w(C) = j + \sum_{a \in F} W_{F_a}(C) + \sum_{\xi=2}^{\lambda} W_{G_\xi}(C) \,.$$

Proof. Every edge with both vertices in the same G_ξ is counted once. The only edges of weight one between different G_ξ are the edges $\{t_a^\xi, t_a^{\xi+1}\}$ and $\{t_a^\lambda, t_a^1\}$. These are counted with one half in both $W_{G_\xi}(C)$ and $W_{G_{\xi+1}}(C)$ for $2 \le \xi \le \lambda - 1$ or one half in both $W_{G_\xi}(C)$ and $W_{F_a}(C)$ for $\xi \in \{2, \lambda\}$. □

In a legal L-cycle cover $\tilde{C}$ as described in Lemma 3.3.6, we have $W_{G_\xi}(\tilde{C}) = 2\sigma m$ for all $\xi \in \{2, \dots, \lambda\}$ since every vertex in G_ξ is only incident to edges of weight one of the cycle cover by construction.

Next, we show some properties of the edge gadgets. These properties resemble the gadget requirement in the previous section with 2σ as the gadget parameter.

Lemma 3.3.8. *Let C be an arbitrary L-cycle cover of G and let $a = \{x, y\} \in F$ be an arbitrary edge of H. Then the following properties hold:*

1. $W_{F_a}(C) \leq 2\sigma - 1$.
2. *If there are 2α external edges at F_a in C, then $W_{F_a}(C) \leq 2\sigma - \alpha$.*
3. *If there is one junction of x and one junction of y at F_a in C, then $W_{F_a}(C) \leq 2\sigma - 2$.*
4. *Let C' be an arbitrary subset of edges of G that legally connect F_a. Assume that there are 2α junctions ($\alpha \in \{1, 2\}$) at F_a in C'. Let C'' be obtained from C' by making the two clamps in F_a healthy, i.e. C'' differs from C' only in F_a's internal edges. Then $W_{F_a}(C'') = 2\sigma - \alpha$.*
5. *Assume that C' is a legal subset of the edges of G, that for all $a \in F$ both clamps of F_a are healthy, and that C' traverses $G_2, \dots, G_\lambda$ in exactly the same way as G_1. Then C' is an L-cycle cover.*

Proof. If $W_{F_a}(C) > 2\sigma - 1$, then $W_{F_a}(C) = 2\sigma$. Then there would be no external edges in C at F_a and F_a would contain an L-cycle cover consisting solely of weight one edges. This would imply that X_a^1 must absorb x_a^1 and Y_a^1 must absorb y_a^1. Thus, z_a^1 is incident to two edges of weight zero contradicting $W_{F_a} = 2\sigma$.

Since F_a consists of 2σ vertices, the second claim holds.

If there is one junction of x and one junction of y at F_a in C and there are other external edges at F_a in C, then $W_{F_a}(C) \leq 2\sigma - 2$ according to the second claim. If there is an internal edge of F_a in C that has weight zero, we are done as well. Otherwise, z_a^1 is incident to some vertex in X_a^1 and thus $w(X_a^1) \leq \sigma - 2$ according to Lemma 3.3.3(4), which proves the third claim of the lemma.

The fourth claim follows from the construction and Lemma 3.3.3(3).

The fifth claim follows from the construction: C' consists solely of cycles and every cycle is either inside a healthy L-clamp or has length $\lambda \in L$. □

Let us now instantiate Generic Lemma 3.1.2.

Lemma 3.3.9. *Given an arbitrary L-cycle cover C, we can compute a legal L-cycle cover $\tilde{C}$ with $w(\tilde{C}) \geq w(C)$ in polynomial time.*

Proof. We proceed similarly as in the proof of Lemma 3.2.4:

1. Let C' be C with all illegal edges with at least one endpoint in G_1 removed.

2. For all $x \in X$ in arbitrary order: If at least one junction of x is in C, then put all junctions of x into C'.

3. For all $a = \{x, y\} \in F$ in arbitrary order: If neither the junctions of x nor the junctions of y are in C', choose arbitrarily one vertex of a, say x, and add all junctions of x to C'.

4. Rearrange C' within G_1 such that all clamps are healthy in C'.

5. Rearrange C' such that all $G_2, \ldots, G_\lambda$ are traversed exactly like G_1.

6. For all $a \in F$: If $t_a^1, \ldots, t_a^\xi$ are not already absorbed by clamps, let them form a cycle of length λ. Call the result $\tilde{C}$.

The running time of the algorithm is obviously polynomial. Moreover, $\tilde{C}$ is a legal L-cycle cover according to Lemma 3.3.8(5). What remains is to prove $w(\tilde{C}) \geq w(C)$.

Let $w(C) = j + \sum_{a \in F} W_{F_a}(C) + \sum_{\xi=2}^{\lambda} W_{G_\xi}(C)$ be the weight of C according to Lemma 3.3.7, i.e. C contains j junctions of weight one. Analogously, let $w(\tilde{C}) = \tilde{j} + \sum_{a \in F} W_{F_a}(\tilde{C}) + \sum_{\xi=2}^{\lambda} W_{G_\xi}(\tilde{C})$, i.e. $\tilde{j}$ is the number of weight one edges in $\tilde{C}$.

All illegal edges have weight zero, and we do not remove any junctions. We have $W_{G_\xi}(\tilde{C}) = 2\sigma m$ for all ξ, which is maximal. Thus, no weight is lost in this way. What remains is to consider the internal edges of the gadgets and the t-edges.

Let $a = \{x, y\}$ be an arbitrary edge of H. If $W_{F_a}(C) \leq W_{F_a}(\tilde{C})$, then nothing has to be shown. Those gadgets F_a with $W_{F_a}(C) > W_{F_a}(\tilde{C})$ remain to be considered. We have $W_{F_a}(\tilde{C}) \geq 2\sigma - 2$ according to Lemma 3.3.8(4) and $W_{F_a}(C) \leq 2\sigma - 1$ according to Lemma 3.3.8(1). Thus, $W_{F_a}(C) = 2\sigma - 1$ and $W_{F_a}(\tilde{C}) = 2\sigma - 2 = W_{F_a}(C) - 1$ for all $a \in F$ with $W_{F_a}(C) > W_{F_a}(\tilde{C})$. As in the proof of Lemma 3.2.4, our aim is to prove that for all such gadgets, there is a junction of weight one in $\tilde{C}$ that is not in C and can thus compensate for the loss of weight one in F_a. This means that we have to prove that $\tilde{j}$ is at least j plus the number of edges a with $W_{F_a}(C) > W_{F_a}(\tilde{C})$.

If $W_{F_a}(C) = 2\sigma - 1$, then according to Lemma 3.3.8(3), the junctions at F_a in C (if there are any) belong to the same vertex. Since $W_{F_a}(\tilde{C}) = 2\sigma - 2$, all four junctions at F_a are in $\tilde{C}$. Thus, while executing the above algorithm there is a moment at which at least one of, say, y's junctions at F_a is in C', and the junctions of x are added in the next step. The notion that a vertex $\boldsymbol{x}$ **compensates** $\boldsymbol{F_a}$ is defined in exactly the same way as in the proof of Lemma 3.2.4. Thus, every gadget F_a with $W_{F_a}(\tilde{C}) < W_{F_a}(C)$ is compensated by some vertex $x \in a$.

It remains to be shown that the number of gadgets that are compensated by some vertex is at most equal to the number of weight one junctions added to C'. Let $\eta \in \{0, \ldots, \lambda\}$ be the number of junctions of x in C. If $\eta = \lambda$, then x does

not compensate any gadget. If $\eta = 0$, i.e. C does not contain any of x's junctions, then the junctions of x are added during Step 3 of the algorithm because there is some edge $a \in F$ with $x \in a$ such that there is no junction at all in C' at F_a before adding x's junctions. Thus, x does not compensate F_a. At most $\lambda - 1$ gadgets are compensated by x, and $\lambda - 1$ junctions of x have weight one. The case that remains is $\eta \in [\lambda - 1]$. Then $\lambda - \eta$ junctions of x are added and at least $\lambda - \eta - 1$ of them have weight one. On the other hand, there are at least $\eta + 1$ gadgets F_a such that at least one junction of x at F_a is already in C: Every junction is at two gadgets, and thus η junctions are at $\eta + 1$ or more gadgets. Thus, at most $\lambda - \eta - 1$ gadgets are compensated by x.

The lemma is proved since $\tilde{C}$ is a legal k-cycle cover with $w(\tilde{C}) \geq w(C)$. □

Finally, we prove the following counterpart to Lemma 3.3.6. This lemma instantiates Generic Lemma 3.1.3.

Lemma 3.3.10. *Let $\tilde{C}$ be the L-cycle cover constructed as described in the proof of Lemma 3.3.9. Choose $\tilde{n}$ such that $w(\tilde{C}) = 2\sigma\lambda m - \tilde{n}$. Let $\tilde{X} = \{x \mid x$'s junctions are in $\tilde{C}\} \subseteq X$ be the subset obtained from $\tilde{C}$. Then $|\tilde{X}| = \tilde{n}$.*

Proof. The proof is similar to the proof of Lemma 3.3.6. We set the weight of all junctions to one. With respect to the modified edge weights, the weight of $\tilde{C}$ is $2\sigma\lambda m$. Thus, $\tilde{n}$ is the number of weight zero junctions in $\tilde{C}$, which is just $|\tilde{X}|$. □

All three generic lemmas are instantiated, and we obtain the following result from Lemma 3.1.4.

Theorem 3.3.11. *For all $L \subseteq \mathcal{U}$ with $\overline{L} \not\subseteq \{3, 4\}$, Max-$L$-UCC is* APX*-hard.*

3.3.3 Clamps in Directed Graphs

The aim of this section is to prove a counterpart to Lemma 3.3.2 (for the existence of L-clamps) for directed graphs. Let $K = (V, E)$ be a directed graph and $u, v \in V$. Again, K_{-u}, K_{-v}, and K_{-u-v} denote the graphs obtained by deleting u, v, and both u and v, respectively. For $k \in \mathbb{N}$, K_u^k denotes the following graph: Let $y_1, \ldots, y_k \notin V$ be new vertices and add edges $(u, y_1), (y_1, y_2), \ldots, (y_k, v)$. For $k = 0$, we add the edge (u, v). The graph K_v^k is similarly defined, except that we now start at v, i.e. we add the edges $(v, y_1), (y_1, y_2), \ldots, (y_k, u)$. K_v^0 is K with the additional edge (v, u).

Now we can define clamps for directed graphs.

Definition 3.3.12 (Directed L-Clamp). *Let $L \subseteq \mathcal{D}$. A directed graph $K = (V, E)$ with $u, v \in V$ is a directed L-clamp with connectors u and v if the following properties hold:*

- *Both K_{-u} and K_{-v} contain an L-cycle cover.*

- *Neither K nor K_{-u-v} nor K_u^k nor K_v^k for any $k \in \mathbb{N}$ contains an L-cycle cover.*

Let us now prove that directed L-clamps exist for almost all L.

Theorem 3.3.13. *Let $L \subseteq \mathcal{D}$ be non-empty. Then there exists a directed L-clamp if and only if $L \neq \mathcal{D}$.*

Proof. We first prove that directed L-clamps exist for all non-empty sets $L \subseteq \mathcal{D}$ with $L \neq \mathcal{D}$. We start by considering finite L and postpone the two cases that $\overline{L}$ is finite and that both L and $\overline{L}$ are infinite.

If L is finite, $\max(L) = \Lambda$ exists. For $L = \{2\}$, the graph shown in Figure 3.3.4(a) is a directed L-clamp: either u or v forms a cycle of length two with x_1, and there are no other possibilities. Otherwise, we have $\Lambda \geq 3$. Figure 3.3.4(b) shows a directed L-clamp for this case, which is a directed variant of the undirected clamp shown in Figure 3.3.1: $x_1, \ldots, x_{\Lambda-1}$ must be on the same cycle. Since Λ is the maximum length allowed, these vertices form a cycle of length Λ with either u or v. Again, there are no other possibilities.

Now we consider finite $\overline{L}$. We start by considering the special case of $\overline{L} = \{2\}$. In this case, Figure 3.3.4(c) shows an L-clamp: x_1, x_2, and x_3 must be on the same cycle since length two is forbidden. This cycle must include u or v. If it includes u, then x_1 is left via (x_1, u) and x_3 is entered via (u, x_3). Thus, we have a cycle of length four in this case. If the cycle includes v, we can argue similarly.

Otherwise, $\max(\overline{L}) = \Lambda \geq 3$ and $\Lambda + 2 \in L$ and the graph shown in Figure 3.3.4(d) is an L-clamp: The vertices $x_1, \ldots, x_{\Lambda-1}$ must all be on the same cycle. Thus, either (y, x_1) or (z, x_1) is in the cycle cover. By symmetry, it suffices to consider the first case. Since $\Lambda \notin L$, the edge $(x_{\Lambda-1}, y)$ cannot be in the cycle cover. Thus, (v, y) and $(x_{\Lambda-1}, z)$ and hence (z, v) are in the cycle cover.

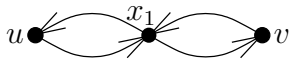
The case that remains to be considered is that both L and $\overline{L}$ are infinite. We distinguish two subcases. Either there exists a $\Lambda \geq 4$ with $\Lambda, \Lambda + 2 \notin L$ and $\Lambda + 1 \in L$. In this case, the graph shown in Figure 3.3.4(e) is an L-clamp: $x_1, \ldots, x_{\lfloor \Lambda/2 \rfloor}$ and $x_{\lfloor \Lambda/2 \rfloor + 1}, \ldots, x_\Lambda$ must be on the same cycle. Since the lengths Λ and $\Lambda + 2$ are not allowed, either v or u is expelled and the other vertex is absorbed.

Or there does not exist a Λ with $\Lambda, \Lambda + 2 \notin L$ and $\Lambda + 1 \in L$. Since both L and $\overline{L}$ are infinite, there exists a $\Lambda \geq 3$ with $\Lambda \notin L$ and $\Lambda + 2 \in L$ and we can use the graph already used for finite $\overline{L}$ (Figure 3.3.4(d)) as a directed L-clamp.

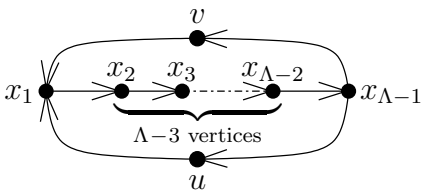
From Lemma 3.3.14 below we obtain the fact that $\mathcal{D}$-clamps do not exist, which completes the proof. □

Lemma 3.3.14. *Let $G = (V, E)$ be a directed graph and let $u, v \in V$. If G_{-u} and G_{-v} both contain a cycle cover, then*

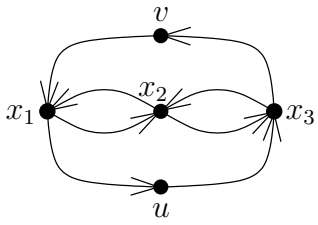
- *both G and G_{-u-v} contain cycle covers or*



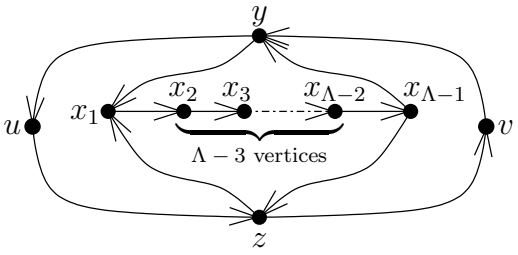
(a) A $\{2\}$-clamp.



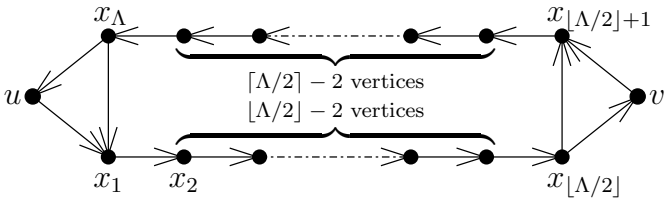
(b) An L-clamp for finite L with $\max(L) = \Lambda \geq 3$.



(c) A $\overline{\{2\}}$-clamp.



(d) An L-clamp for $\Lambda \notin L$ and $\Lambda + 2 \in L$ with $\Lambda \geq 3$.



(e) An L-clamp for $\Lambda, \Lambda + 2 \notin L$ and $\Lambda + 1 \in L$ with $\Lambda \geq 4$.

Figure 3.3.4: Directed L-clamps. The connectors are u and v, the internal vertices are $x_1, x_2, \ldots$ and y, z.

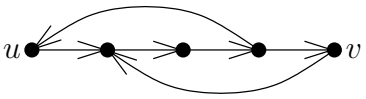
- *all G_u^k and G_v^k for $k \in \mathbb{N}$ contain cycle covers.*

Proof. Let E_{-u} and E_{-v} be the sets of edges of the cycle covers of G_{-u} and G_{-v}, respectively. We construct two sequences of edges $P = (e_1, e_2, \ldots)$ and $P' = (e'_1, e'_2, \ldots)$. These sequences can be viewed as *augmenting paths* and we use them to construct cycle covers of G_{-u-v} and G or G_u^k and G_v^k. The sequence P is given uniquely by traversing edges of E_{-v} forwards and edges of E_{-u} backwards:

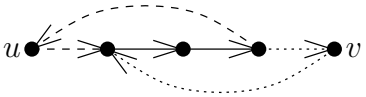
- $e_1 = (u, x_1)$ is the unique outgoing edge of $u = x_0$ in E_{-v}.
- If $e_i = (x_{i-1}, x_i) \in E_{-v}$, i.e. if i is odd, then $e_{i+1} = (x_{i+1}, x_i) \in E_{-u}$ is the unique incoming edge of x_i in E_{-u}.
- If $e_i = (x_i, x_{i-1}) \in E_{-u}$, i.e. if i is even, then $e_{i+1} = (x_i, x_{i+1}) \in E_{-v}$ is the unique outgoing edge of x_i in E_{-v}.
- If in any of the above steps no extension of P is possible, then stop.

Let $P = (e_1, \ldots, e_\ell)$. We observe two properties of the sequence P.

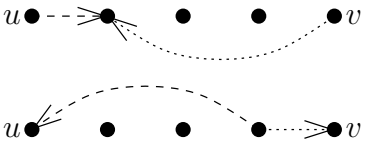
Lemma 3.3.15. *1. No edge appears more than once in P.*



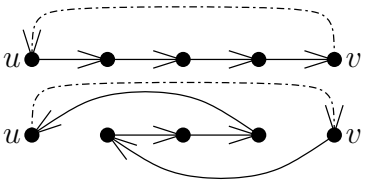
(a) A graph G.



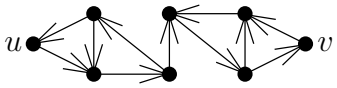
(b) Cycle covers of G_{-v} (dashed and solid) and G_{-u} (dotted and solid).



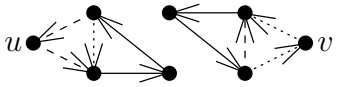
(c) P (top) and P' (bottom). Dashed and dotted edges belong to the cycle covers of G_{-v} and G_{-u}, respectively.



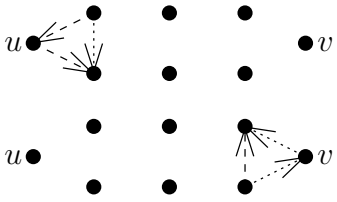
(d) Cycle covers of G_v^0 (top) and G_u^0 (bottom).



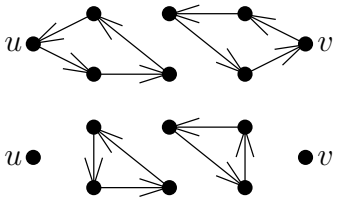
(e) Another graph G.



(f) Cycle covers of G_{-v} (dashed and solid) and G_{-u} (dotted and solid).



(g) P (top) and P' (bottom).



(h) Cycle covers of G (top) and G_{-u-v} (bottom).

Figure 3.3.5: Constructing new cycle covers from the sequences P and P'.

2. *If ℓ is odd, i.e. $e_\ell \in E_{-v}$, then $e_\ell = (x_{\ell-1}, u)$. If ℓ is even, i.e. $e_\ell \in E_{-u}$, then $e_\ell = (v, x_{\ell-1})$.*

Proof. Assume the contrary of the first claim and let $e_i = e_j$ $(i \neq j)$ be an edge that appears at least twice in P such that i is minimal. If $i = 1$, then $e_j = (u, x_1) \in E_{-v}$. This would imply $e_{j-1} = (u, x_{j-2}) \in E_{-u}$, a contradiction. If $i > 1$, then assume $e_i = (x_{i-1}, x_i) \in E_{-v}$ without loss of generality. Since $\text{outdeg}_{E_{-u}}(x_{i-1}) = 1$, the edge $e_{i-1} = e_{j-1}$ is uniquely determined, which contradicts the minimality of i.

Let us now prove the second claim. Without loss of generality, we assume that the last edge e_ℓ belongs to E_{-v}. Let $e_\ell = (x_{\ell-1}, x_\ell)$. The path P cannot be extended, which implies that there does not exist an edge $(x_{\ell+1}, x_\ell) \in E_{-u}$. Since E_{-u} is a cycle cover of G_{-u}, this implies $x_\ell = u$ and completes the proof. □

Now we build the sequence P' analogously, except that we start with the edge $e'_1 = (x'_1, v) \in E_{-u}$. Again, we traverse edges of E_{-v} forwards and edges of E_{-u}

backwards. Let $P' = (e'_1, \ldots, e'_{\ell'})$.

No edge appears in both P and P' as can be proved similarly to the first observation above. Moreover, either P ends at u and P' ends at v or vice versa: $e_\ell = (x_{\ell-1}, u)$ if and only if $e'_{\ell'} = (v, x_{\ell'-1})$ and $e_\ell = (v, x_{\ell-1})$ if and only if $e'_{\ell'} = (x_{\ell'-1}, u)$.

Let $P_{-u} \subseteq E_{-u}$ denote the set of edges of E_{-u} that are part of P. The sets P_{-v}, P'_{-u}, P'_{-v} are defined similarly.

Two examples are shown in Figure 3.3.5: Figures 3.3.5(a) and 3.3.5(b) show a graph with its cycle covers, while Figure 3.3.5(c) depicts P and P', the former starting at u and ending at v and the latter starting at v and ending at u. Figures 3.3.5(e), 3.3.5(f), and 3.3.5(g) show another example graph, this time P starts and ends at u and P' starts and ends at v.

We distinguish two cases. Let us start with the case that P starts at u and ends at v and, consequently, P' starts at v and ends at u. Then

$$E_u^0 = (E_{-v} \setminus P_{-v}) \cup P_{-u} \cup \{(u, v)\}$$

is a cycle cover of G_u^0. To prove this, we have to show $\text{indeg}_{E_u^0}(x) = \text{outdeg}_{E_u^0}(x) = 1$ for all $x \in V$:

- We removed the outgoing edge of u in E_{-v}, which is in P_{-v}. The incoming edge of u in E_{-v} is left. P_{-u} does not contain any edge incident to u and (u, v) is an outgoing edge of u. Thus, $\text{indeg}_{E_u^0}(u) = \text{outdeg}_{E_u^0}(u) = 1$.
- There is no edge incident to v in E_{-v}. P_{-u} contains an outgoing edge of v and (u, v) is an incoming edge of v. Thus, $\text{indeg}_{E_u^0}(v) = \text{outdeg}_{E_u^0}(v) = 1$.
- For all $x \in V \setminus \{u, v\}$, either both P_{-v} and P_{-u} contain an incoming edge of x or none of them does. Analogously, either both P_{-v} and P_{-u} contain an outgoing edge of x or none of them does. Thus, replacing P_{-v} by P_{-u} changes neither $\text{indeg}(x)$ nor $\text{outdeg}(x)$.

By replacing the edge (u, v) by a path $(u, y_1), \ldots, (y_k, v)$, we obtain a cycle cover of G_u^k for all $k \in \mathbb{N}$.

A cycle cover of G_v^0 is obtained similarly:

$$E_v^0 = (E_{-u} \setminus P_{-u}) \cup P_{-v} \cup \{(v, u)\} .$$

As above, we obtain cycle covers of G_v^k by replacing the edge (v, u) by a path $(v, y_1), \ldots, (y_k, u)$.

Figure 3.3.5(d) shows an example of how the new cycle covers are obtained.

The case that remains to be considered is that P starts and ends at u and P' starts and ends at v. In this case,

$$(E_{-v} \setminus P_{-u}) \cup P_{-v} \text{ and}$$
$$(E_{-u} \setminus P'_{-v}) \cup P'_{-u}$$

are cycle covers of G and

$$(E_{-v} \setminus P_{-v}) \cup P_{-u} \text{ and}$$
$$(E_{-u} \setminus P'_{-u}) \cup P'_{-v}$$

are cycle covers of G_{-u-v}. The proof is similar to the previous case above. Figure 3.3.5(h) shows an example. □

For directed clamps on edge-weighted graphs, we have the same properties as for undirected graphs: Lemma 3.3.3 holds also for directed graphs.

3.3.4 *L*-Cycle Covers in Directed Graphs

From the hardness results in the previous sections and the work by Hell et al. [56], we obtain the NP-hardness and APX-hardness of L-DCC and Max-L-DCC, respectively, for all L with $2 \notin L$ and $\overline{L} \not\subseteq \{2, 3, 4\}$: we use the same reduction as for undirected cycle covers and replace every undirected edge $\{u, v\}$ by a pair of directed edges (u, v) and (v, u). However, this does not work if $2 \in L$ and also leaves open the cases when $\overline{L} \subsetneq \{2, 3, 4\}$. If $L = \{2\}$, then L-DCC, Max-L-DCC, and Max-W-L-DCC can easily be solved in polynomial time: Replace every pair of edges (u, v) and (v, u) by an edge $\{u, v\}$ of weight $w(u, v)+w(v, u)$ and compute a matching of maximum weight on the undirected graph thus obtained. $\mathcal{D}$-DCC, Max-$\mathcal{D}$-DCC, and Max-W-$\mathcal{D}$-DCC can also be solved in polynomial time.

