Diss. ETH No. 21003

Analysis of Hard Problems in Reoptimization and Online Computation

A dissertation submitted to the

ETH ZURICH

for the degree of

DOCTOR OF SCIENCES

presented by

Andreas Sprock
Dipl.-Inform. Rheinisch-Westfälische Technische Hochschule Aachen
Born on September 14, 1979
citizen of Germany

accepted on the recommendation of
Prof. Dr. Juraj Hromkovič, examiner
Prof. Dr. Peter Widmayer, co-examiner
Priv.-Doz. Dr. Walter Unger, co-examiner

2013

Zusammenfassung

In dieser Arbeit untersuchen wir die Komplexität von schweren Problemen, wenn zusätzliche Informationen gegeben sind. Zu diesem Zweck untersuchen wir Reoptimierungsprobleme sowie Online-Probleme mit gegebenem Advice als zwei mögliche Wege, zusätzlich gegebene Informationen in die Standard-Berechnungsmodelle zu integrieren.

Bei der Reoptimierung untersuchen wir das Szenario, in dem eine Instanz für ein schweres Problem sowie eine optimale Lösung für diese Instanz gegeben ist. Hierbei besteht das Problem darin, eine lokal veränderte Instanz für dasselbe Problem zu lösen. Es wurde bereits für viele verschiedene schwere Probleme gezeigt, dass entsprechende Reoptimierungsvarianten $\mathcal{NP}$-schwer bleiben oder dass es sogar schwer bleibt, sie zu approximieren. Oft können hier allerdings bessere Approximationsgüten erreicht werden.

In dieser Arbeit beschäftigen wir uns mit der Reoptimierung von Steinerbäumen auf Graphen mit verschärfter β-Dreiecksungleichung. Wir betrachten dabei als lokale Modifikation das Hinzufügen eines neuen Knotens zu einer gegebenen Instanz sowohl als Nicht-Terminal als auch als Terminal. Für diese Variante geben wir einen Linearzeit-Algorithmus an mit einer garantierten Approximationsgüte von $(1/2+\beta)$. Darüber hinaus geben wir einen 2β-Approximationsalgorithmus für das klassische Steinerbaum-Problem an. Dieses Resultat verbessert die bis dato bekannten Ergebnisse für Werte von $\beta < 1/2+\ln(3)/4 \approx 0.775$.

Ausserdem geben wir eine kurze Übersicht über die bereits bekannten Resultate im Bereich der Reoptimierung und untersuchen eine Verallgemeinerung des Konzepts der Reoptimierung, wobei mehr als nur eine optimale Lösung gegeben ist. Wir zeigen für einige Varianten des Steinerbaum- sowie des Traveling-Salesman-Problems, dass die bekannten Resultate zur Schwere auch für dieses allgemeinere Konzept gelten. Zusätzlich zeigen wir am Beispiel des allgemeinen Traveling-Salesman-Problems, dass es Klassen von Problemen gibt, bei denen nicht einmal die Menge *aller* optimalen Lösungen hilft, die generelle Approximierbarkeit zu verbessern. Das gilt sogar dann, wenn es sich dabei um exponentiell viele optimale Lösungen handelt. Weiter zeigen wir, dass lokale Suche für das allgemeine TSP auch dann nicht erfolgreich ist, wenn mehrere optimale Lösungen gegeben sind.

Mit der Advice-Complexity eines Online-Problems beschreiben wir die zusätzliche Information, die notwendig und ausreichend für einen Online-Algorithmus ist, um eine bestimmte Qualität zu erreichen. In unserem Modell kennt ein Orakel die gesamte Eingabe, bevor sie dem Online-Algorithmus gegeben wird. Abhängig von der Eingabe stellt das Orakel einen Advice-String zur Verfügung, der vom Online-Algorithmus teilweise gelesen werden kann. Die Anzahl an Advice-Bits, die ein Algorithmus liest, um eine gegebene Competitive Ratio zu erreichen, gibt uns ein verfeinertes Mass für die Komplexität des Problems.

Im zweiten Teil der Arbeit untersuchen wir Online-Probleme mit Advice. Wir beginnen mit dem Problem der Online-Färbung von Graphen, wobei der Graph Knoten für Knoten online präsentiert wird. Hier befassen wir uns mit der gesamten Klasse der dreifärbbaren Graphen sowie deren Teilklassen der chordalen und der maximal aussenplanaren Graphen. Wir zeigen, dass für die ersten beiden Klassen grundsätzlich $\log_2 3$ Advice-Bits pro Knoten (BpK) notwendig und hinreichend sind, um Graphen dieser Klasse optimal zu färben. Für die Klasse der maximal aussenplanaren Graphen zeigen wir die Existenz einer unteren Schranke von 1.0424 BpK und einer oberen Schranke von 1.2932 BpK. Anschliessend geben wir verschiedene Algorithmen an, um Graphen dieser Graphklassen mit 4 Farben zu färben. Hierbei reichen 0.9865 BpK aus, um chordale und maximal aussenplanare Graphen mit vier Farben zu färben, und für generell dreifärbbare Graphen reichen weniger als 1.1583 BpK aus.

Ein weiterer Beitrag dieser Arbeit ist eine generelle Beweistechnik, um untere Schranken für die Menge an Advice aufzuzeigen, die notwendig ist, um eine angestrebte Qualität zu erreichen. Dafür untersuchen wir das String-Rateproblem als ein generelles Problem und nutzen es, um eine untere Schranke für die Advice-Komplexität des Online-Maximum-Clique-Problems anzugeben. Darüber hinaus definieren wir eine formale Reduktion für Online-Minimierungsprobleme und verbessern mit deren Hilfe die besten bekannten Resultate für das Online-Set-Cover-Problem.

Abstract

In this thesis, we investigate the complexity of hard problems when additional information is given. For this, we consider the scenario of reoptimization and online scenarios with advice as two different approaches to integrate certain additional information into the standard computation models.

In reoptimization, we consider the following scenario: Given an instance of a hard optimization problem together with an optimal solution for it, we want to solve a locally modified instance of the problem. It has recently been shown for several hard optimization problems that their corresponding reoptimization variants remain $\mathcal{NP}$-hard or even hard to approximate whereas they often admit improved approximation ratios.

In this thesis, we deal with the local modification of integration a new vertex as terminal and as non-terminal into instances of the Steiner tree problem in graphs obeying a sharpened β-triangle inequality. For the upper bounds, for these local modifications, we design linear-time $(1/2 + \beta)$-approximation algorithms. As a building block for these algorithms, we employ a 2β-approximation algorithm for the classical Steiner tree problem on such instances, which might be of independent interest since it improves over the previously best known ratio for any $\beta < 1/2 + \ln(3)/4 \approx 0.775$.

Additionally, we briefly survey the known results about reoptimization and we investigate a generalization of the reoptimization concept where we are given not only one optimal solution but multiple optimal solutions for an instance. We prove, for some variants of the Steiner tree problem and the traveling salesman problem, that the known reoptimization hardness results carry over to this generalized setting.

Furthermore, we use the traveling salesman problem as an example that, even in the case where the set of *all* optimal solutions for an instance is available for free, there exist problems for which this additional knowledge does not help at all for improving the approximability. Moreover, we consider the performance of local search strategies on reoptimization problems. We show that local search does not work for solving TSP reoptimization, even in the presence of multiple solutions.

The advice complexity of an online problem describes the additional information both necessary and sufficient for online algorithms to compute solutions of certain quality. In this model, an oracle inspects the input before it is processed by an online algorithm. Depending on the input string, the oracle prepares an advice bit string that may be accessed sequentially by the algorithm. The number of advice bits that are read to achieve some specific competitive ratio can then serve as a fine-grained complexity measure.

In the second part of the thesis, we study online algorithms with advice. We start with the problem of coloring graphs which are presented online vertex by vertex. Here, we consider the class of all 3-colorable graphs and its sub-classes of chordal and maximal outerplanar graphs, respectively. We show that, in the case of the first two classes, for coloring optimally, essentially $\log_2 3$ advice bits per vertex (bpv) are necessary and sufficient. In the case of maximal outerplanar graphs, we show a lower bound of 1.0424 bpv and an upper bound of 1.2932 bpv. Finally, we develop algorithms for 4-coloring in these graph classes. The algorithm for 3-colorable chordal and outerplanar graphs uses 0.9865 bpv, and in case of general 3-colorable graphs, we obtain an algorithm using < 1.1583 bpv.

Another contribution of this thesis is to develop a powerful method for proving lower bounds on the number of advice bits necessary. To this end, we analyze the string guessing problem as a generic online problem and show a lower bound on the number of advice bits needed to obtain a small competitive ratio. We use special reductions from string guessing to give a lower bound on the advice complexity of the online maximum clique problem. Additionally, we define a formal model of reduction for online minimization problems and improve the best known lower bound for the online set cover problem.

Acknowledgment

First of all, I would like to deeply thank my supervisor Juraj Hromkovič, who enabled me to write this thesis in his group. During my time at ETH, I learned a lot from him, also exceeding the field of theoretical computer science. With his advice and support, he inspired me and showed me the way to do research as well as the beauty of the mountains and allowed with numerous excursions the conquer of breathtaking peaks.

I want to express my special gratitude to Hans-Joachim Böckenhauer for the many fruitful discussions. He never became tired of listening to my ideas and providing constructive criticism. I am deeply in debt to him for his abundance of patience in proofreading my text.

I would like to thank Peter Widmayer and Walter Unger for reviewing this thesis. Furthermore, I thank Walter Unger and Sebastian Seibert for the wonderful time in Aachen and the many discussions until late night, not only about research topics.

I am grateful to Tobias Mömke and Dennis Komm for the common, also sportive, challenges and the interesting discussions, that were not restricted to theoretical computer science. I also thank Giovanni Serafini and Sacha Krug not only for being great office mates, but also good friends. Thanks to the whole research group for the nice time I had.

I am thankful to Alexander Lachmann. If it were not for him, I would probably have never started my PhD at ETH.

Thanks to Tomáš Hrúz for encouraging me in doing this PhD, his support, and the nice time in our common project ‘milliPay’.

Thanks to my flat mate Nicola Pasquale for more than four years of living together in Zurich, and the innumerable evenings with interesting, not rarely philosophical, discussions accompanied by delicious Italian food and wine.

Most importantly, I thank my mom, my aunts and my Deniz for their support, advice, and for believing in me all the time.

Contents

Chapter 1

Introduction

A lot of real-world problems, especially in the field of logistics are hard in the sense that they are not optimally solvable within reasonable time. In operations research, such problems are very essential and constitute a wide research field. We are traditionally concerned with finding optimal solutions to practically relevant problem instances about which nothing is known in advance. Unfortunately, finding such optimal solutions is computationally hard in many cases, and thus we have to use different approaches like heuristics or approximation algorithms for computing good (but not necessarily optimal) feasible solutions. But, in many cases, even computing a solution of a satisfactory approximation quality remains a hard task.

When formalizing a real world problem as an instance of a computational problem, we are usually left with a lot of additional information that was not incorporated in the formal model. This information could be used to provide better solutions to many types of hard problems. As an example, consider the problem of finding an optimal train schedule (for some railway network, objective function, and under some constraints). If we succeeded in computing an optimal schedule, it is natural to expect that we can profit somehow from this schedule when an additional railway station is opened and we have to find a new optimal schedule.

In this thesis, we investigate two theoretical approaches to integrate certain kinds of additional knowledge into our models. As a first approach, we analyze approximation scenarios, where a very special kind of extra information is given, namely optimal or near-optimal solutions for similar instances. The concept of *reoptimization* formally describes the following approach: Given a problem instance together with an optimal (or approximate) solution for it and a locally modified new problem instance, what can we say about the optimal solution for the new instance? Does the knowledge about the old optimal solution help when computing a solution for the new instance? How much can it improve the running time or the quality of the output? Even if many optimization problems are better approximable with information in form of optimal solutions for similar instances, here, we highlight the limits of additional knowledge in this concept. We will see that, for many reoptimization problems, even an exponential number of given optimal solutions does not help.

In the second approach, we turn to a very general model of additional information. Here, we are interested in measuring the amount of information that is needed to compute an optimal (or near optimal) solution. We analyze this concept in the context of online problems. Here, we deal with the natural situation that the input arrives piecewise. In contrast to the offline case, the online algorithm must compute a part of the solution for the already given piece of input at every time step. Once a part of the solution is computed, it cannot be changed. In each step, the algorithm has to produce a part of the output without knowing the whole instance. Here, the extra information is given by an oracle that knows the future and writes helpful information onto an advice "tape".

The standard way to measure the quality of an online algorithm is the *competitive analysis*. The quality of the solution computed by the online algorithm is compared to the quality of the best possible solution computable offline, i.e., when knowing the whole input. Measuring the quality of an online algorithm by comparing its output to an optimal offline solution gives an universally applicable yardstick. However, it has the disadvantage that, for many problems, the output of the best possible online algorithms is still far from the offline solutions. To establish a more fine-grained measure of the hardness of online problems, one introduces an advisor (an oracle that knows the whole input) that provides an *advice tape* with an unlimited advice bit string to the online algorithm. For any online algorithm with advice, the *advice complexity* measures the number of bits read by the online algorithm. The advice complexity of a problem is defined as the advice complexity of the best online algorithm (achieving a given competitive ratio).

The advice complexity of an online problem describes the information content of an online problem, and thus a new measure for its hardness. This measure might be used to give an insight into the gap between online and offline algorithms even with unlimited computing power[1]. In this thesis, we focus on online algorithms with advice which have a linear running time. In this situation, the advice complexity might give us a new view on the hardness of such problems. Note that every problem solvable in polynomial time using a logarithmic number of advice bits is also solvable offline in polynomial time by simulating the computation for each of the possible advice strings.

In Section 1.1 and 1.2, we give some additional information and explanation about the reoptimization approach and the idea of online algorithms. The common formal definitions needed in this thesis are given in Section 3.4.

[1]Due to the unlimited computing power of the advisor, the online algorithm may be used on hard problems, provided that enough advice is used.

1.1 Reoptimization Approach

The concept of reoptimization was mentioned for the first time in [73] in the context of postoptimality analysis for a scheduling problem. In postoptimality analysis, one studies the related question of how much a given instance of an optimization problem may be altered without changing the set of optimal solutions, see for example the paper by van Hoesel and Wagelmans [77]. Since then, the concept of reoptimization has been investigated for several different problems like the traveling salesman problem [2, 6, 7, 15, 27], the Steiner tree problem [10, 17, 23, 43], knapsack problems [3], covering problems [13], and the shortest common superstring problem [11, 12]. In these papers, it was shown that, for some problems, the reoptimization variant is exactly as hard as the original problem, whereas reoptimization can help a lot in improving the approximation ratio for other problems. For an overview of some results, see also [4, 24, 81]. These results show that the reoptimization concept gives new insight into the hardness of the underlying optimization problems and allows for a more fine-grained complexity analysis. In this context, we usually measure the quality of a solution in terms of the approximation ratio, i.e., the quotient of the value of the computed solution and the optimal value.

While the reoptimization variants of $\mathcal{NP}$-hard optimization problems usually remain $\mathcal{NP}$-hard, the approximability might essentially improve in the reoptimization setting [24]. For example, the approximation ratio of the metric TSP reoptimization, where the considered local modification consists of changing the cost of a single edge, can be improved to 4/3 [7], whereas without reoptimization the best known approximation ratio is 1.5 due to Christofides' algorithm [35]. On the other hand, for some reoptimization problems, inapproximability results have been proven [15, 16, 25, 27].

In Chapter 2, we give some positive results on the Steiner tree reoptimization on graphs with sharpened triangle inequality. The Steiner tree problem is a very prominent optimization problem with many practical applications, especially in network design, see for example [54, 69]. The problem is known to be APX-hard, even if the edge costs are restricted to 1 and 2 [8]. In [18], the hardness of even more restricted input instances was shown. More precisely, cases where the edge costs are restricted to the values 1 and $1 + \gamma$, for any $0 < \gamma$, were considered. Also this restricted problem variant is known to be $\mathcal{APX}$-hard [50]. In particular, restricting the edge costs in the described way also implies the same hardness results for the class of Steiner tree problems where the edge-costs satisfy the sharpened β-triangle inequality, i. e., where the cost function satisfies the condition $cost(v_1, v_2) \leq \beta \cdot (cost(v_1, v_3) + cost(v_3, v_2))$, for some $1/2 \leq \beta < 1$ and for all vertices v_1, v_2, and v_3.

The graphs satisfying a sharpened triangle inequality form a subclass of the class of all metric graphs. Intuitively speaking, for vertices that are points in the Euclidean plane, a parameter value $\beta < 1$ prevents that three vertices can be placed on the same line. For more details and motivation of the sharpened triangle inequality, see [19].

A minimum spanning tree on the terminal vertices (w. r. t. the metric closure of the edge costs) is sufficient for achieving a 2-approximation (see, e. g., [69]), and the best currently known approximation ratio for the Steiner tree problem is 1.39 for general edge costs [33] and 1.28 for edge costs 1 and 2 [71].

To analyze how the transition from metric graphs ($\beta = 1$) to graphs with sharpened β-triangle inequality influences the computational hardness of the Steiner tree problem, we consider, in Chapter 2, the question whether additional knowledge about the input is helpful to find a good solution [17, 18].

In Chapter 3, we first give a brief overview of known reoptimization results showing that the answers to questions concerning the value of additional knowledge are very dependent on the optimization problem and the type of local modification. On the one hand, as already mentioned, there exist $\mathcal{APX}$-hard optimization problems where the knowledge of an optimal solution can help to improve the achievable approximation ratio on a locally modified instance, sometimes even to design a PTAS for the reoptimization variant of the problem. On the other hand, there exist problems for which the reoptimization variant is exactly as hard as the original problem.

In this thesis, we also consider a generalization of the reoptimization approach in Chapter 2. We assume that we are given not only one optimal solution for a locally modified problem instance, but the k best solutions for some k which might even be exponentially large in the size of the input. We also study a reoptimization situation where we are given a TSP instance together with the set of *all* optimal solutions. As a local modification, the cost of one edge is increased. We show that, even with this extra knowledge, it remains $\mathcal{NP}$-hard to compute a 2^n-approximate solution. Furthermore, we consider TSP reoptimization, where, in addition to all optimal solutions, we are given exponentially many near-optimal solutions. We prove that this multi-solution reoptimization variant, even if all solutions are very different, is as hard to approximate as the TSP itself.

Local search is another method which is often used for solving hard optimization problems. In a classical paper, Papadimitriou and Steiglitz [67] have shown that there exist instances of the general TSP that are very hard for local search with respect to the neighborhood defined by exchanging up to k edges in a Hamiltonian tour in the following sense: Their instances have a unique optimal solution, but exponentially many second-best solutions of exponentially higher cost which furthermore differ in many edges from the optimal solution. We extend their graph construction to prove an analogous result for the multi-solution reoptimization variant of the problem [25, 26].

1.2 Online Versions of $\mathcal{NP}$-hard Problems with Advice

Numerous computational problems arise in so-called *online environments*, where the input arrives piecewise in successive time steps. An *online algorithm* has to answer a query by a part of the final output without knowing anything about the future requests (the rest of the input). The concept of *competitive analysis* was introduced by Sleator and Tarjan in 1985 [76], for a more detailed introduction we refer to the standard literature, e.g., [31, 52].

To adjust for the disadvantage that the output of the best possible online algorithms is far from offline solutions, we provide additional information. We ask how much additional information an online algorithm needs to close this gap, respectively, which progress can be made with limited advice. Here, the advice is provided as an infinite tape of advice bits from which the online algorithm can read as many bits as it needs.

For this, we study the model of *computing with advice.* In order to better understand and quantify this gap, the idea of online algorithms with advice was introduced in [39] and has been further investigated for various online problems, e.g., in [9, 28–30, 41, 42, 46, 53, 61, 62, 70]. We follow the most general and exact model from [53] in this thesis. For a detailed overview, we recommend [60].

Coloring vertices of a graph such that no two adjacent vertices get the same color is a very well known and intensively studied problem which we address in Chapter 4. For an online version, the following set-up can be studied: In every step, a new vertex gets revealed, together with all edges to the previously revealed vertices. Now, the newly revealed vertex has to be colored before the next one is revealed. It turns out that online coloring is hard and no constant competitive ratio is possible [64]. For the class of k-colorable graphs on n vertices, it has been proven that any online coloring algorithm needs $\Omega\left((\log n/(4k))^{k-1}\right)$ colors in the worst case [78]. For an overview of classical online coloring see [58, 59].

A first study of a special case of online path coloring with advice was done in [46]. Further studies of online coloring of bipartite graphs were done in [9]. In Chapter 4, we study online coloring with advice on the class of all 3-colorable graphs, and on its sub-classes of 3-colorable chordal and outerplanar graphs, respectively. We want to know how much advice is necessary and sufficient in order to color these graphs optimally. We also investigate how much advice can be saved if we allow the use of a fourth color. The results mentioned above imply that using only a constant number of colors is a big improvement over coloring without advice. Furthermore, we show lower and upper bounds on the advice complexity of online coloring of general 3-colorable graphs, 3-colorable chordal graphs, and maximal outerplanar graphs. For the upper bounds, we present polynomial-time online algorithms. It turns out that online 3-coloring with advice in general is equally hard with respect to the advice complexity as online 3-coloring of chordal graphs. Moreover, we analyze the advice needed for coloring 3-colorable graphs with a competitive ratio of 4/3 [74, 75]. In other words, we want to color 3-colorable graphs with four colors. Note that the offline version of this problem is also known to be $\mathcal{NP}$-hard [48, 57].

In Chapters 5 and 6, we are especially interested in *lower bounds* on the advice complexity of optimization problems. Such lower bounds do not only tell us something about the information content [53] of online problems, but they also carry over to a randomized setting where they imply lower bounds on the number of random decisions needed to compute a good solution [61]. But, similar to most other computing models, lower bounds on the advice complexity are hard to prove. Thus, it is desirable to have some generic proof methods for establishing lower bounds. In Chapters 5 and 6, we take a first step towards this goal by using a generic online problem and showing how to transfer lower bounds on its advice complexity to lower bounds for other online problems.

In Chapter 5, we study the *string guessing problem* with respect to its necessary advice for reaching a certain quality. As already mentioned, this problem is very generic with respect to proving lower bounds on the advice complexity. Here, a string of length n over an alphabet of size q has to be guessed. More specifically, we define two versions of the problem where, in the first case, the algorithm gets immediate feedback which decisions would have been correct up to the current time step, and in the second case, this feedback is not supplied. This problem is, of course, not $\mathcal{NP}$-hard, but we can use it to improve the known lower bound on the advice complexity of SetCover. First, we prove a lower bound on the advice necessary to achieve some specific number of correct guesses for both versions. Then, we show that the extra information of knowing the history, i.e., of the correct answers to the previous questions, does not help much for this class of problems. Additionally, we analyze the size of the advice depending on both n and q. Employing this result, we use the string guessing problem as a technique to prove lower bounds for other well-studied online problems in Chapter 6. It seems to be a promising approach to use string guessing this way to prove the hardness of further online problems.

In Chapter 6, we analyze the advice complexity of the *online maximum clique problem* (MaxClique) and of the *online set cover problem* (SetCover) in the unweighted variant. In MaxClique, introduced by Demange et al. (see [38]), in every time step, a vertex is given together with all edges to vertices that were already revealed in previous time steps, and an online algorithm `A` has to decide whether the newly revealed vertex becomes part of the solution or not.

SetCover was introduced by Alon et al. in [1]. At the beginning, a ground set X and family $\mathcal{S} \subseteq Pot(X)$ with $\bigcup_{S \in \mathcal{S}} S = X$ are given. In any step, one element x_i of X is given, and the online algorithm has to select a set $S_i \in \mathcal{S}$ such that x_i gets covered, if x_i is not yet covered by any already selected set. The cost function measures the number of selected sets. Our result [21, 22] closes an exponential gap between the lower and upper bounds given in [62].

1.3 Basic Definitions

Most problems in this thesis are formulated in terms of graphs. We use the following graph-theoretic notation following the books by Brandstädt et al. [32] and West [79]. Let $G = (V, E)$ be a finite graph with $|V| = n$. We refer to the edges and vertices of a graph G by $E(G)$ and $V(G)$. Two vertices $u, v \in V(G)$ are **adjacent** in G if there exists an edge $\{u, v\} \in E(G)$. A vertex u is **incident** to an edge e, if $u \in e$. A graph $G' = (V', E')$ is a **subgraph** of G if $V' \subseteq V$ and $E' \subseteq E$. A subgraph G' of G is **induced** by V' if $E' = \{\{u, v\} \mid \{u, v\} \in E \text{ and } u, v \in V'\}$. We denote such a subgraph by $G_{V'}$. A **path** P in G is a sequence $P = x_1, x_2, \ldots, x_k$ of pairwise distinct vertices such that $\{x_i, x_{i+1}\} \in E$ for all $i \in \{1, \ldots, k-1\}$. For a path $P = x_1, \ldots, x_k$, let $V(P) = \{x_1, \ldots, x_k\}$ denote the set of vertices of P and let $E(P) = \{\{x_i, x_{i+1}\} \mid 1 \leq i \leq k-1\}$ denote the set of edges of P. We usually identify a path P with the subgraph $(V(P), E(P))$. The **length** of a path is the number $|E(P)|$ of its edges. A **Hamiltonian path** in G is a path of length $n-1$. A **cycle** $H = x_1, \ldots, x_k$ is a path $P = x_1, \ldots, x_k$ for $k \geq 3$, extended by an edge from the first vertex x_1 to the last vertex x_k, $E(H) = E(P) \cup \{x_k, x_1\}$. Analogous to Hamiltonian path, a **Hamiltonian cycle** in G is a cycle on n vertices.

We call a graph $G = (V, E)$ **connected** if, for each pair of vertices $v_i, v_j \in V(G)$, there exists a path in G with v_i and v_j as its endpoints. A **tree** $T = (V, E)$ is a connected graph that contains no cycle.

A **chord** of a cycle $H = x_1, x_2, \ldots, x_k$ is an edge $\{x_i, x_j\}$, that is no edge of the cycle: $\{x_i, x_j\} \notin E(H)$. A cycle is chordless if it has no chords.

For any vertex $v \in V$, we denote by $\mathit{Neigh}(v) = \{w \in V \mid \{v, w\} \in E\}$ the set of neighbor vertices of v in G. If G is directed, E contains ordered pairs, and the set of predecessors of v is $\mathit{Pred}(v) = \{w \in V \mid (w, v) \in E\}$.

An undirected graph G is called **edge-weighted** if there exists a cost function $\mathit{cost} : E(G) \to \mathbb{Q}$ which defines the cost of every edge in G. For the cost of an edge $\{x, y\}$, we use the notation $\mathit{cost}(x, y)$ instead of $\mathit{cost}(\{x, y\})$.