We will show that $L = \{2\}$ and $L = \mathcal{D}$ are the only cases in which directed L-cycle covers can be computed efficiently by proving the NP-hardness of L-DCC and the APX-hardness of Max-L-DCC for all other L. Thus, we completely settle the complexity for directed graphs.

Let us start by proving the APX-hardness.

Theorem 3.3.16. *Let $L \subseteq \mathcal{D}$ be a non-empty set. If $L \neq \{2\}$ and $L \neq \mathcal{D}$, then Max-L-DCC and Max-W-L-DCC are* APX*-hard.*

Proof. We adapt the proof presented in Section 3.3.2. Since $L \neq \{2\}$, there exists a $\lambda \in L$ with $\lambda \geq 3$. Thus, Min-Vertex-Cover(λ) is APX-complete (Theorem 3.4.1). All we need is such a λ and a directed L-clamp. Then we can reduce Min-Vertex-Cover(λ) to Max-L-DCC.

Let $H = (X, F)$ be a λ-regular graph. The edge gadget F_a for an edge $a = \{x, y\} \in F$ is shown in Figure 3.3.6. It consists of two directed L-clamps X^1_a and Y^1_a. The connectors of X^1_a are x^1_a and z^1_a while the connectors of Y^1_a are y^1_a and z^1_a. Again, t^1_a can also serve as a connector of Y^1_a.

The edge gadgets build the graph G_1: Let $x \in X$ be a vertex of H and $a_1, \ldots, a_\lambda \in F$ be the edges incident to x in H (in arbitrary order). Then we assign weight one to the edges $(x^1_{a_\xi}, x^1_{a_{\xi+1}})$ for all $\xi \in [\lambda - 1]$. The edge $(x^1_{a_\lambda}, x^1_{a_1})$ has weight zero. These λ edges are called the junctions of x.

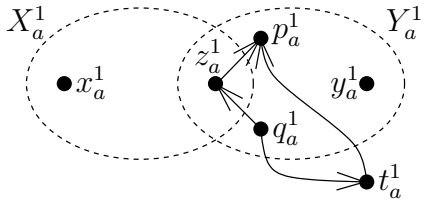


Figure 3.3.6: The directed edge gadget for $a = \{x, y\} \in F$.

Again, $G_2, \ldots, G_\lambda$ are exact copies of G_1 except that $(x_{a_\lambda}^\xi, x_{a_1}^\xi)$ is assigned weight one for all $\xi \in \{2, 3, \ldots, \lambda\}$.

The t-vertices remain to be connected. For all edges $a \in F$, we assign weight one to all λ edges $(t_a^\xi, t_a^{\xi+1})$ for $\xi \in \{1, 2, \ldots, \lambda - 1\}$ and (t_a^λ, t_a^1).

We assign weight zero to all edges that are not mentioned.

The remainder of the proof goes along the same lines as the APX-hardness proof for undirected L-cycle covers. □

We now prove that for all $L \notin \{\{2\}, \mathcal{D}\}$, L-DCC is NP-hard. This does not follow directly from the APX-hardness of Max-L-DCC: A famous counterexample is 2SAT, for which it is APX-hard to maximise the number of simultaneously satisfied clauses [70], although testing whether a 2CNF formula is satisfiable, i.e. whether all clauses are satisfiable simultaneously, takes only polynomial time [69, Section 9.2].

Theorem 3.3.17. *Let $L \subseteq \mathcal{D}$ be a non-empty set. If $L \neq \{2\}$ and $L \neq \mathcal{D}$, then L-DCC is* NP*-hard.*

Proof. As in the proof of the APX-hardness of Max-L-DCC, all we need is an L-clamp and some $\lambda \in L$ with $\lambda \geq 3$. We present a reduction from λ-DM (which is NP-complete since $\lambda \geq 3$) that is similar to the reduction used by Hell et al. [56] to prove the NP-hardness of L-UCC for L with $\overline{L} \not\subseteq \{3, 4\}$.

Let (X, F) be an instance of λ-DM. Note that we will construct a directed graph G as an instance of L-DCC, i.e. G is neither complete nor edge weighted. For each $x \in X$, we have a vertex in G that we again call x. For $a = \{x_1, \ldots, x_\lambda\} \in F$, we construct a cycle of length λ consisting of the vertices $a_1, \ldots, a_\lambda$. Then we add λ L-clamps $K_a^{x_\eta}$ with a_η and x_η as connectors for all $\eta \in [\lambda]$. See Figure 3.3.7 for an example.

Lemma 3.3.18. *$G \in L$-DCC if and only if $(X, F) \in \lambda$-DM.*

Proof. Assume first that $(X, F) \in \lambda$-DM. Thus, there exists a subset $\tilde{F} \subseteq F$ such that $\bigcup_{a \in \tilde{F}} a = X$ and every element $x \in X$ is contained in exactly one set of $\tilde{F}$. We construct an L-cycle cover of G in which all clamps are healthy as follows:

- Let $a = \{x_1, \ldots, x_\lambda\} \in \tilde{F}$. Then let $K_a^{x_\eta}$ expel a_η and absorb x_η for all $\eta \in [\lambda]$, and let $a_1, a_2, \ldots, a_\lambda$ form a cycle of length λ.

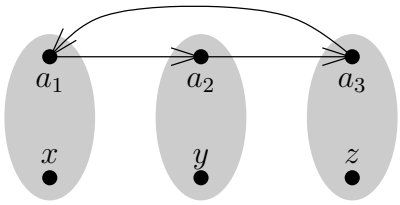


Figure 3.3.7: The construction for the NP-hardness of L-DCC from the viewpoint of $a = \{x, y, z\} \in F$. The L-clamps are coloured grey.

- For all $a = \{x_1, \ldots, x_\lambda\} \notin \tilde{F}$, let $K_a^{x_\eta}$ expel x_η and absorb a_η for all $\eta \in [\lambda]$.

We have to prove that all connectors are absorbed by exactly one clamp or are covered by a cycle of length λ. For every $x \in X$, there is a unique $a \in \tilde{F}$ with $x \in a$. Thus, x is absorbed by K_a^x. Consider now any $a = \{x_1, \ldots, x_\lambda\} \in F$. Either $a \in \tilde{F}$, which implies that $a_1, \ldots, a_\lambda$ form a cycle of length $\lambda \in L$. Or $a \in F \setminus \tilde{F}$, which implies that $K_a^{x_\eta}$ absorbs a_η for all $\eta \in [\lambda]$.

Now we prove the reverse direction. Assume that $G \in L$-DCC, and let C be an L-cycle cover of G. Then every clamp of G is healthy in C, i.e. it absorbs one of its connectors and expels the other one.

Let $a = \{x_1, \ldots, x_\lambda\} \in F$ and assume that $K_a^{x_\eta}$ expels a_η. Since a_η must be part of a cycle in C, $(a_{\eta-1}, a_\eta)$ and $(a_\eta, a_{\eta+1})$ must be in C. Thus, $K_a^{x_{\eta-1}}$ and $K_a^{x_{\eta+1}}$ expel $a_{\eta-1}$ and $a_{\eta+1}$, respectively. By repeatedly applying this argument, we can show that either all $a_1, \ldots, a_\lambda$ are absorbed by $K_a^{x_1}, \ldots, K_a^{x_\lambda}$ or that all are expelled by $K_a^{x_1}, \ldots, K_a^{x_\lambda}$.

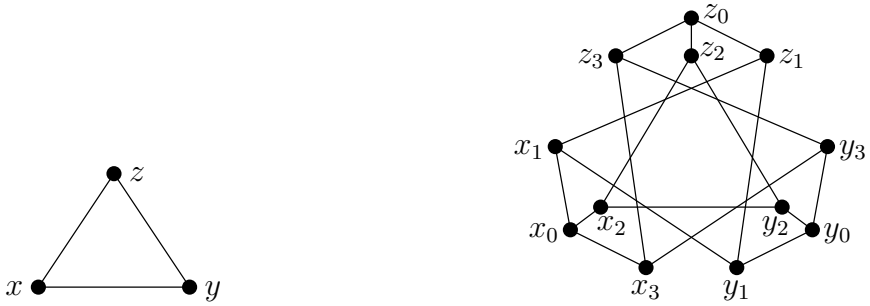
Now consider any $x \in X$ and let $a_1, a_2, \ldots, a_\ell \in F$ be all the sets that contain x. All clamps $K_{a_1}^x, \ldots, K_{a_\ell}^x$ are healthy, C is an L-cycle cover of G, and x is not incident to any further edges. Hence, there must be a unique a_i such that $K_{a_i}^x$ absorbs x. Hence,

$$\tilde{F} = \{a = \{x_1, \ldots, x_\lambda\} \in F \mid K_a^{x_\eta} \text{ absorbs } x_\eta \text{ for all } \eta \in [\lambda]\}$$

is indeed a λ-dimensional matching, proving $(X, F) \in \lambda$-DM. □

Lemma 3.3.18 proves that the construction presented is indeed a many-one reduction from λ-DM to L-DCC, hence L-DCC is NP-hard. □

If the language $\{1^\lambda \mid \lambda \in L\}$ is in NP, then L-DCC is also in NP and therefore NP-complete if $L \notin \{\{2\}, \mathcal{D}\}$: We can nondeterministically guess a cycle cover and then check if $\lambda \in L$ for every cycle length λ occurring in that cover . Conversely, if $\{1^\lambda \mid \lambda \in L\}$ is not in NP, then L-DCC is not in NP either since there is a straightforward reduction of $\{1^\lambda \mid \lambda \in L\}$ to L-DCC: On input $x = 1^\lambda$, construct a graph G on λ vertices that consists solely of a Hamiltonian cycle. Then $x \in L$ if and only if $G \in L$-DCC.



(a) A 2-regular graph H.

(b) The 3-regular graph G obtained from H.

Figure 3.4.1: An example of the construction in Theorem 3.4.1. For readability, H is 2-regular, although Min-Vertex-Cover(2) can be solved in polynomial time.

3.4 Vertex Cover in Regular Graphs

In this section, we prove that Min-Vertex-Cover(λ) is APX-complete for every $\lambda \geq 3$. Previously, this was only known for cubic, i.e. three-regular, graphs [4]. We need the APX-hardness of Min-Vertex-Cover(λ) in Section 3.3, where we uniformly prove the APX-hardness of Max-L-UCC and Max-L-DCC for almost all L.

Theorem 3.4.1. *For every $\lambda \in \mathbb{N}$, $\lambda \geq 3$, Min-Vertex-Cover(λ) is APX-complete.*

Proof. Since Min-Vertex-Cover is in APX (it can be approximated with factor 2 [92, Section 14.3]), Min-Vertex-Cover(λ) is in APX as well. We prove the APX-hardness of Min-Vertex-Cover(λ) for all $\lambda \geq 3$ by induction on λ. The base case, i.e. the APX-hardness of Min-Vertex-Cover(3), has been proved by Alimonti and Kann [4].

Our induction hypothesis is that Min-Vertex-Cover(λ) is APX-hard for some $\lambda \geq 3$. To show the APX-hardness of Min-Vertex-Cover($\lambda + 1$), we L-reduce Min-Vertex-Cover(λ) to Min-Vertex-Cover($\lambda + 1$). Let $H = (X, F)$ be a λ-regular graph as an instance of Min-Vertex-Cover(λ) with $|X| = n$. We create a graph $G = (V, E)$ as an instance of Min-Vertex-Cover($\lambda + 1$) as follows. Let $H_1, \ldots, H_{\lambda+1}$, with $H_i = (X_i, F_i)$ be $\lambda + 1$ copies of H, i.e. $H_i = \{x_i \mid x \in X\}$ and $F_i = \{\{x_i, y_i\} \mid \{x, y\} \in F\}$. Furthermore, let $X_0 = \{x_0 \mid x \in X\}$ and $F_0 = \{\{x_0, x_i\} \mid x \in X, i \in [\lambda + 1]\}$, i.e. the vertex x_0 is connected to all other copies of x. Then $V = \bigcup_{i=0}^{\lambda+1} X_i$ and $E = \bigcup_{i=0}^{\lambda+1} F_i$. Let $k = |V| = (\lambda + 2) \cdot n$. Figure 3.4.1 illustrates the construction.

The graph G thus constructed is $(\lambda + 1)$-regular: Every x_i for $x \in X$ and $i \in [\lambda + 1]$ is adjacent to x_0 and λ vertices of X_i since H is λ-regular, and every $x_0 \in X_0$ is adjacent to $\lambda + 1$ vertices $x_1, \ldots, x_{\lambda+1}$. Given H, the graph G can easily be constructed in polynomial time.

Lemma 3.4.2. *If $H = (X, F)$ has a vertex cover $\tilde{X}$ of size $\tilde{n}$, then $G = (V, E)$ has a vertex cover $\tilde{V}$ of size $n + \lambda\tilde{n}$ with $n = |X|$.*

Proof. Let $\tilde{V} = \{x_i \mid x \in \tilde{X} \wedge i \in [\lambda+1]\} \cup \{x_0 \mid x \notin \tilde{X}\}$. Then $|\tilde{V}| = (\lambda+1)\cdot|\tilde{X}|+n-|\tilde{X}| = n+\lambda\tilde{n}$. It remains to be proved that $\tilde{V}$ is a vertex cover of G. Every edge in every F_i for $i \in [\lambda+1]$ is covered since $\tilde{X}$ is a vertex cover of H and thus $\{x_i \mid x \in \tilde{X}\} \subseteq \tilde{V}$ is a vertex cover of H_i. The only other edges of G are the edges in F_0. For all $x \in X$, either all $x_1, \ldots, x_{\lambda+1}$ are in $\tilde{V}$ or x_0 is in $\tilde{V}$. Hence, all edges in F_0 are covered. □

Lemma 3.4.3. *Let $\tilde{V}$ be an arbitrary vertex cover of G of size $\tilde{k}$. Then we can compute a vertex cover $\tilde{X}$ of H of size $\tilde{n} \leq (\tilde{k}-n)/\lambda$ in polynomial time.*

Proof. Let $\tilde{X}_i = \tilde{V} \cap X_i$ for $i \in [\lambda+1]$. Then $\tilde{X}_i$ is a vertex cover of H_i because $\tilde{V}$ has to cover all edges in F_i and these edges are not adjacent to any vertices outside X_i. Choose $j \in [\lambda+1]$ such that $|\tilde{X}_j|$ is minimal. Let $\tilde{X} = \{x \mid x_j \in \tilde{X}_j\}$ and $\tilde{n} = |\tilde{X}|$. The set $\tilde{X}$ is a vertex cover of H. For all $x \notin \tilde{X}$, we have $x_0 \in \tilde{V}$ since otherwise the edge $\{x_0, x_j\}$ is not covered by $\tilde{V}$. Thus, there are at least $n-\tilde{n}$ vertices of X_0 in $\tilde{V}$.

What remains is to estimate the size $\tilde{n}$ of $\tilde{X}$. We have

$$\tilde{k} = |\tilde{V}| = \sum_{i=0}^{\lambda+1} |X_i| \geq \sum_{i=1}^{\lambda+1} |X_i| + n - \tilde{n} \geq (\lambda+1)\cdot\tilde{n} + n - \tilde{n} = \lambda\tilde{n} + n\,,$$

hence $\tilde{n} \leq (\tilde{k}-n)/\lambda$. Finally, given $\tilde{V}$, the set $\tilde{X}$ can easily be constructed in polynomial time. □

It remains to be proved that the construction described yields an L-reduction. Since H is λ-regular, we have $\mathrm{opt}(H) \geq n/\lambda$. Thus,

$$\mathrm{opt}(G) \leq |V| = (\lambda+2)\cdot n \leq (\lambda+2)\cdot\lambda\cdot\mathrm{opt}(H)\,.$$

On the other hand, let $\tilde{V}$ be an arbitrary vertex cover of G and $\tilde{X}$ be the vertex cover of H constructed as described in the proof of Lemma 3.4.3. According to the two lemmas above, we have $\mathrm{opt}(G) = \lambda\cdot\mathrm{opt}(H)+n$. This together with the inequality of Lemma 3.4.3 yields

$$\Big||\tilde{X}| - \mathrm{opt}(H)\Big| \leq \frac{1}{\lambda}\cdot\Big||\tilde{V}| - \mathrm{opt}(G)\Big|\,.$$

Overall, Min-Vertex-Cover(λ) $\leq_{\mathrm{L}}$ Min-Vertex-Cover($\lambda+1$) for all $\lambda \geq 3$, which proves the theorem. □

CHAPTER

FOUR

Algorithms for Cycle Covers

4.1 Approximation Algorithms

The goal of this section is to devise approximation algorithms for Max-W-L-UCC and Max-W-L-DCC that work for arbitrary L. The catch is that for most L it is impossible to decide whether some cycle length is in L or not.

One possibility would be to restrict ourselves to sets L such that $\{1^\lambda \mid \lambda \in L\}$ is in P. For such L, Max-W-L-UCC and Max-W-L-DCC are NP optimisation problems. Another possibility for circumventing the problem is to include the permitted cycle lengths in the input. However, it turns out that such restrictions are not necessary.

A necessary and sufficient condition for a complete graph with n vertices to have an L-cycle cover is that there exist (not necessarily distinct) lengths $\lambda_1, \dots, \lambda_k \in L$ for some $k \in \mathbb{N}$ with $\sum_{i=1}^{k} \lambda_i = n$. We call such an n ***L*-admissible** and define $\langle L \rangle = \{n \mid n \text{ is } L\text{-admissible}\}$. Although L can be arbitrarily complicated, $\langle L \rangle$ always allows efficient membership testing.

Lemma 4.1.1. *For all $L \subseteq \mathbb{N}$, there exists a finite set $L' \subseteq L$ with $\langle L' \rangle = \langle L \rangle$.*

Proof. Let $L_{\leq \ell} = \{n \in L \mid n \leq \ell\} \subseteq L$. We denote by $g(\ell)$ the greatest common divisor of all numbers in $L_{\leq \ell}$. Then $g(\ell) \geq g(\ell + 1) \geq 1$ for all $\ell \in \mathbb{N}$. Hence, g converges to some $g_L \in \mathbb{N}$ and there exists an ℓ_0 with $g(\ell) = g_L$ for all $\ell \geq \ell_0$. Without loss of generality, we assume that $g_L = 1$. Otherwise, we "scale" L down to $\tilde{L} = \{\lambda \mid \lambda g_L \in L\}$.

If $1 \in L$, then $\langle \{1\} \rangle = \langle L \rangle$ and we are done. We therefore assume that $1 \notin L$. There exist $\xi_1, \dots, \xi_k \in \mathbb{Z}$ and $\lambda_1, \dots, \lambda_k \in L_{\leq \ell_0}$ for some $k \in \mathbb{N}$ with $\sum_{i=1}^{k} \xi_i \lambda_i = 1$. Let $\xi = \min_{i \in [k]} \xi_i$. We have $\xi < 0$ since $1 \notin L$. Choose any $\lambda \in L_{\leq \ell_0}$ and let $\ell = \lambda \cdot (-\xi) \cdot \sum_{i=1}^{k} \lambda_i$. Let $n \in \langle L \rangle$ with $n \geq \ell$, and let

$m = \mathrm{mod}(n-\ell,\lambda) < \lambda$. Then we can write n as

$$\begin{aligned} n &= \lambda \cdot \left\lfloor \frac{n-\ell}{\lambda} \right\rfloor + m + \ell = \lambda \cdot \left\lfloor \frac{n-\ell}{\lambda} \right\rfloor + m \cdot \underbrace{\sum_{i=1}^{k} \xi_i \lambda_i}_{=1} - \lambda\xi \cdot \sum_{i=1}^{k} \lambda_i \\ &= \lambda \cdot \left\lfloor \frac{n-\ell}{\lambda} \right\rfloor + \sum_{i=1}^{k} (m\xi_i - \lambda\xi) \cdot \lambda_i \,. \end{aligned}$$

We have $(m\xi_i - \lambda\xi) \geq 0$ for all i since $m < \lambda$ and $\xi_i \geq \xi < 0$. Hence, $\langle L_{\leq \ell_0} \rangle$ contains all elements $n \in \langle L \rangle$ with $n \geq \ell$. Elements of $\langle L \rangle$ smaller than ℓ are contained in $\langle L_{\leq \ell} \rangle \supseteq \langle L_{\leq \ell_0} \rangle$. Hence, $\langle L_{\leq \ell} \rangle = \langle L \rangle$ and $L' = L_{\leq \ell}$ is the finite set we are looking for. □

For every fixed L, we can not only test in time polynomial in n whether n is L-admissible, but we can, provided that $n \in \langle L \rangle$, also find numbers $\lambda_1, \ldots, \lambda_k \in L'$ that add up to n, where $L' \subseteq L$ denotes a finite set with $\langle L \rangle = \langle L' \rangle$. This can be done via dynamic programming in time $O(n \cdot |L'|)$, which is $O(n)$ for fixed L.

Although $\langle L \rangle = \langle L' \rangle$, there are clearly graphs for which the weights of an optimal L-cycle cover and an optimal L'-cycle cover differ: Let $\lambda \in L \setminus L'$ and consider a graph on λ vertices. We assign weight one to λ edges that form a Hamiltonian cycle, all other edges are assigned weight zero.

Moreover, in contrast to NP optimisation problems, for most L it is impossible to compute a maximum weight L-cycle cover, even if we allow, say, exponential time. The problem is that it is in general impossible to decide whether a cycle cover is an L-cycle cover or not. Thus, Item 2 of the definition of NP optimisation problems (Definition 2.4.2) is violated in general.

This does not matter for our approximation algorithms since they construct L'-cycle covers whose weight is at least a certain fraction of the weight of an optimal cycle cover without any restrictions. The weight of an optimal cycle cover without any restrictions is an upper bound for the weight of an optimal L-cycle cover.

Instead of computing L'-cycle covers in the following two sections, we assume without loss of generality that L is already a finite set.

The main idea of the two approximation algorithms is as follows: We start by computing a cycle cover C^{init} of maximum weight. Then we take a subset S of the edges of C^{init} that weighs as much as possible under the restriction that there exists an L-cycle cover that includes all edges of S. We add edges to obtain an L-cycle cover $C^{\text{apx}} \supseteq S$. Let $C^\star$ be an L-cycle cover of maximum weight, and assume that we can guarantee $\rho \cdot w(S) \geq w(C^{\text{init}})$ for some $\rho \geq 1$. Then $w(C^\star) \leq w(C^{\text{init}}) \leq \rho \cdot w(S) \leq \rho \cdot w(C^{\text{apx}})$. Thus, we have computed a factor ρ approximation to an L-cycle cover of maximum weight.

4.1.1 Approximating Restricted Undirected Cycle Covers

The input of our algorithm for undirected graphs is an undirected complete graph $G = (V, U(V))$ with $|V| = n$ and an edge weight function $w : U(V) \to \mathbb{N}$.

The main idea of the approximation algorithm is as follows: Every cycle of length λ can be divided into $\lfloor \lambda/3 \rfloor$ vertex-disjoint paths of length two, which are just two adjacent edges. This can be done as follows: Let $(e_1, \ldots, e_\lambda)$ be the cycle, then skip $e_3, e_6, \ldots, e_{3\cdot\lfloor\lambda/3\rfloor}$. If λ is not divisible by three, then additionally skip $e_{3\cdot\lfloor\lambda/3\rfloor+1}, \ldots, e_\lambda$. In this way, we obtain the paths $(e_1, e_2), (e_4, e_5), \ldots$ and one or two isolated vertices if λ is not divisible by three.

Conversely, every collection of $\xi \leq \lfloor \lambda/3 \rfloor$ vertex-disjoint paths of length two plus $\lambda - 3\xi$ isolated vertices can be joined to form a cycle of length λ.

Consider a cycle cover consisting of cycles of lengths $\lambda_1, \ldots, \lambda_k$. Let n be the number of vertices. Such a cycle cover can be divided into $\sum_{i=1}^{k} \lfloor \lambda_i/3 \rfloor$ paths of length two. Since $\lfloor \lambda/3 \rfloor \geq \lceil \lambda/5 \rceil$ for $\lambda \geq 3$, we obtain $\sum_{i=1}^{k} \lfloor \lambda_i/3 \rfloor \geq \sum_{i=1}^{k} \lceil \lambda_i/5 \rceil \geq \lceil n/5 \rceil$.

The next lemma states that for every cycle cover C, and thus in particular for C^{init}, there exists a set $P \subseteq C$ of edges consisting of $\lceil n/5 \rceil$ paths of length two such that $2.5 \cdot w(P) \geq w(C)$. Joining the paths in P to obtain an L-cycle cover, which can be done according to Lemma 4.1.3, yields a factor 2.5 approximation.

Lemma 4.1.2. *Let C be any cycle cover of G. Then there exists a subset $P \subseteq C$ such that*

- *the graph (V, P) consists solely of vertex-disjoint paths of length two and isolated vertices,*
- *$|P| = 2 \cdot \lceil n/5 \rceil$, i.e. P contains $\lceil n/5 \rceil$ paths, and*
- *$w(P) \geq 0.4 \cdot w(C)$.*

Proof. Let $m = \lceil n/5 \rceil$ for short. We prove the lemma by a probabilistic argument: We devise a probability distribution on $\mathcal{P}(C)$ to randomly construct $P \subseteq C$ with the following properties:

1. For every edge e, $\mathbb{P}(e \in P) \geq 0.4$.
2. P consists of m paths of length two.

By linearity of expectation, we obtain

$$\mathbb{E}(w(P)) = \sum_{e \in C} \mathbb{P}(e \in P) \cdot w(e) \geq 0.4 \cdot \sum_{e \in C} \cdot w(e) = 0.4 \cdot w(C) \,.$$

Thus, there exists a P as demanded.