An edge-weighted graph $G = (V, E, \mathit{cost})$ satisfies a **sharpened triangle inequality**, if its edge-costs satisfy the sharpened β-triangle inequality, i. e., if there exists a $1/2 \leq \beta < 1$, such that the cost function cost satisfies the condition

$$\mathit{cost}(v_1, v_2) \leq \beta \cdot (\mathit{cost}(v_1, v_3) + \mathit{cost}(v_3, v_2)), \tag{1.1}$$

for all vertices v_1, v_2, and v_3.

The graphs satisfying a sharpened triangle inequality form a subclass of the class of all metric graphs (i.e., the graphs whose edge costs satisfy (1.1) with $\beta = 1$). Intuitively speaking, for vertices that are points in the Euclidean plane, a parameter value $\beta < 1$ prevents that three vertices can be placed on the same line. For more details and further motivation of the sharpened triangle inequality, see [20].

Definition 1.1 (Steiner tree) *Given a graph $G = (V, E)$ and a subset $S \subseteq V$ of vertices, called* ***terminals****, a* ***Steiner tree*** *for (G, S) is a subtree T of G spanning all terminals, i. e., $T = (V(T), E(T))$ is a tree such that $S \subseteq V(T) \subseteq V$ and $E(T) \subseteq E$. The vertices in $V \setminus S$ are called* ***non-terminals****.*

Definition 1.2 (Coloring) *Given a graph $G = (V, E)$, a* ***coloring function*** *col is a function that maps every vertex $v_i \in V$ to some color from $\{1, \ldots, k\}$, for some $k \in \mathbb{N}$, such that $\text{col}(v_i) \neq \text{col}(v_j)$, for all v_i, v_j with $\{v_i, v_j\} \in E(G)$.*

Additionally, we deal with the graph classes of chordal and maximal outerplanar graphs. Now, we give a formal definition.

Definition 1.3 *A graph G is* ***chordal*** *if each cycle in G of length at least 4 has at least one chord.*

Definition 1.4 (Planar graph) *A graph $G = (V, E)$ is called* ***planar****, if it can be drawn in the plane such that no two edges intersect. Such a drawing of a planar graph in the plane is called a* ***planar embedding****, or* ***embedding*** *for short.*

A planar graph $G = (V, E)$ is **maximal planar** if, for every non-adjacent pair of vertices v_i, v_j, the graph $G' = (V, E \cup \{v_i, v_j\})$ is not planar anymore.

Definition 1.5 (outerplanar graph) *A graph $G = (V, E)$ is called* ***outerplanar****, if it has a planar embedding, such that no vertex is totally surrounded by edges.*

An outerplanar graph is **maximal outerplanar** if there are no two non-adjacent vertices v_i, v_j such that the graph $G' = (V, E \cup \{v_i, v_j\})$ is still outerplanar.

1.3.1 Optimization and Reoptimization Problems

In Chapter 2 and 3, we deal with a special variant of optimization problems, the reoptimization problems. In the case of reoptimization, the input consists of a problem instance together with a locally modified instance, according to some type of local modification, and an optimal solution for the second one. The analysis of reoptimization problems is concerned with the question how useful this extra information can be. To this end, we give a formal definition of an optimization problem and of a reoptimization problem. For more details, see [51].

Definition 1.6 (optimization problem) *An* ***optimization problem*** *is a 6-tuple $U = (\Sigma_I, \Sigma_O, L, \mathcal{M}, cost, goal)$, where*

i. Σ_I is an alphabet, called the input alphabet of U, *ii.* Σ_O *is an alphabet, called the output alphabet of* U,

iii. $L \subseteq \Sigma_I^*$ *is the language of feasible problem instances,*

iv. $\mathcal{M}$ *is a function from* L *to* $Pot(\Sigma_O^*)$, *where* $Pot(x)$ *denotes the power set of* x, *and, for every instance* $x \in L$, $\mathcal{M}(x)$ *is called the set of feasible solutions for* x,

v. $cost$ *is a cost function which, for every pair* (u, x) *with* $x \in L$ *and every* $u \in \mathcal{M}(x)$, *assigns a positive real number* $cost(u, x)$,

vi. $goal \in \{minimum, maximum\}$ *is the optimization goal.*

For an instance $x \in L$, *a feasible solution* $y \in \mathcal{M}(x)$ *is called an* ***optimal solution*** *for* x *and* U *if*

$$cost(y, x) = goal\{cost(z, x) \mid z \in \mathcal{M}(x)\},$$

if this optimum exists. In the following, we denote the cost of an optimal solution $y \in \mathcal{M}(x)$ *for the instance* x *by* $\boldsymbol{Opt_U(x)}$.

We say that an algorithm $\mathtt{A}$ **solves** U if, for every instance $x \in L$ the output of $\mathtt{A}$ satisfies $\mathtt{A}(x) \in \mathcal{M}(x)$.

In this thesis, we deal with several types of optimization problems. Now, we give a formal definition of them.

In Chapter 2 and 3, we analyze the minimum Steiner tree problem. The **minimum Steiner tree problem, Min-STP** for short, is the following optimization problem: Given an undirected complete graph $G = (V, E)$ with a edge cost function $cost : E \to \mathbb{Q}^{>0}$ and a subset $S \subseteq V$ of **terminals**, the goal is to find a minimum-cost Steiner tree of G.

Additionally, in Chapter 3, we analyze several variants of the traveling salesman problem. The **traveling salesman problem, TSP** for short, is the problem to find a minimum-cost Hamiltonian cycle in a given undirected complete graph $G = (V, E)$ with an arbitrary edge-cost function $cost : E \to \mathbb{Q}^{>0}$.

In Chapter 4, we analyze an online variant of the minimum vertex coloring of graphs. The **minimum vertex coloring problem, Coloring** for short, is the problem to find a coloring of a given undirected unweighted graph $G = (V, E)$ with a minimum number of colors.

Finally, in Chapter 6, we analyze the online variants of the maximum clique problem and the minimum set cover problem. The **maximum clique problem, MaxClique** for short, is the problem to find a clique, i.e., a complete subgraph G', of a given undirected unweighted graph $G = (V, E)$ containing a maximum number of vertices.

The **minimum set cover problem, SetCover**, is the following minimization problem. Given a ground set X and a set family $\mathcal{F} \subseteq Pot(X)$ such that $X = \bigcup_{S \in \mathcal{F}} S$, the problem is to find a family $\mathcal{C} \subseteq \mathcal{F}$ of minimal size with $X = \bigcup_{S \in \mathcal{C}} S$.

Analogously to the classes $\mathcal{P}$ and $\mathcal{NP}$ for decision problems, there exist the classes $\mathcal{PO}$ and $\mathcal{NPO}$ for optimization problems.

Definition 1.7 ($\mathcal{NPO}$) *The class* $\boldsymbol{\mathcal{NPO}}$ *is the class of optimization problems* $U = (\Sigma_I, \Sigma_O, L, \mathcal{M}, \mathit{cost}, \mathit{goal})$ *satisfying the following constraints:*

i. $L \in \mathcal{P}$.

ii. There exists a polynomial function p_U, *such that*

  (a) for each $x \in L$ *and each* $y \in \mathcal{M}(x)$, $|y| \leq p_U(|x|)$,

  (b) there exists a polynomial-time algorithm that decides, for each $y \in \Sigma_O^*$ *and for each* $x \in L$ *with* $|y| \leq p_U(|x|)$, *whether* $y \in \mathcal{M}(x)$ *or not, and*

iii. the cost function is computable in polynomial time.

Morover, we can define the class $\mathcal{PO}$ as follows.

Definition 1.8 ($\mathcal{PO}$) *The class* $\boldsymbol{\mathcal{PO}}$ *is the class of all optimization problems* $U = (\Sigma_I, \Sigma_O, L, \mathcal{M}, \mathit{cost}, \mathit{goal}) \in \mathcal{NPO}$ *for which an algorithm exists, that solves* U *in polynomial time.*

For the classes $\mathcal{PO}$ and $\mathcal{NPO}$, we have $\mathcal{PO} \subsetneq \mathcal{NPO}$, unless $\mathcal{P} = \mathcal{NP}$. All problems from above, STP, TSP, Coloring, MaxClique, SetCover, are contained in $\mathcal{NPO} \setminus \mathcal{PO}$, unless $\mathcal{P} = \mathcal{NP}$.

For a connection between optimization problems and decision problems, we define the threshold language.

Definition 1.9 (Threshold language) *We define the* ***threshold language*** *for any optimization problem* $U = (\Sigma_I, \Sigma_O, L, \mathcal{M}, \mathit{cost}, \mathit{goal}) \in \mathcal{NPO}$ *as*

$$\mathit{Lang}_U = \{(x, a) \in L \times \Sigma_{\mathit{bool}}^* \mid \mathit{Opt}_U(x) \leq \mathit{Number}(a)\}$$

if goal = minimum and

$$\mathit{Lang}_U = \{(x, a) \in L \times \Sigma_{\mathit{bool}}^* \mid \mathit{Opt}_U(x) \geq \mathit{Number}(a)\}$$

if goal = maximum, where $\mathit{Number}(a)$ *denotes the number represented by the binary word* a.

An optimization problem is called $\mathcal{NP}$*-hard if* Lang_U *is* $\mathcal{NP}$*-hard.*

One approach for dealing with $\mathcal{NP}$-hard optimization problems are approximation algorithms. Even if finding an optimal solution is hard, it can be much easier to find an adequate good solution. We use the worst-case analysis of an algorithm to give a lower bound on the quality of the solution of an algorithm. To measure the quality of such a solution, we use the approximation ratio which is defined in the following.

Definition 1.10 (approximation ratio) *Let $U = (\Sigma_I, \Sigma_O, L, \mathcal{M}, cost, goal)$ be an optimization problem. For each algorithm* A *that solves U and for each $x \in L$, we define the **approximation ratio $R_{\mathtt{A}}(x)$ of* A *on x** as*

$$R_{\mathtt{A}}(x) = \max\left\{\frac{cost(A(x))}{Opt_U(x)}, \frac{Opt_U(x)}{cost(A(x))}\right\}.$$

*For all $n \in \mathbb{N}$, the **approximation ratio of*** A *is defined as*

$$R_{\mathtt{A}}(n) = \max\{R_{\mathtt{A}}(x) \mid x \in L \cap (\Sigma_I)^n\}.$$

For any $\delta \in \mathbb{R}_{>1}$, we call A a **$\delta$-approximation algorithm for U** if $R_{\mathtt{A}}(x) \leq \delta$, for all $x \in L$.

For a function $f : \mathbb{N} \to \mathbb{Q}$, we call A an **$f(n)$-approximation algorithm for U** if $R_{\mathtt{A}}(n) \leq f(n)$, for all $n \in \mathbb{N}$.

Additionally, there exist optimization problems in $\mathcal{NPO}$ for which there exist approximation algorithms where an arbitrary ratio can be reached by increasing the running time for the algorithm.

Definition 1.11 (PTAS, FPTAS) *Let $U = (\Sigma_I, \Sigma_O, L, \mathcal{M}, cost, goal) \in \mathcal{NPO}$ be an optimization problem. An algorithm* A *is called a **polynomial-time approximation scheme (PTAS)** for U, if, for every input pair $(x, \varepsilon) \in L \times \mathbb{R}^+$,* A *computes a solution for U with an approximation ratio $R_{\mathtt{A}} \leq 1+\varepsilon$ and $Time_{\mathtt{A}}(x)$ can be bounded by a function that is polynomial in $|x|$ (but possibly exponential in ε^{-1}).*

*If $Time_{\mathtt{A}}(x)$ can be bounded by a function that is polynomial in both $|x|$ and ε^{-1}, we say that A is a **fully polynomial-time approximation scheme (FPTAS)** for U.*

With this we can give a hierarchy of complexity classes. The first class $\mathcal{FPTAS}$ contains all problems for which an FPTAS exists. One member of this class is the knapsack problem (see [51]). The next class is $\mathcal{PTAS}$ which contains all problems that allow a PTAS. Every problem in $\mathcal{FPTAS}$ lies also in $\mathcal{PTAS}$, but there exist some problems in $\mathcal{PTAS}$ where no FPTAS exists, unless $\mathcal{P} = \mathcal{NP}$. The makespan scheduling problem (see [51]) is such a problem. We also know the class $\mathcal{APX}$, containing all problems for which a δ-approximation algorithm exists, for some constant δ. Here, one prominent example is the Δ-TSP. In the class $\mathcal{LOGAPX}$, all optimization problems are contained, for which an $f(n)$-approximation algorithm exists, for some function $f(n) \in \mathcal{O}(\log n)$. The problems SetCover and Coloring lie in this class. There remain problems in $\mathcal{NPO}$ that are not in $\mathcal{LOGAPX}$. The general TSP and MaxClique are examples. Unless $\mathcal{P} = \mathcal{NP}$, we get the following order of the problem classes

$$\mathcal{PO} \subsetneq \mathcal{FPTAS} \subsetneq \mathcal{PTAS} \subsetneq \mathcal{APX} \subsetneq \mathcal{LOGAPX} \subsetneq \mathcal{NPO}.$$

In this thesis, we analyze $\mathcal{NP}$-hard problems in scenarios where some extra information is given. We start with the concept of reoptimization. Here, besides the instance x, a local modification $lm(x)$ of it, i.e., a slightly different instance, together with its optimal solution, are given as a kind of extra information. In the following, we give a formal definition of a reoptimization problem.

Definition 1.12 (reoptimization problem) *Let $U = (\Sigma_I, \Sigma_O, L, \mathcal{M}, cost, goal)$ be an optimization problem. For a local modification lm on the instances of U, the **reoptimimization problem lm-U** is defined as:*

Input: *Two problem instances x and y where y is a local modification of x according to lm, and an optimal solution s_x for the instance x.*

Problem: *Find an optimal solution $s_y \in \mathcal{M}(y)$ for the instance y.*

In graph problems such local modification could be, e.g., the insertion and deletion of vertices, or increasing or decreasing the cost of a single edge.

In Chapter 2, we deal with different reoptimization variants of the Steiner tree problem. Now, we give a formal definition.

Definition 1.13 (Min-Δ_β-STP) *Given is a connected, undirected, and edge-weighted graph $G = (V, E, cost)$ and a subset $S \subseteq V$ of **terminals**. If the cost function cost satisfies the β- triangle inequality, the minimum Steiner tree problem on $(G, S, cost)$ is called **Min-Δ_β-STP.** Similar to the Min-Δ_β-STP, we consider the problem **Min-$(1, 1+\gamma)$-STP**, where only edges of cost 1 and $1+\gamma$ are allowed, for some constant γ with $0 < \gamma \leq 1$.*

The relation of Min-Δ_β-STP and Min-$(1, 1+\gamma)$-STP is as follows.

Lemma 1.1 (Böckenhauer et al. [18]) *For any graph $G = (V, E)$ and any $0 < \gamma$, any cost function $cost : E \to \{1, 1+\gamma\}$ satisfies the $(1+\gamma)/2$-triangle inequality.*

Based on the Steiner tree variants Min-STP and Min-Δ_β-STP, we define the following reoptimization variants.

Definition 1.14 (Reopt-STP-*lm*) *The **minimum Steiner tree reoptimization problem with the local modification lm (Reopt-STP-lm)** is the following optimization problem. The goal is to find a minimum Steiner tree for an input instance $(G', S', cost')$, given an optimal Steiner tree T_{Old} for the instance $(G, S, cost)$, where $(G', S', cost') = lm(G, S, cost)$. We consider the following local modifications.*

(AddNonTerm) *When adding a non-terminal v_{New}, $V(G') = V(G) \uplus \{v_{\text{New}}\}$, $S' = S$, and cost is the restriction of $cost'$ to $V(G)$.*

(AddTerm) *When adding a terminal vertex v_{New}, $V(G') = V(G) \uplus \{v_{\text{New}}\}$, $S' = S \uplus \{v_{\text{New}}\}$, and cost is the restriction of $cost'$ to $V(G)$.*

The corresponding problem variants where the edge cost function $cost'$ satisfies the sharpened β-triangle inequality for some $1/2 < \beta \leq 1$, and the variant with edge costs in $\{1, 1+\gamma\}$, for $\gamma > 0$, are denoted by Reopt-Δ_β-STP-lm and Reopt-$(1, 1+\gamma)$-STP-lm, respectively.

1.3.2 Online Algorithms and Advice Complexity

The second part of this thesis deals with online algorithms, where the algorithm gets the input piecewise, and irrevocably produces, in every step, a part of the output. To measure the quality of an online algorithm, we use the concept of an **adversary** that produces the input corresponding to the deterministic online algorithm, in order to force the algorithm to produce an as bad as possible output. This concept relates to the idea of a worst-case analysis for optimization problems. Our definitions follow the notation from [29, 53]. We start with the formal definition of an online problem.

Definition 1.15 (online optimization problem) *An **online optimization problem** consists of a set $\mathcal{I}$ of inputs and a cost function. Every input $I \in \mathcal{I}$ is a sequence of requests $I = (x_1, x_2, \ldots, x_n)$. A feasible output $O \in \mathcal{O}$ is a sequence of answers $O = (y_1, y_2, \ldots, y_n)$ where, for every subsequence $I_i = (x_1, \ldots, x_i)$, the corresponding output subsequence $O_i = (y_1, \ldots, y_i)$ is also a feasible answer. The cost function $cost\colon \mathcal{I} \times \mathcal{O} \to \mathbb{Q}^{>0}$ defines a positive real value $cost(I, O)$ for every pair of an input I and any feasible output O for I.*

For an easier notation, when the input is clear from the context, we denote the cost of (I, O) by $cost(O)$.

An established measurement for the quality of an output of an online algorithm is the **competitive ratio** [31,45,52,55]. Now, we give a formal definition of an online algorithm and the competitive ratio, which can be defined analogously to the approximation ratio.

Definition 1.16 (online algorithm, competitive ratio)
*Let $I = (x_1, x_2, \ldots, x_n)$ be an input of an online optimization problem. An **online algorithm** $\mathtt{A}$ computes the output sequence $\mathtt{A}(I) = (y_1, \ldots, y_n)$ such that y_i is computed from $x_1, \ldots, x_i$. For some output sequence O, $cost(O)$ denotes the cost of O.*

*For all $c \in \mathbb{R}{>}\ 1$, an algorithm A is **c-competitive** if there exists some non-negative constant α such that, for every I,*

$$cost(\mathtt{A}(I)) \leq c \cdot cost(\mathtt{Opt}(I)) + \alpha$$

if I is an input of a minimization problem, and

$$cost(\mathtt{A}(I)) \geq \frac{1}{c} \cdot cost(\mathtt{Opt}(I)) - \alpha$$

*if I is an input of a maximization problem. Here, $\mathtt{Opt}(I)$ denotes an optimal solution for I. We call c the **competitive ratio** of $\mathtt{A}$. If $\alpha = 0$, then $\mathtt{A}$ is called **strictly** c-competitive. $\mathtt{A}$ is optimal if it is strictly 1-competitive.*

All algorithms we construct in Part II are analyzed with respect to strict competitiveness. The given algorithm for our positive results from Chapter 4 have polynomial running time. For the lower bounds, we neglect the runtime of our online algorithms, as it is usually done [31, 55]. The concept of competitive analysis was introduced by Sleator and Tarjan in 1985 [76], although is was probably used for the first time implicitly by Yao in 1980 [80]. The term competitive ratio was introduced by Karlin et al. [56].

The competitive analysis of online algorithms is a kind of worst-case analysis in terms of the competitive ratio that an algorithm can guarantee. For the analysis, we use an assumed **adversary** denoted by `Adv`. The adversary constructs the input using its knowledge about what the algorithm will do. Since we analyze invariably deterministic algorithms, `Adv` can predict what the algorithm will do.

If we can prove, for an online problem P, the existence of an adversary that can produce, for every algorithm, an instance for which the algorithm cannot be better than c-competitive, we say that c is a lower bound on the competitiveness of P.

Similar to the concept of reoptimization, where extra information in the form of an optimal solution for a locally modified instance is given, also in the online environment, we investigate a scenario with extra information. Here, we assume the existence of an oracle that knows the whole input from the beginning. The oracle, called the **advisor**, produces an infinitely long advice tape. The online algorithm with advice is allowed to read, during the online computation, advice from the tape, in order to produce a better output. To measure the advice complexity of an online algorithm, we analyze the number of advice bits read by the algorithm. Now, we give a formal definition of online algorithms with advice following [53].

Definition 1.17 (Online Algorithm with Advice) *Let I be an input of an online optimization problem with $I = (x_1, \ldots, x_n)$. An **online algorithm** $\mathtt{A}$ **with advice** computes the output sequence $\mathtt{A}^{\phi}(I) = (y_1, \ldots, y_n)$ such that y_i is computed from $\phi, x_1, \ldots, x_i$, where ϕ is the content of the advice tape, i. e., an infinite binary sequence. $\mathtt{A}$ is **c- competitive with advice complexity $b(n)$** if, for every n and for each input sequence I of length at most n, there exists some ϕ such that $\mathtt{A}^{\phi}(I)$ is c-competitive and at most the first $b(n)$ bits of ϕ have been accessed during the computation of $\mathtt{A}^{\phi}(I)$.*

We can see an advice of b advice bits as a possibility for the algorithm to choose from 2^b different strategies. Here, the advice addresses which of the 2^n variants should be used. We say that c is a lower bound on the **advice complexity** of P, if we can prove the existence of an adversary that can produce, for any set of 2^b deterministic algorithms, an instance for which none of the algorithms can be better than c-competitive.

Part I

Reoptimization of Hard Problems

Chapter 2

Reoptimization of the Steiner Tree Problem

2.1 Introduction

The Steiner tree reoptimization problem in general weighted graphs was previously investigated in [10, 14, 18, 23, 43] for various types of local modifications. For the $\mathcal{APX}$-hard variant of the Steiner tree problem with edge costs restricted to integers from a fixed interval, the corresponding reoptimization problem even admits a polynomial-time approximation scheme [23].

Eight reoptimization variants (insertion and deletion of terminal or non-terminal vertices, increasing and decreasing edge costs, and changing the status of vertices from terminal to non-terminal and vice versa) been shown to be $\mathcal{NP}$-hard on graphs with edge costs restricted to 1 and $1 + \gamma$ [18]. The best approximation algorithms for the four reoptimization variants considered in [10] (a terminal becomes a non-terminal or vice versa; the cost of an edge increases or decreases) achieve a constant approximation ratio in metric graphs. Additionally, on β-metric graphs, for any $\beta < 1$, all of these four cases permit, in contrast to the non-reoptimization problem, a PTAS. When the local modification, however, consists in removing vertices, that the Steiner tree reoptimization is as hard to approximate as the original problem.

The two algorithmically most interesting reoptimization variants are the addition of terminal and of nonterminal vertices. For these modifications, the $\mathcal{APX}$-hardness of the corresponding reoptimization variants has also been shown in [18], which solves also the analogous open problem for reoptimization in Steiner trees with arbitrary edge costs. We cite the corresponding theorems in Section 2.2.

Escoffier et al. [43] designed simple linear-time algorithms for metric input instances ($\beta = 1$) which achieve an approximation ratio of 3/2. Using the same algorithms, but a much more complex and technically involved analysis, we prove a $(1/2 + \beta)$-approximation for graphs satisfying a sharpened β-triangle inequality. Note that the ratio $(1/2 + \beta)$ tends to $3/2$ for β tending to 1 and to 1 for β tending to $1/2$. These proofs employ a 2β-approximation algorithm for the classical non-reoptimization version of the Steiner tree problem in β-metric graphs which may be of independent interest since it improves over the previously best known ratio for any $\beta < 1/2 + \ln(3)/4 \approx 0.775$. These results are also published in [17, 18].

2.2 Reoptimization Hardness

For the hardness analysis of the Reopt-$(1, 1 + \gamma)$-STP-*lm* for $\gamma > 0$ and the Reopt-Δ_β-STP-*lm* for $\beta > 1/2$ and $lm \in \{AddTerm, AddNonTerm\}$, a special variant of the SAT problem was used, namely E*s*-OCC-MaxE*k*SAT.

MaxSAT is the optimization problem of finding a variable assignment satisfying a maximum number of clauses in a given Boolean formula in conjunctive normal form. By E*s*-OCC-MaxE*k*SAT, we denote the restriction of MaxSAT to input formulas where all clauses contain exactly k literals and every variable occurs exactly s times.

For showing the hardness of approximation of the $(1, 1 + \gamma)$-Steiner tree reoptimization problem with adding vertices, a gap-preserving reduction from E*s*-OCC-MaxE*k*SAT was used.

With this, showing the APX-hardness for adding vertices can be done by using the $\mathcal{APX}$-hardness of E5-OCC-MaxE3SAT [44]. For removing a terminal or a non- terminal, a simple argumentation shows the corresponding reoptimization problem to be as hard as the original problem. This leads to the following two theorems, for details see [18].

Theorem 2.1 (Böckenhauer et al. [18]) *The Reopt-*$(1, 1 + \gamma)$*-STP-lm for any* $\gamma > 0$ *and the Reopt-*Δ_β*-STP-lm for* $lm \in \{AddTerm, AddNonTerm\}$ *and* $\beta > 1/2$ *are* $\mathcal{APX}$*-hard.*

Note that Theorem 2.1 implicitly provides an alternative proof for the $\mathcal{APX}$-hardness of the problems Min-$(1, 1 + \gamma)$-STP and Min-Δ_β-STP.

All reductions require a transformation of an instance of the satisfiability problem into a $(1, 1 + \gamma)$-Steiner tree instance.

2.3 Approximation Algorithm

We start this section with an approximation algorithm for the Min-Δ_β-STP. This algorithm, besides being interesting by itself, will be useful for some of the subsequent approximation algorithms for reoptimization.

In any instance $(G, S, cost)$ consisting of terminals only, i. e., if $S = V(G)$, any minimum spanning tree is an optimal solution to the minimum Steiner tree problem. Intuitively speaking, the minimum Steiner tree problem can be viewed as the problem of finding a subset Y of non-terminals which minimizes the cost of a minimum spanning tree on $S \cup Y$ over all possible choices of Y. But also in those graphs in which the optimal solutions contain non-terminal vertices, the minimum spanning tree on the set of terminals gives a useful approximation of the minimum Steiner tree. To estimate the quality of this approximation, we need the following technical lemma for the subsequent approximability result.

Lemma 2.1 *Given an input instance $(G, S, cost)$ for Min-Δ_β-STP, for $1/2 \leq \beta \leq 1$, and a minimum Steiner tree T_{Opt} for this instance, let $T_1, \ldots, T_k$ be the maximal subtrees of T_{Opt} consisting of non-terminals only. For any T_i, let $N(T_i)$ be the set of neighbors of T_i in T_{Opt}. Then there exists a connected subgraph H of G with the following properties:*

(1) $V(H) = S$,

(2) H contains a cycle on the vertices of $N(T_i)$ for all $i \in \{1, \ldots, k\}$, and

(3) $cost(H) \leq 2\beta \cdot cost(T_{\text{Opt}})$.