Consider a cycle $c = (e_0, \ldots, e_{\lambda-1})$ and let $\xi \leq \lceil \lambda/5 \rceil$. We randomly obtain ξ paths of length two of c as follows: Draw $i \in \{0, 1, \ldots, \lambda-1\}$ uniformly at random,

and then take the ξ paths $(e_i, e_{i+1}), (e_{i+3}, e_{i+4}), \ldots, (e_{i+3(\xi-1)}, e_{i+3(\xi-1)+1})$, where addition is modulo λ. In this way, each edge of c is part of a path with probability $2\xi/\lambda$.

Let $c_1, \ldots, c_k$ be the cycles of C and let λ_i be the length of c_i for $i \in [k]$. We can achieve a selection P of m paths by putting either $\lfloor \lambda_i/5 \rfloor$ or $\lceil \lambda_i/5 \rceil$ paths of the cycle c_i into P, because $\sum_{i=1}^{k} \lceil \lambda_i/5 \rceil \geq m \geq \sum_{i=1}^{k} \lfloor \lambda_i/5 \rfloor$.

For all cycles c_i whose length λ_i is divisible by five, we are done since $\lceil \lambda_i/5 \rceil = \lfloor \lambda_i/5 \rfloor = \lambda_i/5$: We choose $\lambda_i/5$ paths, thus every edge of c_i is in P with probability 0.4.

Let $c_1, \ldots, c_r$ be the cycles whose lengths are not divisible by five. If we take $\lceil \lambda_i/5 \rceil = \lfloor \lambda_i/5 \rfloor + 1$ cycles of c_i, we call c_i **abundant**. Otherwise, we call c_i **deficient**. We have to fix a probability p_i for c_i to be abundant.

Let $E = m - \sum_{i=1}^{k} \lfloor \lambda_i/5 \rfloor$. Then E cycles of $c_1, \ldots, c_r$ have to be abundant. If we choose the p_i such that $\sum_{i=1}^{r} p_i = E$, then we have m paths in expectation.

Let $q_i = \lambda_i/5 - \lfloor \lambda_i/5 \rfloor \in [0, 1]$. Then

$$q_i \cdot \lceil \lambda_i/5 \rceil + (1 - q_i) \cdot \lfloor \lambda_i/5 \rfloor = \lambda_i/5 \,.$$

That is, if c_i were abundant with probability q_i, each edge of c_i would be chosen with probability 0.4. Now we choose rational numbers $p_i \in [q_i, 1]$ (guaranteeing $\mathbb{P}(e \in P) \geq 0.4$) such that $\sum_{i=1}^{r} p_i = E$.

Let X be a random subset of $\mathcal{P}(\{c_1, \ldots, c_r\})$ that contains the abundant cycles. We need a probability distribution for X such that X is always of cardinality E and $\mathbb{P}(c_i \in X) = p_i$ for all $i \in [r]$. Such a probability distribution exists according to Lemma A.3.1.

Overall, we first choose randomly the set of abundant cycles. Then we know how many paths we need to take from every cycle. We choose these paths randomly as described above. The probability distribution on $\mathcal{P}(C)$ obtained in this way has the desired properties:

1. For every $i \in [r]$ and every edge e of c_i, we have

$$\begin{aligned} \mathbb{P}(e \in P) &= (1 - p_i) \cdot \frac{2\lfloor \lambda_i/5 \rfloor}{\lambda_i} + p_i \cdot \frac{2\lceil \lambda_i/5 \rceil}{\lambda_i} \\ &\geq (1 - q_i) \cdot \frac{2\lfloor \lambda_i/5 \rfloor}{\lambda_i} + q_i \cdot \frac{2\lceil \lambda_i/5 \rceil}{\lambda_i} \geq 0.4 \,. \end{aligned}$$

   For every $i \in [k] \setminus [r]$, every edge of c_i is in P with probability 0.4.

2. The set P contains m paths of length two.

This completes the proof of the lemma. □

Given a cycle cover C, a subset $P \subseteq C$ with maximum weight among all subsets of C that consists of $\lceil n/5 \rceil$ paths of length two can be computed in time $O(n^2)$ via dynamic programming.

Input: an undirected graph $G = (V, U(V))$ with $|V| = n$ and an edge weight function $w : U(V) \to \mathbb{N}$
Output: an L-cycle cover C^{apx} of G if n is L-admissible, $\perp$ otherwise
1: **if** $n \in \langle L \rangle$ **then**
2: compute a cycle cover C^{init} of maximum weight
3: compute a subset $P \subseteq C^{\text{init}}$ such that (V, P) consists of $\lceil n/5 \rceil$ paths of length two and $n - 3 \cdot \lceil n/5 \rceil$ isolated vertices and $w(P)$ is maximum among all such sets
4: join the paths in P to obtain an L-cycle cover C^{apx}
5: **return** C^{apx}
6: **else**
7: **return** $\perp$

Algorithm 4.1.1: A factor 2.5 approximation algorithm for Max-W-L-UCC.

Now assume that n is L-admissible (which is necessary for graphs with n vertices to contain an L-cycle cover). Then for every collection P of $\lceil n/5 \rceil$ paths of length two, an L-cycle cover C with $C \supseteq P$ exists, as we will show now.

Lemma 4.1.3. *Let $P \subseteq E$ be a set of edges such that (V, P) consists of $\lceil n/5 \rceil$ vertex-disjoint paths of length two and $n - 3 \cdot \lceil n/5 \rceil$ isolated vertices. Let $L \subseteq \mathcal{U}$ such that $n \in \langle L \rangle$. Then there exists an L-cycle cover C with $P \subseteq C$.*

Proof. Since $n \in \langle L \rangle$, there exist $\lambda_1, \ldots, \lambda_k \in L$ with $\sum_{i=1}^{k} \lambda_i = n$. Then $\sum_{i=1}^{k} \lceil \lambda_i/5 \rceil \geq \lceil n/5 \rceil$.

We build C as follows: For $i = 1, \ldots, k$, build a cycle of length λ_i that consists of $\lceil \lambda_i/5 \rceil$ paths from P and $\lambda_i - 3 \cdot \lceil \lambda_i/5 \rceil$ isolated vertices. If $\sum_{i=1}^{k} \lceil \lambda_i/5 \rceil > \lceil n/5 \rceil$, P does not contain enough paths for all cycles. In this case, we build the remaining cycles solely from isolated vertices. (There is possibly one cycle of length λ_i with at least one but less than $\lceil \lambda_i/5 \rceil$ paths from P.) Overall, the set C of edges thus constructed is an L-cycle cover with $C \supseteq P$. □

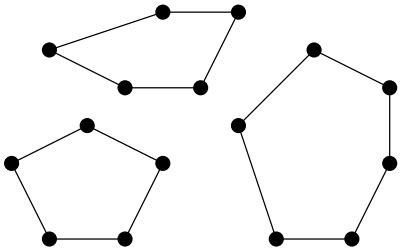
Algorithm 4.1.1 begins by constructing an initial cycle cover, then computes a set P as described above and finally joins the paths and isolated vertices to obtain a cycle cover. Figure 4.1.1 shows an example of how the algorithm works.

Theorem 4.1.4. *For every fixed L, Algorithm 4.1.1 is a factor 2.5 approximation algorithm for Max-W-L-UCC with running time $O(n^3)$.*

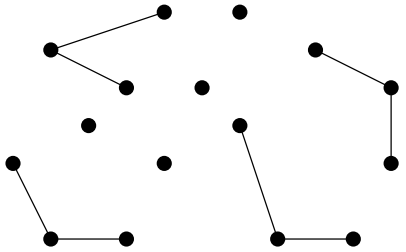
Proof. If L is infinite, we replace L by a finite set $L' \subseteq L$ with $\langle L' \rangle = \langle L \rangle$ according to Lemma 4.1.1. Algorithm 4.1.1 returns $\perp$ if and only if $n \notin \langle L \rangle$. Otherwise, an L-cycle cover C^{apx} is returned.

Let $C^{\star}$ denote an L-cycle cover of maximum weight of G. We have

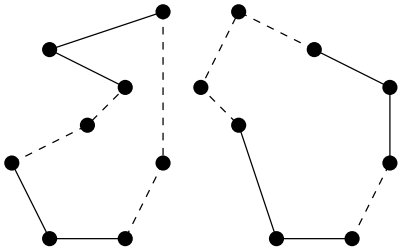
$$w(C^{\star}) \leq w(C^{\text{init}}) \leq 2.5 \cdot w(P) \leq 2.5 \cdot w(C^{\text{apx}}) .$$



(a) The initial cycle cover C^{init}.



(b) Obtaining a set P of $\lceil 16/5 \rceil = 4$ paths of length two from C^{init}.



(c) Joining the paths of P to obtain C^{apx}.

Figure 4.1.1: Computing an $\{8\}$-cycle cover. For readability, edge weights are omitted, and only the edges of the cycle covers are shown.

Hence, the algorithm computes a factor 2.5 approximation.

The running-time is dominated by the time needed to compute C^{init}, which takes time $O(n^3)$ [3, Chap. 12]. All other operations take time $O(n^2)$. □

4.1.2 Approximating Restricted Directed Cycle Covers

We now consider directed graphs. We present an approximation algorithm for Max-W-L-DCC that achieves an approximation ratio of 3. The input of our algorithm consists of a directed complete graph $G = (V, D(V))$ with $|V| = n$ and an edge weight function $w : D(V) \to \mathbb{N}$.

Consider a cycle of length λ. By omitting every other edge, we obtain a set of $\lfloor \lambda/2 \rfloor$ edges that do not share any vertex, i.e. they form a matching. Conversely, $\xi \leq \lfloor \lambda/2 \rfloor$ pairwise non-adjacent edges together with $\lambda - 2\xi$ isolated vertices can be joined to form a cycle of length λ.

Consider a cycle cover consisting of cycles of lengths $\lambda_1, \ldots, \lambda_k$. From such a cycle cover, we can obtain a matching of cardinality $\sum_{i=1}^{k} \lfloor \lambda_i/2 \rfloor$. We have $\lfloor \lambda_i/2 \rfloor \geq \lceil \lambda_i/3 \rceil$ for all $i \in [k]$ since $\lambda_i \geq 2$. Hence, $\sum_{i=1}^{k} \lfloor \lambda_i/2 \rfloor \geq \sum_{i=1}^{k} \lceil \lambda_i/3 \rceil \geq \lceil n/3 \rceil$.

Every cycle cover C, and thus in particular a cycle cover C^{init} of maximum weight, induces such a matching that weighs at least one third of $w(C)$.

Lemma 4.1.5. *Let C be an arbitrary cycle cover of G. Then there exists a matching $M \subseteq C$ of cardinality $\lceil n/3 \rceil$ with $w(M) \geq w(C)/3$.*

Proof. The proof is similar to the proof of Lemma 4.1.2.

Let $m = \lceil n/3 \rceil$ for short. We devise a probability distribution on $\mathcal{P}(C)$ to randomly construct a matching $M \subseteq C$ such that every edge of C is included in M with probability at least $1/3$ and M consists of m edges. Then the lemma follows by linearity of expectation.

Let $c = (e_0, \ldots, e_{\lambda-1})$ be a cycle of length λ and $\xi \leq \lceil \lambda/3 \rceil$. We randomly obtain ξ pairwise non-adjacent edges of c as follows: First, we draw $i \in \{0, 1, \ldots, \lambda - 1\}$ uniformly at random, and then we take the ξ edges $e_i, e_{i+2}, \ldots, e_{i+2(\xi-1)}$, where addition is modulo λ. In this way, each edge of c is taken with probability ξ/λ.

For all cycles c_i whose length λ_i is divisible by three, we are done since $\lceil \lambda_i/3 \rceil = \lfloor \lambda_i/3 \rfloor = \lambda_i/3$. Thus, every edge of c is chosen with probability $1/3$.

Let $c_1, \ldots, c_r$ be the cycles whose lengths are not divisible by three. If we take $\lceil \lambda_i/3 \rceil = \lfloor \lambda_i/3 \rfloor + 1$ edges of c_i, we call c_i **abundant**, otherwise c_i is **deficient**. What remains is to fix a probability p_i for c_i to be abundant.

To obtain a matching of cardinality m, $E = m - \sum_{i=1}^{k} \lfloor \lambda_i/3 \rfloor$ cycles have to be abundant. If we choose p_i such that $\sum_{i=1}^{r} p_i = E$, then we have m edges in expectation.

Let $q_i = \lambda_i/3 - \lfloor \lambda_i/3 \rfloor$ for $i \in [r]$. Then $q_i \cdot \lceil \lambda_i/5 \rceil + (1 - q_i) \cdot \lfloor \lambda_i/5 \rfloor = \lambda_i/5$. Now we choose rational numbers $p_i \in [q_i, 1]$ such that $\sum_{i=1}^{r} p_i = E$.

Let X be a random subset of $\mathcal{P}(\{c_1, \ldots, c_r\})$ that contains the abundant cycles. We need a probability distribution for X such that X is always of cardinality E and $\mathbb{P}(c_i \in X) = p_i$. Such a distribution exists according to Lemma A.3.1.

To summarise: We first choose randomly the set of abundant cycles. Then we know how many edges of each cycle we need, and we choose these edges as described above. The probability distribution on $\mathcal{P}(C)$ obtained in this way has the desired properties:

1. For every $i \in [r]$ and every edge e of cycle c_i,

$$\mathbb{P}(e \in M) = (1 - p_i) \cdot \lfloor \lambda_i/3 \rfloor + p_i \cdot \lceil \lambda_i/3 \rceil \geq 1/3 \,.$$

   Edges of the other cycles are contained in M with probability $1/3$.

2. $|M| = m$.

Thus, the lemma is proved. □

Now we prove the counterpart of Lemma 4.1.3 above: Given any matching M of cardinality at most $\lceil n/3 \rceil$, an L-cycle cover $C \supseteq M$ exists. Moreover, such a matching can be computed efficiently.

Lemma 4.1.6. *Let M be a matching of G of cardinality $\lceil n/3 \rceil$. Then there exists an L-cycle cover $C \supseteq M$.*

Input: a directed graph $G = (V, D(V))$ with $|V| = n$ and an edge weight function $w : D(V) \to \mathbb{N}$
Output: an L-cycle cover C^{apx} of G if n is L-admissible, $\perp$ otherwise
1: **if** $n \in \langle L \rangle$ **then**
2: compute a maximum weight matching M^{init} of G of cardinality $\lceil n/3 \rceil$
3: join the edges in M^{init} to obtain an L-cycle cover C^{apx} as described in Lemma 4.1.6
4: **return** C^{apx}
5: **else**
6: **return** $\perp$

Algorithm 4.1.2: A factor 3 approximation algorithm for Max-W-L-DCC.

(a) Matching M^{init} of cardinality $\lceil 10/3 \rceil$. (b) Joining the edges of M^{init} to get C^{apx}.

Figure 4.1.2: Computing a $\{5\}$-cycle cover. For readability, edge weights are omitted, and only the edges of the cycle covers are shown.

Proof. Since $n \in \langle L \rangle$, there exist $\lambda_1, \ldots, \lambda_k \in L$ with $\sum_{i=1}^{k} \lambda_i = n$. Then $\sum_{i=1}^{k} \lceil \lambda_i/3 \rceil \geq \lceil n/3 \rceil$.

We build C as follows: For $i = 1, \ldots, k$, build a cycle of length λ_i that consists of $\lceil \lambda/3 \rceil$ edges of M and possibly one isolated vertex. If $\sum_{i=1}^{k} \lceil \lambda_i/3 \rceil > \lceil n/3 \rceil$, then M does not contain enough edges for all cycles. In this case, we build the remaining cycles solely from isolated vertices. (There is possibly one cycle of length λ_i built with at least one but less than $\lceil \lambda_i/3 \rceil$ edges from M.) Overall, the set C of edges thus constructed is an L-cycle cover and fulfils $C \supseteq M$. □

Given M, the construction of C thus described can clearly be carried out in linear time. Overall, Algorithm 4.1.2 is a factor 3 approximation algorithm for Max-W-L-DCC. Instead of computing an initial cycle cover, we directly compute a matching M^{init} of cardinality $\lceil n/3 \rceil$. Figure 4.1.2 shows an example of how the algorithm works.

Theorem 4.1.7. *For every fixed L, Algorithm 4.1.2 is a factor 3 approximation algorithm for Max-W-L-UCC with running time $O(n^3)$.*

Proof. If L is infinite, we replace L by a finite set $L' \subseteq L$ with $\langle L' \rangle = \langle L \rangle$ according to Lemma 4.1.1. Algorithm 4.1.2 returns $\perp$ if and only if $n \notin \langle L \rangle$. Otherwise, an L-cycle cover C^{apx} is returned.

Let $C^\star$ be an L-cycle cover of maximum weight of G and C' be a cycle cover of maximum weight of G. Let $M' \subseteq C'$ denote a matching of cardinality $\lceil n/3 \rceil$

that has maximum weight among all such matchings. We have

$$w(C^\star) \leq w(C') \leq 3 \cdot w(M') \leq 3 \cdot w(M^{\text{init}}) \leq 3 \cdot w(C^{\text{apx}}) \,.$$

Hence, Algorithm 4.1.2 computes a factor 3 approximation.

The running time is dominated by the time needed to compute the initial matching M^{init}, which takes time $O(n^3)$ [3, Chap. 12]. All other operations take time $O(n^2)$. □

4.2 Solving Max-4-UCC in Polynomial Time

The aim of this section is to show that Max-4-UCC can be solved deterministically in polynomial time. To do this, we exploit Hartvigsen's algorithm for computing a maximum-cardinality triangle-free two-matching. We will show how to obtain a 4-cycle cover of maximum weight from a maximum-cardinality triangle-free two-matching.

A **two-matching** of an undirected graph G is a spanning subgraph in which every vertex of G has degree *at most* two. Thus, two-matchings consist of disjoint simple cycles and paths. A two-matching is a relaxation of a cycle cover (or two-factor): In a cycle cover, every vertex has degree *exactly* two.

A **triangle-free two-matching** is a two-matching in which each cycle has a length of at least four. The paths can have arbitrary lengths. A triangle-free two-matching is of maximum cardinality if no other triangle-free two-matching contains more edges. The problem of finding a **maximum-cardinality triangle-free two-matching** can be solved in time $O(n^3)$, where n is the number of vertices, as was proved by Hartvigsen [51, Chap. 3].

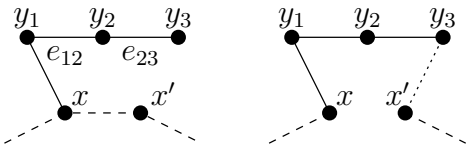
As for cycle covers, we can also consider complete graphs with edge weights zero and one: We replace edges by weight one edges and non-edges by weight zero edges. Then a maximum-cardinality triangle-free two-matching corresponds to a triangle-free two-matching of maximum weight.

We want to solve Max-4-UCC, i.e. we want to find a 4-cycle cover of maximum weight: All cycles must have a length of at least four and no paths are allowed.

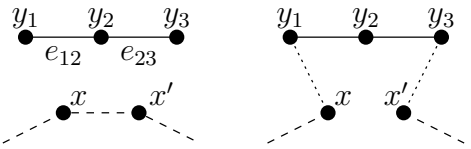
Let M be a maximum weight triangle-free two-matching of a graph G of n vertices. We can assume that M does not contain any edges of weight zero since we can omit such edges without losing any weight and still have a triangle-free two-matching. If M does not contain any paths, then M is already a 4-cycle cover. Since two-matchings are relaxations of cycle covers, M is a 4-cycle cover of maximum weight.

Let ℓ be the number of vertices of G that lie on paths in M. If $\ell \geq 4$, then we connect these paths to get a cycle of length ℓ. No weight is lost in this way, and the result is a maximum weight 4-cycle cover.

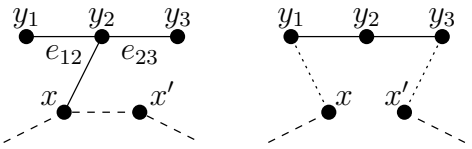
We run into trouble if $\ell \in \{1, 2, 3\}$. Let $Y = \{y_1, \ldots, y_\ell\}$ be the set of vertices that lie on paths in M. These vertices are also referred to as the ***y*-vertices**. Let



(a) Case 1: No weight is lost by using the edge $\{y_1, x\}$.



(b) Case 2: Weight one is lost, the y-vertices are connected to an arbitrary cycle.



(c) Case 3: Weight one is lost although $\{y_2, x\}$ weighs one.

Figure 4.2.1: The three cases for $\ell = 3$ and $\ell' = 2$. In each figure, the left-hand side shows the two-matching, the right-hand side shows how the y-vertices are connected to obtain a 4-cycle cover. Only edges of weight one are drawn except for dotted edges which are part of the 4-cycle cover and may weigh zero. The dashed edges belong to the cycle that contains x and x'.

ℓ' be the number of edges of weight one in M that connect two y-vertices. Then $0 \leq \ell' \leq \ell - 1$ and $w(M) = n - \ell + \ell' \leq n - 1$.

An obvious way to obtain a cycle cover from M is to break one edge of one cycle and connect the y-vertices to this cycle. Unfortunately, breaking an edge might cause a loss of weight one. This yields the aforementioned approximation within an additive error of one (see Section 2.5.1). We need a more careful analysis to prove the following: Either we can avoid the loss of weight one, or indeed a maximum weight 4-cycle cover has only weight $w(M) - 1$.

By assumption, there are no edges of weight zero in M. If $\ell' = 1$, then we assume that $e_{12} = \{y_1, y_2\}$ is in M. If $\ell' = 2$, then we assume that e_{12} and $e_{23} = \{y_2, y_3\}$ are in M. We distinguish three cases (see Figure 4.2.1):

Case 1: There is a $y_i \in Y$ that is an endpoint of a path in M and an $x \notin Y$ such that $w(\{y_i, x\}) = 1$. We remove one edge, say $\{x, x'\} \in M$, incident to x, add $\{x, y_i\}$, and connect the remaining y-vertices to obtain a new cycle. This new cycle is of length $4 + \ell$. No weight is lost and we have thus obtained a 4-cycle cover of maximum weight.

Case 2: All edges connecting y-vertices to vertices not in Y have weight zero. Then we break one arbitrary edge $\{x, x'\}$ of one cycle and connect the y-vertices to obtain a new cycle. The 4-cycle cover obtained has weight $n - \ell + \ell' - 1$. In every 4-cycle cover, the y-vertices are incident to at least $\ell + 1$ different edges. Otherwise, they would form a cycle of length $\ell < 4$. At least $\ell - \ell' + 1$ of these edges have weight zero. Hence, our 4-cycle cover is of maximum weight.

Input: an undirected graph $G = (V, U(V))$ with edge weights $w : U(V) \to \{0, 1\}$
Output: a 4-cycle cover C of maximum weight
1: compute a maximum weight triangle-free two-matching M of G
2: remove all edges of weight zero from M
3: let $Y = \{y_1, \ldots, y_\ell\}$ be the set of vertices that lie on paths in M
4: **if** $\ell = 0$ **then**
5: $\quad C = M$
6: **else if** $\ell \geq 4$ **then**
7: $\quad$ construct $C \supseteq M$ from M by connecting the vertices in Y to a cycle
8: **else**
9: $\quad$ construct C from M as described in the case distinction
10: **return** C

Algorithm 4.2.1: A polynomial-time algorithm for Max-4-UCC.

Case 3: The case that remains is that $\ell = 3$, $e_{12}, e_{23} \in M$, and there is an edge $\{y_2, x\}$ of weight one with $x \notin Y$. In this case, $w(M) = n - 1$.

Neither y_1 nor y_3 is incident to a weight one edge except for e_{12} and e_{23}. Otherwise Case 1 can be applied. Furthermore, $w(\{y_1, y_3\}) = 0$. Otherwise, we can replace e_{23} by $\{y_1, y_3\}$ without losing any weight and apply Case 1 again.

We break one edge $\{x, x'\} \in M$ and add $\{y_1, x\}$ and $\{y_3, x'\}$. Weight one is lost, and we obtain a 4-cycle cover of weight $n - 2$. We prove that this is optimal by considering a maximum weight 4-cycle cover C and distinguishing three cases:

3a: Both e_{12} and e_{23} are in C. Since C is triangle-free, $\{y_1, y_3\} \notin C$. Hence, both y_1 and y_3 are incident to an edge of weight zero, which implies $w(C) \leq n - 2$.

3b: Either e_{12} or e_{23} is in C, the other is not. Assume $e_{12} \in C$. Then y_3 is incident to two edges of weight zero and we have $w(C) \leq n - 2$.

3c: Both e_{12} and e_{23} are not in C. Then there are at least three different weight zero edges incident to y_1 or y_3. Thus, $w(C) \leq n - 3$.

Overall, we have proved that Algorithm 4.2.1 solves Max-4-UCC exactly. The running time of the algorithm is $O(n^3)$ since Hartvigsen's algorithm takes time $O(n^3)$ and the modifications to M can easily be done in time $O(n^2)$. Thus, we have obtained the following result.

Theorem 4.2.1. *Max-4-UCC* $\in$ PO. □

CHAPTER

FIVE

Open Problems on Cycle Covers

We have considered the complexity and approximability of computing restricted cycle covers in both directed and undirected graphs. For almost all L, the decision problem is NP-hard and the optimisation problem is APX-hard. Although computing restricted cycle covers is generally very hard, we have proved that L-cycle covers of maximum weight can be approximated within a constant factor in polynomial time.

For directed graphs, we have settled the complexity of computing L-cycle covers and obtained a dichotomy: If $L = \{2\}$ or $L = \mathcal{D}$, then L-DCC, Max-L-DCC, and Max-W-L-DCC are solvable in polynomial time, otherwise they are intractable and hard to approximate.

For undirected graphs, the status of only five cycle cover problems remains open: 5-UCC, $\overline{\{4\}}$-UCC, Max-5-UCC, Max-$\overline{\{4\}}$-UCC, and Max-W-4-UCC.