Proof. If there exists an optimal solution T_{Opt} without non-terminals, then $H = T_{\text{Opt}}$ is a minimum spanning tree and thus $cost(H) = cost(T_{\text{Opt}}) \leq 2\beta \cdot cost(T_{\text{Opt}})$.

Let $T_1, \ldots, T_k$ be the maximal subtrees of T_{Opt} consisting of non-terminals only. For all $i \in \{1, \ldots, k\}$, let T'_i be the subtree of T_{Opt} containing exactly the vertices $V(T_i) \cup N(T_i)$. Let C_i be the cycle on the vertices of $N(T_i)$ as ordered by a depth-first traversal of T'_i (see Fig. 2.1). In T'_i, there are no neighboring terminals, which means that there do not exist any two terminals $v_1, v_2 \in V(T'_i)$ such that $\{v_1, v_2\} \in E(T'_i)$. This implies that every edge in C_i is a shortcut of at least two edges from T_{Opt}. From this, and since the depth-first traversal visits every edge exactly two times, the cost of C_i can be estimated by $cost(C_i) \leq 2\beta \cdot cost(T'_i)$.

Let $E_{\text{circus}} = \bigcup_{1 \leq i \leq k} E(C_i)$ be the set of all cycle edges and let $E_{\text{arbor}} = E_{Opt} - \bigcup_{1 \leq i \leq k} E(T'_i)$ be the set of all edges between terminals in T_{Opt}. The graph H is the union of the cycles C_i, for all $i \in \{1, \ldots, k\}$, together with the subtrees of T_{Opt} which contain terminals only, i. e., $V(H) = S$ and $E(H) = E_{\text{circus}} \cup E_{\text{arbor}}$. Thus, H satisfies the constraints (1) and (2). The graph H is connected since the terminals inside the subtrees T'_i are connected by the cycle C_i. Moreover, the subtrees T'_i are connected either by a common vertex or by edges from E_{arbor}.

We know that $cost(T_{\text{Opt}}) = \sum_{i=1}^{k} cost(T'_i) + cost((E_{arbor}))$. By definition of T'_i, any two trees T'_j, T'_l are edge-disjoint, therefore $cost(E_{\text{circus}}) \leq$

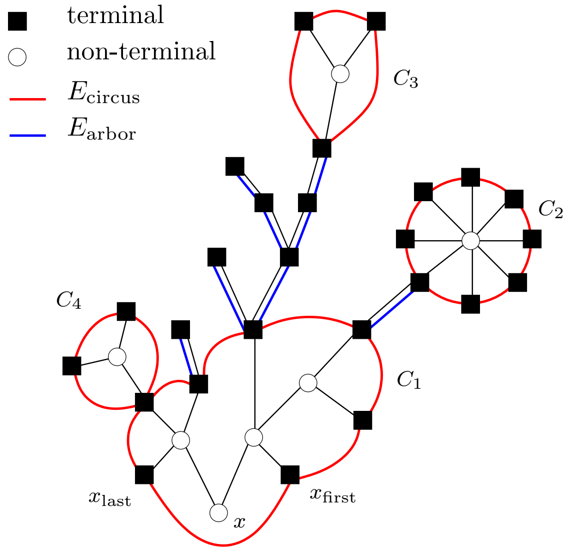


Figure 2.1: The construction in the proof of Lemma 2.1

$2\beta \cdot \sum_{i=1}^{k} cost(T_i')$. This implies

$$\begin{aligned} cost(H) &= cost(E_{\text{circus}}) + cost(E_{\text{arbor}}) \\ &\leq 2\beta \cdot \sum_{i=1}^{k} cost(T_i') + cost(E_{\text{arbor}}) \leq 2\beta \cdot cost(T_{\text{Opt}}), \end{aligned}$$

which proves that H also satifies constraint (3). □

Now, we are ready for the following theorem.

Theorem 2.2 *Let G be graph with cost function cost satisfying the β- triangle inequality for $1/2 \leq \beta \leq 1$. Let S be a set of terminals. Then the minimum spanning tree on S is a 2β-approximation of the minimum Steiner tree for the instance $(G, S, cost)$.*

Proof. This follows directly from constraints (1), (2), and (3) of Lemma 2.1. □

2.4 Reoptimization Algorithms

Now, we consider the Steiner tree reoptimization problem with the local modifications of adding a non-terminal and adding a terminal to the graph.

Algorithm 2.1 Approximation algorithm for Reopt-Δ_β-STP-AddNonTerm

```
Input: $(G = (V, E), S, cost)$, $T_{\text{Old}}$, $(G_{\text{New}} = (V \cup v_{\text{New}}, E \cup \{\{v_{\text{New}}, x\} \mid x \in V\}), S, cost)$
1: Compute the minimum spanning tree $T_{\text{MST}}$ on the vertex set $S \cup \{v_{\text{New}}\}$.
2: Compute the best solution $T_{\text{Alg}}$ between $T_{\text{Old}}$ and $T_{\text{MST}}$.
Output: The Steiner tree $T_{\text{Alg}}$.
```

2.4.1 Adding a Non-Terminal

First, we investigate the Steiner tree reoptimization problem with the local modification of adding a non-terminal. We assume that the new instance satisfies the β-triangle inequality for the same β as the old instance. We design an algorithm for Reopt-Δ_β-STP-AddNonTerm that outputs the better of the following two feasible solutions: The first feasible solution is simply the given optimal solution to the old instance, the second is obtained by computing a minimum spanning tree on the terminals together with the newly inserted vertex. This procedure is shown in Algorithm 2.1.

For analyzing the cost of T_{MST} as computed in Algorithm 2.1, we want to compare it to an optimal solution T_{Opt} for the new instance. For this comparison, we deal with every subtree of T_{Opt} rooted in a neighbor of v_{New}, together with its connection to v_{New}, separately. For our estimations, we need the following technical lemma which is a generalization of Lemma 2.1.

Lemma 2.2 *Let G be a graph and let $S \subseteq V$ be a set of terminals in G. Let T be a subtree of G rooted in some non-terminal vertex x of degree ≥ 2, and let $S_T = S \cap V(T)$ be the set of terminals in T. Let $v_{\text{New}} \in V - V(T)$ be one additional vertex. Let x_{first} be the first and x_{last} be the last terminal in T as found by a depth-first search starting from x. Then there exists a connected subgraph H with $V(H) = \{v_{\text{New}}\} \cup S_T$ with costs $cost(H) \leq \beta \cdot cost(v_{\text{New}}, x) + 2\beta \cdot cost(T) - \beta \cdot cost(x, x_{\text{last}})$.*

Proof. According to Lemma 2.1, there exists a connected subgraph H' of G on the vertices of S_T satisfying $cost(H') \leq 2\beta \cdot cost(T)$.

There exists a maximal subtree T_x of T consisting of non-terminals only which contains x as described in the proof of Lemma 2.1. Moreover, there exists a cycle C_x on the neighbors of T_x such that the edge $\{x_{\text{first}}, x_{\text{last}}\}$ is contained in C_x.

From C_x, we construct a path P_x containing $N(T_x)$ by deleting the edge $\{x_{\text{first}}, x_{\text{last}}\}$. The desired graph H is now obtained from H' by substituting P_x for C_x and adding the edge $\{v_{\text{New}}, x_{\text{first}}\}$ (see Fig. 2.2). It remains to show that the cost of H satisfies the constraint of the lemma. By T'_x we denote the subtree of T on the vertices from $V(T_x) \cup N(T_x)$.

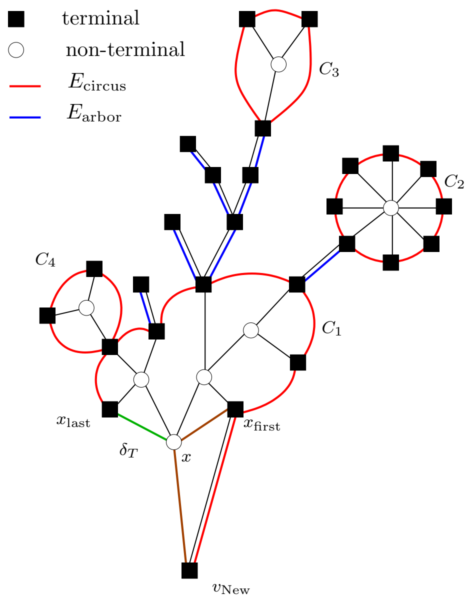


Figure 2.2: The construction in the proof of Lemma 2.2

As already shown in the proof of Lemma 2.1, analyzing a depth-first traversal of T'_x leads to $cost(C_x) \leq 2\beta \cdot cost(T'_x)$. Without loss of generality, such a depth-first traversal may start from vertex x. The vertices x_{first} and x_{last} are the first and the last terminals on the traversal path. This traversal path starts with a simple path $P_{x,x_{\text{first}}}$ from x to x_{first} and ends with a simple path $P_{x_{\text{last}},x}$ from x_{last} to x.

In the proof of Lemma 2.1, we have seen that shortening the subpath of the traversal path between two consecutive terminals on the traversal path generates an edge whose cost can be bounded from above by β times the cost of the corresponding subpath.

Summing over all these shortened edges on the path P_x from x_{first} to x_{last} yields a cost of at most $2\beta \cdot cost(T'_x) - \beta \cdot cost(P_{x,x_{\text{first}}}) - \beta \cdot cost(P_{x_{\text{last}},x})$, since every edge of T'_x is part of exactly two of the corresponding subpaths for the complete cycle C_x, and only the edges on the paths $P_{x,x_{\text{first}}}$ and $P_{x_{\text{last}},x}$ are used exactly once, since the cost of the edge from x_{last} to x_{first} is not included in the above sum.

As already mentioned above, H is obtained by adding the edge $\{v_{\text{New}}, x_{\text{first}}\}$ to P_x. Thus,

$$cost(H) = cost(P_T) + cost(v_{\text{New}}, x_{\text{first}}), \text{ moreover}$$
$$cost(v_{\text{New}}, x_{\text{first}}) \leq \beta \cdot (cost(v_{\text{New}}, x) + cost(P_{x_{\text{first}},x})).$$

From this, we have

$$\begin{aligned} cost(H) \leq & 2\beta \cdot cost(T) - \beta \cdot cost(P_{x,x_{\text{first}}}) - \beta \cdot cost(P_{x_{\text{last}},x}) \\ & + \beta \cdot (cost(v_{\text{New}}, x) + cost(P_{x_{\text{first}},x})) \\ \leq & 2\beta \cdot cost(T) + \beta \cdot cost(v_{\text{New}}, x) - \beta \cdot cost(x, x_{\text{last}}) \end{aligned}$$ □

In the following, we estimate the approximation ratio of Algorithm 2.1.

Theorem 2.3 *For any $1/2 \leq \beta \leq 1$, Algorithm 2.1 is a linear-time $\left(\frac{1}{2} + \beta\right)$-approximation algorithm for Reopt-Δ_β-STP-AddNonTerm.*

Proof. Constructing a spanning tree T_{MST} can be done in linear time, where the size of the input is measured in the number of edges.[1] Let T_{Opt} be an optimal solution for the new instance (G_{New}, S, c), and let T_{Alg} be the Steiner tree computed by Algorithm 2.1. If v_{New} does not occur in T_{Opt}, then T_{Old} and thus also T_{Alg} is an optimal solution.

Thus, we assume that $v_{\text{New}} \in V(T_{\text{Opt}})$. Let $\{x_1, \ldots x_k\}$ be the set of neighbors of v_{New} in T_{Opt}. By removing v_{New} from T_{Opt}, we get a forest of k trees $T_1, \ldots, T_k$, where T_i is rooted in x_i. Let γ_1 denote the cost of all edges adjacent to v_{New} in T_{Opt}, i. e., $\gamma_1 = \sum_{i=1}^{k} cost(v_{\text{New}}, x_i)$, and let $\gamma_2 = \sum_{i=1}^{k} cost(T_i)$ denote the sum of costs of all trees T_i. Thus, the cost of the optimal solution satisfies $cost(T_{\text{Opt}}) = \gamma_1 + \gamma_2$. We get a solution for the old instance by connecting the vertices x_1 to x_k by a path $P = (x_1, \ldots, x_k)$. The cost of this path can be estimated as $cost(P) \leq 2\beta \cdot \gamma_1$. The path P together with the trees T_i constitute a solution of cost greater than or equal to $cost(T_{\text{Old}})$, this implies

$$cost(T_{\text{Old}}) \leq cost(P) + \sum_{i=1}^{k} cost(T_i) \leq 2\beta \cdot \gamma_1 + \gamma_2. \tag{2.1}$$

For each tree T_i, we can construct a graph H_i on the terminals of T_i together with v_{New} as described in Lemma 2.2. If the vertex x_i of T_i is a non-terminal,

$$cost(H_i) \leq \beta \cdot cost(v_{\text{New}}, x_i) + 2\beta \cdot cost(T_i) \tag{2.2}$$

follows from Lemma 2.2, for all $\frac{1}{2} \leq \beta \leq 1$. If the root vertex x_i is a terminal, adding the edge $\{v_{\text{New}}, x_i\}$ to the graph constructed according to Lemma 2.1 yields

$$cost(H_i) \leq cost(v_{\text{New}}, x_i) + 2\beta \cdot cost(T_i), \tag{2.3}$$

for all $\frac{1}{2} \leq \beta \leq 1$. Note that (2.2) implies (2.3) also for the trees rooted in a non-terminal.

[1]Note that the minimum spanning tree on a graph $G = (V, E)$ can be computed in $O(|E| + |V| \log |V|)$ [37]. We deal with complete graphs, so the size of the input is $O(|E|) = O(|V|^2)$.

Algorithm 2.2 Approximation algorithm for Reopt-Δ_β-STP-AddTerm

```
Input: $(G = (V, E), S, cost)$, $T_{\text{Old}}$, $(G_{\text{New}} = (V \cup v_{\text{New}}, E \cup \{\{v_{\text{New}}, x\} \mid x \in V\}), S \cup \{v_{\text{New}}\}, cost)$
1: Compute the minimum spanning tree $T_{\text{MST}}$ over the set of vertices $S \cup \{v_{\text{New}}\}$.
2: Compute $T_{\text{Old}^+}$ from $T_{\text{Old}}$ by choosing the cheapest edge connecting $v_{\text{New}}$ with $S$.
3: Compute the best solution $T_{\text{Alg}}$ between $T_{\text{Old}}$ and $T_{\text{MST}}$.
Output: The Steiner tree $T_{\text{Alg}}$.
```

Let H be the union of the graphs $H_1, \ldots, H_k$ as a subgraph of G, i. e., $V(H) = S \cup \{v_{\text{New}}\}$ and $E(H) = \bigcup_{i=1}^{k} E(H_i)$. Then

$$\begin{aligned} cost(T_{\text{MST}}) \leq cost(H) = \sum_{i=1}^{k} cost(H_i) &\leq \sum_{i=1}^{k} (cost(v_{\text{New}}, x_i) + 2\beta \cdot cost(T_i)) \\ &\leq \sum_{i=1}^{k} cost(v_{\text{New}}, x_i) + 2\beta \cdot \sum_{i=1}^{k} cost(T_i) \\ &\leq \gamma_1 + 2\beta \cdot \gamma_2. \end{aligned} \tag{2.4}$$

Summing Equations (2.1) and (2.4) yields

$$\begin{aligned} 2 \cdot cost(T_{\text{Alg}}) &\leq cost(T_{\text{Old}}) + cost(T_{\text{MST}}) \\ &\leq 2\beta \cdot \gamma_1 + \gamma_2 + \gamma_1 + 2\beta \cdot \gamma_2 \\ &\leq (1 + 2\beta) \cdot (\gamma_1 + \gamma_2) = (1 + 2\beta) \cdot cost(T_{\text{New}}), \end{aligned}$$

and thus $cost(T_{\text{Alg}}) \leq \frac{1+2\beta}{2} \cdot cost(T_{\text{New}}) = \left(\frac{1}{2} + \beta\right) \cdot cost(T_{\text{New}})$. □

2.4.2 Adding a Terminal

Now, we consider the case where the inserted vertex is a terminal. Here, the old optimal solution is not feasible for the new instance. Therefore, we analyze two different candidates for a good feasible solution. The first candidate is a minimum spanning tree for the new instance on all terminals including the newly added one. The second candidate is obtained by connecting the inserted terminal to a terminal in the old optimal solution by the cheapest edge possible. This procedure is shown in Algorithm 2.2.

For analyzing the approximation ratio of Algorithm 2.2, we compare the costs of the computed solutions T_{MST} and T_{Old^+} to the costs of an optimal solution for the new instance. We distinguish two cases for the proof, according to

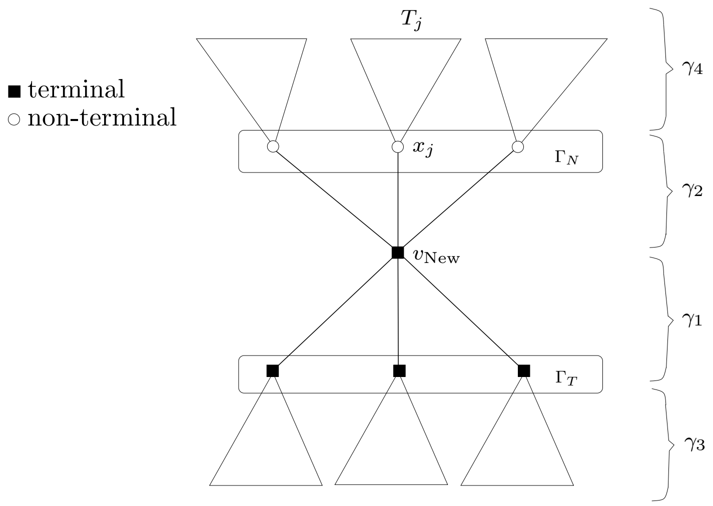


Figure 2.3: The structure of the new optimal solution in the proof of Theorem 2.4

whether there exists a non-terminal among the neighbors of v_{New} in an optimal solution T_{Opt} for the new instance or not. The proof for the case where there is a neighboring non-terminal will again make use of Lemma 2.2.

Theorem 2.4 *For any $\frac{1}{2} \leq \beta \leq 1$, Algorithm 2.2 is a linear-time $\left(\frac{1}{2}+\beta\right)$-approximation algorithm for Reopt-Δ_β-STP-AddTerm.*

Proof. The construction of both T_{MST} and $T_{\mathrm{Old+}}$ is obviously possible in linear time. Let T_{Opt} be an optimal solution of the new minimum Steiner tree instance. As in the proof of Theorem 2.3, let T_{Old} be the optimal solution of the old instance and let T_{MST} and $T_{\mathrm{Old+}}$ be the outputs of step 2 and step 3 of Algorithm 2.2, respectively. Let $\{x_1, \ldots, x_k\}$ be the neighbors of v_{New} in T_{Opt}. By deleting v_{New} from T_{Opt}, we get a set of trees $T_1, \ldots, T_k$. In the following, we denote the set of terminal neighbors of v_{New} in T_{Opt} by Γ_{T} and the set of non- terminal neighbors by Γ_{N}. We denote by γ_1 the sum of the costs of edges connecting v_{New} with a terminal in T_{Opt}, i.e., $\gamma_1 = \sum_{x_i \in \Gamma_{\mathrm{T}}} cost(v_{\mathrm{New}}, x_i)$, and by γ_2 the sum of costs of edges connecting v_{New} with a non-terminal in T_{Opt}, i.e., $\gamma_2 = \sum_{x_i \in \Gamma_{\mathrm{N}}} cost(v_{\mathrm{New}}, x_i)$. Let γ_3 be the sum of the costs of all trees T_i where the root $x_i \in \Gamma_{\mathrm{T}}$, and let γ_4 be the sum of the costs of all trees T_j where the root $x_j \in \Gamma_{\mathrm{N}}$ (see Fig. 2.3). Hence, $cost(T_{\mathrm{Opt}}) = \gamma_1 + \gamma_2 + \gamma_3 + \gamma_4$.

We distinguish two cases according to the number of non-terminals adjacent to v_{New}.

Case 1: $|\Gamma_{\mathrm{N}}| = 0$, $(\gamma_2 = \gamma_4 = 0)$. First we look at the case where all vertices x_i in the neighbourhood of v_{New} are terminals. This implies $cost(T_{\mathrm{New}}) =$

$\gamma_1 + \gamma_3$. In the case where there is only one terminal x in the neighborhood of v_{New} in T_{Opt}, Algorithm 2.2 connects v_{New} by the edge $\{v_{\text{New}}, x\}$ in step 3 and finds the optimal solution. Thus, we may assume that $|\Gamma_{\text{T}}| \geq 2$. In order to estimate $\mathit{cost}(T_{\text{MST}})$, for each tree T_i, we first construct a graph H_i containing the terminals of T_i. By connecting v_{New} to H_i by the edge $\{v_{\text{New}}, x_i\}$, we get a graph H which consists of all terminals in G_{New}, i.e., $V(H) = S \cup \{v_{\text{New}}\}$. It is obvious that the cost of T_{MST} is smaller than $\mathit{cost}(H)$. Now we can estimate $\mathit{cost}(T_{\text{MST}})$ as follows:

$$\mathit{cost}(T_{\text{MST}}) \leq \sum_{i=1}^{k} \mathit{cost}(v_{\text{New}}, x_i) + \sum_{i=1}^{k} \mathit{cost}(H_i). \tag{2.5}$$

According to Lemma 2.1, the cost of H_i satisfies $\mathit{cost}(H_i) \leq 2\beta \cdot \mathit{cost}(T_i)$, and thus $\sum_{i=1}^{k} \mathit{cost}(H_i) \leq 2\beta \cdot \gamma_3$. Together with Equation (2.5), we get

$$\mathit{cost}(T_{\text{MST}}) \leq \gamma_1 + 2\beta \cdot \gamma_3. \tag{2.6}$$

From this, we get $\mathit{cost}(T_{\text{Old}^+}) = \mathit{cost}(T_{\text{Old}}) + \mathit{cost}(v_{\text{New}}, x_{\text{first}})$, where $\{v_{\text{New}}, x_{\text{first}}\}$ is the cheapest edge from v_{New} to a terminal in G. To estimate T_{Old^+}, we connect the terminals $x_1, \ldots, x_k$ by a path P. The union of P with the trees T_i constitutes a feasible solution for the old instance. So we know that the cost of the old optimal solution is at most the cost of the union of P and all T_i, i.e., $\mathit{cost}(T_{\text{Old}}) \leq \mathit{cost}(P) + \sum_{i=1}^{k} \mathit{cost}(T_i)$. For the cost of P, this implies $\mathit{cost}(P) \leq 2\beta \cdot \sum_{i=1}^{k} \mathit{cost}(v_{\text{New}}, x_i) - \beta \cdot \mathit{cost}(v_{\text{New}}, x_l) - \beta \cdot \mathit{cost}(v_{\text{New}}, x_m)$, where $x_l \neq x_m$, and thus $\mathit{cost}(T_{\text{Old}}) \leq 2\beta \cdot \gamma_1 + \gamma_3 - \beta \cdot \mathit{cost}(v_{\text{New}}, x_l) - \beta \cdot \mathit{cost}(v_{\text{New}}, x_m)$. Without loss of generality, let $\mathit{cost}(v_{\text{New}}, x_l) < \mathit{cost}(v_{\text{New}}, x_m)$. Then we can estimate $\mathit{cost}(T_{\text{Old}^+})$ as

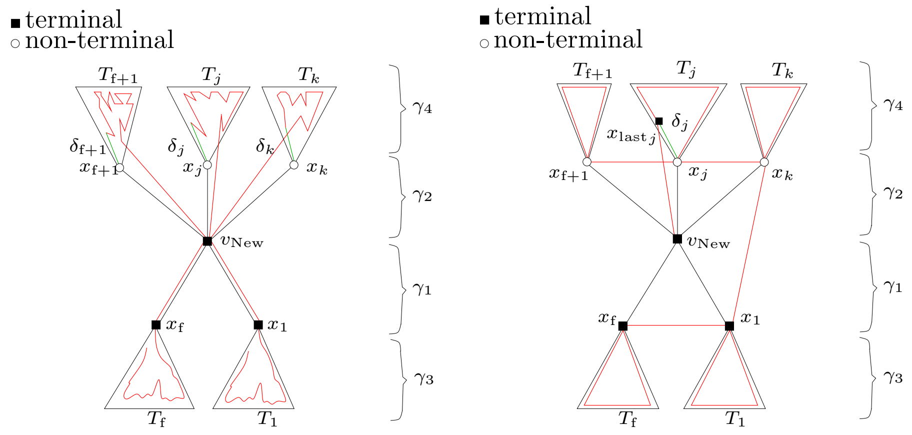
$$\begin{aligned} \mathit{cost}(T_{\text{Old}^+}) &\leq 2\beta \cdot \gamma_1 + \gamma_3 - 2\beta \cdot \mathit{cost}(v_{\text{New}}, x_l) + \mathit{cost}(v_{\text{New}}, x_l) \\ &\leq 2\beta \cdot \gamma_1 + \gamma_3. \end{aligned} \tag{2.7}$$

By adding (2.6) and (2.7), we get

$$\begin{aligned} 2 \cdot \mathit{cost}(T_{\text{Alg}}) &\leq \mathit{cost}(T_{\text{MST}}) + \mathit{cost}(T_{\text{Old}^+}) \leq \gamma_1 + 2\beta \cdot \gamma_3 + 2\beta \cdot \gamma_1 + \gamma_3 \\ &\leq (1 + 2\beta) \cdot (\gamma_1 + \gamma_3) \qquad = (1 + 2\beta) \cdot \mathit{cost}(T_{\text{New}}), \end{aligned}$$

and thus $\mathit{cost}(T_{\text{Alg}}) \leq (1 + 2\beta)/2 \cdot \mathit{cost}(T_{\text{New}}) = (1/2 + \beta) \cdot \mathit{cost}(T_{\text{New}})$.

Case 2: $|\boldsymbol{\Gamma}_{\mathbf{N}}| \geq 1$. Let $\Gamma_{\text{T}} = \{x_1, \ldots, x_f\}$ and $\Gamma_{\text{N}} = \{x_{f+1}, \ldots, x_k\}$ be the sets of terminals and non-terminals in the neighborhood of v_{New}, respectively, for some $0 \leq f < k$ in T_{Opt}. To estimate the cost of T_{MST}, we first construct, for each tree T_i with $i \in \{1, \ldots, f\}$, a spanning graph as already described in case 1. For each tree T_j with $j \in \{f + 1, \ldots, k\}$, we construct a spanning graph



(a) Case 2: construction of the MST

(b) Case 2: Construction of T_{Old^+}

Figure 2.4: The two feasible solutions in case 2 of the proof of Theorem 2.4

connected with v_{New} as described in Lemma 2.2 (see Fig. 2.4(a)). This implies

$$cost(T_{\mathrm{MST}}) \leq \gamma_1 + 2\beta\gamma_3 + \beta\gamma_2 + 2\beta \cdot \gamma_4 - \sum_{j=f+1}^{k} \beta cost(x_j, x_{\mathrm{last}\,j}), \tag{2.8}$$

where $x_{\mathrm{last}\,j}$ is the last terminal of a depth-first traversal of T_j as described in Lemma 2.2, for $j \in \{f+1, \ldots, k\}$.

For estimating the cost of T_{Old^+} (see Fig. 2.4(b)), we connect the vertices $x_1, \ldots, x_k$ by a path P, analogously to case 1. The union of P and the trees T_i, for $i = 1, \ldots, k$, gives a feasible solution for the old instance. Thus, $cost(T_{\mathrm{Old}}) \leq cost(P) + \gamma_3 + \gamma_4$. The cost of P can be bounded from above by $cost(P) \leq 2\beta \cdot (\gamma_1 + \gamma_2) - \beta \cdot cost(v_{\mathrm{New}}, x_j)$, where $x_j \in \{f+1, \ldots, k\}$.

From this, we get

$$\begin{aligned} cost(T_{\mathrm{Old}^+}) &\leq 2\beta(\gamma_1+\gamma_2) + \gamma_3 + \gamma_4 - \beta \cdot cost(v_{\mathrm{New}}, x_i) + cost(v_{\mathrm{New}}, x_{\mathrm{last}\,i}) \\ &\leq 2\beta(\gamma_1+\gamma_2) + \gamma_3 + \gamma_4 - \beta \cdot cost(v_{\mathrm{New}}, x_i) \\ &\quad + \beta \cdot (cost(v_{\mathrm{New}}, x_i) + cost(x_i, x_{\mathrm{last}\,i})) \\ &\leq 2\beta(\gamma_1+\gamma_2) + \gamma_3 + \gamma_4 + \beta \cdot cost(x_i, x_{\mathrm{last}\,i}). \end{aligned} \tag{2.9}$$

Adding (2.8) and (2.9) yields

$$\begin{aligned} 2 \cdot cost(T_{\text{Alg}}) &\leq cost(T_{\text{MST}}) + cost(T_{\text{Old}^+}) \\ &\leq (2\beta + 1) \cdot \gamma_1 + 3\beta \cdot \gamma_2 + (2\beta + 1) \cdot (\gamma_3 + \gamma_4) \\ &\quad + \beta \cdot cost(x_i, x_{\text{last}\,i}) - \sum_{j=f+1}^{k} \beta cost(x_j, x_{\text{last}\,j}) \\ &\leq (1 + 2\beta)(\gamma_1 + \gamma_2 + \gamma_3 + \gamma_4) \leq (1 + 2\beta) \cdot cost(T_{\text{New}}), \end{aligned}$$

and thus $cost(T_{\text{Alg}}) \leq \frac{(1+2\beta)}{2} cost(T_{\text{New}}) \leq (1/2 + \beta)\, cost(T_{\text{New}})$. □

2.5 Discussion

In the model of reoptimization, the approximation hardness of the Steiner tree problem in graphs with sharpened triangle inequality heavily depends on the considered local modification. For removing vertices from the graph, the reoptimization problem stays exactly as hard as the original Steiner tree problem and adding a vertex leads to an APX-hard problem, whereas changing the status of a vertex from terminal to a nonterminal or vice versa or changing the cost of an edge leads to a PTAS. In the case of adding a vertex, we improved the constant approximation ratio over that of the original Steiner tree problem. It remains as an open problem whether the reoptimization variant of changing edge-costs or the status of a vertex is APX-hard in general graphs.

Chapter 3

Hardness of Reoptimization with Multiple Given Solutions

3.1 Introduction

As the main result of this chapter, we prove an even stronger inapproximability result. It is well known that the traveling salesman problem (TSP) is not approximable with an approximation ratio of 2^n in an n-vertex graph with arbitrary edge weights [72], unless $\mathcal{P} = \mathcal{NP}$. Even the reoptimizations variant of changing the cost of one edge is not approximable [15].

Here, we consider a generalization of the reoptimization approach. We assume that we are given not only one optimal solution for a locally modified problem instance, but the k best solutions for some k which might even be exponentially large in the size of the input.

In Section 3.2, we give an overview of the reoptimization results and in Section 3.3, we consider a reoptimization situation where we are given a TSP instance together with the set of *all* optimal solutions. As a local modification, the cost of one edge is increased. We show that, even with this extra knowledge, it remains $\mathcal{NP}$-hard to compute a 2^n-approximate solution.

In Sections 3.6 and 3.7, we show that knowing all optimal solutions to a locally modified TSP instance does not help for TSP reoptimization.

We also prove in Section 3.5 that the $\mathcal{APX}$-hardness of Steiner tree reoptimization, where the local modification is the insertion of a new terminal vertex, carries over to the case of exponentially many given solutions. The proof technique used here can be easily generalized for many other optimization problems as well.

Then, in Section 3.6 and 3.7, we consider TSP reoptimization where, in addition to all optimal solutions, we are given exponentially many near-optimal solutions. We prove that even this multi-solution reoptimization variant, even if all solutions are very different, is as hard to approximate as the TSP itself.

3.2 Overview of Reoptimization Results

The results obtained in these chapter show that the hardness of reoptimization problems varies a lot. Obviously, lm-Reopt-U might be an easier problem than U, since we have access to an optimal solution for the original problem instance for free. Nevertheless, in most cases, the reoptimization variant of an $\mathcal{NP}$-hard optimization problem remains $\mathcal{NP}$-hard. This is due to the fact that, for most problems, it is possible to transform any given instance into a trivially solvable one by using a sequence of only polynomially many local modifications, see the paper by Böckenhauer *et al.* [24] for more details.

For some optimization problems and some local modifications, the corresponding reoptimization problem trivially admits a very good approximation since the old optimal solution itself is a good approximate solution for the new instance. For example, adding a single edge in an instance of a graph coloring problem can increase the cost of an optimal solution by at most one.

Let ΔTSP denote the restriction of the TSP to complete edge-weighted graphs where the edge-weights obey the *triangle inequality*, i. e., to instances $G = (V, E, cost)$ with

$$cost(u, v) \leq cost(u, w) + cost(w, v)$$

for all $u, v, w \in V$. It is well known that the ΔTSP is $\mathcal{APX}$-hard [68], and the best known approximation algorithm for it is due to Christofides [35] and achieves an approximation ratio of 3/2. We consider the local modification of increasing the cost of a single edge in such a way that the triangle inequality is still satisfied. We denote the resulting reoptimization problem by Inc-Edge-Reopt-ΔTSP. This reoptimization problem is $\mathcal{NP}$-hard [15, 16], but it admits an approximation algorithm which improves over the one from Christofides.

Theorem 3.1 (Böckenhauer *et al.* [15, 16]) *There exists a polynomial-time approximation algorithm for the Inc-Edge-Reopt-ΔTSP with an approximation ratio of* 7/5.

The proof of this theorem is based on the following idea which can be used for several other reoptimization problems as well. The algorithm considers two possible solutions. One of these solutions is the given optimal solution for the old instance, possibly adapted slightly to become feasible for the new instance. The other solution is based on guessing (by exhaustive search) a small part of the new optimal solution which can be proven to have relatively high costs and to use some known approximation algorithm to approximate the rest of the new solution. Usually, it can be shown that the first solution is good if the local modification does not change the cost of the optimal solution too much, and the second solution can be proven to be good in the case that the value of the optimal solution changes a lot after applying the local modification.

For some problems, the concept of reoptimization can help even more in lowering the approximation ratio. There exist $\mathcal{APX}$-hard optimization problems for which some corresponding reoptimization problems admit a PTAS. As an example, we consider the Steiner tree problem on graphs with bounded edge weights. The *Steiner tree problem (STP)* is known to be $\mathcal{APX}$-hard [8], even if all edge costs are drawn from the set $\{1, 2, \ldots, r\}$ for some integer constant $r \geq 2$. By Inc-Term-Reopt-r-STP, we denote the reoptimization variant of the STP on graphs with edge costs from $\{1, 2, \ldots, r\}$, where a non-terminal vertex of the graph becomes a terminal. The Inc-Term-Reopt-r-STP is $\mathcal{NP}$-hard [24], but it admits a PTAS.

Theorem 3.2 (Böckenhauer *et al.* [24]) *Let r be an arbitrary positive integer, $r \geq 2$. There exists a polynomial-time approximation scheme for the Inc-Term-Reopt-r-STP.*

The proof of this theorem is based on the following idea. For a desired approximation ratio of $1 + \varepsilon$, the algorithm computes the number $m = r \cdot \lceil 1/\varepsilon \rceil$. If there are few terminals (i. e., less than m), then the optimal Steiner tree can be computed using the Dreyfus-Wagner algorithm [40] whose running time is exponential only in the number of terminals. Otherwise, in the case of many terminals, just adding one edge from the new terminal to the old optimal solution, if necessary, gives a solution which can be proven to be $(1+\varepsilon)$-approximative.

On the other hand, for some problems, reoptimization does not help at all. As an example, consider the TSP with arbitrary edge weights. As already mentioned above, it is well known that the TSP is not approximable within a polynomial approximation ratio [72]. This result generalizes to the Inc-Edge-Reopt-TSP as follows.

Theorem 3.3 (Böckenhauer *et al.* [15, 16]) *It is $\mathcal{NP}$-hard to approximate the Inc-Edge-Reopt-TSP with an approximation ratio which is polynomial in the input size.*

The proof of this theorem is based on a *diamond graph construction* similar to the one used by Papadimitriou and Steiglitz for constructing instances of the TSP that are hard for local search [67]. We will not go into detail here since we will use a similar technique in the next section to prove an even stronger inapproximability result.

3.3 Knowing All Optimal Solutions

In this section, we consider a generalization of the reoptimization model. We assume that we are not only given a single optimal solution for the old instance, but that the set of *all* (possibly exponentially many) optimal solutions is available for free. We use the general TSP as an example to show that there exist optimization problems for which even this additional knowledge does not help at all for improving the approximability. We start with a formal definition of this TSP reoptimization variant.

Definition 3.1 (*Inc-Edge-Reopt$_{ALL}$*-TSP) *Let $G = (V, E)$ be a graph. We define **Inc-Edge-Reopt$_{ALL}$-TSP** as the problem given two edge weight functions $cost_{\mathrm{old}}$ and $cost_{\mathrm{new}}$ such that $(G, cost_{\mathrm{old}})$ and $(G, cost_{\mathrm{new}})$ are both admissible inputs for the TSP and such that $cost_{\mathrm{old}}$ and $cost_{\mathrm{new}}$ coincide, except for one edge $e_{change} \in E$ where $cost_{\mathrm{old}}(e_{change}) < cost_{\mathrm{new}}(e_{change})$. Moreover, we are given all optimal TSP solutions for $(G, cost_{\mathrm{old}})$. The goal is to compute an optimal TSP solution for $(G, cost_{\mathrm{new}})$.*

We measure the size of an instance of *Inc-Edge-Reopt$_{ALL}$*-TSP by the size of $(G, cost_{\mathrm{new}})$ only. The size needed for representing all (possibly exponentially many) optimal solutions for $(G, cost_{\mathrm{old}})$ is not taken into account.

For proving inapproximability, we give a reduction from the Hamiltonian cycle problem (HC) to *Inc-Edge-Reopt$_{ALL}$*-TSP. The HC problem is the problem to decide whether a given undirected unweighted graph G contains a Hamiltonian cycle or not.

For this reduction we employ a diamond graph construction similar to the construction of Papadimitriou and Steiglitz [67].

We start with the definition of the diamond graph.

Definition 3.2 (diamond graph) *The **diamond graph $D = (V, E)$** is a graph with 8 vertices and 9 edges with $V = \{N, S, W, E, u, x, y, z\}$ and $E = \{\{W, x\}, \{x, N\}, \{N, u\}, \{u, z\}, \{z, S\}, \{S, y\}, \{y, E\}, \{W, z\}, \{u, E\}\}$, see also Figure 3.1.*

For our proof, we need the following lemma from Papadimitriou and Steiglitz [67], see also Hromkovič [51].

Lemma 3.1 (Papadimitriou and Steiglitz [67]) *If the diamond graph D is an induced subgraph of a graph G with a Hamiltonian cycle C, then C traverses D in exactly one of the following two modes:*

1. *from north N to south S (or vice versa): $\ldots, N, x, W, z, u, E, y, S, \ldots$ (see Fig. 3.2(a)), or*
2. *from west W to east E (or vice versa): $\ldots, W, x, N, u, z, S, y, E, \ldots$ (see Fig. 3.2(b)).*

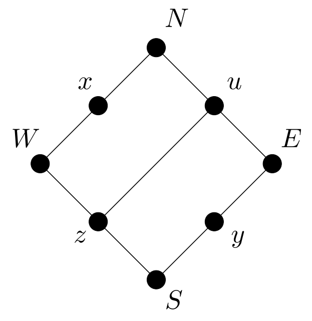


Figure 3.1: Diamond graph

That is, if a cycle C enters the diamond from the north, it must leave it from the south; and similarly with respect to the east-west vertices.

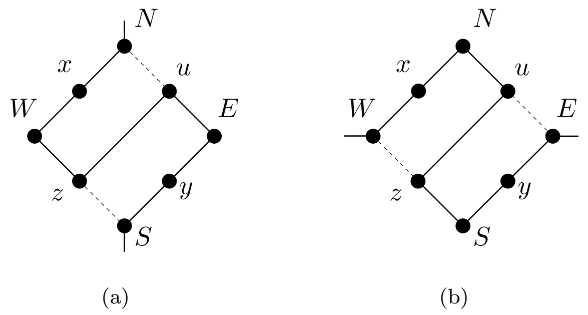


Figure 3.2: (a) Traversal of a diamond in north-south direction (b) traversal of a diamond in west-east direction

Theorem 3.4 *Under the assumption of $\mathcal{P} \neq \mathcal{NP}$, there does not exist any polynomial-time approximation algorithm with an approximation ratio of 2^n for Inc-Edge-Reopt$_{ALL}$-TSP, where n is the number of vertices in the instance.*

Proof. For proving this theorem, we give a reduction from the Hamiltonian cycle problem (HC) to the *Inc-Edge-Reopt*$_{ALL}$-TSP.

Let G_{HC} be an input instance for HC with $V(G_{HC}) = \{v_1, \ldots, v_k\}$ and $E(G_{HC}) = \{e_1, \ldots, e_m\}$. To construct an instance of *Inc-Edge-Reopt*$_{ALL}$-TSP, we first build an unweighted graph G_{TSP}. As a first step, we substitute every vertex $v_i \in V(G_{HC}), i \in \{1, ..., k\}$, by a diamond graph D_i with $V(D_i) = \{N_i, S_i, W_i, E_i, u_i, x_i, y_i, z_i\}$ as shown in Fig. 3.1.

Secondly, we add edges from every S- and N-vertex of every diamond D_i to every S- and N-vertex of all other diamonds D_j, $j \neq i$ (see Fig. 3.3(a)). Furthermore, for every edge $e_i = \{v_j, v_k\} \in E(G_{HC})$, the corresponding diamonds D_j and D_k are connected with two more edges: one between the W-vertex W_j of D_j and the E-vertex E_k of D_k and one edge between W_k and E_j (see Fig. 3.3(b)).

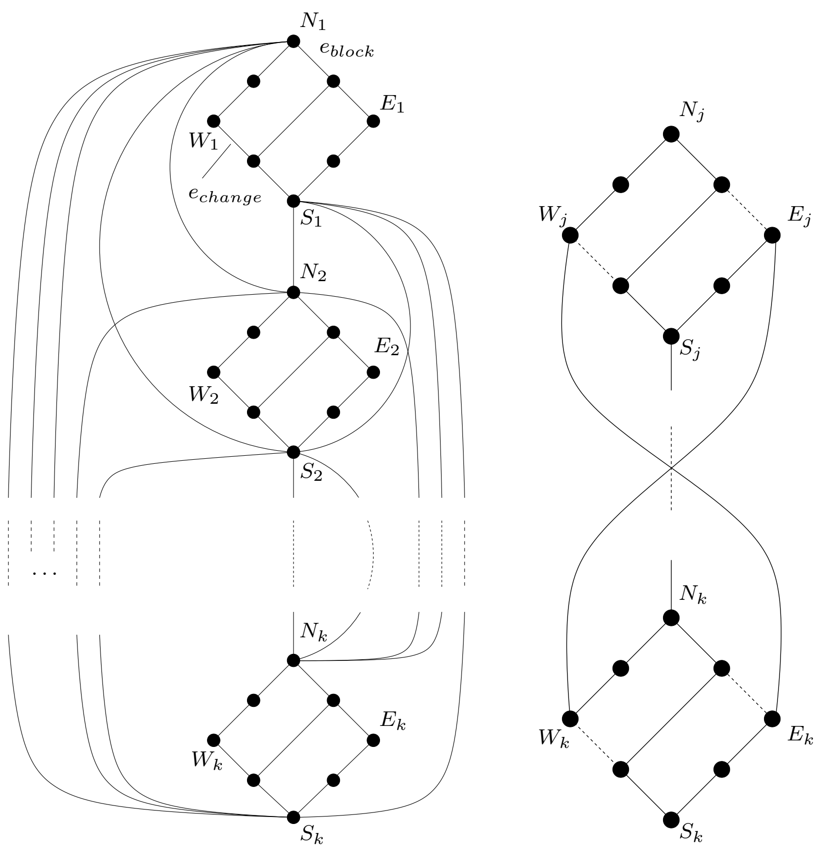


Figure 3.3: (a) North-south connections in the graph G_{TSP} (b) west-east connections in the graph G_{TSP} if $\{v_j, v_k\} \in E(G_{HC})$

Now we transform G_{TSP} into an instance $(G_{Reopt}, cost_{\text{old}}, cost_{\text{new}})$ of *Inc-Edge-Reopt*$_{ALL}$-TSP. Let $G_{Reopt} = (V, E) = K_{8k}$ be the complete graph on $8k$ vertices. We define $cost_{\text{old}}$ as follows: The edge $e_{block} = \{N_1, u_1\}$ plays a special role in our argumentation, we set $cost_{\text{old}}(e_{block}) = 1 + \varepsilon$. For all other edges

$e_j \in \{\{v_l, v_k\} \mid v_l, v_k \in V(G_{TSP})\}, e_j \neq e_{block}$, we define

$$cost_{\text{old}}(e_j) = \begin{cases} 1 & \text{if } e_j \in E(G_{TSP}) - \{e_{block}\} \\ M & \text{otherwise,} \end{cases}$$

where $M = 2^{9k}$.

We now consider the local modification of changing the cost of one edge in $cost_{old}$, leading to the new TSP instance $(G_{Reopt}, cost_{\text{new}})$. For this, we change the cost of the edge $e_{change} = \{W_1, z_1\}$ in D_1 from $cost_{\text{old}}(e_{change}) = 1$ to $cost_{\text{new}}(e_{change}) = M$.

In addition to $(G_{Reopt}, cost_{\text{old}}, cost_{\text{new}})$, we have to specify the set of all optimal solutions for $(G_{Reopt}, cost_{\text{old}})$. These are all the $2^k \cdot (k-1)!$ many Hamiltonian tours which traverse every diamond in north-south or south-north direction.

In the graph G_{TSP}, independently from the original graph G_{HC}, there exist $2^k \cdot (k-1)!$ many Hamiltonian cycles traversing all diamonds in north-south or south-north direction: The order of the diamonds can be varied arbitrarily, yielding $(k-1)!$ different possibilities, and all diamonds can be traversed either in north- south or south-north direction which leads to $(2^k \cdot (k-1)!)$ different Hamiltonian tours in north-south direction overall. Any of these Hamiltonian cycles leads to an optimal solution in $(G_{Reopt},\ cost_{\text{old}})$ with cost of $8 \cdot k$ because, for all edges $e_i \neq e_{block}$, $e_i \in E(G_{TSP})$, $cost_{\text{old}}(e_j) = 1$ holds and the edge e_{block} is not traversed by a run through the diamond D_1 in north-south direction.

Furthermore, if and only if there exists a Hamiltonian cycle in G_{HC}, there also exist one or more (exactly as many as in G_{HC}) Hamiltonian cycles in G_{TSP} that traverse every diamond in west-east direction. These Hamiltonian cycles lead to second-best solutions in G_{Reopt} of cost $8 \cdot k + \varepsilon$ because every traversal of the diamond D_1 in west-east direction has to use the edge e_{block}. Note that these west-east solutions do not use the edge e_{change}. A Hamiltonian cycle in G_{Reopt} traversing some diamonds in north-south direction and some other in west-east direction uses at least two edges of cost M due to the construction, leading to a cost of at least $8 \cdot k - 2 + 2 \cdot M$.

By increasing the edge cost of e_{change} from $cost_{\text{old}}(e_{change}) = 1$ to $cost_{\text{new}}(e_{change}) = M$, all optimal solutions in $(G_{Reopt}, cost_{\text{old}})$ get a cost of $8 \cdot k - 1 + M$ in $(G_{Reopt}, cost_{\text{new}})$. Therefore, in $(G_{Reopt}, cost_{\text{new}})$ there exists an optimal solution with cost of $8 \cdot k + \varepsilon$ if and only if there is a Hamiltonian cycles in G_{HC}. Otherwise, the old optimal solutions in $(G_{Reopt}, cost_{\text{old}})$ stay optimal in $(G_{Reopt}, cost_{\text{new}})$.

Thus, an approximation algorithm with an approximation ratio smaller than $(M + 8 \cdot k - 1)/(8 \cdot k + \varepsilon)$ would solve the HC problem. Due to $M = 2^{9k}$, we have

$$\frac{M + 8 \cdot k - 1}{8 \cdot k + \varepsilon} > 2^{8k}$$

for almost all values of k. Since the constructed graph G_{Reopt} has $8k$ vertices, the claim follows. $\square$

3.4 Reoptimization with k Given Solutions

We consider the following reoptimization variant of the STP with the local modification *lm* (e. g., adding a terminal vertex to the graph). Several local modifications have been considered in the literature, e. g., changing a terminal to a non-terminal or vice versa, changing the cost of an edge, adding a terminal or non-terminal, or removing a terminal or non-terminal. We will focus on the addition of a terminal vertex (together with all its incident edges) here. Formally, we consider the local modification $lm = AddTerm$, i. e., $V(G') = V(G) \cup \{v_{\text{New}}\}$, for some $v_{\text{New}} \notin V(G)$, $S' = S \cup \{v_{\text{New}}\}$, and c is the restriction of c' to $V(G)$.

Allowing the presence of more than one solution for the old instance leads to the multiple-solution variant of STP reoptimization which can be defined as follows.

Definition 3.3 (k-Sol-Reopt-STP-*lm*) *The **minimum Steiner tree k-solution reoptimization problem** with the local modification lm, **k-Sol-Reopt-STP-lm** for short, is the following optimization problem. The input consists of a STP instance $(G, S, cost)$, a sequence of the k best Steiner trees $T^{(1)}_{\text{Old}}, \ldots, T^{(k)}_{\text{Old}}$ for it (with ties broken arbitrarily), and a locally modified STP instance $(G', S', cost')$, with respect to the local modification lm. The goal is to find a minimum Steiner tree for $(G', S', cost')$.*

*We also allow the number of given solutions to depend on the size n of the input graph, in this case we call the problem **f(n)-Sol-Reopt-STP-lm**, for some non-decreasing function $f : \mathbb{N} \to \mathbb{N}$.*

Also for the TSP, we consider a multiple-solution reoptimization variant.

Definition 3.4 (k-Sol-Reopt-TSP-*lm*) *The **k-solution reoptimization TSP** with the local modification lm, **k-Sol-Reopt-TSP-lm** for short, is the following optimization problem. The input consists of a TSP instance $(G, cost_{old})$, a sequence of the k best Hamiltonian tours in G (with ties broken arbitrarily), and a locally modified TSP instance $(G', cost_{\text{new}})$, with respect to the local modification lm. The goal is to find a minimum-cost Hamiltonian tour for $(G', cost_{\text{new}})$.*

*We again allow the number of given solutions to depend on the size n of the input graph, in this case we call the problem **$f(n)$-Sol-Reopt-TSP-lm**, for some non-decreasing function $f : \mathbb{N} \to \mathbb{N}$.*

Also for TSP reoptimization, various different local modifications have been considered in the literature, like adding or deleting a vertex or changing the cost of an edge. Here, we will focus on the modification of increasing the cost of a single edge e_{change}, we write $lm = IncEdge$. Formally, for $G = (V, E)$ and $e_{change} \in E$, we define $G' = G$ and $cost_{\text{new}}(e) = cost_{old}(e)$ for all $e \neq e_{change}$ and $cost_{\text{new}}(e_{change}) > cost_{old}(e_{change})$.

Note that we explicitly allow the function f to be exponential. Nevertheless, we do not count the size of the representation of the given solutions as a part of the input size. This is justified by the observation that the set of given solutions often can be given in a compact representation.

3.5 Approximating the Multiple-Solution Steiner Tree Reoptimization Problem is Hard

In this section, we establish an inapproximability result for k-Sol-Reopt-STP-AddTerm. It was shown in [18] that Reopt-STP-AddTerm is $\mathcal{APX}$-hard. This means that there does not exist any polynomial-time approximation scheme for Reopt-STP-AddTerm, unless $\mathcal{P} \neq \mathcal{NP}$. For an introduction to the theory of approximation hardness, we refer to the books [5, 51]. We use a reduction from Reopt-STP-AddTerm to prove that, for any k, the problem k-Sol-Reopt-STP-AddTerm is exactly as hard to approximate.

For any (multiple-solution) Reopt-STP-AddTerm instance, let n denote the number of vertices of the old graph, i. e., of the graph for which the optimal solution is given.

Theorem 3.5 *The n^{n-2}-Sol-Reopt-STP-AddTerm is $\mathcal{APX}$-hard.*

Proof. We prove the claim by an AP-reduction from the Min-STRP-AddTerm, i. e., by a polynomial-time reduction preserving the approximation ratio (see [5,51] for a formal definition of AP-reductions). Consider an arbitrary Reopt-STP-AddTerm instance $I = (G = (V, E), S, c, T_{\text{Old}}, G' = (V', E'), S', c')$ where $V' = V \cup \{x_{\text{new}}\}$ for some $x_{\text{new}} \notin V$, $S' = S \cup \{x_{\text{new}}\}$, $E' = E \cup \{\{x_{\text{new}}, v\} \mid v \in V\}$ and c is a restriction of c' to the edges from E. From this, we construct a n^{n-2}-Sol-Reopt-STP-AddTerm instance and prove that any α-approximative solution for it can be re-transformed into an α-approximative solution for the Reopt-STP-AddTerm instance.