There are some reasons for optimism that 5-UCC, $\overline{\{4\}}$-UCC, Max-5-UCC, and Max-$\overline{\{4\}}$-UCC are solvable in polynomial time: Hartvigsen [52] presented a polynomial time algorithm for finding cycle covers without cycles of length four in bipartite graphs. (For bipartite graphs, additionally forbidding cycles of length three does not change the problem.) Moreover, there are augmenting path theorems for L-cycle covers for all L with $\overline{L} \subseteq \{3, 4\}$, which includes the two cases that are known to be polynomial time solvable. Augmenting path theorems are often a building block for matching algorithms. Of course, this does not prove that 5-UCC and $\overline{\{4\}}$-UCC are indeed in P. In fact, there are also augmenting path theorems for L-cycle covers for $L \subseteq \{3, 4\}$ [80], even though L-UCC is NP-complete and Max-L-UCC and Max-W-L-UCC are APX-complete in these cases.

Results by Cunningham and Wang [34] suggest that a complete polyhedral characterisation of 4-cycle covers might be difficult (cf. Cunningham [33]). Such a characterisation would possibly lead to algorithms based on linear programming.

However, Hartvigsen's algorithm for 4-UCC is quite complicated. Furthermore, the complexity of the five open problems has remained unsettled for more

than two decades. This might be an indication that polynomial time algorithms for these five problems, if existent, are intricate.

Part II

Smoothed Analysis of Binary Search Trees

CHAPTER

SIX

Smoothed Analysis and Binary Search Trees

In the first part of this thesis, we considered the worst case complexity and approximability of computing restricted cycle covers. The term "efficient" in the context of approximability refers to the existence of polynomial-time approximation algorithms.

In this part of the thesis, we consider binary search trees, which are among the most fundamental data structures and, as such, are building blocks for many advanced data structures (see e.g. Aho et al. [1, 2] and Knuth [61]). *Efficient* in the context of data structures based on trees means that elements can be accessed in logarithmic time with respect to the size of the tree. The maximum access time of an element in a tree is proportional to the height of the tree.

Unfortunately, binary search trees are very inefficient in the worst case: The height of a tree generated from the sorted sequence is equal to the number of its elements. In contrast, a binary search tree constructed from a sequence drawn uniformly at random has logarithmic height in expectation. But in practice, we usually cannot assume that our sequences are completely random.

There is a huge discrepancy between the worst height and the average height of binary search trees. To close this gap, we consider the *smoothed height of binary search trees*: Instead of considering worst-case sequences or drawing sequences completely at random, we slightly perturb sequences given by an adversary.

We start this chapter by reviewing the history of and previous results on smoothed analysis and binary search trees. After that, we define some notations needed in subsequent chapters (Section 6.3) and state our new results (Section 6.4).

6.1 Existing Results for Smoothed Analysis

Santha and Vazirani introduced the semi-random model [82], in which an adversary adaptively chooses a sequence of bits, each of which is corrupted inde-

pendently with some fixed probability. They showed how to obtain sequences of quasi-random bits from such semi-random sources. Their work inspired research on semi-random graphs [21, 44], which can be viewed as a forerunner of the smoothed analysis of discrete problems.

Spielman and Teng introduced smoothed analysis as a hybrid of average-case and worst-case complexity [86, 89]. They showed that the simplex algorithm for linear programming with the shadow vertex pivot rule has polynomial smoothed complexity. This means that the running time of the algorithm is expected to be polynomial in terms of the input size and the variance of the Gaussian perturbation. Since then, smoothed analysis has been applied to a variety of fields, for instance several variants of linear programming [22, 41, 88], properties of moving objects [35], online and other algorithms [11,83,84], property testing [87], discrete optimisation [12, 79], graph theory [45], and computational geometry [36].

Banderier, Beier, and Mehlhorn [10] applied the concept of smoothed analysis to combinatorial problems. In particular, they analysed the number of left-to-right maxima of a sequence, which is the number of maxima seen when scanning a sequence from left to right. The worst case is the sequence $1, 2, \ldots, n$, which yields n left-to-right maxima. On average, we expect $\sum_{i=1}^{n} 1/i \approx \ln n$ left-to-right maxima. Banderier et al. used the perturbation model of *partial permutations*, where each element of the sequence is independently selected with a probability of $p \in [0, 1]$ and then a random permutation on the selected elements is performed (see Section 7.1 for a precise definition).

Banderier et al. proved that the number of left-to-right maxima under partial permutations is $O(\sqrt{(n/p)\log n})$ in expectation for $0 < p < 1$. Furthermore, they showed a lower bound of $\Omega(\sqrt{n/p})$ for $0 < p \leq 1/2$.

6.2 Existing Results for Binary Search Trees

Given a sequence $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ of n distinct elements from any ordered set, we obtain a binary search tree $T(\sigma)$ by iteratively inserting the elements $\sigma_1, \sigma_2, \ldots, \sigma_n$ into the initially empty tree (this is formally described in Section 6.3.1). Beyond being an important data structure, binary search trees play a central role in the analysis of algorithms. For instance, the height of $T(\sigma)$ equals the number of levels of recursion required by Quicksort when sorting σ if the first element is always chosen as the pivot (see e.g. Cormen et al. [29]).

The worst-case height of a binary search tree is obviously n: just take $\sigma = (1, 2, \ldots, n)$. (We define the length of a path as the number of vertices it contains.) The expected height of the binary search tree obtained from a random permutation (with all permutations being equally likely) has been the subject of a considerable amount of research in the past. We briefly review some results. Let the random variable $H(n)$ denote the height of a binary search tree obtained from a random permutation of n elements. Robson [75] proved that

$\mathbb{E}H(n) \approx c\ln(n) + o(\ln(n))$ for some $c \in [3.63, 4.3112]$ and observed that $H(n)$ does not vary much from experiment to experiment [76]. Pittel [72] proved the existence of a $\gamma > 0$ with $\gamma = \lim_{n\to\infty} \frac{\mathbb{E}H(n)}{\ln(n)}$. Devroye [37] then proved that $\lim_{n\to\infty} \frac{\mathbb{E}H(n)}{\ln(n)} = \alpha$ with $\alpha \approx 4.31107$ being the larger root of $\alpha \ln(2e/\alpha) = 1$. The variance of $H(n)$ was shown to be $O((\operatorname{llog} n)^2)$ by Devroye and Reed [38] and by Drmota [39]. Robson [77] proved that the expectation of the absolute value of the difference between the height of two random trees is constant. Thus, the height of random trees is concentrated around the mean. A climax was the result discovered independently by Drmota [40] and Reed [74] that the variance of $H(n)$ is actually $O(1)$. Furthermore, Reed [74] proved that the expectation of $H(n)$ is $\alpha \ln n + \beta \ln(\ln n) + O(1)$ with $\beta = \frac{3}{2\ln(\alpha/2)} \approx 1.953$. Finally, Robson [78] proved strong upper bounds on the probability of large deviations from the median. His results suggest that all moments of $H(n)$ are bounded from above by a constant.

Although the worst-case and average-case height of binary search trees are very well understood, nothing is known in between, i.e. when the sequences are not completely random but the randomness is limited.

6.3 Preliminaries

For general preliminaries, we refer to Section 2.1.

Let $\sigma = (\sigma_1, \ldots, \sigma_n) \in S^n$ for some ordered set S. We call σ a **sequence**. Usually, we assume that all elements of σ are distinct, i.e. $\sigma_i \neq \sigma_j$ for all $i \neq j$. The length of σ is n. In most cases, σ will simply be a permutation of $[n]$. We denote the sorted sequence $(1, 2, \ldots, n)$ by σ^n_{sort}. When considering partial alterations (see Section 7.2), we define $\sigma^n_{\text{sort}} = (0.5, 1.5, \ldots, n - 0.5)$ instead (this will be clear from the context).

Let $\tau = (\tau_1, \ldots, \tau_t)$. We call τ a **subsequence of $\boldsymbol{\sigma}$** if there are indexes $i_1 < i_2 < \ldots < i_t$ with $\tau_j = \sigma_{i_j}$ for all $j \in [t]$. Let $\mu = \{i_1, \ldots, i_t\} \subseteq [n]$. Then $\sigma_\mu = (\sigma_{i_1}, \ldots, \sigma_{i_t})$ denotes the subsequence consisting of all elements of σ at positions in μ. For instance, $\sigma_{[k]}$ denotes the prefix of length k of σ. In an abuse of notation, we sometimes use σ_μ to mean the set of elements at positions in μ, i.e. in this case $\sigma_\mu = \{\sigma_i \mid i \in \mu\}$. Whether we consider σ_μ to be a sequence or a set will always be clear from the context. For $\mu \subseteq [n]$, we define $\overline{\mu} = [n] \setminus \mu$.

6.3.1 Binary Search Trees and Left-to-right Maxima

Let $\sigma = (\sigma_1, \ldots, \sigma_n)$ be a sequence. We obtain a **binary search tree $\boldsymbol{T(\sigma)}$** from σ by iteratively inserting the elements $\sigma_1, \sigma_2, \ldots, \sigma_n$ into the initially empty tree as follows:

- The root of $T(\sigma)$ is the first element σ_1 of σ.

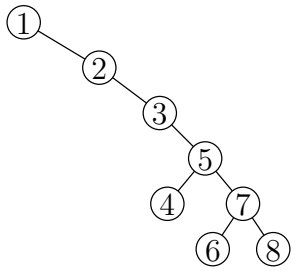


Figure 6.3.1: The binary search tree $T(\sigma)$ obtained from $\sigma = (1, 2, 3, 5, 7, 4, 6, 8)$. We have $\text{height}(\sigma) = 6$.

- Let $\sigma_< = \sigma_{\{i|\sigma_i<\sigma_1\}}$ be σ restricted to elements smaller than σ_1. The left subtree of the root σ_1 of $T(\sigma)$ is obtained inductively from $\sigma_<$.

  Analogously, let $\sigma_> = \sigma_{\{i|\sigma_i>\sigma_1\}}$ be σ restricted to elements greater than σ_1. The right subtree of the root σ_1 of $T(\sigma)$ is obtained inductively from $\sigma_>$.

Figure 6.3.1 shows an example. We denote the height of $T(\sigma)$ by $\mathbf{height(\sigma)}$, i.e. $\text{height}(\sigma)$ is the number of nodes on the longest path from the root to a leaf.

The element σ_i is called a **left-to-right maximum** of σ if $\sigma_i > \sigma_j$ for all $j \in [i-1]$. Let $\mathbf{ltrm(\sigma)}$ denote the number of left-to-right maxima of σ. We have $\text{ltrm}(\sigma) \leq \text{height}(\sigma)$ since the number of left-to-right maxima of a sequence is equal to the length of the right-most path in the tree $T(\sigma)$.

6.3.2 Probability Theory

To bound large deviations from the mean of binomially distributed random variables, we will frequently use the following lemma, which is based on Chernoff bounds and proved in Appendix A.2.

Lemma 6.3.1. *Let $k \in \mathbb{N}$, $\alpha > 1$ and $p \in [0, 1]$. Assume that we have mutually independent random variables $X_1, \ldots, X_k$ that assume values in $\{0, 1\}$. Assume further that $\mathbb{P}(X_i = 1) = p = 1 - \mathbb{P}(X_i = 0)$ for all $i \in [k]$. Let $X = \sum_{i=1}^{k} X_i$. Then*

$$\mathbb{P}\left((X > \alpha pk) \vee (X < \alpha^{-1}pk)\right) \leq 2 \cdot \exp\left(-2(1-\alpha^{-1})^2 p^2 k\right) .$$

6.4 New Results

We will consider the height of binary search trees subject to slight perturbations (*smoothed height*), i.e. the expected height under limited randomness. The height of a binary search tree obtained from a sequence of elements depends only on the ordering of the elements. Therefore, one should use a perturbation model that slightly perturbs the order of the elements of the sequence.

6.4.1 Perturbation Models

We consider three perturbation models, which we formally define in Chapter 7.

Partial permutations, introduced by Banderier et al. [10], rearrange some elements, i.e. they randomly permute a small subset of the elements of the sequence: Every element is marked independently with probability p, and then all marked element are randomly permuted.

The other two perturbation models are new.

Partial alterations do not move elements, but replace some elements by new elements chosen at random. Thus, they change the rank of the elements. More precisely: As for partial permutations, every element is marked with probability p, and then all marked elements are replaced by random elements.

Partial deletions remove some of the elements of the sequence without replacement, i.e. they shorten the input: Again, every element is marked with probability p, and then all marked elements are removed. This model turns out to be useful for analysing the other two models.

6.4.2 Lower and Upper Bounds

We show matching lower and upper bounds for the expected height of binary search trees and the expected number of left-to-right maxima under all three models in Chapter 8. For all $p \in (0, 1)$ and all sequences of length n, the expectation of the height of a binary search tree obtained via p-partial permutation is at most $6.7 \cdot (1-p) \cdot \sqrt{n/p}$ for sufficiently large n. On the other hand, the expectation of the height of a binary search tree obtained from the sorted sequence via p-partial permutation is at least $0.8 \cdot (1-p) \cdot \sqrt{n/p}$. This lower bound matches the upper bound up to a constant factor.

For the number of left-to-right maxima under partial permutations, we are able to prove an even better upper bound of $3.6 \cdot (1-p) \cdot \sqrt{n/p}$ for all sufficiently large n and a lower bound of $0.4 \cdot (1-p) \cdot \sqrt{n/p}$.

All these bounds hold for partial alterations as well.

Thus, under limited randomness, the behaviour of binary search trees and the number of left-to-right maxima differ markedly from both the worst-case and the average-case.

For partial deletions, we obtain $(1-p) \cdot n$ for both the lower and upper bound for the height of binary search trees and the number of left-to-right maxima.

6.4.3 Smoothed Analysis and Stability

In smoothed analysis one analyses how fragile worst case instances are. We suggest examining also the dual property: given a good (or best case) instance, how much can the complexity increase if the instance is perturbed slightly? In other words, how stable are best-case instances under perturbations?

The lower and upper bound for partial deletions are straightforward. The main reason for considering this model is that we can bound the expected height under partial alterations and permutations by the expected height under partial deletions (Section 9.1). The converse holds as well, we only have to blow up the sequences quadratically.

We exploit this when considering the stability of the perturbation models in Section 9.2: We prove that partial deletions and, thus, partial permutations and partial alterations as well are quite unstable, i.e. they can cause best-case instances to become much worse. More precisely: There are sequences of length n that yield trees of height $O(\log n)$, but the expected height of the tree obtained after smoothing is $n^{\Omega(1)}$.

CHAPTER

SEVEN

Perturbation Models for Permutations

Since we deal with ordering problems, we need perturbation models that slightly change a given permutation of elements. There seem to be two natural possibilities: Either *change the positions* of some elements, or *change the elements* themselves.

Partial permutations implement the first of these possibilities: A subset of the elements is chosen at random, and then these elements are randomly permuted.

The second possibility is realised by partial alterations. Again, a subset of the elements is chosen randomly. These elements are then replaced by random elements.

The third model, partial deletions, also starts by randomly choosing a subset of the elements. These elements are then removed without replacement.

We will formally define all three perturbation models in the following three sections. In Section 7.4, we show some properties of binary search trees and partial permutations and alterations.

For all three models, we obtain the random subset as follows. Let σ be a sequence of length n and $p \in [0,1]$ be a probability. Every element of σ is marked independently of the others with probability p. More formally: The random variable M_p^n is a random subset of $[n]$ with $\mathbb{P}(i \in M_p^n) = p$ for all $i \in [n]$. For any $\mu \subseteq [n]$ we have $\mathbb{P}(M_p^n = \mu) = p^{|\mu|} \cdot (1-p)^{|\overline{\mu}|}$.

Let $\mu \subseteq [n]$ be the set of marked positions. If $i \in \mu$, then we say that **position i** and **element σ_i are marked**. Thus, σ_μ is the sequence (or set) of all marked elements.

By $\textbf{height-perm}_p(\sigma)$, $\textbf{height-alter}_p(\sigma)$, and $\textbf{height-del}_p(\sigma)$ we denote the expected height of the binary search tree $T(\sigma')$, where σ' is obtained from σ by performing a p-partial permutation, alteration, or deletion on σ, respectively, on σ. Analogously, by $\textbf{ltrm-perm}_p(\sigma)$, $\textbf{ltrm-alter}_p(\sigma)$, and $\textbf{ltrm-del}_p(\sigma)$ we denote the expected number of left-to-right maxima of the sequence σ' obtained from σ via p-partial permutation, alteration, and deletion, respectively.

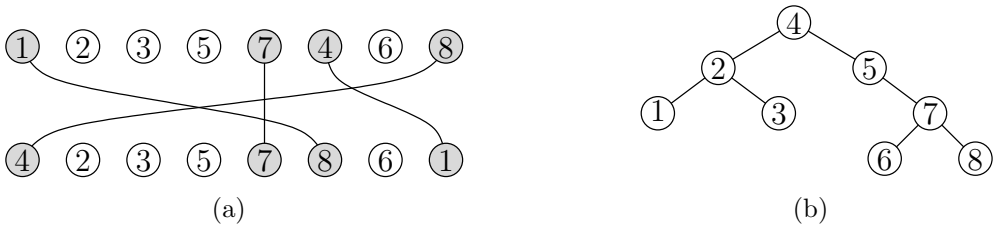


Figure 7.1.1: An example of a partial permutation. (a) Top: The sequence $\sigma = (1, 2, 3, 5, 7, 4, 6, 8)$; Figure 6.3.1 shows $T(\sigma)$. The first, fifth, sixth, and eighth element is (randomly) marked, thus $\mu = M_p^n = \{1, 5, 6, 8\}$. Bottom: The marked elements are randomly permuted. The result is the sequence $\sigma' = \Pi(\sigma, \mu)$, in this case $\sigma' = (4, 2, 3, 5, 7, 8, 6, 1)$. (b) $T(\sigma')$ with $\text{height}(\sigma') = 4$.

7.1 Partial Permutations

The notion of ***p*-partial permutations** was introduced by Banderier et al. [10]. Given a random subset M_p^n of $[n]$, the elements at positions in M_p^n are permuted according to a permutation drawn uniformly at random: Let $\sigma = (\sigma_1, \ldots, \sigma_n)$ and $\mu \subseteq [n]$. Then the sequence $\sigma' = \Pi(\sigma, \mu)$ is a random variable with the following properties:

- Π chooses a permutation π of μ uniformly at random and
- sets $\sigma'_{\pi(i)} = \sigma_i$ for all $i \in \mu$ and $\sigma'_i = \sigma_i$ for all $i \notin \mu$.

Note that $i \in \mu$ does not necessarily mean that σ_i is at a position different from i in $\Pi(\sigma, \mu)$; the random permutation can of course map $\pi(i)$ to i.

Example 7.1.1. *Figure 7.1.1 shows an example.*

By varying p, we can interpolate between the average and the worst case: for $p = 0$, no element is marked and $\sigma' = \sigma$, while for $p = 1$, all elements are marked and σ' is a random permutation of the elements of σ with all permutations being equally likely.

Let us show that partial permutations are indeed a suitable perturbation model by proving that the distribution of $\Pi(\sigma, M_p^n)$ favours sequences close to σ. To do this, we have to introduce a metric on sequences. Let σ and τ be two sequences of length n. Without loss of generality, we assume that both are permutations of $[n]$. Otherwise, we replace the jth smallest element of either sequence by j for $j \in [n]$. We define the distance $d(\sigma, \tau)$ between σ and τ as $d(\sigma, \tau) = |\{i \mid \sigma_i \neq \tau_i\}|$, thus d is a metric. Note that $d(\sigma, \tau) = 1$ is impossible since there are no two permutations that differ in exactly one position.

The distribution of $\Pi(\sigma, M_p^n)$ is symmetric around σ with respect to d, i.e. the probability that $\Pi(\sigma, M_p^n) = \tau$ for some fixed τ depends only on $d(\sigma, \tau)$.

Lemma 7.1.2. *Let $p \in (0,1)$, and let σ and τ be permutations of $[n]$ with $d = d(\sigma,\tau)$. Then*

$$\mathbb{P}(\Pi(\sigma, M_p^n) = \tau) = \sum_{k=0}^{n-d} p^{k+d} \cdot (1-p)^{n-d-k} \cdot \binom{n-d}{k} \cdot \frac{1}{(k+d)!}\,.$$

Proof. All d positions where σ and τ differ must be marked. This happens with probability p^d. The probability that k of the remaining positions are marked is $\binom{n-d}{k} \cdot p^k \cdot (1-p)^{n-d-k}$. Thus, the probability that $k+d$ positions are marked, d of which are positions where σ and τ differ, is $\binom{n-d}{k} \cdot p^{k+d} \cdot (1-p)^{n-d-k}$.

If $k+d$ positions are marked overall, the probability that the "right" permutation is chosen is $1/(k+d)!$. □

Let $\mathbb{P}_d = \sum_{k=0}^{n-d} p^{k+d} \cdot (1-p)^{n-d-k} \cdot \binom{n-d}{k} \cdot \frac{1}{(k+d)!}$ be the probability that $\Pi(\sigma, M_p^n) = \tau$ for a fixed sequence τ with distance d to σ. Then $\mathbb{P}_d$ tends exponentially to zero with increasing d. Thus, the distribution of $\Pi(\sigma, M_p^n)$ is highly concentrated around σ.

Lemma 7.1.3. *Let $p \in (0,1)$. There exists a constant $c < 1$ such that for all sufficiently large n, we have $\mathbb{P}_2 \le c \cdot \mathbb{P}_0$ and $\mathbb{P}_{d+1} \le c \cdot \mathbb{P}_d$ for all d with $2 \le d < n$.*

Proof. By omitting the last summand, we obtain

$$\mathbb{P}_d \ge \sum_{k=0}^{n-d-1} p^{k+d} \cdot (1-p)^{n-d-k} \cdot \binom{n-d}{k} \cdot \frac{1}{(k+d)!}\,.$$

Thus,

$$\begin{aligned}
\frac{\mathbb{P}_{d+1}}{\mathbb{P}_d} &\le \frac{\sum_{k=0}^{n-d-1} p^{k+d+1} \cdot (1-p)^{n-(d+1)-k} \cdot \binom{n-(d+1)}{k} \cdot \frac{1}{(k+d+1)!}}{\sum_{k=0}^{n-d-1} p^{k+d} \cdot (1-p)^{n-d-k} \cdot \binom{n-d}{k} \cdot \frac{1}{(k+d)!}} \\
&\le \max_{0 \le k \le n-d-1} \left(\frac{p^{k+d+1} \cdot (1-p)^{n-d-1-k} \cdot \binom{n-d-1}{k} \cdot \frac{1}{(k+d+1)!}}{p^{k+d} \cdot (1-p)^{n-d-k} \cdot \binom{n-d}{k} \cdot \frac{1}{(k+d)!}} \right) \\
&\le \frac{p}{1-p} \cdot \max_{0 \le k \le n-d-1} \left(\frac{n-d-k}{(n-d) \cdot (k+d+1)} \right) \le \frac{p}{1-p} \cdot \frac{1}{d+1}\,.
\end{aligned}$$

The second inequality holds because $\sum_{i \in I} a_i / \sum_{i \in I} b_i \le \max_{i \in I} a_i/b_i$ for any set I and nonnegative numbers a_i and b_i ($i \in I$). This proves the lemma for all d with $d+1 > \frac{1-p}{p}$.

What remains is to consider $d \le \frac{1-p}{p} - 1 = \frac{1}{p} - 2$. Fix $\alpha > 1$ arbitrarily with $\alpha p < 1$. Then $\mathbb{P}_{d+1} = \sum_{k=0}^{n-d-1} p^{k+d+1} \cdot (1-p)^{n-d-1-k} \cdot \binom{n-d-1}{k} \cdot \frac{1}{(k+d+1)!}$ is dominated by the summands with $k < \alpha p n$ as follows: Let

$$\mathbb{P}'_{d+1} = \sum_{0 \le k < \alpha p n} p^{k+d+1} \cdot (1-p)^{n-d-1-k} \cdot \binom{n-d-1}{k} \cdot \frac{1}{(k+d+1)!}\,,$$

then $\mathbb{P}_{d+1} \leq (1 - o(1)) \cdot \mathbb{P}'_{d+1}$. Furthermore, we define

$$\mathbb{P}'_d = \sum_{0 \leq k < \alpha p n} p^{k+1+d} \cdot (1-p)^{n-d-k-1} \cdot \binom{n-d}{k+1} \cdot \frac{1}{(k+1+d)!} \leq \mathbb{P}_d \,.$$

Now we have $\frac{\mathbb{P}_{d+1}}{\mathbb{P}_d} \leq (1 - o(1)) \cdot \frac{\mathbb{P}'_{d+1}}{\mathbb{P}'_d}$ and

$$\begin{aligned} \frac{\mathbb{P}'_{d+1}}{\mathbb{P}'_d} &\leq \max_{0 \leq k < \alpha p n} \left(\frac{p^{k+d+1} \cdot (1-p)^{n-d-1-k} \cdot \binom{n-d-1}{k} \cdot \frac{1}{(k+d+1)!}}{p^{k+1+d} \cdot (1-p)^{n-d-k-1} \cdot \binom{n-d}{k+1} \cdot \frac{1}{(k+1+d)!}} \right) \\ &\leq \max_{0 \leq k < \alpha p n} \left(\frac{k+1}{n-d} \right) = \frac{\alpha p n}{n-d} \leq \alpha p + o(1) \end{aligned}$$

for sufficiently large n. The last inequality holds because $d \leq \frac{1}{p} - 2 \in O(1)$. Thus, there exists a $c < 1$ with $\mathbb{P}_{d+1}/\mathbb{P}_d \leq \alpha p + o(1) \leq c$ for sufficiently large n. Finally, the proof above yields $\mathbb{P}_2/\mathbb{P}_0 \leq \frac{\mathbb{P}_2 \cdot \mathbb{P}_1}{\mathbb{P}_1 \cdot \mathbb{P}_0} \leq c^2 \leq c < 1$, which completes the proof. □

7.2 Partial Alterations

Let us now introduce ***p*-partial alterations**. For this perturbation model, we restrict the sequences of length n to be permutations of $[n - \frac{1}{2}] = \{\frac{1}{2}, \frac{3}{2}, \ldots, n - \frac{1}{2}\}$.

Every element at a position in M_p^n is replaced by a real number drawn uniformly and independently at random from $[0, n)$ to obtain a sequence σ'. All elements in σ' are distinct with probability one.