The idea of the construction is to substitute one terminal z of G (but not the newly added one) by a clique of n terminals which are very close to each other. Then many optimal solutions for this new graph can be computed by just replacing z by an arbitrary spanning tree on the newly introduced vertices. Since the overall structure of the graph does not change by this transformation, the knowledge of all of these exponentially many optimal solutions does not help to solve the new instance.

Formally, the constructed n^{n-2}-Sol-Reopt-STP-AddTerm instance $\hat{I}$ can be described as follows. The given old instance consists of a graph $\hat{G} = (\hat{V}, \hat{E})$, a terminal set $\hat{S}$ and an edge-cost function $\hat{c}$. Here, $\hat{V} = (V \setminus \{z\}) \cup \{z_1, \ldots, z_n\}$ for some $z \in S$ and new vertices $z_1, \ldots, z_n \notin V$, $\hat{S} = (S \setminus \{z\}) \cup \{z_1, \ldots, z_n\}$, and

$$\hat{cost}(e) = \begin{cases} cost(e) & \text{for all } e \in E \text{ such that } z_1, \ldots, z_n \notin e \\ cost(z, x) & \text{for } e = \{z_i, x\}, i \in \{1, \ldots, n\}, x \in V \setminus \{z\} \\ 0 & \text{for } e = \{z_i, z_j\}, i, j \in \{1, \ldots, n\} \end{cases}$$

According to Cayley's formula [34], there are n^{n-2} different trees on n distinguishable vertices. Let $k = n^{n-2}$. We construct k different optimal Steiner trees for $\hat{G}$ by substituting z in the given optimal Steiner tree T_{Old} for G by each of the k spanning trees on $\{z_1, \ldots, z_n\}$. We call the resulting trees $\hat{T}^{(1)}_{\text{old}}, \ldots, \hat{T}^{(k)}_{\text{old}}$.

The new instance is given by $\hat{G}' = (\hat{V}', \hat{E}')$, a terminal set $\hat{S}'$ and an edge-cost function $\hat{c}'$. Here, $\hat{V}' = \hat{V} \cup \{x_{\text{new}}\}$, $\hat{S}' = \hat{S} \cup \{x_{\text{new}}\}$, and

$$\hat{cost}'(e) = \begin{cases} \hat{cost}(e) & \text{for all } e \in \hat{E} \\ cost'(e) & \text{for all } e \in E' \\ cost'(\{z, x_{\text{new}}\}) & \text{for } e = \{z_i, x_{\text{new}}\}, i \in \{1, \ldots, n\} \end{cases}$$

A graphical illustration of this construction is given in Fig. 3.4.

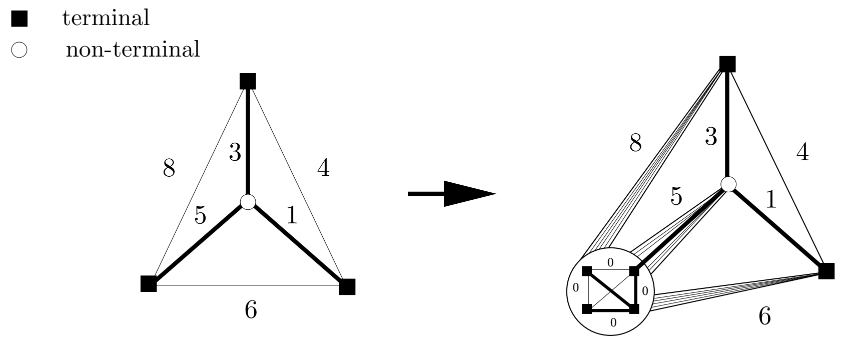


Figure 3.4: The construction from the proof of Theorem 3.5. An optimal Steiner tree is drawn in bold.

We first prove that the cost of an optimal solution for $\hat{I}$ equals the cost of an optimal solution for I. By construction, any spanning tree on $z_1, \ldots, z_n$ has zero cost. Thus, replacing z in a solution for I by any such tree yields a valid Steiner tree for $\hat{I}$ with the same cost. This implies that the costs of an optimal solution for $\hat{I}$ are at most the costs of an optimal solution for I. On the other hand, consider an arbitrary Steiner tree for $\hat{G}'$. Contracting the vertices $z_1, \ldots, z_n$ into one vertex z does not change the cost and yields a substructure of G' that spans all terminals from T'. This spanning substructure might contain some cycles, but deleting some edges to break the cycles does not increase the cost. This means that, for any feasible solution for $\hat{I}$, we can construct a feasible solution for I of at most the same cost. Hence, the costs of an optimal solution for $\hat{I}$ are at least as high as those of an optimal solution for I.

Now consider some α-approximative solution $\hat{T}_{\text{approx}}$ for $\hat{I}$. Following the argument above, we can transform $\hat{T}_{\text{approx}}$ into a feasible solution T_{approx} for I without increasing the cost. Let $Opt(I)$ and $Opt(\hat{I})$ denote the costs of optimal solutions for I and $\hat{I}$, respectively. Then, we have

$$cost(T_{\text{approx}}) \leq cost(\hat{T}_{\text{approx}}) \leq \alpha \cdot Opt(\hat{I}) = \alpha \cdot Opt(I),$$

meaning that also T_{approx} is α-approximative. □

Note that the edges of cost 0 in the above proof could be also replaced by edges of positive cost, as long as their total cost does not exceed the cost of one of the other edges. This construction can also easily be generalized to the other local modifications that were considered in [18].

The used technique of substituting one vertex by a large but cheap subgraph can also been applied to extend reoptimization hardness results to the multiple-solution case for other problems besides the STP.

3.6 Multi-Dimensional Diamond Graphs

We have seen in Section 3.3 that, knowing all optimal solutions to a locally modified instance, TSP is also hard to approximate, furthermore, we have seen an example in Section 3.5 where a large number of given optimal or near-optimal solutions does not help to improve the approximability of a reoptimization problem. But all the exponentially many given solutions in this example were quite similar in their global structure. Now, we will give some examples showing that also a large number of rather different solutions does not necessarily improve the solvability of a reoptimization problem.

As a prerequisite, we need an extension of the diamond graph construction from Papadimitriou and Steiglitz [67] as described in Definition 3.2 to what we call multi-dimensional diamond graphs. Using these multi-dimensional diamond graphs as building blocks, we will show in the following sections that the multiple-solution reoptimization TSP is neither approximable nor tractable by local search according to the well-known exchange neighborhood.

The main property of a diamond graph (see Lemma 3.1) is that, whenever it is a subgraph of a graph G with a Hamiltonian cycle and this cycle enters the diamond via the vertex I_i, then it has to traverse the diamond on the unique Hamiltonian path from I_i to O_i in D, for $i \in \{1, 2\}$. We call this a traversal of D in *dimension i* (see Fig. 3.2(a) for a traversal of D in dimension 1 and Fig. 3.2(b) for a traversal of D in dimension 2, were $I_1 = N, O_1 = S, I_2 = W$, and $O_2 = E$.).

We now extend the notation of a diamond graph to more than two dimensions. We call a graph D a k-dimensional diamond graph if there are k pairs of input and output vertices in D such that, if D is a subgraph of a Hamiltonian graph G, the Hamiltonian cycle C has to enter D via an input vertex, traverse all vertices in D and leave D via the corresponding output vertex. More precisely, we define the diamond graph property as follows.

Definition 3.5 (k-dimensional diamond graph) *Let $D = (V, E)$ be a subgraph of a graph G which contains a Hamiltonian cycle C. Let $I_D = \{I_1, \ldots, I_k\} \subset V(D)$ be a set of k vertices called **input vertices**, and let $O_D = \{O_1, \ldots, O_k\} \subseteq V(D)$, where $I_D \cap O_D = \emptyset$, be a set of k vertices called **output vertices**. Assume that D is connected to $G - D$ only via I_D and O_D. The graph D is called a **k-dimensional diamond graph**, if D has the following properties:*

(i) For every pair (I_k, O_k), there exists exactly one Hamiltonian path H_k from I_k to O_k in D.

(ii) Every Hamiltonian cycle C in G that enters D via a vertex $I_k \in I_D$ has to follow H_k and to leave D via O_k.

We now want to construct a 2^i-dimensional diamond graph D_i for every $i \geq 1$. We define D_i recursively as follows. The graph D_1 is the two-dimensional diamond from Definition 3.2. The graph D_i includes a graph D_{i-1} as a subgraph for $i \geq 2$. As an example, D_2 is shown in Fig. 3.5, including D_1 as a subgraph. Formally, we define D_i as follows.

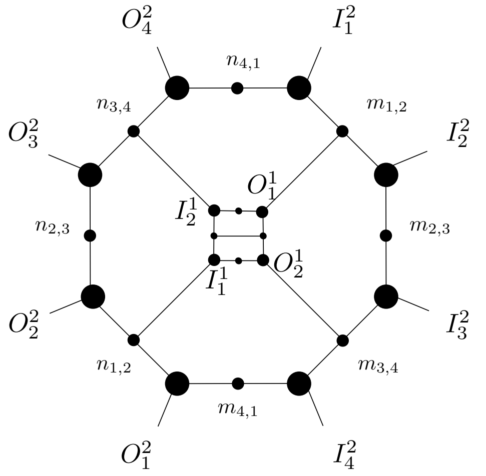


Figure 3.5: The 4-dimensional diamond graph D_2.

Definition 3.6 *The graph* $\boldsymbol{D_i = (V_i, E_i)}$*, for* $i \geq 2$*, contains* 2^{i+2} *vertices in an external cycle and a copy of* D_{i-1} *as a subgraph in the middle, connected by* 2^i *edges to the external cycle.*

The external cycle contains 2^i *input vertices* $I_1^i, \dots, I_{2^i}^i$*,* 2^i *output vertices* $O_1^i, \dots, O_{2^i}^i$ *and* 2^{i+1} *intermediate vertices* $m_{1,2}, m_{2,3}, \dots, m_{2^i,1}$*,* $n_{1,2}, \dots, n_{2^i,1}$*.*

In the external cycle, there are 2^{i+2} *edges* $\{I_1, m_{1,2}\}, \{m_{1,2}, I_2\}, \{I_2, m_{2,3}\}, \dots, \{I_{2^i}, m_{2^i,1}\}, \{m_{2^i,1}, O_1\}$*,*$\{O_1, n_{1,2}\}$*,*$\dots$*,*$\{O_{2^i}, n_{2^i,1}\}$*,*$\{n_{i,1}, I_1\}$*.*

Furthermore, there are edges for every odd $j \in \{1, 3, \dots, 2^i - 1\}$*, from* $n_{j,j+1}$ *to the input vertex* $I_{(j+1)/2}^{i-1}$ *of the* D_{i-1} *in the middle and from* $m_{j,j+1}$ *to the output vertex* $O_{(j+1)/2}^{i-1}$ *of the* D_{i-1} *in the middle, called* ***cross edges****.*

We now count the number of vertices and edges of D_i.

Observation 3.1 *For* $i \geq 1$*, the graph* D_i *contains* $2^{i+3} - 8$ *vertices.*

Proof. We prove the claim by induction. Obviously, $|V(D_1)| = 8 = 2^4 - 8$. By the induction hypothesis, $|V(D_i)| = 2^{i+3} - 8$. We know that the graph D_{i+1} has $2 \cdot 2^{i+1} + 2^{i+2} = 2^{i+3}$ vertices on the external cycle and $|V(D_i)|$ vertices in the middle. This yields $|V(D_{i+1})| = 2^{i+3} + |V(D_i)| = 2^{i+3} + 2^{i+3} - 8 = 2^{i+4} - 8$. □

Observation 3.2 *For any* $i \geq 1$*, the graph* D_i *contains* $2^{i+3} + 2^{i+1} - 11$ *edges.*

Proof. We prove the claim by induction. The graph D_1 has $9 = 2^4 + 2^2 - 11$ edges. By the induction hypothesis, $|E(D_i)| = 2^{i+3} + 2^{i+1} - 11$. We know that D_{i+1} has 2^{i+3} edges in the external cycle and 2^{i+1} cross edges plus the edges of D_i. This yields

$$\begin{aligned} |E(D_{i+1})| &= 2^{i+3} + 2^{i+1} + |E(D_i)| \\ &= 2^{i+3} + 2^{i+1} + 2^{i+3} + 2^{i+1} - 11 = 2 \cdot 2^{i+3} + 2 \cdot 2^{i+1} - 11 \\ &= 2^{i+4} + 2^{i+2} - 11. \end{aligned}$$ □

We now prove that D_i indeed is a 2^i-dimensional diamond graph.

Lemma 3.2 *The graph* D_i *is a* 2^i*-dimensional diamond graph with* $I = \{I_k^i \mid k \in \{1, \dots, 2^i\}\}$ *and* $O = \{O_k^i \mid k \in \{1, \dots, 2^i\}\}$*.*

Proof. We prove the claim inductively. We know that the claim holds in the case $i = 1$ due to [67]. We assume in the following that it holds for D_{i-1}. Let D_i, for $i \geq 2$, be a subgraph of a graph G with a Hamiltonian cycle C, such that D is connected to $G - D$ only via vertices from I and O. The cycle C can enter D_i only in one of the input or output vertices I_j, O_j, for $j \in \{1, \dots, 2^i\}$.

We call an edge $e = \{x, y\}$ of D_i *active* with respect to some path P in D_i, if both P and e can be used in C. In particular, every edge incident to an inner vertex of P (but not in $E(P)$) is inactive since every vertex has degree 2 in C.

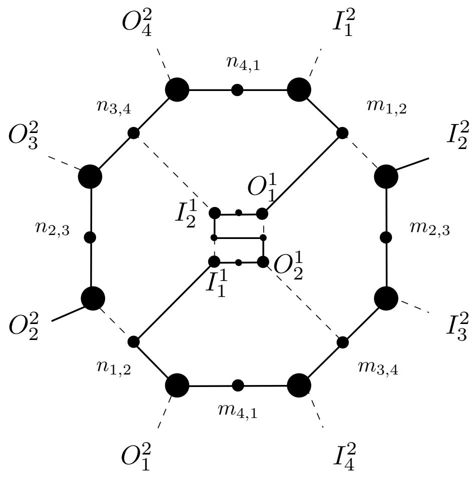


Figure 3.6: An example of a Hamiltonian path in D_2.

The *active degree* of a vertex x with respect to path P is the number of active edges incident to x.

If C enters D_i in the vertex I_k^i (without loss of generality, assume $k \in \{2, 4, \ldots, 2^i - 2\}$, the other cases can be handled symmetrically), there are only two reachable vertices, namely $m_{k-1,k}$ and $m_{k,k+1}$. The vertex $m_{k,k+1}$ has degree 2. Because of this, C has to proceed to this vertex, otherwise it cannot pass through it later.

The vertex $m_{k-1,k}$ has active degree 2, since the edge $\{m_{k-1,k}, I_k^i\}$ is not active with respect to C since C already uses two other edges incident to I_k^i.

It follows that the vertex $m_{k-1,k}$ lies on a path from the vertex $O_{\lceil k/2 \rceil}^{i-1}$ of the inner diamond to I_{k-1}^i. Thus, from the induction hypothesis, it follows that this path has to pass through the inner diamond D_{i-1} to the vertex $I_{\lceil k/2 \rceil}^{i-1}$. This means that all cross edges in D_i from the external cycle to the inner diamond except for $\{I_{\lceil k/2 \rceil}^{i-1}, n_{k-1,k}^i\}$ are not active with respect to $H_{k/2}^{i-1}$.

Then, all the intermediate vertices $m_{k,k+1}^i, \ldots, m_{2^i,1}^i, n_{1,2}, \ldots, n_{k-1,k}^i$ and $n_{k+1,k+2}^i, \ldots, n_{2^i,1}^i, m_{1,2}, \ldots, m_{k-2,k-1}^i$ have active degree 2. Hence, C has to use the paths from I_k^i to O_{k-1}^i and from O_k^i to I_{k-1}^i on the external cycle to visit all of these vertices. The edge $\{n_{k-1,k}^i, O_k^i\}$ cannot be part of C, since it closes a cycle together with the paths from O_k^i to I_{k-1}^i on the external cycle and the path from $m_{k-1,k}$ to $n_{k-1,k}$ via D_{i-1}, both of which have to be parts of C as seen above. Thus, C has to use the edge $\{O_{k-1}^i, n_{k-1,k}^i\}$.

This leads to the unique Hamiltonian path $H_k^i = I_k^i, \ldots, O_{k-1}^i, n_{k-1,k}^i, I_{\lceil k/2 \rceil}^{i-1}, H_{k/2}^{i-1}, O_{\lceil k/2 \rceil}^{i-1}$, $m_{k-1,k}$, $I_{k-1}^i, \ldots, O_k^i$ from I_k^i to O_k^i in D_i (see Fig. 3.6 for an example), proving condition (i) of the diamond graph definition. Condition (ii) also follows immediately from the above discussion since entering D_i twice would necessarily leave one of the intermediate vertices on the external cycle unvisited. □

The proof of Lemma 3.2 directly implies the following observation.

Observation 3.3 *Let D_i be a diamond with the sets I_i and O_i of input and output vertices. Let the diamond D_{i-1} be a subgraph of D_i with the sets I_{i-1} and O_{i-1} of input and output vertices. Then, the path $H^{i-1}_{\lceil j/2 \rceil}$ is a subpath of H^i_j.* □

Note that Definition 3.6 does not provide the only possible construction of a multi-dimensional diamond graph. Independently, another definition of multi-dimensional diamonds was given by Freiermuth and Stewénius [47], where the authors constructed multi-dimensional diamonds for every possible dimension $k \geq 3$ and additionally tried to minimize their size.

In the following, we analyze in how many edges two Hamiltonian paths H_p and H_r differ. We first define a measurement of distance between two input vertices. For two input vertices I^i_p and I^i_r let $\Delta_i(I^i_p, I^i_r) = |r - p|$.

Observation 3.4 *Let $I^i_p, I^i_r \in I_D$, for $p < r$, be two input vertices in D_i. Let H^i_p and H^i_r be the corresponding Hamiltonian paths in D_i. Let $O^{i-1}_{\lceil p/2 \rceil}$ and $I^{i-1}_{\lceil p/2 \rceil}$ be the first and last vertex of the diamond D_{i-1} on the path H^i_p and let the vertices $O^{i-1}_{\lceil r/2 \rceil}$ and $I^{i-1}_{\lceil r/2 \rceil}$ be the first and last vertex of D_{i-1} on the path H^i_r. If $\Delta_i(I^i_p, I^i_r) = k$, then $\Delta_{i-1}(I^{i-1}_s, I^{i-1}_t) \geq \lfloor \frac{k}{2} \rfloor$.*

Proof. Let be $\Delta_i\left(I^i_p, I^i_r\right) = k$, then an easy calculation shows, that $\Delta_{i-1}\left(I^{i-1}_{\lceil p/2 \rceil}, I^{i-1}_{\lceil r/2 \rceil}\right) = \left|\lceil \frac{p}{2} \rceil - \lceil \frac{r}{2} \rceil\right| \geq \lfloor \frac{k}{2} \rfloor$. □

We now compare two Hamiltonian paths in D starting from two different input vertices. In the following observation, we analyze the first part of the compared paths from the input vertex to the input vertex of the inner diamond.

Observation 3.5 *Let H^i_p and H^i_r be two Hamiltonian paths in the diamond D_i starting from two input vertices I^i_p and I^i_r (without loss of generality, assume $p < r$). Let P^i_p denote the union of the two subpaths in $H^i_p - H^{i-1}_{\lceil p/2 \rceil}$, i. e., the parts of H^i_p outside D_{i-1}, let P^i_r analogously denote the parts of H^i_r outside D_{i-1}.The subgraphs P^i_p and P^i_r differ in 2 edges if p is an odd number and $r = p + 1$, or in 4 edges in all other cases.*

Proof. All edges from the external cycle of D_i except the two edges $e_{1,p}, e_{2,p}$ are included in P^i_p (see Fig. 3.7), where

$$e_{1,p} = \begin{cases} \{I_p, m_{p-1,p}\} & \text{if } p \text{ is even} \\ \{I_p, m_{p,p+1}\} & \text{else} \end{cases}$$

and

$$e_{2,p} = \begin{cases} \{O_p, n_{p-1,p}\} & \text{if } p \text{ is even} \\ \{O_p, n_{p,p+1}\} & \text{else.} \end{cases}$$

In both cases, $e_{1,p} \neq e_{1,r}$ and $e_{2,p} \neq e_{2,r}$ if $r \neq p$.

If p is an odd number and $r = p + 1$, both paths P_r^i and P_p^i use the edges $\{n_{p,r}, I_{r_2}^{i-1}\}$ and $\{m_{p,r}, O_{r_2}^{i-1}\}$. Otherwise, P_r^i and P_p^i differ also in the cross edges from the external cycle to the inner diamond D_{i-1}. $\square$

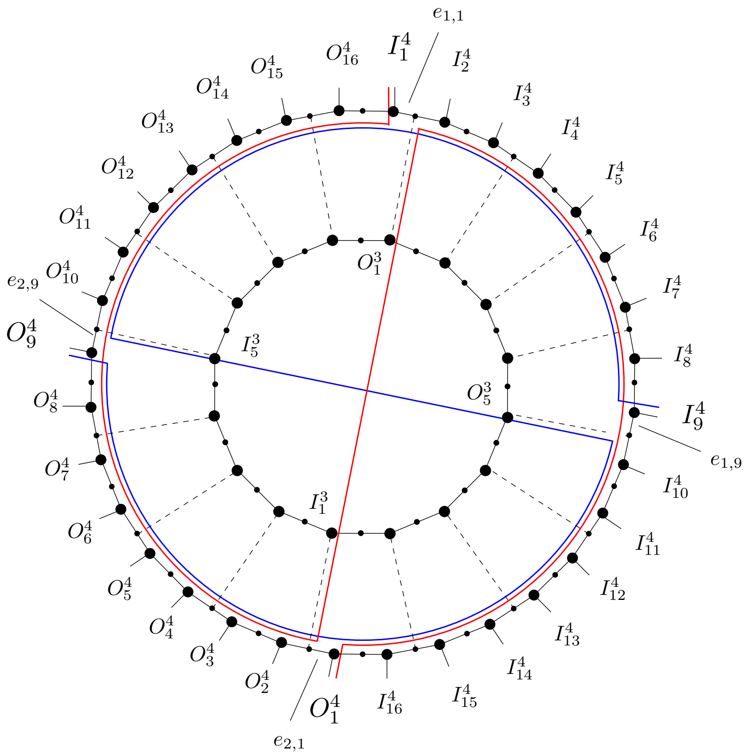


Figure 3.7: Two Hamiltonian paths in the 16-dimensional diamond graph D_4.

Lemma 3.3 *Let $I_p^i, I_r^i \in I_{D_i}$, $p, r \in \{2, 4, 6, \dots, 2^i\}$, $p < r$, be two input vertices in D_i with $\Delta_i(p,r) = r - p = 2^j$. Then, the corresponding Hamiltonian paths H_p^i and H_r^i differ in at least $4(j-1)+2$ edges.*

Proof. Let $I_p^i, I_r^i \in I_{D_i}$, $p, r \in \{2, 4, 6, \dots, 2^i\}, p < r$, be 2 input vertices in the diamond D_i with $\Delta_i = 2^j(p,r)$, and let H_p^i and H_r^i be the two corresponding Hamiltonian paths. We know from Observation 3.3 that the path $H_{\lceil p/2 \rceil}^{i-1}$ is a subpath of H_p^i and that $H_{\lceil r/2 \rceil}^{i-1}$ is a subpath of H_r^i. Obviously, the distance decreases in every step from D_l to D_{l-1} by at most the half, for $1 < l \leq i$.

At earliest after step j, both paths H_p^i and H_r^i enter the diamond D_{i-j} via the same vertex. It follows that the distance between the input vertices of D_{i-j+1} on H_p^i and H_r^i is 1. Because of this, both paths differ in 4 edges in each step from D_l to D_{l-1} for $i \geq l \geq i-j+2$ and in at least 2 edges from D_{i-j+1} to D_{i-j}. Overall, the paths differ in at least $(i-1) \cdot 4 + 2$ edges. □

We can now use Lemma 3.3 to exhibit a large set of pairwise distant Hamiltonian paths.

Lemma 3.4 *In the diamond graph D_{i^2}, there are 2^i different Hamiltonian paths which differ pairwise in at least $4 \cdot (i-1) + 2$ edges.*

Proof. Consider the graph D_{i^2}. We denote the vertex $I_{1+s \cdot 2^i}^{i^2}$ by $\hat{I}_s$ for all $s \in \{0, \dots, 2^i\}$. The set $\hat{I}_D = \{\hat{I}_s \mid s \in \{1, \dots, 2^i\}\}$ is the new set of input vertices.

There are 2^i many paths $P_{1+s \cdot 2^i}^{i^2}$ for $s \in \{0, 1, \dots, 2^i\}$, starting from the points $\hat{I}_s$.

For two starting points $\hat{I}_s, \hat{I}_t \in \hat{I}_D$, $s, t \in \{1, \dots, 2^i\}$, for $s \neq t$, the distance is $\Delta_{i^2}(\hat{I}_s, \hat{I}_t) \geq 2^i$. It follows by Lemma 3.3 that the two paths $P_{1+s \cdot 2^i}^{i^2}, P_{1+t \cdot 2^i}^{i^2}$, $s \neq t$, differ in at least $4 \cdot (i-1) + 2$ edges. □

A diamond D_{i^2} restricted to the 2^i input vertices with distance i^2 leads us to the definition of a new class of multi-dimensional diamonds $\hat{D}_i$.

Definition 3.7 *For $i \geq 1$, $\hat{D}_i$ denotes the diamond graph D_{i^2}, where the set of input vertices is restricted to $\hat{I}_{\hat{D}_i} = \{\hat{I}_s = I_{1+s \cdot 2^i}^{i^2} \mid s \in \{1, \dots, 2^i\}\}$ and the set of output vertices is restricted to $\hat{O}_{\hat{D}_i} = \{\hat{O}_s = O_{1+s \cdot 2^i}^{i^2} \mid i \in \{1, \dots, 2^i\}\}$. Furthermore, we denote the Hamiltonian cycle $H_p^{i^2}$ starting in the input vertex $p \in \hat{I}_{\hat{D}_i}$ by $\hat{H}_p$.*

It is obvious that also $\hat{D}_i$ is a diamond graph in the sense of Definition 3.5.

Observation 3.6 *For any dimension p, there exists an edge $e_{\mathrm{not}p}$ in $\hat{D}_i$ that is contained in the Hamiltonian paths of all other dimensions except in that of dimension p in $\hat{D}_i$.*

Proof. Let $\hat{H}_p$ be the unique Hamiltonian path in $\hat{D}_i$ in dimension p, i. e., starting in the input vertex $p \in \hat{I}_{\hat{D}_i}$. The edge from the input vertex $\hat{I}_p = I^{i^2}_{1+p\cdot 2^i}$ to the right neighbor vertex on the external cycle, $e_p = \{I^{i^2}_{1+p\cdot 2^i}, n_{1+p\cdot 2^i, 1+p\cdot 2^i+1}\}$, is not part of $\hat{H}_p$ but a part of all other Hamiltonian cycles $\hat{H}_r$, for $r \in \{1,\ldots,2^i\}, r \neq p$, see also Fig. 3.7. □

3.7 Multiple-Solution Reoptimization of TSP is Hard to Approximate

In this section, we prove that $\lfloor 2^{\sqrt{n}} \cdot (\sqrt{n})! \rfloor$-Sol-Reopt-TSP-IncEdge is not approximable within a ratio of 2^n. For this, we generalize the proof technique used in Section 3.3 by using the multi-dimensional diamond graphs defined in the previous section.

In Section 3.3, it was proven that the reoptimization variant of TSP, where the cost of a single edge is increased and all optimal solutions to the old instance are given, is not approximable within a ratio of 2^n on input graphs with n vertices. The proof was done using a reduction from the well-known $\mathcal{NP}$-hard *Hamiltonian cycle problem* (HC), which asks to decide whether a given undirected graph has a Hamiltonian cycle or not. From an HC instance G, a TSP reoptimization instance G' was constructed by replacing every vertex of G by a 2- dimensional diamond. By using an extension of this idea and replacing the vertices by higher-dimensional diamonds, we prove that even knowing additionally all near-optimal solutions does not help for approximating this reoptimization variant of TSP.

Theorem 3.6 *Let $\beta = \lfloor 2^{\sqrt{n}} \cdot (\sqrt{n})! \rfloor$. Under the assumption of $\mathcal{P} \neq \mathcal{NP}$, there does not exist any polynomial-time approximation algorithm with an approximation ratio of 2^n for β-Sol-Reopt-TSP-IncEdge, where n is the number of vertices in the instance.*

Proof. We prove the claim by giving a reduction from the Hamiltonian cycle problem (HC) to the β-Sol-Reopt-TSP-IncEdge.

Let G_{HC} be an input instance for HC with $V(G_{HC}) = \{v_1, \ldots, v_k\}$. To construct an instance of β-Sol-Reopt-TSP-IncEdge, we first build an unweighted graph G_{TSP}. As a first step, we substitute every vertex $v_i \in V(G_{HC})$, $i \in \{1, ..., k\}$, by a copy $D^{(i)}$ of the diamond graph D_{z+1} of dimension $z+1$ for some value of z that will be chosen later. We denote the set of all edges from $D^{(1)}, \ldots, D^{(k)}$ by E_{Dia}.

Then, we add a set $E_{z+1} = E_{HC}$ of edges. Here, for every edge $e_i = \{v_l, v_m\} \in E(G_{HC})$, we add two edges between the corresponding diamonds $D^{(l)}$ and $D^{(m)}$ in dimension $z+1$: one between the vertex $I^{(l)}_{z+1}$ and $O^{(m)}_{z+1}$ and one between the vertex $O^{(l)}_{z+1}$ and $I^{(m)}_{z+1}$.

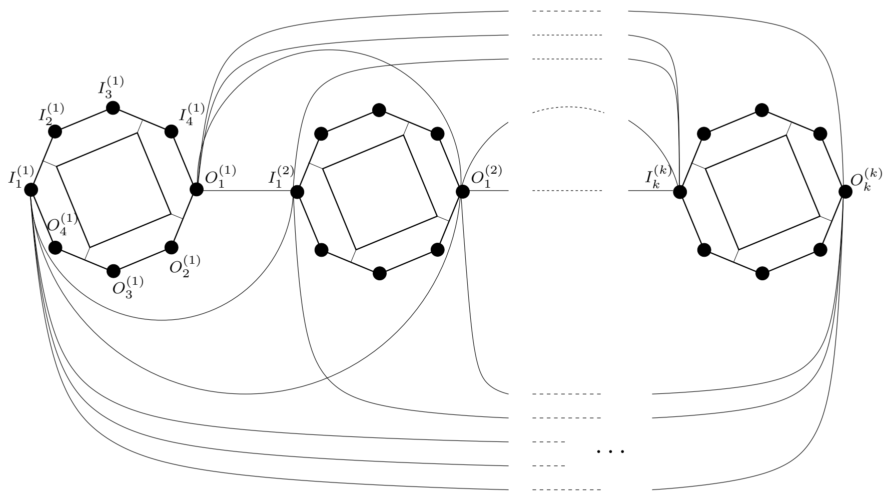


Figure 3.8: Connections in the graph G_{TSP} in dimension 1

Furthermore, for every other dimension $g \in \{1, \ldots, z\}$, we add a set E_g of edges from every $I_g^{(i)}$ and $O_g^{(i)}$ vertex of every diamond $D^{(i)}$ to every $I_g^{(j)}$ and $O_g^{(j)}$ vertex of all other diamonds $D^{(j)}$ for $j \neq i$.

We call an edge e an *edge of dimension* d, if it is a connection between input and output vertices in dimension d, i. e., if $e \in E_d$. We say that a Hamiltonian cycle H in G_{TSP} lies in dimension d if, for all edges e of H, we have $e \in E_{Dia} \cup E_d$. In other words, it contains only edges inside the diamonds and edges of dimension d.

Now, we transform G_{TSP} into an instance $(G_{Reopt}, cost_{\text{old}}, cost_{\text{new}})$ of β-Sol-Reopt-TSP-IncEdge. Let $G_{Reopt} = (V, E)$ be a complete graph with $V = V(G_{TSP})$ on $z' \cdot k$ vertices, where $z' = 2^{\lceil \log_2(z+1) \rceil + 3} - 8$ denotes the number of vertices of one $(z+1)$-dimensional diamond. We define $cost_{old}$ as follows:

All edges in the set E_{Dia} get cost 1, all edges of set E_g get cost of g, for $g \in \{1, \ldots, z\}$, and the edges of set E_{HC} get cost of $z+1$. All other edges get cost of $M = 2^{2z' \cdot k}$.

We now consider the local modification of increasing the cost of one edge in $cost_{\text{old}}$, leading to the new TSP instance $(G_{Reopt}, cost_{new})$. For this, we change the cost of the edge $e_{change} = e_{\text{not}(z+1)}$ in the diamond D_1 from $cost_{\text{old}}(e_{change}) = 1$ to $cost_{\text{new}}(e_{change}) = M$.

In addition to $(G_{Reopt}, cost_{\text{old}}, cost_{\text{new}})$, we have to specify a set of given solutions for $(G_{Reopt}, cost_{\text{old}})$. For this, we take all Hamiltonian tours in the first z dimensions.

In the graph G_{TSP}, independent of the original graph G_{HC}, there exist $z \cdot (2^k \cdot (k-1)!)$ many Hamiltonian cycles traversing all diamonds in the first

z dimensions: In every dimension $g \in \{1, \ldots, z\}$, the order of the diamonds can vary arbitrarily, yielding $(k-1)!$ different possibilities in this dimension, and every diamond D_i can be traversed in dimension g either from $I_{i,g}$ to $O_{i,g}$ or vice versa which leads to $(2^k \cdot (k-1)!)$ different Hamiltonian tours in each dimension g and thus to $z \cdot (2^k \cdot (k-1)!)$ different tours overall. Furthermore, if and only if there exists a Hamiltonian cycle in G_{HC}, there also exist one or more (exactly as many as in G_{HC}) Hamiltonian cycles in G_{TSP} traversing all diamonds in dimension $z+1$.

Any of the Hamiltonian cycles in dimension 1 lead to an optimal solution in $(G_{Reopt}, cost_{\text{old}})$ with cost of $z' \cdot k$ because, for all edges $e_i \in E(G_{TSP})$, $cost_{\text{old}}(e_j) = 1$ holds. With the same argument, we see that every Hamiltonian cycle in dimension g leads to a g-best solution in $(G_{Reopt}, cost_{\text{old}})$ with cost of $(z' + g - 1)k$, for all $g \in \{1, \ldots, z\}$, because it traverses all of the k diamonds with overall cost z', and every edge in dimension g from one diamond to another has cost g.

If there exists a Hamiltonian cycle in G_{HC}, this cycle leads to one solution with cost $(z' + z) \cdot k$.

A Hamiltonian cycle in G_{Reopt} traversing some diamonds in dimension b and some other in dimension c with $b \neq c$ uses at least two edges of cost M due to the construction, leading to a cost of at least $z' \cdot k - 2 + 2 \cdot M$.

By increasing the edge cost of e_{change} from $cost_{\text{old}}(e_{change}) = 1$ to $cost_{\text{new}}(e_{change}) = M$, all z-best solutions in $(G_{Reopt}, cost_{\text{old}})$ get a cost of at least $z' \cdot k - 1 + M$ in $(G_{Reopt}, cost_{\text{new}})$. Therefore, in $(G_{Reopt}, cost_{\text{new}})$, there exists an optimal solution with cost of $(z' + z) \cdot k$ if and only if there is a Hamiltonian cycle in G_{HC}. Otherwise, the old optimal solutions in $(G_{Reopt}, cost_{\text{old}})$ stay optimal in $(G_{Reopt}, cost_{\text{new}})$.

Thus, an approximation algorithm with an approximation ratio smaller than

$$\frac{z' \cdot k - 1 + M}{(z' + z) \cdot k}$$

would solve the HC problem. Due to $M = 2^{2z'k}$, we have

$$\frac{z' \cdot k - 1 + M}{(z' + z) \cdot k} > 2^{z'k} = 2^n, \tag{3.1}$$

for almost all values of k.

To prove our claim, we now have to specify a value for z. The value of $z+1$ should be a power of two, since we only defined 2^i-dimensional diamond graphs and it has to satisfy the inequality

$$z \cdot 2^k \cdot (k-1)! \geq \left\lfloor \left(\sqrt{n}\right)! \cdot 2^{\sqrt{n}} \right\rfloor \tag{3.2}$$

We know that $n = k \cdot z'$ and $z' = 2^{\lceil \log_2(z+1) \rceil + 3} - 8$, thus (3.2) is equivalent to

$$\begin{aligned} z \cdot 2^k \cdot (k-1)! &\geq \left\lfloor \left(\sqrt{\left(2^{\lceil \log_2(z+1) \rceil + 3} - 8\right) \cdot k} \right)! \cdot 2^{\sqrt{\left(2^{\lceil \log_2(z+1) \rceil + 3} - 8\right) \cdot k}} \right\rfloor \\ &\geq \left\lfloor \left(\sqrt{8 \cdot z \cdot k} \right)! \cdot 2^{\sqrt{8 \cdot z \cdot k}} \right\rfloor \qquad (3.3) \end{aligned}$$

An easy calculation shows that $\overline{z} = \frac{1}{8}k - \frac{1}{4}$ satisfies (3.3), thus choosing $z+1$ as the largest power of 2 that is smaller or equal to $\overline{z} - 1$ proves the claim. □

Note that (3.1) additionally proves that also a variant of multiple-solution TSP reoptimization, where we are given all optimal and all near-optimal solutions starting from the z-best ones, is equally hard to approximate.

3.8 Local Search Does Not Work for Multiple-Solution TSP Reoptimization

We now give an example of an instance for the multiple-solution TSP reoptimization which shows the hardness of local search with respect to the exchange neighborhood. According to the old cost function, the instance has exponentially many optimal solutions, all of which can be given. The new instance has a unique optimal solution and exponentially many second-best solutions, which differ from the unique optimal solution in many edges.

Theorem 3.7 *Let $\beta = 2^{\sqrt{n+8\sqrt{n}}} \cdot \left(\sqrt{n+8\sqrt{n}}\right)!$, let n be such that $k = 2\sqrt{n}$ and $z = \sqrt{\log_2\left(\frac{1}{16}\sqrt{n}+1\right)} - 1$ are integers. Then $n = k \cdot z'$, where $z' = 2^{(z+1)^2+3} - 8$. Furthermore, let $M = 2^n$. Then there exists an instance (K_n, $cost_{old}$, $cost_{\mathrm{new}}$) for β-Sol-Reopt-TSP-IncEdge with n vertices with the following properties.*

(i) There exists a set H_{Opt} of optimal Hamiltonian cycles in $(K_n, cost_{old})$ of size $z \cdot 2^k \cdot (k-1)!$, where each cycle $H \in H_{Opt}$ has cost $cost_{old}(H) = z' \cdot k$.

(ii) The set of H_{Opt} can be partitioned into subsets $Z_1, Z_2, \ldots, Z_z \subseteq H_{Opt}$ with $|Z_1| = |Z_2| =, \ldots, = |Z_z| = 2^k \cdot (k-1)!$ where two solutions $H_i \in Z_i$ and $H_j \in Z_j$, for $i \neq j$, differ in at least $(4z+3) \cdot k$ edges.

(iii) In the modified instance $(K_n, cost_{\mathrm{new}})$, all solutions in H_{Opt} become second-best solutions of cost $z' \cdot k - 1 + M$.

(iv) There is a new unique optimal solution in the modified instance $(K_n, cost_{\mathrm{new}})$ of cost $(z'+1)k$ which differs from every Hamiltonian cycle $H_i \in H_{Opt}$ in at least $(4z+3) \cdot k$ edges.

Proof. An easy calculation shows that indeed $n = k \cdot z'$. To construct the instance $(K_n, cost_{old}, cost_{\text{new}})$, we first construct a graph G_{TSP} with several different Hamiltonian tours, and then extend it to a complete graph by defining the cost functions $cost_{old}$ and $cost_{\text{new}}$. We build G_{TSP} as a graph consisting of k diamonds $\hat{D}_{z+1}^{(i)}$ for $i \in \{1, \ldots, k\}$. The vertices of each diamond are connected as described in Definition 3.7. We denote the set of these vertices by E_{Dia}.

In the next step, we build a cycle through the k diamonds in the first dimension. For this, we connect the diamonds $D^{(l)}$ und $D^{(l+1)}$, for $l \in \{1, \ldots, k-1\}$, by edges $\{\{O_1^{(l)}, I_1^{(l+1)}\} \mid l \in \{1, \ldots, k-1\}\}$ and close the cycle with the edge $\{O_1^{(k)}, I_1^{(1)}\}$. Let E_1 denote the set of these edges from dimension 1 (see Figure 3.9).

Furthermore, we add, for every other dimension $g \in \{2, \ldots, z+1\}$, a set E_g of edges from every $I_g^{(i)}$ and $O_g^{(i)}$ vertex of every diamond $D^{(i)}$ to every $I_g^{(j)}$ and $O_g^{(j)}$ vertex of all other diamonds $D^{(j)}$, for $j \neq i$.

Now, we transform G_{TSP} into an instance $(G_{Reopt}, cost_{\text{old}}, cost_{\text{new}})$ of *Inc-Edge-Reopt*$_{ALL}$-TSP. For this let $G_{Reopt} = (V, E) = K_{z' \cdot k}$ be the complete graph on $z' \cdot k$ vertices.

All edges in the set E_{Dia} get cost 1, all edges of set E_g for $g \in \{2, \ldots, z+1\}$ get cost of 1 and the edges of set E_1 get cost of 2. All other edges get cost of M.

The local modification of changing the cost of one edge in $cost_{\text{old}}$ leads to the new TSP instance $(G_{Reopt}, cost_{\text{new}})$. Here, we change the cost of the edge $e_{change} = e_{\text{not1}}$ in the diamond D_1 (according to Observation 3.6) from $cost_{\text{old}}(e_{change}) = 1$ to $cost_{\text{new}}(e_{change}) = M$.

Additionally, we have to specify the set of all optimal solutions in $(G_{Reopt}, cost_{\text{old}})$. In each dimension $g \in \{2, \ldots, z+1\}$ in G_{TSP}, there are $2^k \cdot (k-1)!$ many Hamiltonian cycles. Summing up, there are $z \cdot (2^k \cdot (k-1)!)$ Hamiltonian tours in the z dimensions.

Any of these Hamiltonian tours leads to an optimal solution in $(G_{Reopt}, cost_{\text{old}})$ with cost $z'k$ because it traverses any of the k diamonds with cost z' and the connecting edges from one diamond to an other have cost 1 as well. Overall there are $z \cdot 2^k \cdot (k-1)!$ optimal solutions.

Furthermore, there is one Hamiltonian tour in dimension 1 in G_{TSP}. This tour leads to the second best solution in $(G_{Reopt}, cost_{\text{old}})$ with cost $(z'+1)k$. It traverses all diamonds with cost z' each and the connection from one diamond to another in dimension 1 has cost 2. This Hamiltonian tour differs from all optimal solutions by at least $(4z+3) \cdot k$ edges: The Hamiltonian paths through a diamond differ in at least $(4z+2)$ edges per diamond to the other Hamiltonian paths according to and the k edges in between the diamonds differ to all optimal solutions as well.

Note that a tour that leaves a diamond in one dimension and enters another dimension has to take at least two edges of cost M.

By increasing the cost of e_{change} to $cost_{\text{new}}(e_{change}) = M$, all optimal solutions in $(G_{Reopt}, cost_{\text{old}})$ get cost of $z'k - 1 + M$ and therefore become second-best solutions. The new unique optimal solution is the Hamiltonian tour in dimension 1 with cost $(z' + 1)k$.

To prove the correctness of our claim, it remains to show that the parameters β, k and z are properly chosen.

The instance contains $z \cdot 2^k \cdot (k-1)!$ optimal Hamiltonian cycles according to $cost_{old}$. It is easy to verify that

$$z \cdot 2^k \cdot (k-1)! \geq \beta = 2^{\sqrt{n+8\sqrt{n}}} \cdot \left(\sqrt{n + 8\sqrt{n}}\right)!$$

holds for almost all n, thus proving our claim. □

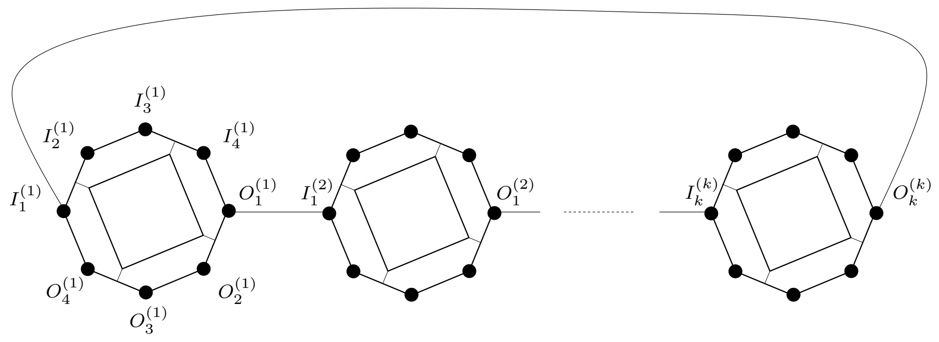


Figure 3.9: The Hamiltonian tour in G_{TSP} in dimension 1 in the proof of Theorem 3.7

Theorem 3.7 immediately implies that it is hard for a local search algorithm to escape from a local minimum for $(G_{Reopt}, cost_{\text{new}})$ when using a neighborhood that is defined by exchanging less than $(4z+3) \cdot k$ edges of a Hamiltonian tour.

3.9 Discussion

In this chapter, we have given some examples where the presence of multiple optimal or near-optimal solutions does not help to improve the solvability of reoptimization problems. It would be interesting to use the proof techniques to extend these hardness results also to other optimization problems. Another important question is whether these hardness results can be complemented by some examples of problems that become easier to solve within this generalized framework of reoptimization.

Part II

Online Computation with Advice

Chapter 4

Online Coloring of 3-Colorable Graphs with Advice

4.1 Introduction

In this chapter, we analyze the advice needed for coloring 3- colorable graphs with a competitive ratio of 4/3. In other words, we want to color 3-colorable graphs with four colors. Note that the offline version of this problem is also known to be $\mathcal{NP}$-hard [48, 57]. Here, we show how to obtain a 4-coloring for any 3-colorable graph with 1.1583 advice bits per vertex.

Additionally, we develop an algorithm to color 3-colorable chordal graphs with four colors by using less than one bit (0.9865) advice per vertex. An overview of our results is shown in Table 4.1.

Table 4.1: Overview of the results on the number of bits per vertex for online coloring.

number of bits per vertex	lower bounds	upper bounds	
	3-coloring	3-coloring	4-coloring
3-colorable graphs	$\log_2 3 - \varepsilon^2$	1.5863	1.1583
3-colorable chordal graphs	$\log_2 3 - \varepsilon$	1.5863	0.9865
maximal outerplanar graphs	1.0424	1.2932	0.9865

[2] $\log_2 3 \geq 1.5849$

4.2 Preliminaries

We use the following notation for online algorithms analogous to [29], for coloring graphs, and for the problem at hand, respectively.

Definition 4.1 (OColA) *The **Online Coloring Problem with Advice, in Vertex-Revealing Mode (OColA)** is the following online problem: The input is an unweighted, undirected graph $G = (V, E)$ with $|V(G)| = n$ and an order $\prec$ of revealing on the set of vertices. The goal is to find a minimum-cost coloring function $c : V \to \{1, \ldots, n\}$ for the vertices in G.*

In each time step i, the next vertex $v_i \in V$ (in the order $\prec$) is revealed, together with all edges $\{\{v_i, v_j\} \mid j < i\}$, and the online algorithm has to decide which color $c(v_i)$ the vertex v_i gets. To this end, it can read a certain number of advice bits.

For every instance $I = (G, \prec)$, where $G = (V, E)$, we get a directed graph $G^{\prec} = (V, E')$ by giving a direction on every edge $e \in E$ depending on the order of revealing the vertices. Every edge $e = \{v_i, v_j\} \in E$ is directed from v_i to v_j, i.e., $(v_i, v_j) \in E'$, iff v_i is revealed before v_j.

For developing algorithms to color a 3-colorable graph optimally, we need a method to read a one-out-of-three decision from a Boolean advice string. For this, we use the following lemma.

Lemma 4.1 *Reading several one-out-of-three decisions from a bit string costs $46/29 < 1.5863 \approx \log_2 3$ bits on average.*

Proof. When the first one-out-of-three decision is necessary, 46 bits get read from the advice string. By these 46 bits and the corresponding 2^{46} different possible bit allocations, 29 three-way decisions can be encoded, because $2^{46} \geq 3^{29}$. With this, for the first three-way decision, the algorithm gets the results of the next 28 three-way decisions at the same time and keeps them in its memory. This leads to an average of the information needed for a three-way decision of $46/29 < 1.5863 \approx \log_2 3$ bits. In general, if n one-out-of-three decisions have to be done, this costs at most $1.5863(n-1) + 46 = 1.5863n + d$ bits, for some constant d. $\square$

4.3 Lower Bounds on the Advice Complexity

In this section, we first turn to the lower bounds. We will show that more than one bit of advice per vertex is necessary to color a maximal outerplanar graph optimally, i.e., by three colors.

Theorem 4.1 *For any $k \in \mathbb{N}$, there exists a set G_k of maximal outerplanar graphs $G(u, v, w, x, y, z)$ on $4k + 9$ vertices for $k = u + v + w + x + y + z$ such that every deterministic online algorithm for OColA on G_k needs at least*

$\frac{1}{2} \cdot \left(\log_2 3 + \frac{1}{2}\right) \cdot n - 12 > 1.0424 \cdot n - 12$ *advice bits to generate an optimal coloring.*

Proof. We prove the claim by constructing a class $\mathcal{G}_{hard}$ of hard input instances. For every 6-tuple of numbers $(u, v, w, x, y, z) \in \mathbb{N}^6$, we construct an outerplanar graph $G(u, v, w, x, y, z)$ with $4 \cdot (u+v+w+x+y+z)+9$ vertices (see Figure 4.1).

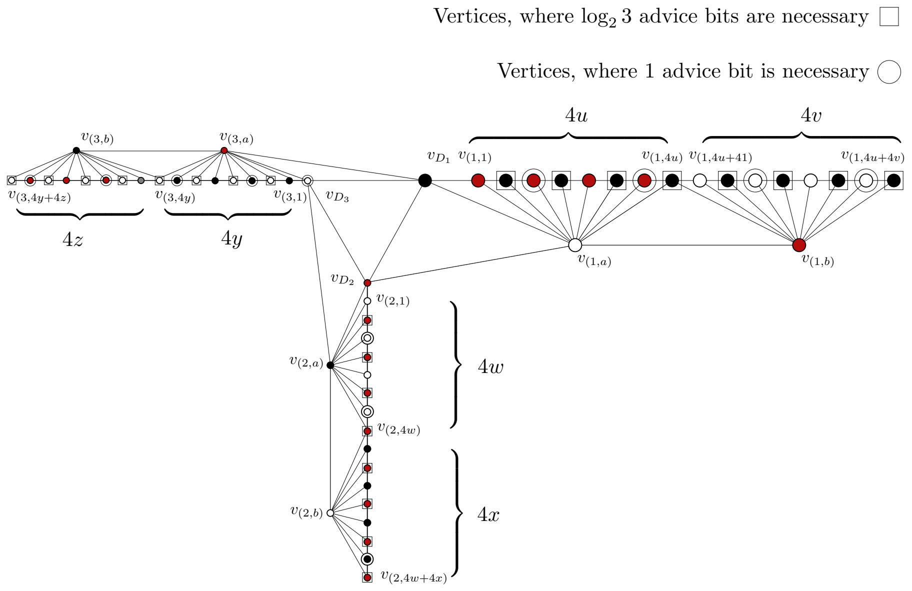


Figure 4.1: Example of a hard instance for the parameters $u = v = w = x = y = z = 2$.

For an easier notation, we use $t = u+v+w+x+y+z$. At first, we describe the graph G for a given tuple (u, v, w, x, y, z). The graph G of an input instance in $\mathcal{G}_{hard}$ contains three paths P_1, P_2, P_3. The path $P_1 = v_{(1,1)}, v_{(1,2)}, \dots, v_{(1,4u)}, v_{(1,4u+1)}, \dots, v_{(1,4u+4v)}$ contains $4 \cdot (u+v)$ vertices, the path $P_2 = v_{(2,1)}, \dots, v_{(2,4w+4x)}$ contains $4 \cdot (w+x)$ vertices and $P_3 = v_{(3,1)}, \dots, v_{(3,4y+4z)}$ contains $4 \cdot (y+z)$ many vertices.

Every path P_i, for $i \in \{1, 2, 3\}$, is, at the beginning, extended by one vertex v_{D_i} (see Figure 4.1). The three vertices v_{D_1}, v_{D_2} and v_{D_3} are connected to each other and form a triangle. Additionally, every vertex in a path P_i is connected to one of the two vertices $v_{(i,a)}, v_{(i,b)}$. The vertex $v_{(1,4u)}$ (resp. $v_{(2,4w)}, v_{(3,4y)}$) is connected to both vertices $v_{(1,a)}$ and $v_{(1,b)}$ (resp. $v_{(2,a)}$ and $v_{(2,b)}$, $v_{(3,a)}$ and $v_{(3,b)}$).