Instead of considering permutations of $[n - \frac{1}{2}]$, we could also consider permutations of $[n]$ and draw the random values from $[\frac{1}{2}, n + \frac{1}{2})$. This would not change the results. Another possibility would be to consider permutations of $[n]$ and draw the random values from $[0, n + 1)$. This would not change the results by much either. However, for technical reasons, we consider partial alterations as introduced above.

Example 7.2.1. *Let* $\sigma = (0.5, 1.5, 2.5, 4.5, 6.5, 3.5, 5.5, 7.5)$ *(which is the sequence of Example 7.1.1 with* 0.5 *subtracted from each element) and* $\mu = \{1, 5, 6, 8\}$*. By replacing the marked elements with random numbers, we may obtain the sequence* $(3.96..., 1.5, 2.5, 4.5, 7.22..., 7.95..., 5.5, 0.67...)$.

Like partial permutations, partial alterations interpolate between the worst case ($p = 0$) and the average case ($p = 1$). Partial alterations are somewhat easier to analyse: The majority of results on the average case height of binary search trees is actually not obtained by considering random permutations. Instead, the binary search trees are grown from a sequence of n random variables that are uniformly and independently drawn from $[0, 1)$. This corresponds to partial

alterations for $p = 1$. There is no difference between partial permutations and partial alterations for $p = 1$. This appears to hold for all p in the sense that the lower and upper bounds obtained for partial permutations and partial alterations are equal for all p.

The metric introduced above for partial permutations does not yield meaningful results for alterations: replacing a single element can change the rank of all elements. One possible metric is the edit distance: The distance of σ and τ is the minimum number of insertions, deletions, and substitutions by which we obtain a sequence σ' from σ with $\sigma'_i < \sigma'_j$ if and only if $\tau_i < \tau_j$ for all i and j.

7.3 Partial Deletions

As the third perturbation model, we introduce ***p*-partial deletions**: Again, we have a random marking M_p^n as in Section 7.1. Then we delete all marked elements to obtain the sequence $\sigma_{\overline{M_p^n}}$.

Example 7.3.1. *The sequence σ and the marking μ as in Example 7.1.1 yield the sequence* $(2, 3, 5, 6)$.

Partial deletions do not really perturb a sequence: any ordered sequence remains ordered even if elements are deleted. The main reason for considering partial deletions is that they are easy to analyse when considering the stability of perturbation models (Section 9.2). The results obtained for partial deletions then carry over to partial permutations and partial alterations since the expected heights with respect to these three models are closely related (Section 9.1).

7.4 Basic Properties

In this section, we state some properties of partial permutations (Section 7.4.2) and partial alterations (Section 7.4.3) that we will exploit in subsequent chapters. But let us start by considering some properties of binary search trees.

7.4.1 Properties of Binary Search Trees

We start by introducing a new measure for the height of binary search trees. Let $\mu \subseteq [n]$ and let σ be a sequence of length n. The **μ-restricted height of $T(\sigma)$**, denoted by **$\mathrm{height}(\sigma, \mu)$**, is the maximum number of elements of σ_μ on a root-to-leaf path in $T(\sigma)$.

Lemma 7.4.1. *For all sequences σ of length n and $\mu \subseteq [n]$, we have*

$$\begin{aligned} \mathrm{height}(\sigma) &\leq \mathrm{height}(\sigma, \mu) + \mathrm{height}(\sigma, \overline{\mu}) \quad \textit{and} \\ \mathrm{height}(\sigma, \mu) &\leq \mathrm{height}(\sigma_\mu)\,. \end{aligned}$$

Proof. Consider any path of maximum length from the root to a leaf in $T(\sigma)$. This path consists of at most $\text{height}(\sigma, \mu)$ elements of σ_μ and at most $\text{height}(\sigma, \overline{\mu})$ elements of $\sigma_{\overline{\mu}}$, which proves the first part.

For the second part, let a and b be elements of σ_μ that do not lie on the same path from the root to a leaf in $T(\sigma_\mu)$. Assume that $a < b$. Then there exists a c prior to a and b in σ_μ with $a < c < b$. Thus, a and b do not lie on the same root-to-leaf path in the tree $T(\sigma)$ either. Now consider any root-to-leaf path of $T(\sigma)$ with $\text{height}(\sigma, \mu)$ elements of σ_μ. Then all these elements lie on the same root-to-leaf path in $T(\sigma_\mu)$, which proves the second part of the lemma. □

Of course we have $\text{height}(\sigma, \mu) \leq \text{height}(\sigma)$ for all σ and μ. But $\text{height}(\sigma_\mu) \leq \text{height}(\sigma)$, which would imply $\text{height-del}_p(\sigma) \leq \text{height}(\sigma)$, does not hold in general: Consider $\sigma = (c, a, b, d, e)$ (we use letters and their alphabetical ordering instead of numbers for readability) and $\mu = \{2, 3, 4, 5\}$, then $\sigma_\mu = (a, b, d, e)$. Thus, $\text{height}(\sigma) = 3$ and $\text{height}(\sigma_\mu) = 4$. This will be investigated further in Section 9.2, when we consider the stability of the perturbation models.

To bound the smoothed height from above, we will use the following lemma, which is an immediate consequence of Lemma 7.4.1.

Lemma 7.4.2. *For all sequences σ of length n and $\mu \subseteq [n]$, we have*

$$\text{height}(\sigma) \leq \text{height}(\sigma_\mu) + \text{height}(\sigma, \overline{\mu}) .$$

Proof. We have $\text{height}(\sigma) \leq \text{height}(\sigma, \mu) + \text{height}(\sigma, \overline{\mu}) \leq \text{height}(\sigma_\mu) + \text{height}(\sigma, \overline{\mu})$ according to Lemma 7.4.1. □

We can state equivalent lemmas for left-to-right maxima. Let σ be a sequence of length n and $\mu \subseteq [n]$. Then $\mathbf{ltrm(\sigma, \mu)}$ denotes the **μ-restricted number of left-to-right maxima** of σ, i.e. the number of elements σ_i such that $i \in \mu$ and σ_i is a left-to-right maximum of σ. We omit the proof of the following lemma since it is almost identical to the proofs of the lemmas above.

Lemma 7.4.3. *Let σ be a sequence of length n and $\mu \subseteq [n]$. Then*

$$\begin{aligned} \text{ltrm}(\sigma) &\leq \text{ltrm}(\sigma, \mu) + \text{ltrm}(\sigma, \overline{\mu}) , \\ \text{ltrm}(\sigma, \mu) &\leq \text{ltrm}(\sigma_\mu) , \textit{ and} \\ \text{ltrm}(\sigma) &\leq \text{ltrm}(\sigma_\mu) + \text{ltrm}(\sigma, \overline{\mu}) . \end{aligned}$$

7.4.2 Properties of Partial Permutations

Let us now prove some properties of partial permutations. The three lemmas proved in this section are crucial for estimating the smoothed height and the smoothed number of left-to-right maxima under partial permutations. In the next section, we will prove counterparts of these lemmas for partial alterations that will play a similar role in estimating the height under partial alterations.

We start by proving that the expected height under partial permutations depends merely on the elements that are left unmarked. The marked elements contribute at most $O(\log n)$ to the height. Thus, when estimating the expected height in the subsequent sections, we can restrict ourselves to considering the elements that are left unmarked.

Lemma 7.4.4. *Let σ be a sequence of length n and let $p \in (0,1)$. Let $\mu \subseteq [n]$ be a random set of marked positions and $\sigma' = \Pi(\sigma, \mu)$ be the random sequence obtained from σ via p-partial permutation. Then*

$$\text{height-perm}_p(\sigma) = \mathbb{E}(\text{height}(\sigma')) \leq \mathbb{E}\left(\text{height}(\sigma', \overline{\mu})\right) + O(\log n)\,.$$

Proof. We have $\text{height}(\sigma_\mu) \in O(\log n)$ since the elements at positions in μ are randomly permuted. Then the lemma follows from Lemma 7.4.2. □

And again we obtain an equivalent lemma for left-to-right maxima.

Lemma 7.4.5. *Under the assumptions of Lemma 7.4.4, we have*

$$\text{ltrm-perm}_p(\sigma) \leq \mathbb{E}\left(\text{ltrm}(\sigma', \overline{\mu})\right) + O(\log n)\,.$$

The following lemma gives an upper bound for the probability that no element in a fixed set of elements is permuted to a position in a fixed set of positions.

Lemma 7.4.6. *Let $p \in (0,1)$, $\alpha > 1$, let $n \in \mathbb{N}$ be sufficiently large, and let σ be a sequence of length n with elements from $[n]$. Let $\sigma' = \Pi(\sigma, M_p^n)$.*

Let $\ell = a\sqrt{n/p}$ and $k = b\sqrt{n/p}$ with $a, b \in \Omega((\text{polylog}\, n)^{-1}) \cap O(\text{polylog}\, n)$. Let $A = \sigma'_{[\ell]}$ be the set of the first ℓ elements of σ' and let $B \subseteq [n]$ be any subset with $|B| = k$.

Then $\mathbb{P}(A \cap B = \emptyset) \leq \exp(-ab/\alpha)$.

Proof. We choose β with $1 < \beta^3 < \alpha$ arbitrarily. According to Lemma 6.3.1, the probability P that

- $|M_p^n \cap [\ell]| < \beta^{-1}p\ell$, i.e. that too few of the first ℓ positions are marked,
- $|\sigma_{M_p^n} \cap B| < \beta^{-1}pk$, i.e. that too few of the elements of B are marked, or
- $|M_p^n| > \beta pn$, i.e. that too many positions are marked overall

is $O(\exp(-n^\epsilon))$ for an appropriately chosen $\epsilon > 0$ by Lemma 6.3.1. This holds because $a, b \in \Omega((\text{polylog}\, n)^{-1})$.

From now on, we assume that at least $\beta^{-1}p\ell$ of the first ℓ positions of σ are marked, at least $\beta^{-1}pk$ elements in B are marked, and at most βpn positions are

marked overall. The probability that then no element from B is in A is at most

$$\begin{aligned}\left(\frac{\beta pn - \beta^{-1}p\ell}{\beta pn}\right)^{\beta^{-1}pk} &= \left(1 - \frac{\ell}{\beta^2 n}\right)^{\beta^{-1}pk} \\ = \left(\left(1 - \frac{\ell}{\beta^2 n}\right)^{\frac{\beta^2 n}{\ell}}\right)^{\frac{\ell}{\beta^2 n}\cdot\beta^{-1}pk} &\leq \exp\left(-\frac{\ell}{\beta^2 n}\cdot\beta^{-1}pk\right) = \exp\left(-\frac{ab}{\beta^3}\right).\end{aligned}$$

Overall, $\mathbb{P}(A \cap B = \emptyset) \leq \exp(-ab/\beta^3) + P \leq \exp(-ab/\alpha)$ for sufficiently large n since $a, b \in O(\operatorname{polylog} n)$. □

7.4.3 Properties of Partial Alterations

Partial alterations possess roughly the same properties as partial permutations. We state the lemmas and restrict ourselves to pointing out the differences in the proofs.

Lemma 7.4.7. *Let σ be a sequence of length n with elements from $[n - \frac{1}{2}]$ and let $p \in (0,1)$. Let σ' be the random sequence obtained from σ via p-partial alteration and μ be the random set of marked positions. Then*

$$\begin{aligned}\text{height-alter}_p(\sigma) &\leq \mathbb{E}(\text{height}(\sigma', \overline{\mu})) + O(\log n) \quad \textit{and} \\ \text{ltrm-alter}_p(\sigma) &\leq \mathbb{E}(\text{ltrm}(\sigma', \overline{\mu})) + O(\log n)\,.\end{aligned}$$

The following lemma is the counterpart of Lemma 7.4.6.

Lemma 7.4.8. *Let $p \in (0,1)$, $\alpha > 1$, let $n \in \mathbb{N}$ be sufficiently large, and let σ be a sequence with elements from $[n - \frac{1}{2}]$. Let σ' be the random sequence obtained from σ by performing a p-partial alteration.*

Let $\ell = a\sqrt{n/p}$ and $k = b\sqrt{n/p}$ with $a, b \in \Omega((\operatorname{polylog} n)^{-1}) \cap O(\operatorname{polylog} n)$. Let $A = \sigma'_{[\ell]}$ and $B = [x, x+k) \subseteq [0, n)$ for some x.

Then $\mathbb{P}(A \cap B = \emptyset) \leq \exp(-ab/\alpha)$.

Proof. The proof is similar to the proof of Lemma 7.4.6. Choose β arbitrarily with $1 < \beta < \alpha$. Assume that at least $\beta^{-1}p\ell$ of the first ℓ positions of σ are marked. Then the probability that no element in A assumes a value in B is at most

$$\left(\frac{n-k}{n}\right)^{\beta^{-1}p\ell} = \left(\left(1 - \frac{k}{n}\right)^{\frac{n}{k}}\right)^{ab/\beta} \leq \exp(-ab/\beta)\,.$$

The remainder of the proof proceeds as in the proof of Lemma 7.4.6. □

CHAPTER

EIGHT

Tight Bounds for Binary Search Trees

In this chapter, we prove tight lower and upper bounds for the number of left-to-right maxima (Section 8.1) and the height of binary search trees (Section 8.2) under all three perturbation models. Additionally, we present some results of experiments that we performed to estimate the constants in the bounds for the height of binary search trees (Section 8.3). These results have led to Conjecture 10.1.2.

8.1 Bounds for Left-To-Right Maxima

We start by considering the easier problem of left-to-right maxima. For partial permutations and partial alterations, we obtain lower and upper bounds of $0.4 \cdot (1-p) \cdot \sqrt{n/p}$ and $3.6 \cdot (1-p) \cdot \sqrt{n/p}$, respectively. For partial deletions, we easily obtain $(1-p) \cdot n$ as both the lower and upper bound. In the next section, we prove upper bounds, while lower bounds are proved in Section 8.1.2.

8.1.1 Upper Bounds

Theorem 8.1.1. *Let $p \in (0,1)$. Then for all sufficiently large n and for all sequences σ of length n,*

$$\text{ltrm-perm}_p(\sigma) \leq 3.6 \cdot (1-p) \cdot \sqrt{n/p}\,.$$

Proof. The main idea for proving this theorem is to estimate the probability that one of the k largest elements of σ is among the first k elements, which would bound the number of left-to-right maxima by $2k$.

According to Lemma 7.4.5, it suffices to show

$$\mathbb{E}(\text{ltrm}(\sigma', \overline{\mu})) \leq C \cdot (1-p) \cdot \sqrt{n/p}$$

for some $C < 3.6$, where $\mu \subseteq [n]$ is the random set of marked positions and σ' is the sequence obtained by randomly permuting the elements of σ_μ. Then

$$\text{ltrm-perm}_p(\sigma) \leq C \cdot (1-p) \cdot \sqrt{n/p} + O(\log n) \leq 3.6 \cdot (1-p) \cdot \sqrt{n/p}\,.$$

We assume without loss of generality that σ is a permutation of $[n]$.

Let $K_c = c \cdot \sqrt{n/p}$ for $c \in [\log n]$. In this and the following proofs, we assume that K_c is a natural number for the sake of readability. If K_c is not a natural number, then we can replace K_c by $\lceil K_c \rceil$. The proofs remain valid.

Choose α with $1 < \alpha < 1.001$. Let P denote the probability that less than $\alpha^{-1}pK_c$ of the first K_c positions are marked or that less than $\alpha^{-1}pK_c$ of the K_c largest elements are marked for some $c \in [\log n]$ or that more than αpn elements are marked overall. Then, by Lemma 6.3.1, P tends exponentially to zero as n increases.

From now on, we assume that for all $c \in [\log n]$, at least $\alpha^{-1}pK_c$ of the first K_c positions and of the K_c largest elements are marked. Furthermore, we assume that at most αpn positions are marked overall. In this case, we say that the partial permutation is **partially successful**. If a partial permutation is not partially successful, we bound the number of left-to-right maxima by n.

We call σ' ***c*-successful** for $c \in [\log n]$ if one of the K_c largest elements $n, n-1, \ldots, n-K_c+1$ is among the first K_c elements in σ'.

Assume that σ' is c-successful and that $m \in \{n-K_c+1, \ldots, n\}$ is among the first K_c elements of σ'. The only unmarked elements that can contribute to $\text{ltrm}(\sigma', \overline{\mu})$ are those that are among the first K_c positions and those that are larger than m. All other unmarked elements are smaller than m and located behind m in σ', thus they are no left-to-right maxima. The expected number of unmarked elements larger than $n - K_c$ plus the expected number of unmarked positions among the first K_c positions is at most $2 \cdot (1-p) \cdot K_c = Q_c$. Hence, we have $\mathbb{E}(\text{ltrm}(\sigma', \overline{\mu})) \leq Q_c$ if σ' is c-successful.

Let $c \in [\log n]$. The probability that a partially successful partial permutation is not c-successful is at most $\exp(-c^2/\alpha)$ according to Lemma 7.4.6. In particular, the probability that σ' is not $(\log n)$-successful is at most $P' = \exp(-(\log n)^2/\alpha)$. If σ' is not $(\log n)$-successful, we bound the number of left-to-right maxima by n.

If we restrict ourselves to partially successful partial permutations, we have

$$\mathbb{P}(\text{ltrm-perm}_p(\sigma) > Q_c) \leq \exp(-c^2/\alpha)\,.$$

Hence, we can bound $\text{ltrm}(\sigma', \overline{\mu})$ from above by

$$\begin{aligned} &\sum_{c=0}^{\log n} Q_{c+1} \cdot \underbrace{\mathbb{P}(\sigma' \text{ is not } c\text{-successful but } (c+1)\text{-successful})}_{\leq \mathbb{P}(\sigma' \text{ is not } c\text{-successful})} + n \cdot (P + P') \\ \leq\ & 2 \cdot (1-p) \cdot \sqrt{n/p} \cdot \underbrace{\sum_{c \in \mathbb{N}} (c+1) \cdot e^{-\frac{c^2}{\alpha}}}_{< 1.8 \text{ for } \alpha < 1.001} + n \cdot (P + P') \leq C \cdot (1-p) \cdot \sqrt{n/p} \end{aligned}$$

for some $C < 3.6$, which proves the theorem. □

We obtain the same upper bound for the expected number of left-to-right maxima under partial alterations.

Theorem 8.1.2. *Let $p \in (0,1)$. Then for all sufficiently large n and for all sequences σ of length n (where σ is a permutation of $[n - \frac{1}{2}]$),*

$$\text{ltrm-alter}_p(\sigma) \leq 3.6 \cdot (1-p) \cdot \sqrt{n/p}\,.$$

Proof. The main difference between the proof of this theorem and the proof of Theorem 8.1.1 is that we have to use Lemma 7.4.8 instead of Lemma 7.4.6.

The sequence σ' obtained from σ via p-partial alteration is called c-successful if at least one of the first K_c elements of σ' lies in the interval $[n - K_c, n)$. The remainder of the proof proceeds in the same way as the proof of Theorem 8.1.1. □

For partial deletions, we easily obtain the following upper bound.

Theorem 8.1.3. *For all $p \in [0,1]$, $n \in \mathbb{N}$, and sequences σ of length n,*

$$\text{ltrm-del}_p(\sigma) \leq (1-p) \cdot n\,.$$

Proof. Let σ' be the sequence obtained from σ via p-partial deletion. Then σ' consists of $(1-p) \cdot n$ elements in expectation. The number of elements is an upper bound for the number of left-to-right maxima. □

8.1.2 Lower Bounds

The following lemma is an improvement of the lower bound proof for the number of left-to-right maxima under partial permutations presented by Banderier et al. [10]. We obtain a lower bound with a much larger constant that holds for all $p \in (0,1)$ (Theorem 8.1.5); the lower bound provided by Banderier et al. holds only for $p \leq 1/2$.

Lemma 8.1.4. *Let $p \in (0,1)$, $\alpha > 1$, and $c > 0$. Then for all sufficiently large n, there exist sequences σ of length n with*

$$\text{ltrm-perm}_p(\sigma) \geq \exp\left(-c^2\alpha\right) \cdot c \cdot (1-p) \cdot \sqrt{n/p}\,.$$

Proof. Let $K_c = c \cdot \sqrt{n/p}$ and let $\sigma = (n-K_c+1, n-K_c+2, \ldots, n, 1, 2, \ldots, n-K_c)$. We start with a sketch of the proof: The probability that none of the first K_c elements is moved further to the front is bounded from below by $\exp(-c^2\alpha)$ for any fixed $\alpha > 1$. In such a case, all unmarked elements among the first K_c elements are left-to-right maxima, and there are $(1-p) \cdot K_c$ such elements in expectation.

Choose β arbitrarily with $1 < \beta^3 < \alpha$. Let P denote the probability that more than $\beta p K_c$ of the first K_c elements or less than $\beta^{-1}pn$ of the remaining $n - K_c$ elements are selected. P tends exponentially to zero as n increases (Lemma 6.3.1).

Let μ be the set of marked positions and let $\mu_c = \mu \cap [K_c]$ be the set of marked positions among the first K_c positions, $\mu_c = \{i_1, \ldots, i_x\}$ with $i_1 < i_2 < \ldots < i_x$, where $x = |\mu_c|$ is the number of such positions. Let $y = |\mu \setminus \mu_c|$ be the number of remaining positions. Let f be a random permutation of μ. We say that f is **successful** if $f(i) > i$ for all $i \in \mu_c$. Thus, under a successful permutation, all marked elements in $\{n - K_c + 1, \ldots, n\}$ are moved further to the back.

If f is successful, then all $K_c - x$ unmarked elements in $\{n - K_c + 1, \ldots, n\}$ are left-to-right maxima. Provided that at most $\beta p K_c$ of the first K_c elements are marked, i.e. $x \leq \beta p K_c$, the expectation of $K_c - x$ is at least $(1-p) \cdot K_c$.

Let us bound the probability from below that the random permutation f of μ is successful for a given μ: For i_x, y positions are allowed and x positions are not allowed; for i_{x-1}, y are positions allowed (all in $\mu \setminus \mu_c$ plus one for position i_x minus one for position $f(i_x)$) and $x-1$ positions are not allowed; $\ldots$; for i_1, y positions are allowed and one position is not allowed. Thus, the probability that the random permutation is successful is at least

$$\left(\frac{y}{y+x}\right)^x = \Big(\underbrace{\Big(1 - \frac{x}{y+x}\Big)^{\frac{y+x}{x}}}_{\geq e^{-1}\cdot(1-\frac{x}{y+x})}\Big)^{\frac{x^2}{y+x}} \geq \exp\left(\left(\ln\left(1 - \frac{x}{y+x}\right) - 1\right) \cdot \frac{x^2}{y+x}\right) .$$

Provided that $x \leq \beta p K_c$ and $x + y \geq y \geq \beta^{-1} pn$, we obtain a probability that the random permutation is successful of at least

$$\begin{aligned} & \exp\left(\left(\ln\left(1 - \frac{\beta p K_c}{\beta^{-1} pn}\right) - 1\right) \cdot \frac{\beta^2 p^2 K_c^2}{\beta^{-1} pn}\right) \\ = \; & \exp\left(\left(\ln\left(1 - \frac{\beta^2 c}{\sqrt{pn}}\right) - 1\right) \cdot \beta^3 c^2\right) = Q \cdot \exp(-\beta^3 c^2) \end{aligned}$$

for $Q = \big(1 - \frac{\beta^2 c}{\sqrt{pn}}\big)^{\beta^3 c^2}$, which tends to one as n increases. Thus, with a probability of at least $(1-P) \cdot Q \cdot \exp(-\beta^3 c^2)$, all unmarked elements of $\{K_c + 1, \ldots, n\}$ are left-to-right maxima. Furthermore, we have $(1-P) \cdot Q \cdot \exp(-\beta^3 c^2) \geq \exp(-c^2\alpha)$ for sufficiently large n. Since the expectation of the number of unmarked elements among the first K_c elements is at least $(1-p) \cdot K_c$, the lemma is proved. □

The term $\exp(-c^2\alpha) \cdot c$ assumes its maximum for $c = 1/\sqrt{2\alpha}$. Thus, we obtain the strongest lower bound from Lemma 8.1.4 by choosing α close to 1 and $c = 1/\sqrt{2\alpha}$. This yields the following theorem.

Theorem 8.1.5. *For all $p \in (0,1)$ and all sufficiently large n, there exists a sequence σ of length n with*

$$\text{ltrm-perm}_p(\sigma) \geq 0.4 \cdot (1-p) \cdot \sqrt{n/p}\,.$$

□

Theorem 8.1.5 also yields the same lower bound for height-perm$_p(\sigma)$ since the number of left-to-right maxima of a sequence is a lower bound for the height of the binary search tree obtained from that sequence. We can, however, prove a stronger lower bound for the smoothed height of binary search trees (Theorem 8.2.7).

A consequence of Lemma 8.1.4 is that there is no constant c such that the number of left-to-right maxima is at most $c \cdot (1-p) \cdot \sqrt{n/p}$ with high probability, i.e. with a probability of at least $1 - n^{-\Omega(1)}$. Thus, the bounds proved for the expected tree height or the number of left-to-right maxima cannot be generalised to bounds that hold with high probability. A bound for the tree height that holds with high probability can be obtained from Lemma 7.4.6, as we will show in Theorem 8.2.4. Clearly, this bound holds for the number of left-to-right maxima as well.

Let us now prove the counterpart of Lemma 8.1.4 for partial alterations.

Lemma 8.1.6. *Let $p \in (0,1)$, $\alpha > 1$, and $c > 0$. Then for all sufficiently large n, there exist sequences σ of length n with*

$$\text{ltrm-alter}_p(\sigma) \geq \exp\left(-c^2\alpha\right) \cdot c \cdot (1-p) \cdot \sqrt{n/p}\,.$$

Proof. Let $K_c = c \cdot \sqrt{n/p}$. Let $\sigma = (n - K_c + \frac{1}{2}, n - K_c + \frac{3}{2}, \ldots, n - \frac{1}{2}, \frac{1}{2}, \frac{3}{2}, \ldots, n - K_c - \frac{1}{2})$. Choose β arbitrarily with $1 < \beta < \alpha$. Let P denote the probability that more than $\beta p K_c$ of the first K_c positions are marked. By Lemma 6.3.1, P tends exponentially to zero as n increases.