Additionally, the vertex $v_{(1,a)}$ (resp. $v_{(2,a)}, v_{(3,a)}$) is also connected to v_{D_1} and v_{D_2} (resp. v_{D_2}, v_{D_3} or v_{D_3}, v_{D_1}).

We can see that such a graph is maximal outerplanar, because the subgraph $G_{V(P_i^+)}$ which is induced by the vertices $V(P_i^+) = V(P_i) \cup \{v_{(i,a)}, v_{(i,b)}\}$ is maximal outerplanar on its own. The triangle $v_{D_1}, v_{D_2}, v_{D_3}$ is maximal outerplanar and the three extended paths $G_{V(P_i^+)}$ are connected to the triangle twice and planar, hence the whole graph is maximal outerplanar. In Figure 4.1, also an optimal coloring of G is shown. A maximal outerplanar graph is chordal [63]. Because of this, in every maximal outerplanar graph, the coloring is unique up to permutations of the colors.

Now, we consider the set of problem instances of graph coloring that are all based on G_k and they differ in the choice of the numbers u, v, w, x, y, z and in the order in which the vertices of G_k are revealed. We do not consider all permutations of the $4k + 9$ vertices of the graphs in G_k, but only $3^{2t} \cdot 2^t$ special ones.

For counting the numbers of instances which need a different advice string, we separate the input into three phases. In the first phase, every second vertex $v_{(i,2)}, v_{(i,4)}, \ldots$ of every path P_i is revealed as an isolated vertex. In the second phase, when all $2t$ isolated vertices have been revealed, for every pair $(v_{(i,4\cdot c-2)}, v_{(i,4\cdot c)})$ for $c \in \{1, \ldots, u + v\}$ and $i \in \{1, 2, 3\}$, the vertex $v_{(i,4\cdot c-1)}$ connected to both is revealed. Overall, there are t such pairs of isolated vertices. In the last step, the remaining vertices connecting the already revealed subpaths of length 3, as well as the vertices $v_{(i,a)}, v_{(i,b)}$ and v_{D_i} are revealed. Additionally, between the first vertex $v_{(i,2)}$ of each path and the vertex of the middle triangle v_{D_i} the vertex $v_{(i,1)}$ is revealed to connect the path and the triangle. In the example in Figure 4.1, the vertices which are revealed as isolated vertices are marked by a square, and the vertices revealed in the second step are marked by a circle.

Now, we count the number of instances which need a different advice string, such that a deterministic algorithm can be guaranteed to be optimal.

The instance has $2(u + v + w + x + y + z) = 2t$ isolated vertices. In every optimal solution, $2(u + v)$ many vertices $(v_{(1,2)}, v_{(1,4)}, \ldots, v_{(1,4u+4v)})$ have to be colored with the same color, $2(w + x)$ isolated vertices have to be colored with the same color, but different to the first color, and $2(y + z)$ many isolated vertices have to be colored with the third color.

This isolated vertices build t pairs $(v_{(1,2)}, v_{(1,4)}), (v_{(1,6)}, v_{(1,8)}), \ldots$, and each pair gets connected by a vertex $v_{(1,3)}, v_{(1,5)}, \ldots$. In this situation, there are $u+v$ pairs of vertices in one color and u pairs become connected by vertices which have to get the same color and v pairs get connected by vertices, which have all to be colored in the other color. This coloring of the vertices in the middle is unique and determined by the connection to $v_{(i,a)}$ respectively $v_{(i,b)}$, but these two vertices are revealed later.

These means that there are, in any possible input instance, $2t$ many isolated vertices and each vertex can be located in any of the three paths and thus requires exactly one of the three colors. This leads to

$$3^{2t} \text{ many possibilities.}$$

The optimal coloring is unique up to permutations of the colors. For the necessary advice, we have to calculate that 6 optimal colorings can use the same advice due to the 6 possibilities of renaming colors. Thus, we need $3^{2t} \cdot \frac{1}{6}$ different advice strings for an optimal coloring of the $2t$ isolated vertices.

The $2t$ isolated vertices are combined to t pairs. There exist t different pairs of isolated vertices, $u+v$ pairs of vertices with color 1, $w+x$ pairs with color 2 and $y+z$ pairs of color 3.

In the second step, these t pairs get connected by one vertex in the middle each. For every vertex in the middle of a pair, there are two possible colors to choose from, but only one color leads to an optimal coloring of G. So there are $u+v$ pairs with color one, where, in u pairs, the vertex in the middle has to be colored with 2 and, in v pairs, the vertex in the middle has to be colored with 3 to obtain an optimal coloring. The respective statements hold for the $w+x$ pairs colored with 2 and for the $y+z$ pairs colored with 3.

We now count the number of possible variations of the order in which the vertices in the middle of isolated vertices of color 1 are revealed. There are $u+v$ many pairs where, in the middle of u pairs, the revealed vertex has to be colored with color 2 and v revealed vertices which have to be colored with color 3. The two values u and v can be chosen arbitrarily and thus there exist

$$\sum_{i=0}^{u+v} \binom{u+v}{i} = 2^{u+v}$$

many different continuations for each instance from the first phase. Considering all groups of pairs, we get

$$2^{u+v} \cdot 2^{w+x} \cdot 2^{y+z} = 2^t$$

many different continuations for any input string of the first phase, which all need different advice strings. This leads to

$$3^{2t} \cdot 2^t$$

many different input strings for a graph from the class of graphs with $4t+9$ vertices overall.

For the $3^{2t} \cdot 2^t$ many different instances, at least $3^{2t} \cdot 2^t \cdot \frac{1}{6}$ many different advice strings are necessary.

Now we show that, for an instance of $n = 4t+9$ vertices, at least $\frac{1}{2} \cdot \left(\log_2 3 + \frac{1}{2}\right) \cdot n - 12 > 1.0424 \cdot n - 12$ many advice bits are needed to be optimal. We have

$$\begin{aligned}
\log_2\left(3^{2t}\cdot 2^t\cdot\frac{1}{6}\right) &= \log_2\left(3^{2t}\right)+t+\log_2\frac{1}{6}\\
&= 2t\cdot\log_2 3+t-\log_2 6\\
&= t\cdot(2\log_2 3+1)-\log_2 3-1\\
&= \frac{n-9}{4}\cdot(2\log_2 3+1)-\log_2 3-1\\
&= \frac{n}{2}\cdot\left(\log_2 3+\frac{1}{2}\right)-\frac{9}{2}\log_2 3-\frac{9}{4}-\log_2 3-1\\
&= \frac{n}{2}\cdot\left(\log_2 3+\frac{1}{2}\right)-\left(\frac{11}{2}\log_2 3+\frac{13}{4}\right)\\
&> \frac{1}{2}\cdot n\cdot\left(\log_2 3+\frac{1}{2}\right)-11.9673\\
&> \frac{1}{2}\cdot n\cdot\left(\log_2 3+\frac{1}{2}\right)-12\\
&> 1.0424\cdot n-12. \qquad\square
\end{aligned}$$

Additionally, we want show that $\log_2 3$ bits per vertex are necessary for coloring any 3-colorable graph online optimally. For this, we prove the following theorem.

Theorem 4.2 *For any $k \in \mathbb{N}$, there exists a 3-colorable graph G on $n = k+3$ vertices and an ordering $\prec$ such that every deterministic online algorithm for OColA on $(G,\prec)$ needs at least $\log_2 3\cdot(k-1)-1 = \log_2 3\cdot(n-4)-1$ advice bits to be optimal.*

Proof. We prove the claim by constructing a class $\mathcal{G}_{gen}$ of input instances. For every 3-tuple of numbers $(u,v,w)\in\mathbb{N}^3$, we define a graph G with $u+v+w+3$ vertices (see Figure 4.2).

For an easier notation, we use $k = u+v+w$. At first, we describe the graph G for a given tuple (u,v,w). The graph G of an input instance in $\mathcal{G}_{gen}$ contains three sets of vertices V_{c1}, V_{c2}, V_{c3}, with $|V_{c1}| = u, |V_{c2}| = v$, and $|V_{c3}| = w$ vertices, and three more vertices v_1, v_2, and v_3, which build a triangle. All vertices of V_{c1} are connected to v_2 and v_3, the vertices of V_{c2} are connected to v_1 and v_3, and the vertices in V_{c3} are connected to v_1 and v_2 (see Figure 4.2).

For every instance $G \in \mathcal{G}_{gen}$, there is only one optimal coloring, except for renaming of the colors. For the three vertices v_1, v_2 and v_3, the coloring is unique, because they build a triangle and the color of all vertices in V_{ci} for $i \in \{1,2,3\}$ is determined by this.

For counting the numbers of instances which need a different advice string, we separate the input into two phases. In the first phase, every vertex from V_{c1}, V_{c2}, and V_{c3} is revealed as an isolated vertex.

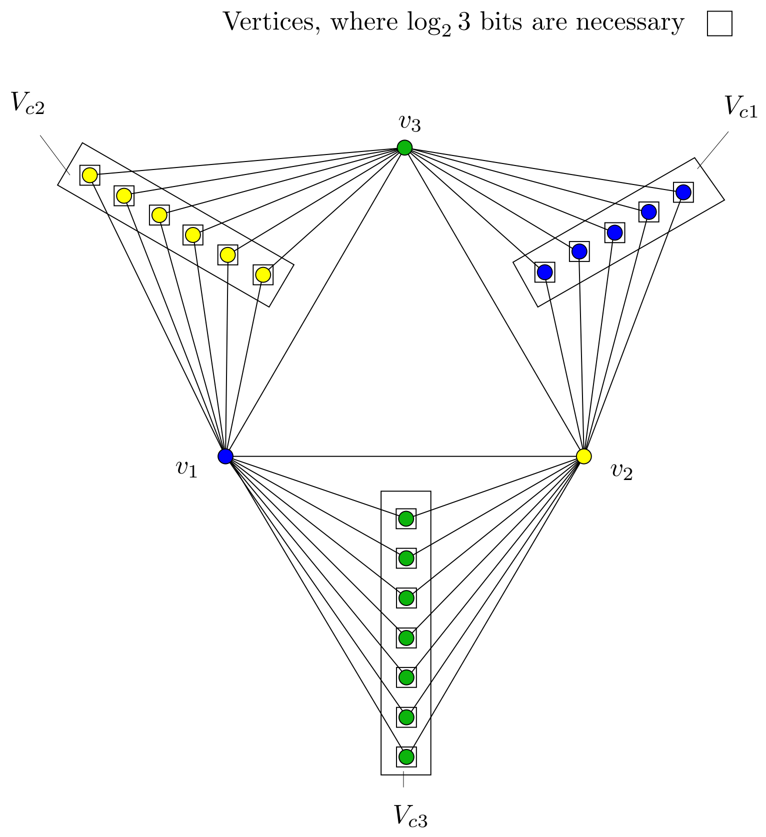


Figure 4.2: Example of a hard instance for the tuple $(u, v, w) = (5, 6, 7)$.

In the second phase, the vertex v_1 is revealed, connected to all vertices from V_{c2} and V_{c3}, the vertex v_2 is revealed, connected to v_1 and all vertices from V_{c1} and V_{c3}, and v_3 is revealed, connected to v_1, v_2 and all vertices from V_{c1} and V_{c2}. In the example in Figure 4.2, the vertices that are revealed as isolated vertices are marked by a square.

Now, we count the number of instances which need a different advice string, such that a deterministic algorithm can be guaranteed to be optimal.

In the first phase, $u + v + w = k$ many isolated vertices are revealed, which can be colored in one of the possible colors $1, 2, 3$. Every isolated vertex has three possibilities to belong to one of the three sets V_{ci}. The variables u, v, w can get arbitrary values with the constraint $u + v + w = k$.
This leads to

$$3^k$$

many possible input strings. Since we have k isolated vertices and there are three colors possible for every position.

In the second phase, there are 6 possibilities for revealing the last three vertices, but here, no advice is needed, because, for an optimal coloring, the colors of the three vertices v_1, v_2 and v_3 are determined by the edges given when the vertices are revealed.

On the other hand, we have to calculate that the 6 optimal colorings, coming up by color renaming, can use the same advice.

Thus, we need $3^k \cdot \frac{1}{6}$ different advice strings for an optimal coloring of the k isolated vertices. Thus, we need

$$\begin{aligned}\log_2\left(3^k \cdot \frac{1}{6}\right) &= k \cdot \log_2 3 + \log_2\left(\frac{1}{6}\right)\\ &= (k-1)\cdot \log_2 3 - 1\\ &= (n-4)\cdot \log_2 3 - 1\end{aligned}$$

advice bits. □

4.4 Online Algorithms for the Coloring of 3-Colorable Graphs

Now we are ready to investigate several algorithms for coloring graphs online with given advice. Let G be a graph with an order $\prec$ of vertices of G, and let $G^{\prec}$ be the corresponding directed graph. Depending on the direction of the edges in $G^{\prec}$, we define the function $p : V(G) \to \{1, 2, 3\}$ for all vertices in $G^{\prec}$, where $p(v) = i$ describes which position v has in a triangle.

Every vertex v_x, that was revealed as isolated, i.e., has outgoing edges only, is the first vertex of any triangle it belongs to. Such a vertex gets the label one, that is $p(v_x) = 1$. Every vertex v_y that is connected to one or more already revealed vertices, but not closing a triangle, has $p(v_y) = 2$. (In any triangle, there is at most one ingoing edge). Finally, every vertex v_z which closes one or more triangles (has two ingoing edges in one triangle) gets $p(v_z) = 3$. That way, we partition the vertices of a given input instance $G^{\prec}$ into three classes $V_i = \{v \in V(G) \mid p(v) = i\}$, for $i \in \{1, 2, 3\}$. For an example showing the different types of vertices, see Figure 4.3.

In every step of the coloring algorithm, when a new vertex v occurs, there exists a set of colors by which v may be colored. We denote, for every vertex v, the set of allowed colors by $C_v = \{1, \ldots, i\} \setminus \{c(w) \mid w \in \mathit{Neigh}(v)\}$, with $i \in \{3, 4\}$ corresponding to a coloring in i colors. Note that, since we use an ordered set of colors, we may speak of a 'smallest' color in C_v.

We start with Algorithm 4.1, which colors an arbitrary 3-colorable graph G online with 3 colors, where G is revealed according to an order $\prec$. For this, we

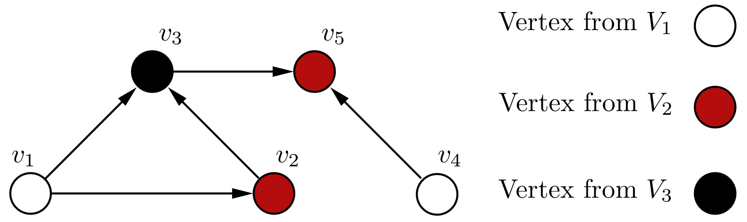


Figure 4.3: Vertices of different types.

need 1.5863 advice bits per vertex on average.

The idea of the first algorithm is quite simple. For every isolated vertex $v \in V_1$, the algorithm asks for the optimal color (one out of three). For every vertex $w \in V_2$ connected to an already colored vertex, the algorithm asks for the correct color from the the remaining colors C_w (at most one out of two), and every vertex $x \in V_3$ gets colored by the only remaining color. This leads to the following lemma.

Lemma 4.2 *Let G be a graph with $\chi(G) = 3$, and let $G^{\prec}$ be an input instance for the OColA. Algorithm 4.1 colors G optimally with at most $(n-3)$ one-out-of-three or one-out-of-two decisions and at most one additional one-out-of-two decision. With this, Algorithm 4.1 uses less than $1.5863 \cdot (n-3) + 1 + 45 = 1.5863 \cdot n + d$ advice bits*[1]*.*

Proof. Let $G^{\prec}$ be an input instance of a graph G with $\chi(G) = 3$ with $|V(G)| = n$ vertices. In the worst case, $G^{\prec}$ contains $n-2$ vertices in V_1. Otherwise, G could not contain a cycle, and it would be a forest and thus two-colorable. Algorithm 4.1 does not use information for the first vertex, because here the coloring can be arbitrary.

For the second revealed vertex, even if it is revealed as isolated, only one bit of advice is necessary for knowing whether it gets the same color as the first vertex or a different one. For all further vertices, except the last one, a one-out-of-three decision might be necessary. Summing up, Algorithm 4.1 needs at most $(n-3)$ one-out-of-three and one one-out-of two decisions. We know from Lemma 4.1 that a one-out-of-three decision needs less than 1.5863 bits on average. This leads to less than $1.5863 \cdot (n-3) + 1 + 45 = 1.5863 \cdot n + d$ bits overall. □

Now, we observe that, since vertices in V_1 have outgoing edges only, no two of them can be connected.

Observation 4.1 *Let $G^{\prec}$ be the directed graph resulting from G and the order $\prec$ of revealing. Then V_1 is an independent set in $G^{\prec}$, respectively in G.*

[1]The constant 45 is a result of Lemma 4.1.

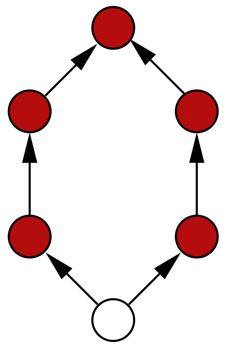
Algorithm 4.1 Optimal online 3-coloring

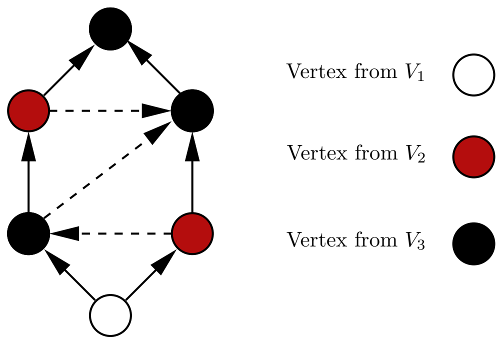
```
Input: Online input instance $G^{\prec}$.
 1: for every revealed vertex $v$ do
 2:   if $v$ is the first revealed vertex then
 3:     Define $c(v) := 1$
 4:   else if $v$ is the second revealed vertex then
 5:     if $v$ is connected to $v_1$ then
 6:       Define $c(v) := 2$
 7:     else
 8:       Read one bit $b$ from the advice tape
 9:       if $b = 0$ then
10:         Define $c(v) := 1$
11:       else
12:         Define $c(v) := 2$
13:       end if
14:     end if
15:     Decide of which type $v$ is
16:     if $v \in V_1$ then
17:       Ask for a one-out-of-three decision from the advice tape $c_v \in \{1, 2, 3\}$
          and define $c(v) := c_v$ ($< 1.5863$ bits)
18:     else if $v \in V_2$ then
19:       Determine the remaining colors $C_v$
20:       if $|C_v| = 1$ then
21:         define $c(v) := c_v \in C_v$
22:       else $\{(|C_v| = 2)\}$
23:         Ask for a one-out-of-two decision from the advice tape $c_v \in C_v$ and
            define $c(v) := c_v$ (1 bit)
24:       end if
25:     else $\{(v \in V_3)\}$
26:       Determine the remaining color $c_v \in C_v$ and define $c(v) := c_v$.
27:     end if
28:   end if
29: end for
Output: The coloring function $c$
```

This leads us to the following lemma, which holds for general chordal graphs.

Lemma 4.3 *Let G be a chordal graph, $(G, \prec)$ be an input instance for OColA, and let $G^{\prec}$ be the corresponding directed graph. For the set $A = V_1 \cup V_2$, the subgraph $G_A^{\prec}$ is a forest.*

Proof. If $G_A^{\prec}$ is not a forest, it contains at least one cycle. This cycle has to be extended by edges to triangles because G is chordal. Hence, such a cycle contains at least three vertices from A which form a triangle. The vertex v_l from this triangle, that is revealed last, has two incoming edges in $G_A^{\prec}$ and is consequently an element of V_3 (see Figure 4.4(a),(b)), which is a contradiction to the assumption. □


(a) A hypothetical cycle in $G^{\prec}$



(b) Types of the vertices when all edges are included

Figure 4.4: Example of a cycle in a chordal graph

In the following, we analyze OColA on the class of outerplanar graphs. Therefore, we need some observations and lemma for this class of graphs. The following observations holds for input instances for OColA in general.

Lemma 4.4 *Let G be an outerplanar graph with $\chi(G) = 3$ on n vertices and let $G^{\prec}$ be an input instance for OColA. For the set V_1 of vertices that are revealed as isolated, $|V_1| \leq 1/2 \cdot n$.*

Proof. Let G be an outerplanar graph and let $G^{\prec}$ be a corresponding input instance. This implies that all vertices in G lie on an outer cycle. We know from Lemma 4.1 that all vertices of type V_1 are independent in G. This implies that, between two vertices $v, w \in V_1$, there has to be at least one vertex $x \in V_2$ to connect them. This yields $|V_2| \geq |V_1| - 1$. Additionally, at least at the end one vertex $y \in V_3$ is necessary to close the cycle, otherwise G would be 2-colorable. □

Using Lemma 4.4, we can analyze Algorithm 4.1 with the following result.

Lemma 4.5 *Let G be a maximal outerplanar graph with $V(G) = n$, and let $G^{\prec}$ be a corresponding input instance for the OColA. Then Algorithm 4.1 colors G optimally, using less than $1.29315 \cdot n + 45$ advice bits.*

Proof. Let G be a maximal outerplanar graph with $V(G) = n$, and let $G^{\prec}$ be a corresponding input instance for the OColA. Let V_1, V_2, and V_3 be the corresponding sets of vertices. According to Lemma 4.4, $|V_1| \leq |V_2| + |V_3|$. Together with Lemma 4.3, it leads to the following inequalities: $\frac{|V_1|}{n} \leq 0.5$ and $\frac{|V_1|}{n} + \frac{|V_2|}{n} < 1$.

For the number of advice bits per vertex A_{BpV} for Algorithm 4.1, we have

$$A_{BpV} \leq \frac{|V_1|}{n} \cdot 1.5863 + \frac{|V_2|}{n}. \tag{4.1}$$

We maximize the right-hand side of (4.1) by setting $\frac{|V_1|}{n} = 0.5$. This implies $\frac{|V_2|}{n} \leq 0.5$, and thus $A_{BpV} \leq 0.5 \cdot 1.5863 + 0.5 = 1.2931$.

For the upper bound on the number of advice bits used by Algorithm 4.1, this means: $A_b \leq 1.29315 \cdot n + d$ where $d \leq 45$ is the number of bits needed to encode the last ≤ 28 one-out-of-three decisions. □

In addition to the results for an optimal coloring, we now give an alternative online algorithm, which colors an arbitrary 3-colorable graph G with 4 colors. The idea is to color all vertices of V_1, which are revealed as isolated, with an additional color 4 and to ask for every revealed vertex from V_2 and V_3 for advice according to an optimal coloring of G using the colors $\{1, 2, 3\}$ (see Algorithm 4.2).

Following this strategy, advice is only necessary for vertices of V_2 (1.5863 bits) and for vertices of V_3 (1 bit). So this strategy is efficient for instances with a high number of isolated vertices.

This leads us to Algorithm 4.3, which combines the strategies of Algorithm 4.1 and Algorithm 4.2. With it, we can color all 3-colorable graphs with at most four colors. To know what to do, the algorithm reads at the beginning the first bit of the advice tape and, depending on this bit, it decides which of the two strategies it follows.

The following lemma shows that Algorithm 4.3 colors G optimally if, for the vertices of $G^{\prec}$, $|V_1|/n \cdot 1.5863 + |V_3|/n \leq 1.15822$. Otherwise it colors G with four colors. In both cases, it needs at most $1.1582196 \cdot n + d$ advice bits.

Lemma 4.6 *Let G be a graph with $V(G) = n$ and $\chi(G) = 3$, and let $G^{\prec}$ be a corresponding input instance for the OColA. There exists an advice tape with which Algorithm 4.3 colors G with four colors, using at most $1.1582196 \cdot n + 46$ bits.*

Algorithm 4.2 Online 4-coloring

```
Input: Online input instance $G^{\prec}$, with $\chi(G) = 3$.
 1: for every revealed vertex $v \in V(G)$ do
 2:   if $v \in V_1$ {isolated vertex} then
 3:     Define $c(v) := 4$
 4:   else if $v \in V_2$ {connected to at least one already colored vertex} then
 5:     Define $C_v$ to be the set of possible colors for $v$ in a 4-coloring
 6:     if $|C_v| = 1$ {only one color remains} then
 7:       Define $c(v) := c_v \in C(V)$
 8:     else if $|C_v| = 2$ {two of the colors $\{1, 2, 3\}$ are possible} then
 9:       Ask for a one-of-two decision from the advice tape $c_v \in C_v$ and define
          $c(v) := c_v$ {1 bit}
10:     else
11:       Ask for a one-of-three decision from the advice tape $c_v \in \{1, 2, 3\}$
          and define $c(v) := c_v$ {$< 1.5863$ bits}
12:     end if
13:   else {$v \in V_3$}
14:     Define $C_v$ to be the set of possible colors for $v$ in a 4-coloring
15:     if $|C_v| = 1$ then
16:       Define $c(v) := c_v \in C(V)$
17:     else {$|C_v| = 2$}
18:       Ask for a one-of-two decision $c_v \in C_v$ and define $c(v) := c_v$ {1 bit}
19:     end if
20:   end if
21: end for
Output: The coloring function $c$
```

Algorithm 4.3 Online graph coloring

```
Input: Online input instance $G^{\prec}$, with $\chi(G) = 3$.
1: Read the first bit $b_1$ of the advice tape
2: if $b_1 = 0$ then
3:   Use Algorithm 4.1
4: else
5:   Use Algorithm 4.2
6: end if
Output: The coloring function $c$
```

Proof. There exists a 4-coloring for G, where all vertices revealed as isolated have the same color, because $\chi(G) = 3$ and the vertices from set V_1 are independent (see Lemma 4.1). In such a coloring, the algorithm needs a one-out-of-three decision for every vertex from V_2 and a one-out-of-two decision for every vertex from V_3, because it is already connected to at least one already colored vertex from V_2.

Now, we compute the maximum of advice bits used, by combining both algorithms. Algorithm 4.1 uses $\leq |V_1| \cdot 1.5863 + |V_2|$ bits, and Algorithm 4.2 uses $\leq |V_2| \cdot 1.5863 + |V_3|$ many bits. This leads to a maximal number of advice bits per vertex at $\frac{|V_1|}{n} = 0.26986$, $\frac{|V_2|}{n} = 0.73014$ and thus to at most 1.1582196 bits per vertex. This leads to an upper bound of $1.1582196 \cdot n + d$ bits overall, for every 3-colorable graph of n vertices. □

For giving the idea of the next algorithm, we analyze, for a chordal graph G, the graph G' which is obtained from G by edge contraction.