Let μ_c be the set of marked positions among the first K_c positions. Let $x = |\mu_c|$ and $\mu_c = \{i_1, \ldots, i_x\}$ with $i_1 < i_2 < \ldots < i_x$. We have $\sigma_{i_j} = n - K_c + i_j - \frac{1}{2}$ for all $j \in [x]$. Let σ' be the sequence obtained from σ by replacing all marked elements with random numbers from $[0, n)$. We say that σ' is successful if $\sigma'_{i_j} \leq n - K_c$ for all $j \in [x]$. If σ' is successful, then all $K_c - x$ unmarked elements among the first K_c elements of σ are left-to-right maxima.

The probability that σ' is successful is at least

$$\left(\frac{n-K_c}{n}\right)^x = \Big(\underbrace{\Big(1 - \frac{K_c}{n}\Big)^{\frac{n}{K_c}}}_{\geq e^{-1} \cdot (1 - \frac{K_c}{n})}\Big)^{\frac{xK_c}{n}} \geq \exp\left(\left(\ln\left(1 - \frac{K_c}{n}\right) - 1\right) \cdot \frac{xK_c}{n}\right)\,.$$

Provided that $x \leq \beta p K_c$, we obtain a probability that σ' is successful of at least

$$\begin{aligned} & \exp\left(\left(\ln\left(1 - \frac{\beta p K_c}{n}\right) - 1\right) \cdot \frac{\beta p K_c^2}{n}\right) \\ = \; & \exp\left(\left(\ln\left(1 - \frac{\beta c}{\sqrt{pn}}\right) - 1\right) \cdot \beta c^2\right) = Q \cdot \exp(-\beta c^2) \end{aligned}$$

for $Q = \left(1 - \frac{\beta c}{\sqrt{pn}}\right)^{\beta c^2}$, which tends to one as n increases. Thus, with a probability of at least $(1 - P) \cdot Q \cdot \exp(-\beta c^2)$, all unmarked elements among the first K_c elements are left-to-right maxima. The expected number of unmarked elements among the first K_c elements is at least $(1 - p) \cdot K_c$. Furthermore, for sufficiently large n, we have $(1-P)\cdot Q\cdot\exp(-\beta c^2) \geq \exp(-\alpha c^2)$, which proves the lemma. □

From the above lemma, we obtain the same lower bounds for the number of left-to-right maxima as for partial permutations, again by choosing α close to 1 and $c = 1/\sqrt{2\alpha}$.

Theorem 8.1.7. *For all $p \in (0, 1)$ and all sufficiently large n, there exists a sequence σ of length n with*

$$\text{ltrm-alter}_p(\sigma) \geq 0.4 \cdot (1 - p) \cdot \sqrt{n/p}\,.$$

□

As for partial permutations, a consequence of Lemma 8.1.6 is that we cannot achieve a bound of $O((1 - p) \cdot \sqrt{n/p})$ that holds with high probability for the number of left-to-right maxima or the height of binary search trees, but we can show that the height after p-partial alteration is $O(\sqrt{(n/p) \cdot \log n})$ with high probability (Theorem 8.2.4).

For partial deletions, we easily obtain a matching lower bound.

Theorem 8.1.8. *For all $p \in [0, 1]$, $n \in \mathbb{N}$,*

$$\text{ltrm-del}_p(\sigma^n_{\text{sort}}) = (1 - p) \cdot n\,.$$

Proof. Let σ' be the sequence obtained from σ via p-partial deletion. Every element of σ' is a left-to-right maximum, and σ' consists of $(1 - p) \cdot n$ elements in expectation. □

8.2 Bounds for Binary Search Trees

We now consider the smoothed height of binary search trees. For both partial permutations and partial alterations, we obtain lower and upper bounds of $0.8 \cdot (1 - p) \cdot \sqrt{n/p}$ and $6.7 \cdot (1 - p) \cdot \sqrt{n/p}$, respectively. The upper and lower bounds shown for the number of left-to-right maxima under partial deletions (Theorems 8.1.3 and 8.1.8) carry over to the height of binary search trees.

8.2.1 Upper Bounds

Theorem 8.2.1. *Let $p \in (0, 1)$. Then for all sufficiently large n and all sequences σ of length n, we have*

$$\text{height-perm}_p(\sigma) \leq 6.7 \cdot (1 - p) \cdot \sqrt{n/p}\,.$$

Proof. The idea is to divide the sequence into blocks $B_1, B_2, \ldots$, where B_d is of size $cd^2\sqrt{n/p}$ for some $c > 0$. Each block B_d is further divided into d^4 parts $A_d^1, \ldots, A_d^{d^4}$, each consisting of $cd^{-2}\sqrt{n/p}$ elements. Assume that on every root-to-leaf path in the tree obtained from the perturbed sequence, there are elements of at most two such A_d^i for every d. Then the height can be bounded from above by

$$\sum_{d=1}^{\infty} 2 \cdot \underbrace{cd^{-2}\sqrt{n/p}}_{\text{size of an } A_d^i} = (c\pi^2/3)\sqrt{n/p}\,.$$

The probability for such an event is roughly $O\big(\exp(-c^2)^2/\big(1-\exp(-c^2)\big)\big)$. We obtain the upper bound claimed in the theorem mainly by carefully applying this bound and by exploiting the fact that only a fraction of $(1-p)$ of the elements are unmarked. Marked elements contribute at most $O(\log n)$ to the expected height of the tree.

According to Lemma 7.4.4, it suffices to show

$$\mathbb{E}(\text{height}(\sigma', \overline{\mu})) \leq C \cdot (1-p) \cdot \sqrt{n/p}$$

for some fixed $C < 6.7$, where $\mu \subseteq [n]$ is the random set of marked positions and σ' is the sequence obtained by randomly permuting the elements of σ_μ. Then

$$\text{height-perm}_p(\sigma) \leq C \cdot (1-p) \cdot \sqrt{n/p} + O(\log n) \leq 6.7 \cdot (1-p) \cdot \sqrt{n/p}$$

for sufficiently large n.

Choose α arbitrarily with $1 < \alpha < 1.01$. Without loss of generality, we assume that σ is a permutation of $[n]$.

We define

$$D(d) = \sum_{i=1}^{d-1} i^2 = \frac{1}{3} \cdot \big(d-1\big) \cdot \big(d-\frac{1}{2}\big) \cdot d\,.$$

Then $D(d) \geq d^3/8$ for $d \geq 2$.

Let $c \in [\log n]$ and $K_c = c \cdot \sqrt{n/p}$. We divide a prefix of the sequence σ into blocks $B_1, B_2, \ldots, B_{(\log n)^2}$. The block B_d consists of $d^2 K_c$ elements: B_1 contains the elements of σ at the first K_c positions, B_2 contains the elements of σ at the next $4K_c$ positions, and so on. Thus,

$$B_d = \sigma_{[D(d+1)\cdot K_c]} \setminus \sigma_{[D(d)\cdot K_c]}\,.$$

Let $B = \bigcup_{d=1}^{(\log n)^2} B_d$ be the set of elements that are contained in any B_d. Let $d' = (\log n)^2 + 1$ and $D' = D(d') \geq (\log n)^6/8$. We have $|B| = D' \cdot K_c \geq \frac{1}{8} \cdot (\log n)^6 \cdot K_c$.

Every block B_d is further divided into d^4 subsets $A_d^1, \ldots, A_d^{d^4}$ of elements as follows: A_d^1 contains the K_c/d^2 smallest elements of B_d, A_d^2 contains the K_c/d^2 next smallest elements of B_d, …, and $A_d^{d^4}$ contains the K_c/d^2 largest elements

of B_d. Figure 8.2.1(a) illustrates the division of σ into blocks $B_1, B_2, \ldots, B_{(\log n)^2}$ and subsets A_d^i for $d \in [(\log n)^2]$ and $i \in [d^2]$.

Finally, we divide $[n]$ into $\log n \cdot \sqrt{np}$ subsets $C_1, \ldots, C_{\log n \cdot \sqrt{np}}$ with

$$C_j = \left\{ \frac{\sqrt{n/p}}{\log n} \cdot (j-1) + 1, \ldots, \frac{\sqrt{n/p}}{\log n} \cdot j \right\} .$$

Thus, C_1 contains the $(\log n)^{-1} \cdot \sqrt{n/p}$ smallest numbers of $[n]$, C_2 contains the $(\log n)^{-1} \cdot \sqrt{n/p}$ next smallest numbers of $[n]$, ..., and $C_{\log n \cdot \sqrt{np}}$ contains the $(\log n)^{-1} \cdot \sqrt{n/p}$ largest elements of $[n]$.

Let $\eta = 1 + n^{-1/6}$. Then

$$\eta^{-1} = \frac{1}{1+n^{-1/6}} = 1 - \frac{n^{-1/6}}{1+n^{-1/6}} \geq 1 - n^{-1/6} . \tag{8.1}$$

We call a set of k positions or elements **partially successful** in μ and σ' if at least $\eta^{-1}pk$ and at most ηpk elements of this set are marked. We say that μ and σ' are partially successful if the following properties are fulfilled:

- for all $c \in [\log n]$, $d \in [(\log n)^2]$, and $i \in [d^4]$, A_d^i is partially successful in μ and σ', and
- for all $j \in [\log n \sqrt{np}]$, C_j is partially successful in μ and σ'.

There are only polynomially many sets of elements that must be partially successful, and every such set is of cardinality $\Omega\big(\sqrt{n/p}/\operatorname{polylog} n\big)$. Hence, there exists some $\epsilon > 0$ such that the probability that μ and σ are partially successful is $O(\exp(-n^\epsilon))$ according to Lemma 6.3.1. Let P denote this probability. If μ and σ' are not partially successful, we bound the height of $T(\sigma')$ by n.

From now on, we assume that μ and σ' are partially successful. When speaking about partial success, we occasionally do not mention μ or σ'.

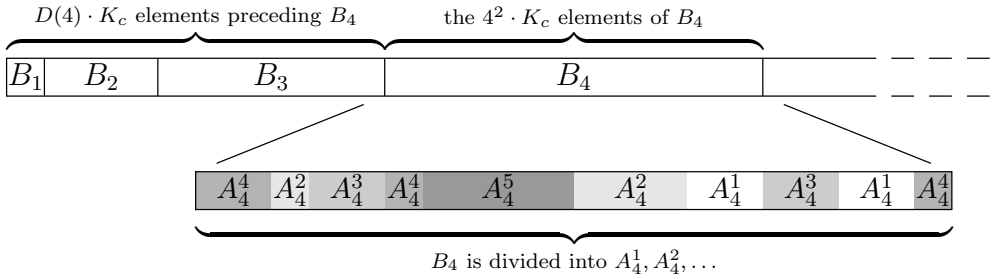
We call a subset A_d^i ***c*-successful** if at least one element of A_d^i is permuted to one of the $D(d) \cdot c \cdot \sqrt{n/p}$ positions that precede B_d. Thus, for all $d \in [(\log n)^2]$, $d \geq 2$, and $i \in [d^4]$, we have

$$\mathbb{P}(A_d^i \text{ is not successful}) \leq \exp(-d^{-2} c D(d) c \alpha^{-1}) \leq \exp(-c^2 d/(8\alpha))$$

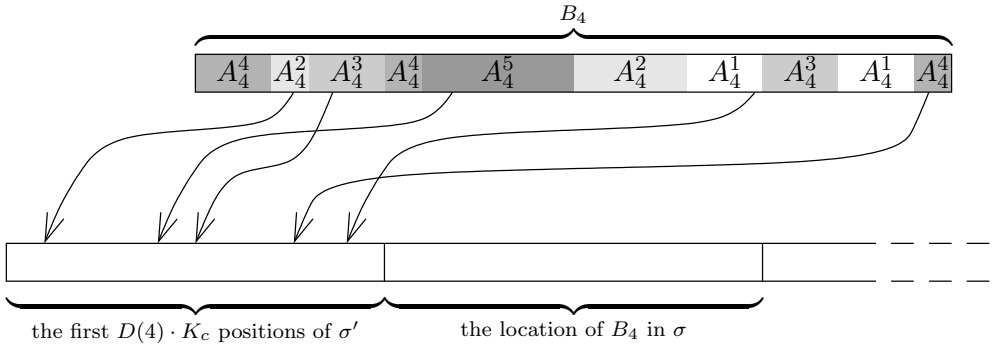
according to Lemma 7.4.6: There are $d^{-2}c\sqrt{n/p}$ elements in A_d^i and $D(d)c\sqrt{n/p}$ positions that precede B_d.

We call a block B_d (for $d \geq 2$) ***c*-successful** if all subsets $A_d^1, \ldots, A_d^{d^4}$ of B_d are c-successful. The probability that B_d is not c-successful is at most $d^4 \cdot \exp(-c^2 d/(8\alpha))$ since there are d^4 subsets $A_d^1, \ldots, A_d^{d^4}$ of B_d. Figure 8.2.1 illustrates c-success.

A subset C_j is called ***c*-successful** if at least one element of C_j is among the first $D' c \sqrt{n/p}$ positions of σ'. The probability that a fixed C_j is not c-successful



(a) Dividing the first $D' \cdot K_c$ elements of σ into blocks $B_1, \ldots, B_{(\log n)^2}$. The subset A_4^1 contains the $K_c/4$ smallest elements of B_4, ..., and A_4^{16} contains the $K_c/4$ largest elements of B_4. (For readability, B_4 is divided into only five subsets in the illustration.)



(b) A subset A_4^i is c-successful if at least one element of A_4^i is among the first $D(4) \cdot K_c$ elements of σ'. The block B_4 is c-successful if all A_4^i are c-successful.

Figure 8.2.1: The division of σ into blocks and subsets (shown here for B_4).

is at most $\exp(-\frac{cD'}{\alpha \log n}) \leq \exp(-\frac{c(\log n)^5}{8\alpha})$. The probability that any C_j is not c-successful is bounded from above by

$$\log n \cdot \sqrt{np} \cdot \exp\left(-\frac{c(\log n)^5}{8\alpha}\right) \leq d'^4 \cdot \exp\left(-\frac{c^2 d'}{8\alpha}\right) \tag{8.2}$$

for sufficiently large n.

Finally, we say that ***σ′* is *c*-successful** if

- all blocks $B_1, B_2, \ldots, B_{(\log n)^2}$ are c-successful and
- all subsets $C_1, \ldots, C_{\log n \sqrt{np}}$ are c-successful.

Let $c \geq 5$. The probability that σ' is not c-successful is at most

$$\begin{aligned} & \sum_{2 \leq d \leq (\log n)^2} d^4 \cdot \exp\bigl(-c^2 d/(8\alpha)\bigr) + \mathbb{P}(\text{some } C_j \text{ is not } c\text{-successful}) \\ \leq & \sum_{2 \leq d \leq (\log n)^2 + 1} d^4 \cdot \exp\bigl(-c^2 d/(8\alpha)\bigr) \;\leq\; \sum_{d \geq 2} \bigl(\exp\bigl(-c^2/(16\alpha)\bigr)\bigr)^d \\ = & \frac{\exp\bigl(-c^2/(16\alpha)\bigr)^2}{1 - \exp\bigl(-c^2/(16\alpha)\bigr)} = E(c, \alpha)\,. \end{aligned} \tag{8.3}$$

The first inequality holds due to Formula 8.2, the second inequality holds since $c \geq 5$. If σ' is not $(\log n)$-successful, which happens with a probability of at most $E(\log n, \alpha) \leq \exp(-(\log n)^2/(16\alpha))$, we bound the height of $T(\sigma')$ by n.

Let $Q_c = \left(c \cdot \frac{\pi^2}{3} + \frac{2}{\log n}\right) \cdot (1 - \eta^{-1}p) \cdot \sqrt{n/p}$.

Lemma 8.2.2. *If σ' is c-successful, then* $\text{height}(\sigma', \overline{\mu}) \leq Q_c$.

Proof. Consider the way in which $T(\sigma')$ is built iteratively from σ'. Let $d \geq 2$. After inserting the first $D(d) \cdot K_c$ elements, the partial tree $\tilde{T}$ grown so far contains at least one element of A_d^i for every $i \in [d^4]$. Except for elements of $\tilde{T}$, there cannot be elements from both B_{j^-} and B_{j^+} for $j^- < i < j^+$ that lie on the same root-to-leaf path of $T(\sigma')$: Let $x \in B_i$ be part of $\tilde{T}$, then all elements of B_{j^-} that are not part of $\tilde{T}$ are to the left of x in $T(\sigma')$, while all elements of B_{j^+} that are not part of $\tilde{T}$ are to the right of x in $T(\sigma')$.

It follows that except for elements of $\tilde{T}$, only elements of two consecutive parts A_d^i and A_d^{i+1} can lie on the same root-to-leaf path of $T(\sigma')$. For every i, there are at most $2 \cdot d^{-2} \cdot K_c$ such elements.

For every d and i, there are at most $(1 - \eta^{-1}p) \cdot d^{-2} \cdot K_c$ unmarked elements in A_d^i since σ' is partially successful. Thus for every d, at most $2 \cdot (1 - \eta^{-1}p) \cdot d^{-2} \cdot K_c$ unmarked elements of B_d are on the same root-to-leaf path in $T(\sigma')$.

Let $\overline{B} = [n] \setminus B$ be the set of elements of σ that are not contained in any A_d^i. There cannot be unmarked elements from both $C_{k^-} \cap \overline{B}$ and $C_{k^+} \cap \overline{B}$ for $k^- < j < k^+$ on the same root-to-leaf path in σ' since there is at least one element of C_j among the first $D' \cdot K_c$ elements of σ'. Thus, there are at most $2 \cdot (1 - \eta^{-1}p) \cdot \frac{\sqrt{n/p}}{\log n}$ unmarked elements of $\overline{B}$ on the same root-to-leaf path in $T(\sigma')$.

The maximum number of unmarked elements on any root-to-leaf path in $T(\sigma')$ is thus at most

$$\begin{aligned} & \sum_{1 \leq d \leq (\log n)^2} 2 \cdot (1 - \eta^{-1}p) \cdot cd^{-2} \cdot \sqrt{n/p} + 2 \cdot (1 - \eta^{-1}p) \cdot (\log n)^{-1} \cdot \sqrt{n/p} \\ \leq & \left(2c \cdot \sum_{d \geq 1} d^{-2} + 2/\log n\right) \cdot (1 - \eta^{-1}p) \cdot \sqrt{n/p} = Q_c \,. \end{aligned}$$

□

According to Lemma 8.2.2 and Formula 8.3, we have $\mathbb{P}\left(\text{height}(\sigma', \overline{\mu}) > Q_c\right) \leq E(c, \alpha)$ for $5 \leq c \leq \log n$. Hence, we can bound the expectation of $\text{height}(\sigma', \overline{\mu})$

from above by

$$
\begin{aligned}
& Q_5 + \sum_{5 \le c \le \log n} Q_{c+1} \cdot \underbrace{\mathbb{P}(\sigma' \text{ is not } c\text{-successful but } (c+1)\text{-successful})}_{\le \mathbb{P}(\sigma' \text{ is not } c\text{-successful})} \\
& + \underbrace{n \cdot (P + E(\log n, \alpha))}_{=X} \\
\le\ & \underbrace{(1 - \eta^{-1} p)}_{\le (1-p) + n^{-1/6} p} \cdot \sqrt{n/p} \cdot \underbrace{\left(5 + \sum_{c=5}^{\infty} \left(\frac{\pi^2}{3}(c+1) + \frac{2}{\log n}\right) \cdot E(c, \alpha)\right)}_{= Y \in O(1)} + X \\
\le\ & \underbrace{(1-p) \cdot \sqrt{n/p}}_{=Z} \cdot Y + \underbrace{n^{2/6} \cdot \sqrt{p} \cdot Y + X}_{\in o(Z)} \\
=\ & Z \cdot \underbrace{\Big(5 + \frac{\pi^2}{3} \cdot \overbrace{\sum_{c \ge 5} (c+1) \cdot E(c, \alpha)}^{< 0.5 \text{ for } \alpha < 1.01}\Big)}_{= C < 6.7 \text{ for } \alpha < 1.01} + o(Z) \le C \cdot (1-p) \cdot \sqrt{n/p}
\end{aligned}
$$

for sufficiently large n and $\alpha < 1.01$. The second inequality holds due to Formula 8.1. The equality holds because $Z \cdot \sum_{c=5}^{\infty} \frac{2E(c,\alpha)}{\log n} \in o(Z)$. This completes the proof. □

The following theorem is obtained via a proof similar to the proof of Theorem 8.2.1.

Theorem 8.2.3. *Let $p \in (0,1)$. Then for all sufficiently large n and all sequences σ of length n (where σ is a permutation of $[n - \frac{1}{2}]$),*

$$\text{height-alter}_p(\sigma) \le 6.7 \cdot (1-p) \cdot \sqrt{n/p}\,.$$

Proof. The main difference between the proof of this theorem and the proof of Theorem 8.2.1 is that we have to use Lemma 7.4.8 instead of Lemma 7.4.6. The blocks B_d and C_j and the subsets A_d^i are defined in the same way. Now for each subset A_i^d we have numbers $a_d^i = \lfloor \min A_i^d \rfloor$ and $b_d^i = \lceil \max A_i^d \rceil$. We say that A_i^d is ***c*-successful** if at least one of the first $D(d) \cdot c \cdot \sqrt{n/p}$ elements is from the interval $[a_d^i, b_d^i)$. The term c-successful for blocks B_d is defined in the same way as in the previous proof. For subsets C_j, the term c-successful is defined just as for A_i^d. The remainder of the proof proceeds along the same lines as the proof of Theorem 8.2.1. □

An upper bound for the height of binary search trees under partial permutation and partial alteration that holds with high probability can be obtained by applying Lemmas 7.4.6 and 7.4.8.

Theorem 8.2.4. *Let $p \in (0,1)$, $\alpha > 1$, $c > 0$, and let $n \in \mathbb{N}$ be sufficiently large. Let σ be a sequence of length n and let σ' be the sequence obtained from σ by performing a p-partial permutation. Then*

$$\mathbb{P}\left(\text{height}(\sigma') > c \cdot \sqrt{(n/p) \cdot \log n}\right) \leq n^{-(c/3)^2/\alpha + 0.5}.$$

The same holds if σ is a permutation of $[n - \frac{1}{2}]$ and σ' is obtained by performing a partial alteration.

Proof. Let $\tilde{c} = c/3$. Let $K_{\tilde{c}} = \tilde{c} \cdot \sqrt{(n/p) \cdot \log n}$. Let B_1 be the set of the $K_{\tilde{c}}$ smallest elements of σ, let B_2 be the set of the $K_{\tilde{c}}$ next smallest elements of σ, ..., and let $B_{n/K_{\tilde{c}}}$ be the set of the $K_{\tilde{c}}$ largest elements of σ.

If at least one element of every B_i is among the first $K_{\tilde{c}}$ elements of σ', then we can bound the height of $T(\sigma')$ as follows.

Lemma 8.2.5. *Assume that for every i, at least one element of B_i is among the first $K_{\tilde{c}}$ elements of σ'.*

Then $\text{height}(\sigma') \leq c \cdot \sqrt{(n/p) \cdot \log n}$.

Proof. Consider the way in which $T(\sigma')$ is built iteratively from σ'. After inserting the first $K_{\tilde{c}}$ elements, the partial tree $\tilde{T}$ grown so far has a height of at most $K_{\tilde{c}}$. The tree $\tilde{T}$ contains at least one element of every B_i. Except for elements of $\tilde{T}$, there cannot be elements from both B_{j^-} and B_{j^+} for $j^- < i < j^+$ that lie on the same root-to-leaf path of $T(\sigma')$: Let $x \in B_i$ be part of $\tilde{T}$, then all elements of B_{j^-} that are not part of $\tilde{T}$ are to the left of x in $T(\sigma')$, while all elements of B_{j^+} that are not part of $\tilde{T}$ are to the right of x in $T(\sigma')$.

It follows that except for elements of $\tilde{T}$, only elements of two consecutive blocks B_i and B_{i+1} can lie on the same root-to-leaf path of $T(\sigma')$. For every i, there are at most $2 \cdot K_{\tilde{c}}$ such elements, yielding a height of at most $2 \cdot K_{\tilde{c}}$. Together with the first $K_{\tilde{c}}$ elements, which build $\tilde{T}$, we obtain $\text{height}(\sigma') \leq 3 \cdot K_{\tilde{c}} = c \cdot \sqrt{(n/p) \cdot \log n}$. □

What remains is to estimate the probability that there is an i such that no element of B_i is among the first $K_{\tilde{c}}$ elements. For every i, the probability that no element of B_i is among the first $K_{\tilde{c}}$ elements in σ' is at most $\exp(-(\tilde{c}^2/\alpha) \cdot \log n) = n^{-\tilde{c}^2/\alpha}$ by Lemma 7.4.6. Thus, the probability that there is any B_i such that no element of B_i is among the first $K_{\tilde{c}}$ elements of σ' is at most

$$(n/K_{\tilde{c}}) \cdot n^{-\tilde{c}^2/\alpha} = \tilde{c}^{-1} \cdot \sqrt{p/\log n} \cdot n^{-\tilde{c}^2/\alpha + 0.5} \leq n^{-\tilde{c}^2/\alpha + 0.5}$$

for sufficiently large n, which completes the proof for partial permutations.

The same bound can be obtained for partial alterations; there are two main differences: We now have to use Lemma 7.4.8, and we have to estimate the probability that for every i, at least one of the first K_c elements is in the interval $[(i-1) \cdot K_c, i \cdot K_c)$. □

The following result follows immediately from the previous theorem.

Corollary 8.2.6. *Let $p \in (0,1)$ and n be sufficiently large. Let σ be a sequence of length n and σ' be the sequence obtained from σ via p-partial permutation. Then*

$$\mathbb{P}\big(\text{height}(\sigma') > 3.7 \cdot \sqrt{(n/p) \cdot \log n}\big) \leq 1/n\,.$$

The same holds if σ is a permutation of $[n - \frac{1}{2}]$ and σ' is obtained via p-partial alteration.

8.2.2 Lower Bounds

Now we turn to lower bounds for the smoothed height. Interestingly, the lower bound is obtained for the sorted sequence, which is not the worst case for the expected number of left-to-right maxima under partial permutation; the expected number of left-to-right maxima of the sequence obtained by partially permuting the sorted sequence σ^n_{sort} is roughly only $O(\log n)$ [10].

Theorem 8.2.7. *For all $p \in (0,1)$ and all sufficiently large $n \in \mathbb{N}$, we have*

$$\text{height-perm}_p(\sigma^n_{\text{sort}}) \geq 0.8 \cdot (1-p) \cdot \sqrt{n/p}\,.$$

Proof. The proof is similar to the proof of Lemma 8.1.4, except that we consider the sorted sequence.

Let again $K_c = c \cdot \sqrt{n/p}$ for $c > 0$. Let σ' be the sequence obtained from σ^n_{sort} via p-partial permutation. We say that σ' is ***c*-successful** if all marked elements among the first K_c elements of σ^n_{sort} are permuted further to the back. According to the proof of Lemma 8.1.4, we have

$$\mathbb{P}(\sigma' \text{ is } c\text{-successful}) \geq \exp(-c^2\alpha)$$

for arbitrarily chosen $\alpha > 1$ and sufficiently large n. If σ' is c-successful, then $\text{height}(\sigma')$ is at least the number of unmarked elements among the first K_c elements. Let $Q = (1-p) \cdot \sqrt{n/p}$ for short. Analogously to Lemma 8.1.4, we obtain

$$\mathbb{P}\left(\text{height}(\sigma') \geq cQ\right) \geq \exp(-c^2\alpha)$$

for sufficiently large n. We compute a lower bound for the expected height of $T(\sigma')$ by considering c-success for all $c \in \{0.1, 0.2, \ldots, 9.9, 10\} = C$. To use more values for c does not make much sense since the changes in the result are negligible. We obtain

$$\begin{aligned}
\mathbb{E}(\text{height}(\sigma')) &\geq Q \cdot \sum_{c \in C} c \cdot \mathbb{P}(cQ \leq \text{height}(\sigma') < (c+0.1) \cdot Q) \\
&\geq Q \cdot \sum_{c \in C} 0.1 \cdot \mathbb{P}(\text{height}(\sigma') \geq cQ) \\
&\geq Q \cdot \underbrace{\sum_{c \in C} 0.1 \cdot \exp(-c^2\alpha)}_{\geq 0.8 \text{ for } \alpha < 1.01} \geq 0.8 \cdot Q
\end{aligned}$$

for sufficiently large n and $\alpha < 1.01$, which proves the theorem. □

We obtain the same lower bound for the height of binary search trees under partial alterations. Again, the lower bound is obtained for the sorted sequence.

Theorem 8.2.8. *For all $p \in (0,1)$ and all sufficiently large $n \in \mathbb{N}$,*

$$\text{height-alter}_p(\sigma^n_{\text{sort}}) \geq 0.8 \cdot (1-p) \cdot \sqrt{n/p}\,.$$

Proof. The proof is almost identical to the proof of Theorem 8.2.7. The only difference is that we have to use Lemma 8.1.6 instead of Lemma 8.1.4. □

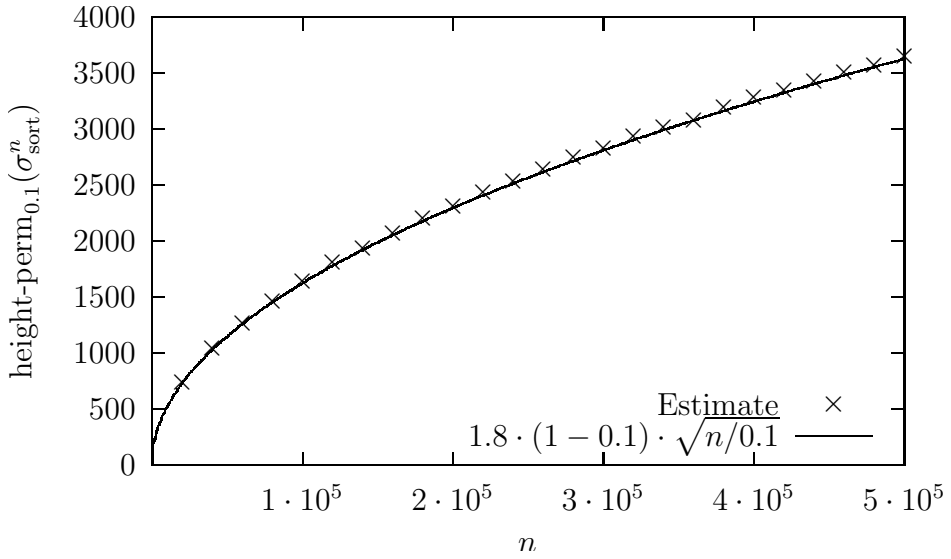
8.3 Experimental Results

We performed experiments to estimate the constants in the bounds for the height of binary search trees.

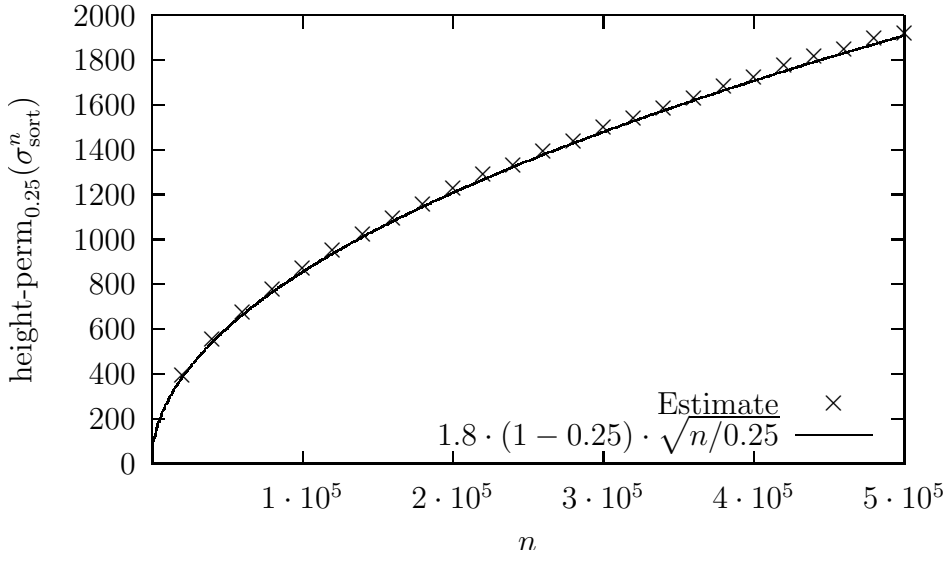
For all $n \in \{20\,000, 40\,000, \ldots, 500\,000\}$ and $p \in \{0.1, 0.25\}$, we randomly performed 5 000 p-partial permutations on the sorted sequence σ^n_{sort}. We then estimated height-perm$_p(\sigma^n_{\text{sort}})$ as the average height of the trees generated by the sequences thus obtained. Figure 8.3.1 shows the results compared to $1.8 \cdot (1-p) \cdot \sqrt{n/p}$.

We performed the same experiment for $n \in \{100\,000, 500\,000\}$ and $p \in \{0.05, 0.10, \ldots, 0.95\}$. Figure 8.3.2 shows the results, again compared to $1.8 \cdot (1-p) \cdot \sqrt{n/p}$.

The experimental results lead us to Conjecture 10.1.2, which states that height-perm$_p(\sigma^n_{\text{sort}})$ is roughly $1.8 \cdot (1-p) \cdot \sqrt{n/p}$. Proving Conjecture 10.1.2 would immediately improve the lower bound of Theorem 8.2.7. Furthermore, it would lead to stronger upper bounds for the smoothed height of binary search trees under partial permutations, provided that Conjecture 10.1.1 holds as well.

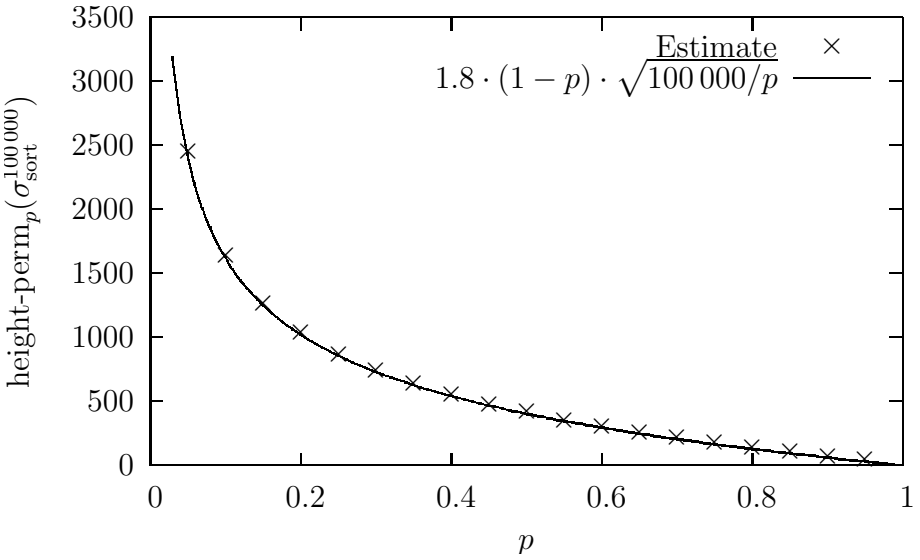


(a) $p = 0.10$

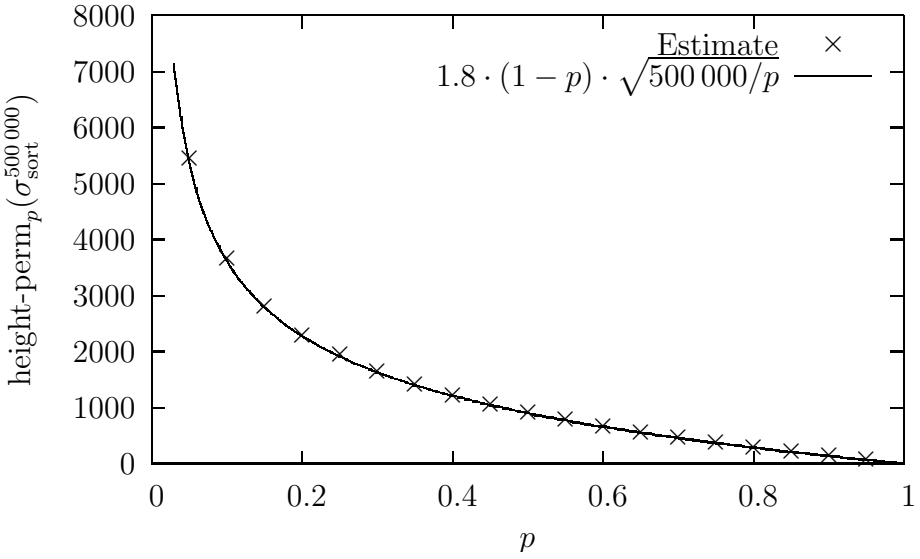


(b) $p = 0.25$

Figure 8.3.1: Experimental data for σ^n_{sort} for $n \in \{20\,000, 40\,000, \ldots, 500\,000\}$ and $p \in \{0.1, 0.25\}$ compared to $1.8 \cdot (1 - 0.25) \cdot \sqrt{n/p}$.



(a) $n = 100\,000$



(b) $n = 500\,000$

Figure 8.3.2: Experimental data for σ_{sort}^n for $n \in \{100\,000, 500\,000\}$ and $p \in \{0.05, 0.10, \ldots, 0.95\}$ compared to $1.8 \cdot (1-p) \cdot \sqrt{n/p}$.

CHAPTER

NINE

Smoothed Analysis and Stability

Smoothed analysis can be viewed as analysing the *fragility* of worst case instances: How much do worst case instances break down under slight perturbations? We suggest examining also the dual property, the *stability* under slight perturbations: Given a good (or best-case) instance, how much can the complexity increase if the instance is perturbed slightly?

We show that all three perturbation models considered are not stable: There are sequences that yield trees of logarithmic height, but slightly perturbing these sequences yields trees of height $n^{\Omega(1)}$.

In the next section, we show how to bound the expected height under partial permutations and partial alterations by the expected height under partial deletions and vice versa. In Section 9.2, we prove that the height of binary search trees under partial deletions is fragile and transfer the results to partial permutations by applying the results of Section 9.1.

9.1 Comparing Partial Deletions with Permutations and Alterations

Partial deletions turn out to be the worst of the three models: Trees are usually expected to be higher under partial deletions than under partial permutations or alterations, even though they contain fewer elements. The expected height under partial deletions yields upper bounds (up to an additional $O(\log n)$ term) for the expected height under partial permutations and alterations. Furthermore, we prove that lower bounds for the expected height under partial deletions yield slightly weaker lower bounds for permutations and alterations. The main advantage of partial deletions over partial permutations and partial alterations is that partial deletions are much easier to analyse.

The following lemma is an immediate consequence of Lemmas 7.4.4, 7.4.5, and 7.4.7, we therefore omit its proof.

Lemma 9.1.1. *For all sequences σ of length n and $p \in [0,1]$,*

$$\begin{aligned} \text{height-perm}_p(\sigma) &\leq \text{height-del}_p(\sigma) + O(\log n) \quad \textit{and} \\ \text{ltrm-perm}_p(\sigma) &\leq \text{ltrm-del}_p(\sigma) + O(\log n)\,. \end{aligned}$$

If σ is a permutation of $[n - \frac{1}{2}]$, then

$$\begin{aligned} \text{height-alter}_p(\sigma) &\leq \text{height-del}_p(\sigma) + O(\log n) \quad \textit{and} \\ \text{ltrm-alter}_p(\sigma) &\leq \text{ltrm-del}_p(\sigma) + O(\log n)\,. \end{aligned}$$

Thus, we can bound the expected height under partial permutations or alterations from above by the expected height under partial deletions. The converse is not true; this follows from the upper bounds for the height of binary search trees under partial permutations and partial alterations (Theorems 8.2.1 and 8.2.3) and the lower bound under partial deletions (Theorem 8.1.8). But we can bound the expected height under partial deletions by the expected height under partial permutations or alterations by padding the sequences considered.

Lemma 9.1.2. *Let $p \in (0,1)$ be fixed and let σ be a sequence of length n with* $\text{height}(\sigma) = d$ *and* $\text{height-del}_p(\sigma) = d'$.

Then there exists a sequence $\tilde{\sigma}$ of length $O(n^2)$ with $\text{height}(\tilde{\sigma}) = d + O(\log n)$ *and* $\text{height-perm}_p(\tilde{\sigma}) \in \Omega(d')$.

Proof. Without loss of generality, we assume that σ is a permutation of $[n]$. The idea is to attach a tail of sufficiently many elements greater than n to the sequence such that all marked elements that are greater than or equal to n will be permuted to this tail. Thus, the overall structure of the remaining elements from $[n]$ will be as if a partial deletion had been carried out.

Choose $K = n^2 p$ and construct $\tilde{\sigma}$ from σ as follows: the first n items of $\tilde{\sigma}$ are just σ; we call this the **head** of $\tilde{\sigma}$. The last $K - n$ items of $\tilde{\sigma}$, which we call the **tail** of $\tilde{\sigma}$, are numbers greater than n such that these numbers build a tree of height $O(\log(K - n)) = O(\log n)$. With a constant probability of, say, c, all elements marked in the head are permuted into the tail (see the proof of Lemma 8.1.4).

Consider the tree obtained from the first n elements after partial permutation under the assumption that all marked head elements are now in the tail. This tree is almost identical to the tree obtained from σ via partial deletion when the same elements are marked. The only difference is that the tree contains some elements greater than n, which only increase the length of the right-most path. Thus, $\text{height-perm}_p(\tilde{\sigma})$ is at least cd', which proves the lemma. □

The following is the analogue of the above lemma for partial alterations. Since its proof is similar to the proof of the previous lemma (the only difference is that we have to use the proof of Lemma 8.1.6 instead of Lemma 8.1.4), we omit it.

Lemma 9.1.3. *Let $p \in (0,1)$ be fixed and let σ be a sequence of length n with elements from $[n - \frac{1}{2}]$. Let $d = \text{height}(\sigma)$ and $d' = \text{height-del}_p(\sigma)$.*

Then there exists a sequence $\tilde{\sigma}$ of length $O(n^2)$ with $\text{height}(\tilde{\sigma}) = d + O(\log n)$ and $\text{height-alter}_p(\tilde{\sigma}) \in \Omega(d')$.

9.2 The (In-)Stability of Perturbations

Having shown in the previous chapter that worst case instances become much better by smoothing, we now provide a family of best-case instances for which smoothing results in an exponential increase in height.

We consider the following family of sequences:

- $\sigma^{(1)} = (1)$.
- $\sigma^{(k+1)} = (2^k, \sigma^{(k)}, 2^k + \sigma^{(k)})$, where $c + \sigma = (c+\sigma_1, \ldots, c+\sigma_n)$ for a sequence σ of length n.

For instance, $\sigma^{(2)} = (2,1,3)$ and $\sigma^{(3)} = (4,2,1,3,6,5,7)$. Let $n = 2^k - 1$. Then $\sigma^{(k)}$ contains the numbers $1, 2, \ldots, n$, and we have

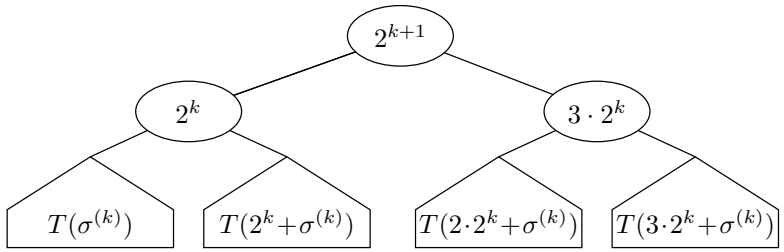
$$\text{height}(\sigma^{(k)}) = \text{ltrm}(\sigma^{(k)}) = k \in \Theta(\log n)\,.$$

Let us estimate the expected number of left-to-right maxima after partial deletion, thus obtaining a lower bound for the expected height of the binary search tree. Deleting the first element of $\sigma^{(k)}$ roughly doubles the number of left-to-right maxima in the resulting sequence. This is the basic idea behind the following theorem; the idea is illustrated in Figure 9.2.1.

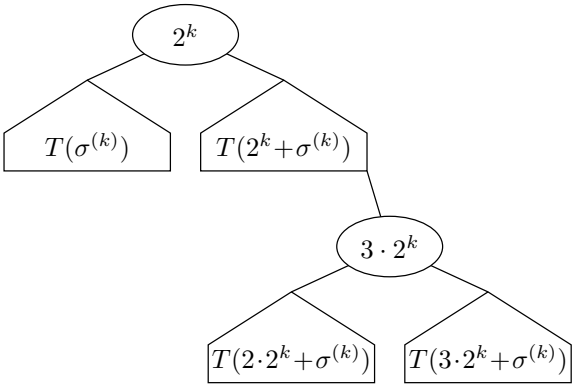
Theorem 9.2.1. *Let $p \in (0,1)$. Then for all $k \in \mathbb{N}$,*

$$\text{ltrm-del}_p(\sigma^{(k)}) = \frac{1-p}{p} \cdot \big((1+p)^k - 1\big)\,.$$

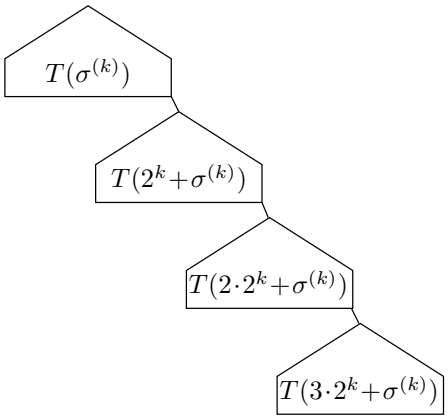
Proof. Let $\ell(k) = \text{ltrm-del}_p(\sigma^{(k)})$ for short. The root 2^{k-1} is deleted with probability p. Then the expected number of left-to-right maxima is just the expectation for the left subtree plus the expectation for the right subtree since all elements in the left subtree are smaller and occur earlier than all elements in the right subtree. Both expectations are $\ell(k-1)$. If the root is not deleted, we expect $1 + \ell(k-1)$ left-to-right maxima: One is the root and $\ell(k-1)$ are expected in the right subtree. The left subtree does not contribute any other maxima since all elements in the left subtree are smaller than the root. We have $\ell(1) = 1 - p$



(a) $T(\sigma^{(k+2)})$.



(b) Removing the root 2^{k+1} roughly doubles the height.



(c) Additionally removing the roots 2^k of $T(\sigma^{(k+1)})$ and $3 \cdot 2^k$ of $T(2^{k+1} + \sigma^{(k+1)})$ increases the height by a factor of four.

Figure 9.2.1: Removing root elements increases the height and the number of left-to-right maxima.

since the single element will be deleted with probability p. Overall, we have

$$\begin{aligned}\ell(k) &= p \cdot 2 \cdot \ell(k-1) + (1-p) \cdot (1 + \ell(k-1)) \\ &= (1+p) \cdot \ell(k-1) + (1-p) = (1-p) \cdot \sum_{i=0}^{k-1} (1+p)^i \\ &= \frac{1-p}{p} \cdot \left((1+p)^k - 1\right).\end{aligned}$$

□

Corollary 9.2.2. *For all $p \in (0,1)$ and all $k \in \mathbb{N}$,*

$$\text{height-del}_p(\sigma^{(k)}) \geq \frac{1-p}{p} \cdot \left((1+p)^k - 1\right).$$

We conclude that there are some best case instances that are quite fragile under partial deletions: From logarithmic height they "jump" via smoothing to

a height of $\Omega(n^{\log(1+p)})$. (We have $\frac{1-p}{p} \cdot ((1+p)^k - 1) \in \Theta(n^{\log(1+p)})$.) Thus, the height increases exponentially.

We can transfer this result to partial permutations and partial alterations due to Lemmas 9.1.2 and 9.1.3. Therefore, we consider sequences $\tilde{\sigma}^{(k)}$ which are obtained from $\sigma^{(k)}$ as described in the proof of Lemma 9.1.2.

Corollary 9.2.3. *Let $p \in (0,1)$ be fixed. Then*

$$\begin{array}{lcl} \text{height}(\tilde{\sigma}^{(k)}) & \in & O(\log n)\ , \\ \text{height-perm}_p(\tilde{\sigma}^{(k)}) & \in & \Omega\big(n^\epsilon\big)\ , \textit{ and} \\ \text{height-alter}_p(\tilde{\sigma}^{(k)}) & \in & \Omega\big(n^\epsilon\big) \end{array}$$

for some fixed $\epsilon > 0$.

For the sake of completeness, let us mention that the number of left-to-right-maxima is maximally fragile, at least asymptotically for any fixed p: There are sequences with one left-to-right maximum for which the expected number of left-to-right maxima after partial permutation is $\Omega(\sqrt{n})$. The same holds for partial alterations. For partial deletions, the number can jump from 1 to $\Omega(n)$. The proofs are straightforward: Take an adversarial sequence of length $n-1$ for proving lower bounds for the expected number of left-to-right maxima under any of these perturbation models and add an n at the front of the sequence. For partial permutations, this n will be marked with constant probability and moved behind the first $\Theta(\sqrt{n/p})$ elements. For the other two models, the proof is similar.

CHAPTER

TEN

Concluding Remarks

We conclude the second part of the thesis with some conjectures regarding binary search trees and prospects for further research in the area of smoothed analysis of discrete problems.

10.1 Conjectures

We have analysed the height of binary search trees obtained from perturbed sequences and obtained asymptotically tight lower and upper bounds of roughly $\Theta(\sqrt{n})$ for the height under partial permutations and alterations. This stands in contrast to both the worst-case and the average-case height of n and $\Theta(\log n)$, respectively. Thus, the height of binary search trees under limited randomness differs significantly from both the average and the worst case. One direction for future work is of course improving the constants of the bounds.

Interestingly, the sorted sequence σ^n_{sort} turns out to be a worst case for the smoothed height of binary search trees in the sense that the lower bounds are obtained for σ^n_{sort} (Theorems 8.2.7 and 8.2.8). This is in contrast to the fact that the expected number of left-to-right maxima of σ^n_{sort} under p-partial permutations is roughly $O(\log n)$ [10]. We believe that for the height of binary search trees, σ^n_{sort} is indeed the worst case.

Conjecture 10.1.1. *For all $p \in [0,1]$, all $n \in \mathbb{N}$, and every sequence σ of length n,*

$$\begin{aligned} \text{height-perm}_p(\sigma) &\leq \text{height-perm}_p(\sigma^n_{\text{sort}}) \text{ and} \\ \text{height-alter}_p(\sigma) &\leq \text{height-alter}_p(\sigma^n_{\text{sort}})\,. \end{aligned}$$

We have performed experiments to estimate the constants in the bounds for the height of binary search trees. For all $n \in \{20\,000, 40\,000, \ldots, 500\,000\}$ and $p \in \{0.1, 0.25\}$, we performed 5 000 partial permutations of σ^n_{sort}. We did the

same thing for $n \in \{100\,000, 500\,000\}$ and $p \in \{0.05, 0.10, \ldots, 0.95\}$. (See Section 8.3 for more details.) The results led to the following conjecture. Proving this conjecture would immediately improve our lower bound. Provided that Conjecture 10.1.1 holds as well, we would also obtain an improved upper bound for the height of binary search trees under partial permutations.

Conjecture 10.1.2. *For $p \in (0, 1)$ and sufficiently large n,*

$$\text{height-perm}_p(\sigma^n_{\text{sort}}) = (\gamma + o(1)) \cdot (1 - p) \cdot \sqrt{n/p}$$

for some constant $\gamma \approx 1.8$.

Throughout this work, the bounds obtained for partial permutations and partial alterations are equal. Moreover, the proofs used to obtain these bounds are almost identical. We suspect that this is always true for binary search trees.

Conjecture 10.1.3. *For all $p \in [0, 1]$ and σ,*

$$\text{height-perm}_p(\sigma) \approx \text{height-alter}_p(\sigma)\,.$$

10.2 Smoothed Analysis of Discrete Problems

In addition to partial permutations and alterations, one could consider other perturbation models for sequences. From a more abstract point of view, a future research direction would be to characterise the properties of perturbation models that lead to upper or lower bounds that are asymptotically different from the average or worst case.

Apart from lower and upper bounds, we have also examined the stability of perturbations, i.e. how much higher a tree can become if the underlying sequence is perturbed. It turns out that all three perturbation models are unstable.

Finally, we are interested in generalising these results to other problems based on permutations, like sorting algorithms (Quicksort under partial permutations has already been investigated by Banderier et al. [10]), routing algorithms, and other algorithms and data structures. Hopefully, this will shed some light on the discrepancy between the worst-case and average-case complexity of these problems.

APPENDIX

A

Technical Lemmas

A.1 L-Reductions imply AP-Reductions

Lemma A.1.1. *Let* Π *and* Π' *be two optimisation problems with* $\Pi \in \mathsf{APX}$. *If* $\Pi \leq_{\mathrm{L}} \Pi'$, *then* $\Pi \leq_{\mathrm{AP}} \Pi'$.

Proof. Since $\Pi = (I, \mathrm{sol}, m, \mathrm{goal}) \leq_{\mathrm{L}} \Pi' = (I', \mathrm{sol}', m', \mathrm{goal}')$, there exist two functions f_{L} and g_{L} and constants α_{L} and β_{L} as described in Definition 2.4.9.

Let us first assume that Π is a minimisation problem, thus $m^\star(x) \leq m(x,y)$ for all $x \in I$ and $y \in \mathrm{sol}(x)$. For the AP-reduction, we choose $f_{\mathrm{AP}}(x,r) = f_{\mathrm{L}}(x)$ and $g_{\mathrm{AP}}(x,y',r) = g_{\mathrm{L}}(x,y')$. What remains to be proved is that there exists some constant $\alpha_{\mathrm{AP}} \geq 1$ such that $R'(x',y') \leq r$ implies $R(x,y) \leq 1 + \alpha_{\mathrm{AP}} \cdot (r-1)$ for all $r > 1$. Let $R'(x',y') \leq r$ and $\alpha_{\mathrm{AP}} = \alpha_{\mathrm{L}}\beta_{\mathrm{L}}$, then

$$\begin{aligned} R(x,y) &= \frac{m(x,y)}{m^\star(x)} = \frac{m(x,y) - m^\star(x)}{m^\star(x)} + 1 \leq \frac{\beta_{\mathrm{L}} \cdot |m'(x',y') - m'^\star(x')|}{\alpha_{\mathrm{L}}^{-1} \cdot m'^\star(x')} + 1 \\ &\leq \alpha_{\mathrm{L}}\beta_{\mathrm{L}} \cdot \frac{\max\{m'^\star(x'), m'(x',y')\} - \min\{m'^\star(x'), m'(x',y')\}}{\min\{m'^\star(x'), m'(x',y')\}} + 1 \\ &\leq \alpha_{\mathrm{AP}} \cdot (R'(x',y') - 1) + 1 \leq \alpha_{\mathrm{AP}} \cdot (r-1) + 1. \end{aligned}$$

Now assume that Π is a maximisation problem. For this case, we have to exploit $\Pi \in \mathsf{APX}$, i.e. there exists a factor γ approximation algorithm for Π for some $\gamma \geq 1$. Let h be the function computed by this approximation algorithm. Then $R(x, h(x)) \leq \gamma$. We choose f_{AP} as above. The function g_{AP} selects the better of the two solutions $h(x)$ and $g_{\mathrm{L}}(x,y')$:

$$y = g_{\mathrm{AP}}(x,y') = \begin{cases} h(x) & \text{if } m(x,h(x)) \geq m(x, g_{\mathrm{L}}(x,y')) \text{ and} \\ g_{\mathrm{L}}(x,y') & \text{otherwise.} \end{cases}$$

Let $R'(x',y') \leq r$ and $\alpha_{\mathrm{AP}} = \alpha_{\mathrm{L}}\beta_{\mathrm{L}}\gamma$, then

$$R(x,y) = \frac{m^\star(x) - m(x,y)}{m(x,y)} + 1 \leq \frac{m^\star(x) - m(x,y)}{\gamma^{-1} \cdot m^\star(x)} \leq \alpha_{\mathrm{AP}} \cdot (r-1) + 1\,,$$

which proves the lemma. □

A.2 Chernoff Bounds

Let $p \in (0,1)$ and let $X_1, X_2, \ldots, X_k$ be mutually independent random variables with $\mathbb{P}(X_i = 1) = 1 - \mathbb{P}(X_i = 0) = p$ and $X = \sum_{i=1}^{k} X_i$. Clearly, $\mathbb{E}(X) = pk$. The probability that X deviates by more than a from its expectation is bounded from above by

$$\mathbb{P}(|X - pk| > a) < 2 \cdot \exp\left(-\frac{2a^2}{k}\right) \tag{A.1}$$

according to Alon et al. [5, Corollary A.7].

Let us now prove Lemma 6.3.1.

Lemma 6.3.1. *Let $k \in \mathbb{N}$, $\alpha > 1$ and $p \in [0,1]$. Assume that we have mutually independent random variables $X_1, \ldots, X_k$ that assume values in $\{0,1\}$. Assume further that $\mathbb{P}(X_i = 1) = p = 1 - \mathbb{P}(X_i = 0)$ for all $i \in [k]$. Let $X = \sum_{i=1}^{k} X_i$. Then*

$$\mathbb{P}\left((X > \alpha pk) \vee (X < \alpha^{-1} pk)\right) \leq 2 \cdot \exp\left(-2(1-\alpha^{-1})^2 p^2 k\right) .$$

Proof. Since $\alpha - 1 \geq 1 - \alpha^{-1}$ for all $\alpha \in \mathbb{R}$, we apply Formula A.1 with $a = (1-\alpha^{-1}) \cdot pk$ and obtain

$$\begin{aligned} \mathbb{P}\left((X > \alpha pk) \vee (X < \alpha^{-1} pk)\right) &\leq \mathbb{P}\left(|X - pk| > (1-\alpha^{-1})pk\right) \\ &< 2 \cdot \exp\left(-\frac{2(1-\alpha^{-1})^2 p^2 k^2}{k}\right) \\ &= 2 \cdot \exp\left(-2(1-\alpha^{-1})^2 p^2 k\right) . \end{aligned}$$

□

A.3 Adjusting Probabilities

Lemma A.3.1. *Let $n, E \in \mathbb{N}$ with $E \leq n$. Let $p_1, \ldots, p_n$ be rational numbers with $0 \leq p_i \leq 1$ for all $i \in [n]$ such that $\sum_{i=1}^{n} p_i = E$.*

Let $X \subseteq [n]$ be a random set. Then there exists a probability distribution on $\mathcal{P}([n])$ with the following properties:

1. *For all $i \in [n]$, $\mathbb{P}(i \in X) = p_i$.*
2. *$\mathbb{P}(|X| = E) = 1$.*

Proof. Since all p_i are rational, there exist $M, P_1, \ldots, P_n \in \mathbb{N}$ with $p_i = P_i/M$ for all $i \in [n]$. Consider a matrix $\mu = (\mu_{i,j})_{i \in [n], j \in [M]}$ with $\mu_{i,j} \in \{0,1\}$ and the following properties:

1. For all $i \in [n]$, $\sum_{j=1}^{M} \mu_{i,j} = P_i$, i.e. the row sum of row i is P_i.
2. For all $j \in [M]$, $\sum_{i=1}^{n} \mu_{i,j} = E$, i.e. the column sum of column j is E.

Then μ corresponds to a distribution as claimed: Choose a $j \in [M]$ uniformly at random and set $X = \{i \mid \mu_{i,j} = 1\}$. Then $\mathbb{P}(i \in X) = \frac{|\{j|\mu_{i,j}=1\}|}{M} = \frac{P_i}{M} = p_i$ and $|X| = \sum_{i=1}^{n} \mu_{i,j} = E$.

Let us quickly check that the sum of the column sums equals the sum of the row sums, which is a necessary condition for such a matrix to exist: The sum of the row sums is $\sum_{i=1}^{n} P_i = M \cdot \sum_{i=1}^{n} p_i = ME$. Each column sum is E and there are M columns, hence the sum of the column sums is ME.

We construct the matrix μ as follows: For all $i \in [n]$, set $\mu_{i,1} = \mu_{i,2} = \ldots = \mu_{i,P_i} = 1$ and $\mu_{i,P_i+1} = \ldots, \mu_{i,M} = 0$. Thus, the row sum of row i is P_i, which is as claimed. Let us now iteratively modify the matrix such that each column sum becomes E and the row sums are left unchanged.

If μ already fulfils the second property, i.e. if every column sum is E, we are done. Otherwise, there exist two columns j^+ and j^- with $\sum_{i=1}^{n} \mu_{i,j^+} > E > \sum_{i=1}^{n} \mu_{i,j^-}$. Hence, there is an $i^\star \in [n]$ with $\mu_{i^\star,j^+} = 1$ and $\mu_{i^\star,j^-} = 0$. We modify μ by setting $\mu_{i^\star,j^+} = 0$ and $\mu_{i^\star,j^-} = 1$. This does not change the row sums. By iterating this, we obtain a matrix μ as claimed. □

BIBLIOGRAPHY

[1] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[2] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, 1983.

[3] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[4] Paola Alimonti and Viggo Kann. Some APX-completeness results for cubic graphs. *Theoretical Computer Science*, 237(1–2):123–134, 2000.

[5] Noga Alon, Joel H. Spencer, and Paul Erdős. *The Probabilistic Method*. John Wiley & Sons, 1992.

[6] Sanjeev Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM*, 45(5):753–782, 1998.

[7] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998.

[8] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM*, 45(1):70–122, 1998.

[9] Giorgio Ausiello, Pierluigi Crescenzi, Giorgio Gambosi, Viggo Kann, Alberto Marchetti-Spaccamela, and Marco Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer, 1999.

[10] Cyril Banderier, René Beier, and Kurt Mehlhorn. Smoothed analysis of three combinatorial problems. In Branislav Rovan and Peter Vojtás, editors, *Proc. of the 28th Int. Symp. on Mathematical Foundations of Computer Science*

(MFCS), volume 2747 of *Lecture Notes in Computer Science*, pages 198–207. Springer, 2003.

[11] Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, Guido Schäfer, and Tjark Vredeveld. Average case and smoothed competitive analysis of the multi-level feedback algorithm. In *Proc. of the 44th Ann. IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 462–471. IEEE Computer Society, 2003.

[12] René Beier and Berthold Vöcking. Typical properties of winners and losers in discrete optimization. In *Proc. of the 36th Ann. ACM Symp. on Theory of Computing (STOC)*, pages 343–352. ACM Press, 2004.

[13] Markus Bläser. Approximationsalgorithmen für Graphüberdeckungsprobleme. Habilitationsschrift, Institut für Theoretische Informatik, Technisch-Naturwissenschaftliche Fakultät, Universität zu Lübeck, Lübeck, Germany, 2002.

[14] Markus Bläser. A 3/4-approximation algorithm for maximum ATSP with weights zero and one. In Klaus Jansen, Sanjeev Khanna, José D. P. Rolim, and Dana Ron, editors, *Proc. of the 7th Int. Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, volume 3122 of *Lecture Notes in Computer Science*, pages 61–71. Springer, 2004.

[15] Markus Bläser and Bodo Manthey. Two approximation algorithms for 3-cycle covers. In Klaus Jansen, Stefano Leonardi, and Vijay Vazirani, editors, *Proc. of the 5th Int. Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, volume 2462 of *Lecture Notes in Computer Science*, pages 40–50. Springer, 2002.

[16] Markus Bläser and Bodo Manthey. Approximating maximum weight cycle covers in directed graphs with weights zero and one. *Algorithmica*, 42(2):121–139, 2005.

[17] Markus Bläser, Bodo Manthey, and Jiří Sgall. An improved approximation algorithm for the asymmetric TSP with strengthened triangle inequality. *Journal of Discrete Algorithms*, to appear.

[18] Markus Bläser and L. Shankar Ram. An improved approximation algorithm for TSP with distances one and two. In Maciej Liśkiewicz and Rüdiger Reischuk, editors, *Proc. of the 15th Int. Symp. on Fundamentals of Computation Theory (FCT)*, volume 3623 of *Lecture Notes in Computer Science*, pages 504–515. Springer, 2005.

[19] Markus Bläser, L. Shankar Ram, and Maxim I. Sviridenko. Improved approximation algorithms for metric maximum ATSP and maximum 3-cycle

cover problems. In Frank Dehne, Alejandro López-Ortiz, and Jörg-Rüdiger Sack, editors, *Proc. of the 9th Workshop on Algorithms and Data Structures (WADS)*, volume 3608 of *Lecture Notes in Computer Science*, pages 350–359. Springer, 2005.

[20] Markus Bläser and Bodo Siebert. Computing cycle covers without short cycles. In Friedhelm Meyer auf der Heide, editor, *Proc. of the 9th Ann. European Symp. on Algorithms (ESA)*, volume 2161 of *Lecture Notes in Computer Science*, pages 368–379. Springer, 2001. Bodo Siebert is the birth name of Bodo Manthey.

[21] Avrim Blum and Joel Spencer. Coloring random and semi-random k-colorable graphs. *Journal of Algorithms*, 19(2):204–234, 1995.

[22] Avrim L. Blum and John D. Dunagan. Smoothed analysis of the perceptron algorithm for linear programming. In *Proc. of the 13th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 905–914. SIAM, 2002.

[23] Hans-Joachim Böckenhauer, Juraj Hromkovic, Ralf Klasing, Sebastian Seibert, and Walter Unger. Approximation algorithms for the TSP with sharpened triangle inequality. *Information Processing Letters*, 75(3):133–138, 2000.

[24] L. Sunil Chandran and L. Shankar Ram. Approximations for ATSP with parameterized triangle inequality. In Helmut Alt and Afonso Ferreira, editors, *Proc. of the 19th Int. Symp. on Theoretical Aspects of Computer Science (STACS)*, volume 2285 of *Lecture Notes in Computer Science*, pages 227–237. Springer, 2002.

[25] Zhi-Zhong Chen and Takayuki Nagoya. Improved approximation algorithms for metric Max TSP. In Gerth Stølting Brodal and Stefano Leonardi, editors, *Proc. of the 13th Ann. European Symp. on Algorithms (ESA)*, volume 3669 of *Lecture Notes in Computer Science*, pages 179–190. Springer, 2005.

[26] Zhi-Zhong Chen, Yuusuke Okamoto, and Lusheng Wang. Improved deterministic approximation algorithms for Max TSP. *Information Processing Letters*, 95(2):333–342, 2005.

[27] Nicos Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, 1976.

[28] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990.

[29] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001.

[30] Gérard P. Cornuéjols and William R. Pulleyblank. A matching problem with side conditions. *Discrete Mathematics*, 29(2):135–159, 1980.

[31] Pierluigi Crescenzi. A short guide to approximation preserving reductions. In *Proc. of the 12th Ann. IEEE Conf. on Computational Complexity (CCC)*, pages 262–273. IEEE Computer Society, 1997.

[32] Pierluigi Crescenzi, Viggo Kann, Riccardo Silvestri, and Luca Trevisan. Structure in approximation classes. *SIAM Journal on Computing*, 28(5):1759–1782, 1999.

[33] William H. Cunningham. Matching, matroids, and extensions. *Mathematical Programming, Series B*, 91(3):515–542, 2002.

[34] William H. Cunningham and Yaoguang Wang. Restricted 2-factor polytopes. *Mathematical Programming, Series A*, 87(1):87–111, 2000.

[35] Valentina Damerow, Friedhelm Meyer auf der Heide, Harald Räcke, Christian Scheideler, and Christian Sohler. Smoothed motion complexity. In Giuseppe Di Battista and Uri Zwick, editors, *Proc. of the 11th Ann. European Symp. on Algorithms (ESA)*, volume 2832 of *Lecture Notes in Computer Science*, pages 161–171. Springer, 2003.

[36] Valentina Damerow and Christian Sohler. Extreme points under random noise. In Susanne Albers and Tomasz Radzik, editors, *Proc. of the 12th Ann. European Symp. on Algorithms (ESA)*, volume 3221 of *Lecture Notes in Computer Science*, pages 264–274. Springer, 2004.

[37] Luc Devroye. A note on the height of binary search trees. *Journal of the ACM*, 33(3):489–498, 1986.

[38] Luc Devroye and Bruce Reed. On the variance of the height of random binary search trees. *SIAM Journal on Computing*, 24(6):1157–1162, 1995.

[39] Michael Drmota. An analytic approach to the height of binary search trees. *Algorithmica*, 29(1–2):89–119, 2001.

[40] Michael Drmota. An analytic approach to the height of binary search trees II. *Journal of the ACM*, 50(3):333–374, 2003.

[41] John D. Dunagan, Daniel A. Spielman, and Shang-Hua Teng. Smoothed analysis of Renegar's condition number for linear programming. arXiv, cs.DS/0302011 v2, 2003. `http://arxiv.org/abs/cs.DS/0302011v2`.

[42] Jack Edmonds. Maximum matching and a polyhedron with 0,1-vertices. *Journal of Research of the National Bureau of Standards*, 69B(1–2):125–130, 1965.

[43] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[44] Uriel Feige and Joe Kilian. Heuristics for semirandom graph problems. *Journal of Computer and System Sciences*, 63(4):639–671, 2001.

[45] Abraham D. Flaxman and Alan M. Frieze. The diameter of randomly perturbed digraphs and some applications. In Klaus Jansen, Sanjeev Khanna, José D. P. Rolim, and Dana Ron, editors, *Proc. of the 8th Int. Workshop on Randomization and Computation (RANDOM)*, volume 3122 of *Lecture Notes in Computer Science*, pages 345–356. Springer, 2004.

[46] Dimitris A. Fotakis and Paul G. Spirakis. A Hamiltonian approach to the assignment of non-reusable frequencies. In Vikraman Arvind and R. Ramanujam, editors, *Proc. of the 18th Conf. on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 1530 of *Lecture Notes in Computer Science*, pages 18–29. Springer, 1998.

[47] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[48] Michael R. Garey, David S. Johnson, and Larry K. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976.

[49] Paul C. Gilmore, Eugene L. Lawler, and David B. Shmoys. Well-solved special cases. In Lawler et al. [63], pages 87–143.

[50] Gregory Gutin and Abraham P. Punnen, editors. *The Traveling Salesman Problem and its Variations*. Kluwer Academic Publishers, 2002.

[51] David Hartvigsen. *An Extension of Matching Theory*. PhD thesis, Department of Mathematics, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, September 1984.

[52] David Hartvigsen. The square-free 2-factor problem in bipartite graphs. In Gérard Cornuéjols, Rainer E. Burkhard, and Gerhard J. Woeginger, editors, *Proc. of the 7th Int. Conf. on Integer Programming and Combinatorial Optimization (IPCO)*, volume 1610 of *Lecture Notes in Computer Science*, pages 234–241. Springer, 1999.

[53] Refael Hassin and Shlomi Rubinstein. An approximation algorithm for maximum triangle packing. In Susanne Albers and Tomasz Radzik, editors, *Proc. of the 12th Ann. European Symp. on Algorithms (ESA)*, volume 3221 of *Lecture Notes in Computer Science*, pages 403–413. Springer, 2004.

[54] Refael Hassin and Shlomi Rubinstein. On the complexity of the k-customer vehicle routing problem. *Operations Research Letters*, 33:1, 71-76 2005.

[55] Pavol Hell. Graph packings. *Electronic Notes in Discrete Mathematics*, 5:170–173, 2000.

[56] Pavol Hell, David G. Kirkpatrick, Jan Kratochvíl, and Igor Kríz. On restricted two-factors. *SIAM Journal on Discrete Mathematics*, 1(4):472–484, 1988.

[57] Dieter Jungnickel. *Graphs, Networks and Algorithms*, volume 5 of *Algorithms and Computation in Mathematics*. Springer, 2nd edition, 2005.

[58] Haim Kaplan, Moshe Lewenstein, Nira Shafrir, and Maxim Sviridenko. Approximation algorithms for asymmetric TSP by decomposing directed regular multigraphs. *Journal of the ACM*, 52(4):602–626, 2005.

[59] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proc. of a Symp. on the Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[60] David G. Kirkpatrick and Pavol Hell. On the complexity of general graph factor problems. *SIAM Journal on Computing*, 12(3):601–609, 1983.

[61] Donald E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, 2nd edition, 1998.

[62] Harold W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[63] Eugene L. Lawler, Jan Karel Lenstra, Alexander H. G. Rinnooy Kan, and David B. Shmoys, editors. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. John Wiley & Sons, 1985.

[64] László Lovász and Michael D. Plummer. *Matching Theory*, volume 121 of *North-Holland Mathematics Studies*. Elsevier, 1986.

[65] Bodo Manthey. On approximating restricted cycle covers. In *Proc. of the 3rd Workshop on Approximation and Online Algorithms (WAOA 2005)*, Lecture Notes in Computer Science. Springer, to appear.

[66] Bodo Manthey and Rüdger Reischuk. Smoothed analysis of binary search trees. In Xiaotie Deng and Dingzhu Du, editors, *Proc. of the 16th Ann. Int. Symp. on Algorithms and Computation (ISAAC)*, volume 3827 of *Lecture Notes in Computer Science*, pages 483–492. Springer, 2005.

[67] Marcin Mucha and Piotr Sankowski. Maximum matchings via Gaussian elimination. In *Proc. of the 45th Ann. IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 248–255. IEEE Computer Society, 2004.

[68] Christos H. Papadimitriou. The Euclidean traveling salesman problem is NP-complete. *Theoretical Computer Science*, 4(3):237–244, 1977.

[69] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[70] Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43(3):425–440, 1991.

[71] Christos H. Papadimitriou and Mihalis Yannakakis. The traveling salesman problem with distances one and two. *Mathematics of Operations Research*, 18(1):1–11, 1993.

[72] Boris Pittel. On growing random binary trees. *Journal of Mathematical Analysis and Applications*, 103(2):461–480, 1984.

[73] Abraham P. Punnen. The traveling salesman problem: Applications, formulations and variations. In Gutin and Punnen [50], pages 585–607.

[74] Bruce Reed. The height of a random binary search tree. *Journal of the ACM*, 50(3):306–332, 2003.

[75] John Michael Robson. The height of binary search trees. *The Australian Computer Journal*, 11(4):151–153, 1979.

[76] John Michael Robson. The asymptotic behaviour of the height of binary search trees. Technical Report TR-CS-81-15, The Australian National University, Department of Computer Science, Canberra, 1981.

[77] John Michael Robson. On the concentration of the height of binary search trees. In Pierpaolo Degano, Roberto Gorrieri, and Alberto Marchetti-Spaccamela, editors, *Proc. of the 24th Int. Coll. on Automata, Languages and Programming (ICALP)*, volume 1256 of *Lecture Notes in Computer Science*, pages 441–448. Springer, 1997.

[78] John Michael Robson. Constant bounds on the moments of the height of binary search trees. *Theoretical Computer Science*, 276(1–2):435–444, 2002.

[79] Heiko Röglin and Berthold Vöcking. Smoothed analysis of integer programming. In Michael Jünger and Volker Kaibel, editors, *Proc. of the 11th Int. Conf. on Integer Programming and Combinatorial Optimization (IPCO)*, volume 3509 of *Lecture Notes in Computer Science*, pages 276–290. Springer, 2005.

[80] Mark Phillip Russell. Restricted two-factors. Master's thesis, University of Waterloo, Waterloo, Ontario, Canada, 2001.

[81] Sartaj Sahni and Teofilo Gonzalez. P-complete approximation problems. *Journal of the ACM*, 23(3):555–565, 1976.

[82] Miklos Santha and Umesh V. Vazirani. Generating quasi-random sequences from semi-random sources. *Journal of Computer and System Sciences*, 33(1):75–87, 1986.

[83] Guido Schäfer. A note on the smoothed complexity of the single-source shortest path problem. Technical Report MPI-I-2003-1-018, Max-Planck-Institut für Informatik, Saarbrücken, October 2003.

[84] Guido Schäfer and Naveen Sivadasan. Topology matters: Smoothed competitiveness of metrical task systems. *Theoretical Computer Science*, 241(1–3):216–246, 2005.

[85] Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer, 2003.

[86] Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. In *Proc. of the 33rd Ann. ACM Symp. on Theory of Computing (STOC)*, pages 296–305. ACM Press, 2001.

[87] Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis: Motivation and discrete models. In Frank Dehne, Jörg-Rüdiger Sack, and Michiel Smid, editors, *Proc. of the 8th Workshop on Algorithms and Data Structures (WADS)*, volume 2748 of *Lecture Notes in Computer Science*, pages 256–270. Springer, 2003.

[88] Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of termination of linear programming algorithms. *Mathematical Programming, Series B*, 97(1–2):375–404, 2003.

[89] Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM*, 51(3):385–463, 2004.

[90] Z. Sweedyk. A $2\frac{1}{2}$-approximation algorithm for shortest superstring. *SIAM Journal on Computing*, 29(3):954–986, 1999.

[91] William T. Tutte. A short proof of the factor theorem for finite graphs. *Canadian Journal of Mathematics*, 6:347–352, 1954.

[92] Vijay V. Vazirani. *Approximation Algorithms*. Springer, 2001.

[93] Sundar Vishwanathan. An approximation algorithm for the asymmetric travelling salesman problem with distances one and two. *Information Processing Letters*, 44(6):297–302, 1992.

[94] Oliver Vornberger. Easy and hard cycle covers. Technical report, Universität/Gesamthochschule Paderborn, 1980.