Lemma 4.7 *Let G be a chordal graph with $\chi(G) = c$. For every graph G', that is obtained by contracting an edge of G, G' is chordal and $\chi(G') \leq c$.*

Proof. Assume that G is a chordal graph with $\chi(G) = c$ and G' is obtained by contracting the edge $\{a, b\}$ in G. Assume that G' contains a vertex-induced cycle of length > 3 containing x, where x is the vertex contracted from edge $\{a, b\}$. Let be $x, v, w, \cdots, z, x$ this cycle, then either $a, v, w, \cdots, z, a$ is also a cycle of length > 3 in G or $a, v, w, \cdots, z, b, a$ is a cycle of length > 4 in G. Both alternatives are a contradiction to our assumption. It follows that G' is chordal as well.

Now we show that $\chi(G') \leq \chi(G)$. Assume $\chi(G') > \chi(G)$. We have that x is in a clique $C = \{x, v_i, v_2, \cdots, v_c\}$ of size $> \chi(G)$. But the clique size of $\{a, v_i, v_2, \cdots, v_c\}$ and $\{b, v_i, v_2, \cdots, v_c\}$ is at most $\chi(G)$. Thus there exists i, j with $\{a, v_i\} \notin E(G)$ and $\{b, v_j\} \notin E(G)$. If $i = j$, then C would not be a clique of size $> \chi(G)$. Thus a, v_i, v_j, b is a vertex-induced cycle of length 4 in G, which is a contradiction. □

Now, we present an algorithm for coloring 3-colorable chordal graphs with 4 colors. For this, we separate the vertices $V(G)$ into two sets $A := V_1 \cup V_2$ and $B := V_3$. We know that, for every chordal graph G, the graph G_A restricted to the vertices in A is a forest (see Lemma 4.3).

The idea is to color each tree of G_A by a pair of colors $(i, 4)$, for some $i \in \{1, 2, 3\}$. The remaining vertices in V_3 will be colored using only colors $\{1, 2, 3\}$. We will show later that such a coloring always exists.

Before we can describe Algorithm 4.4, we have to move a few vertices inside A. It might happen that a vertex v from V_2 is revealed as the first one of a tree in G_A. This can occur when all its predecessors in $G^{\prec}$ are in V_3 (see v_7, v_8 in Figure 4.5). However, in this case, we note that v cannot have a neighbor in V_1. Such a neighbor w would be revealed after v, and consequently the edge

Algorithm 4.4 Online chordal graph coloring

```
Input: Online input instance $G^{\prec}$, with $\chi(G) = 3$.
1: for every revealed vertex $v \in V(G)$ do
2:   if $x \in V_1'$ {$x$ has only predecessors in $V_3$} then
3:     Ask from the advice tape for a one-out-of-three decision, by which pair
       $p_x \in \{(1,4),(2,4),(3,4)\}$ the tree containg $x$ gets colored, and for a one-out-
       of-two decision, by which color $c_x \in p_x$ this vertex gets colored, and define
       $c(x) := c_x$ (2.5863 bits).
4:   else if $y \in V_2'$ {connected to at least one revealed vertex $x \in V_1'$} then
5:     Identify the tree $x$ belongs to and the pair $p(x)$ for this tree from
       $Pred(x)$.
6:     Define $C_y \in p_x \setminus c(x)$, and define $p_y := p_x$.
7:   else {$z \in V_3$}
8:     Define $C_z$ {Set of possible colors for $z$ in a 4-coloring without color 4.}
9:     if $|C_z = 2|$ then
10:      Ask for a one-of-two decision $c_z \in C_z$ and define $c(z) := c_z$ {1 bit}
11:    else {$|C_z = 1$}
12:      Define $c(z) := c_z \in C_z$ {0 bit}
13:    end if
14:  end if
15: end for
Output: The coloring function $c$
```

orientation would be (v, w), an ingoing edge for w, thus $w \notin V_1$. Therefore, we can define

$$V_{21} = \{v \in V_2 \mid Pred(v) \subseteq V_3\},$$

and move V_{21} to V_1, more precisely $V_1' = V_1 \cup V_{21}$ and $V_2' = V_2 \setminus V_{21}$, while still preserving independence of V_1'.

Looking again at Algorithm 4.2, we observe that all it needs from V_1 is that it is an independent set since all vertices from V_1, and only those, are colored by color 4. Consequently, the algorithm works the same when using V_1' instead of V_1. Let us call this variant Algorithm 4.2′.[2]

We are now ready to describe Algorithm 4.4. Here, V_1' contains exactly those vertices from A which are revealed without a predecessor from A, while, for all vertices in V_2', such a predecessor exists. Consequently, Algorithm 4.4 asks, for every vertex $x \in V_1$, for two pieces of advice. First, it wants to know which pair of colors $(i, 4)$ will be used to color the tree x belongs to. Secondly, it asks which of the two colors x gets itself.

There are three possible pairs of colors. This leads to a combination of a one-out-of-three and a one-out-of-two decision. Thus, at most 2.5863 bits are

[2] Algorithm 4.2′ results from Algorithm 4.2 by substituting V_1' and V_2' for V_1 and V_2, respectively.

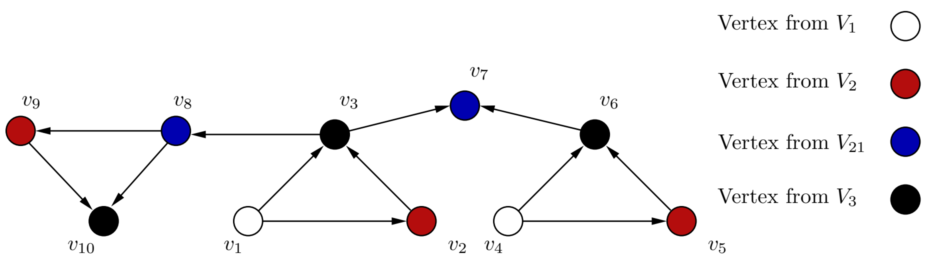


Figure 4.5: Vertices of type V_{21}.

needed for every vertex from V_1.

With this information, obviously the algorithm is able to color all vertices $x \in V_1'$. Also, all vertices from V_2' can be colored because, at the moment a vertex v from V_2' is revealed, it has a predecessor w in A, and, for w, the color pair of the tree both belong to is known as well as the color w gets. Hence, v is colored by the other color from that pair without further advice. Inside the trees of G_A, such a coloring is clearly possible, but we still have to show later that this way a correct coloring of the whole graph is constructed.

Finally, for every vertex $z \in V_3$, which closes one or more triangles, the algorithm asks for a one-out-of-two decision, because such vertices have to be connected to at least two already colored and connected vertices (x, y), with different colors and so, in the worst case, there remain two possible colors for z ($|C_z| \leq 2$). The new algorithm needs at most $|V_1'| \cdot 2.5863 + |V_3| + \mathit{const}$ many bits.

To prove that such a coloring exists for every 3-colorable chordal graph, we give a further algorithm, which describes how an oracle can find the related coloring and with this the right advice tape.

Again, we use $A = V_1 \cup V_2$ and $B = V_3$. We build the graph G' by subsuming every connected component of G_A in one vertex. When G is a 3-colorable chordal graph, the graph G' is 3-colorable as well (see Lemma 4.7). Thus, we use a 3-coloring c' of G'.

The 4-coloring c for G can be derived from c' in the following way. For a vertex $v' \in V(G')$, we distinguish two cases. If v' was constructed by an edge contraction, we color the contracted tree in G with the colors $\{c'(v'), 4\}$, and if v' corresponds directly to a vertex $v \in V(G)$ we define $c(v) := c(v')$. With this procedure, we get an coloring which satisfies the needed properties for Algorithm 4.4. With the corresponding advice tape, Algorithm 4.4 needs $\frac{|V_1|}{n} \cdot 2.5863 + \frac{|V_3|}{n}$ bits of advice per vertex. This leads us to the following lemma.

Lemma 4.8 *Let $G = (V, E)$ be a 3-colorable chordal graph and let $G^{\prec}$ be the corresponding input instance for OColA. There exists a coloring $c : V(G) \rightarrow$*

$\{1, 2, 3, 4\}$ *and with this an advice tape such that Algorithm 4.4 can color $G^{\prec}$ with the coloring function c.*

For proving the claim, we give the algorithm to find such a coloring and prove that, for every 3-colorable chordal graph, such a coloring will be found.

Proof. Let $G^{\prec}$ be the given input instance. From the order of the vertices, we can build three sets of vertices V_1, V_2 and V_3. We build from this two sets $A = V_1 \cup V_2$ and $B = V_3$. We know that the graph G_A is a forest (see Lemma 4.3). Each tree of G_A is two-colorable. In the following, we will color each tree of G_A with two colors, where one of the two colors will be 4 for all trees. To find the corresponding color for each tree, we have to determine a coloring, which fits with a coloring for the vertices in B.

For finding such a coloring, we build the graph G' from G by contraction of edges between vertices of A. This leads us to a surjective function $m : V(G) \to V(G')$. If $T_i \subset A$ is a tree in G_A, then for all $t_j, t_k \in V(T_i), j \neq k$, we have $m(t_j) = m(t_k)$.

We know that G' is 3-colorable and chordal as well as G (see Lemma 4.7), so we can find an optimal coloring c' for G' with three colors, because G is a 3-colorable chordal graph.

We extend the coloring function c' to c for G by coloring all vertices $v \in B \subset V(G)$ which are also vertices in G' with $c(v) := c'(v)$. For all vertices $w_i \in A \subset V(G)$ which are represented by one vertex $x \in V(G')$, with $m(w_i) = x$ we color the corresponding tree alternatingly in $(c'(x), 4)$.

It is obvious that the coloring c needs at most 4 colors because c' was a coloring with 3 colors and the color 4 is used additionally. The new coloring is proper for G because no two vertices with color 4 are connected to each other, and each tree T_i in G_A is colored properly with the two colors $(c'(x), 4)$. If the vertex x represents a tree T_x in G_A, it follows that all vertices in G which are connected to the tree T_x represented by x are connected to x in G'. If c' is a proper coloring for G', it follows that $c'(y) \neq c'(x)$, for all vertices $y \in \mathit{Neigh}_{G'}(x)$. If now the tree which is represented by x in G' is colored only with $c(x)$ and 4, it follows that c is a proper coloring. □

Putting everything together, we can combine the previous algorithms into a final one, Algorithm 4.5. This algorithm uses the first two advice bits to decide which of the Algorithms 4.1, 4.2', 4.4 it shall use. Consequently, it always makes use of the best possible advice-per-vertex ratio among those three. This results in the following analysis.

Theorem 4.3 *Let $G = (V, E)$ be a 3-colorable chordal graph with $|V(G)| = n$ and let $G^{\prec}$ be the corresponding input instance for OColA. Algorithm 4.5 colors $G^{\prec}$ with 4 colors using at most* $0,9865 \cdot n + 47$ *advice bits.*

Proof. We have seen before that there exists an advice tape for any of the three Algorithms 4.1, 4.2', 4.4. Now, we show that, in any case, there is one of the

Algorithm 4.5 Online graph coloring

```
Input: Online input instance $G^{\prec}$, with $\chi(G) = 3$.
1: Read the first two bits $b_1, b_2$ of the advice tape
2: if $b_1 = 1$ then
3:     Use Algorithm 4.1
4: else if $b_1 = 0, b_2 = 1$ then
5:     Use Algorithm 4.2
6: else
7:     Use Algorithm 4.4
8: end if
Output: The coloring function $c$
```

three strategies that colors G using $0,9865 \cdot n + 47$ advice bits. Let $z_1 = \frac{|V_1|}{n}$, $z'_1 = \frac{|V'_1|}{n}$, $z_2 = \frac{|V_2|}{n}$, $z'_2 = \frac{|V'_2|}{n}$ and $z_3 = \frac{|V_3|}{n}$.

For the needed advice bits A_1 for Algorithm 4.1, A_2 for Algorithm 4.2', and A_4 for Algorithm 4.4, we have (with $z_1 + z_2 = z'_1 + z'_2$, $z_1 \leq z'_1$)

$$\begin{aligned} A_1 &\leq 1.5863 \cdot z_1 + z_2 \leq 1.5863 \cdot z'_1 + z'_2 \\ A_2 &\leq 1.5863 \cdot z'_2 + z_3 \\ A_4 &\leq 2.5863 \cdot z'_1 + z_3 \end{aligned}$$

Additionally, $z'_1 + z'_2 + z_3 = 1$. The corresponding convex space has its maximum at $z'_1 = 0.30667$, $z'_2 = 0.5$, and $z_3 = 0.19333$, and there it needs $A_g = 0.98647 \cdot n$ bits.

The additive constant 47 consists of the two bits read at the beginning and the usual 45 bits that can remain in each of the sub-algorithms from the block of bits read for one-out-of-three decisions. □

4.5 Discussion

We introduced first research results for online coloring algorithms with advice for 3-colorable graphs. For planar, chordal and general 3-colorable graphs we presented nearly matching lower and upper bounds on the number of advice bits for the 3-coloring. We also gave 4-coloring online algorithms with advice for those graph classes (see Table 4.1). It remains to extend the lower bounds to the 4-coloring. The extension to other graph classes, k-coloring, and general coloring is also very interesting.

Chapter 5

Bounds on the Advice Complexity of a Generic Online Problem

5.1 Introduction

The concept of advice complexity, as we have seen it in Chapter 4, enables us to perform a much more fine-grained analysis of the hardness of online problems than using the classical competitive analysis. We are especially interested in *lower bounds* on the advice complexity. Such lower bounds do not only tell us something about the information content [53] of online problems, but they also carry over to a randomized setting where they imply lower bounds on the number of random decisions needed to compute a good solution [61]. But, similar to most other computing models, lower bounds on the advice complexity are hard to prove. Thus, it is desirable to have some generic proof methods for establishing lower bounds. This and the next chapter, we take a first step towards this goal by using a generic online problem to show how to transfer lower bounds on its advice complexity to lower bounds for other online problems.

In this chapter, we study the *string guessing problem* with respect to its advice complexity (Section 5.2). This problem is very generic with respect to proving lower bounds on the advice complexity. Here, a string of length n over an alphabet of size q has to be guessed. More specifically, we define two versions of the problem where, in the first case, the algorithm gets immediate feedback which decisions would have been correct up to the current time step and, in the second case, this feedback is not supplied. This problem is similar to the *generalized matching pennies problem* that was introduced in [31]. But the string guessing problem uses a more general cost function that enables reductions from different online problems to it.

First, we prove a lower bound on the advice necessary to achieve some specific number of correct guesses for both versions. Thus, we show that the extra information of knowing the history does not help a lot for this class of problems. Additionally, we analyze the size of the advice depending on both n and q.

Employing this result, we use the string guessing problem as a technique to prove lower bounds for other well-studied online problems in Chapter 6. It seems to be a promising approach to use string guessing this way to show the hardness of further online problems.

5.2 The String Guessing Problem

In many online problems, the question arises whether knowing the history, i. e., the parts of an optimal solution that correspond to the input known at a specific time step, has an effect on the additional information necessary to achieve a certain competitive ratio. We compare these two scenarios on a very generic online problem, called the string guessing problem. In the first variant, the algorithm has to guess a character from some fixed alphabet, then, in the next step, it is told what would have been the correct answer and is asked for the next character. In the second variant, the algorithm also has to guess character by character, but it gets no feedback about whether its answer was correct or not, until the very end of the request sequence. In both cases, the length n of the string is given as the first request and the algorithm then has to guess n characters step by step.

Let us begin by defining the two variants of the string guessing problem formally.

Definition 5.1 (String Guessing with Known History) *We define the **string guessing problem with known history over an alphabet Σ of size $q \geq 2$ (q-SGKH)** is the following online problem. The input $I = (n, d_1, d_2, \ldots, d_n)$ consists of a natural number n and the characters $d_1, d_2, \ldots, d_n$, $d_i \in \Sigma$, that are revealed one by one. The online algorithm $\mathtt{A}$ computes the output sequence $\mathtt{A}(I) = y_1y_2\ldots y_n$, where $y_i = f(n, d_1, \ldots, d_{i-1}) \in \Sigma$, for some computable function f. The cost of a solution $\mathtt{A}(I)$ is the number of wrongly guessed characters, i. e., the Hamming distance* $\mathrm{Ham}(d, \mathtt{A}(I))$ *between $\mathtt{A}(I)$ and $d = d_1d_2\ldots d_n$.*

Definition 5.2 (String Guessing with Unknown History) *We define the **string guessing problem with unknown history over an alphabet Σ of size $q \geq 2$ (q-SGUH)** as the following online problem. The input $I = (n, ?_2, \ldots, ?_n d)$, for $d = d_1, \ldots, d_n \in \Sigma^n$, consists of the input size n in the first request and $n - 1$ subsequent requests "?" carrying no extra information. In each of the first n time steps, the online algorithm $\mathtt{A}$ is required to output one character from Σ, forming the output sequence $\mathtt{A}(I) = y_1y_2\ldots y_n$. In the last request, the string d is revealed. The algorithm is not required to respond with* *any output in this time step. The cost of a solution* $\mathtt{A}(I)$ *is again the Hamming distance between* $\mathtt{A}(I)$ *and* d*.*

For simplicity, we sometimes speak about the input string $d = d_1d_2\dots d_n$ when we mean the input sequence $I = (n, d_1, d_2, \dots, d_n)$ or $I = (n, ?_2, ?_3, \dots, ?_n, d)$ with $n = |d|$. Also, we write $\mathtt{A}(d)$ instead of $\mathtt{A}(I)$.

Since the cost of an optimal solution for any string guessing instance is always 0, it is not meaningful to consider the competitive ratio as a measure for these problems. We will therefore restrict our analysis to the number of errors produced by an algorithm. Our goal is to minimize this number of errors.

5.2.1 Some Special Cases

First, we give some results for algorithms without advice, for the necessary advice to be optimal, and for the case where a constant-size advice is given. It is easy to see that, for every online algorithm without advice, there exists an input string of length n such that the algorithm produces n mistakes. This holds for any alphabet of arbitrary size $q \geq 2$. To see this, consider an adversary `Adv` that, in each step, produces an input character α_i differing from the deterministic output $y_i = f(n, \alpha_1, \dots, \alpha_{i-1})$ of the algorithm. Obviously, no deterministic online algorithm gains anything by knowing the history as its decision, in any case, is purely deterministic also if it is based on the correct answers of previous time steps.

We now consider online algorithms with advice as introduced in Definition 1.17. Let us first look at the number of advice bits sufficient to guess all characters correctly. We can interpret each character as a number and the number obtained by concatenating all correct characters of the n requests as a q-ary number with n digits. An upper bound of $\lceil n \log_2 q \rceil$ can easily be achieved by writing this number of all correct characters in a binary encoding onto the advice tape. We now show that this bound is tight for both problems.

Theorem 5.1 *Every online algorithm* `A` *with advice for* q-SGKH *needs to read at least* $\lceil n \log_2 q \rceil$ *advice bits to be optimal on any input of length* n*.*

Proof. Assume `A` reads $m < \lceil n \log_2 q \rceil$ advice bits. There are q^n possible different input strings, but only $2^m \leq 2^{\lceil n \log_2 q \rceil - 1} < 2^{n \log_2 q} = q^n$ different advices. Thus, at least two different input strings $I_1 = (n, d_1, \dots, d_n)$ and $I_2 = (n, d'_1, \dots, d'_n)$ of length n get the same advice. There is one position in the string where $d = d_1d_2\dots d_n$ and $d' = d'_1d'_2\dots d'_n$ differ for the first time. The algorithm `A` makes a deterministic decision in the corresponding time step that is optimal for at most one of the two inputs and `Adv` can always choose the other one. □

Next, we consider the version of the problem where all decisions have to be made without getting feedback after every time step. Obviously, all lower bounds for q-SGKH directly carry over to q-SGUH.

Theorem 5.2 *Every online algorithm with advice for* q-SGUH *needs to read at least* $\lceil n \log_2 q \rceil$ *advice bits to be optimal on any input of length* n.

Together with our observation preceding Theorem 5.1, the above bounds are tight. Now we analyze the situation that we are given an advice of constant size. Here, we get the following upper bound for q-SGUH which immediately carries over to q-SGKH.

Theorem 5.3 *There exists an online algorithm for* q-SGUH *that guesses at least* $\lceil n/q \rceil$ *positions correctly on an input string of size* n *using* $\lceil \log_2 q \rceil$ *advice bits.*

Proof. In every input string of length n over an alphabet of size q, there is at least one character that occurs at least $\lceil n/q \rceil$ times. With $\lceil \log_2 q \rceil$ bits, this character can be selected, and so an algorithm that outputs this character in every step guesses at least $\lceil n/q \rceil$ positions correctly. □

In the remainder of this section, we will estimate the number of advice bits necessary and sufficient to reach a specific cost. We start with lower bounds on the advice complexity.

5.2.2 Lower Bounds

As we have seen, on the one hand, both problems are arbitrary bad when no advice is given. On the other hand, for both problems, $\lceil n \log_2 q \rceil$ advice bits are necessary and sufficient to be optimal. Indeed, as long as we consider purely deterministic strategies, it is easy to see that knowing the history does not help at all. As stated above, the decisions of the online algorithm are deterministic and it does not matter on what they actually depend. In any case, the adversary knows these decisions in advance. However, as soon as an oracle is involved, the situation changes. In this case, it might be possible that the oracle's advice depends on the history and that this fact is used to compress the advice string in some way.

First, we investigate a lower bound on the number of advice bits necessary to guarantee at most a specific number of wrong answers for q-SGUH. Consider an online algorithm `A` using b advice bits. This can be seen as a collection of 2^b different deterministic algorithms. Since all possible inputs look the same on the first n requests, the behavior of these algorithms can only depend on the advice.

For each of the q^n possible inputs, the oracle can choose between 2^b different algorithms, each of which produces a fixed output string. The oracle has to construct a set of 2^b such strings, which we call center strings, in such a way that the maximum distance of any input string to the nearest of these center strings is minimized. This is exactly the task of constructing a so-called *covering code.* A covering code $K_q(n, r)$ for an alphabet Σ of size q of the strings of length n with radius r is defined as a set of code words (elements of Σ^n) with the property that every string in Σ^n has a distance smaller than or equal to r to at least one code word in $K_q(n,r)$. For an overview of covering codes, we recommend [36]. Thus, the minimum size of a covering code $K_q(n,r)$ gives us the number of different advice strings we need to make sure that the worst-case error over all inputs for q-SGUH is at most r.

To get a simple lower bound on the size of a covering code $K_q(n,r)$, we consider the Hamming balls of radius r around the center strings. A Hamming ball of radius r around a string s in Σ^n consists of all strings t with $\mathrm{Ham}(s,t) \le r$. Due to the symmetry of the Hamming distance, the size of a Hamming ball of radius r around some string s does not depend on s. We denote it by $\mathrm{Vol}_q(n,r)$. Assume that the Hamming balls of radius r around all center strings were pairwise disjoint. Then the number b of advice bits to make sure that no error greater than r occurs for any input string has to satisfy the condition

$$2^b \cdot \mathrm{Vol}_q(n,r) \ge q^n. \tag{5.1}$$

The volume of a Hamming ball is given by

$$\mathrm{Vol}_q(n,r) = \sum_{i=0}^{r} \binom{n}{i} (q-1)^i \tag{5.2}$$

and can be estimated as follows.

Lemma 5.1 (Guruswami et al. [49]) *Let $p \in \mathbb{R}$, $0 < p \le 1 - 1/q$. For sufficiently large n, we obtain $\mathrm{Vol}_q(n,pn) \le q^{H_q(p)n}$, where $H_q(p) = p\log_q(q-1) - p\log_q p - (1-p)\log_q(1-p)$ is the q-ary entropy function.*

Proof. The following proof is taken from [49].

$$\begin{aligned}
1 &= (p + (1-p))^n \\
&= \sum_{i=0}^{n} \binom{n}{i} p^i (1-p)^{n-i} \\
&\ge \sum_{i=0}^{pn} \binom{n}{i} p^i (1-p)^{n-i} \\
&= \sum_{i=0}^{pn} \binom{n}{i} (q-1)^i \left(\frac{p}{q-1}\right)^i (1-p)^{n-i} \\
&= \sum_{i=0}^{pn} \binom{n}{i} (q-1)^i (1-p)^n \left(\frac{p}{(q-1)(1-p)}\right)^i \\
&\ge \sum_{i=0}^{pn} \binom{n}{i} (q-1)^i (1-p)^n \left(\frac{p}{(q-1)(1-p)}\right)^{pn}
\end{aligned}$$

$$= \underbrace{\sum_{i=0}^{pn} \binom{n}{i} (q-1)^i}_{Vol_q(0,pn)} \left(\frac{p}{q-1}\right)^{pn} (1-p)^{(1-p)n}.$$

Together with $q^{-H_q(P)n} = \left(\frac{p}{q-1}\right)(1-p)^{(1-p)n}$ we get

$$1 \geq Vol_q \cdot q^{-H_q(P)n}.$$

The calculation immediately yields

$$\mathrm{Vol}_q(n, pn) \leq q^{H_q(p)n} = \left(\frac{q-1}{p}\right)^{pn} \left(\frac{1}{1-p}\right)^{(1-p)n}, \tag{5.3}$$

which concludes the proof. □

This observation leads to the following lower bound for q-SGUH.

Theorem 5.4 *Consider an input string of length n for q-SGUH, for some $n \in \mathbb{N}$. The minimum number of advice bits that can guarantee some online algorithm to be correct in more than αn characters, for $1/q \leq \alpha < 1$, is*

$$\left(1 + (1-\alpha)\log_q\left(\frac{1-\alpha}{q-1}\right) + \alpha \log_q \alpha\right) n \log_2 q = (1 - H_q(1-\alpha))\, n \log_2 q.$$

Proof. Guessing at least αn characters correctly means there can be at most $(1-\alpha)n$ errors. We know from (5.1) and (5.2) that, in order to guarantee that the algorithm makes less than r errors, we need at least b advice bits such that

$$\frac{q^n}{2^b} \leq \sum_{i=0}^{r} \binom{n}{i} (q-1)^i.$$

To give a lower bound on b, we define $\alpha' = 1 - \alpha$, substitute r by $\alpha' n$ and, together with (5.3), we get

$$\frac{q^n}{2^b} \leq \sum_{i=0}^{\alpha' n} \binom{n}{i} (q-1)^i \leq \left(\frac{q-1}{\alpha'}\right)^{\alpha' n} \left(\frac{1}{1-\alpha'}\right)^{(1-\alpha')n}.$$

After taking the logarithm with base q on both sides, we get

$$\begin{aligned}
& n - \log_q 2^b \leq \alpha' n \log_q\left(\frac{(q-1)n}{\alpha' n}\right) + (n - \alpha' n)\log_q\left(\frac{n}{n-\alpha' n}\right) \\
\iff \quad & -b \log_q 2 \leq -\alpha' n \log_q(\alpha' n) + \alpha' n \log_q(q-1) + \alpha' n \log_q(n - \alpha' n) \\
& \qquad - n - n \log_q(n - \alpha' n) + n \log_q n \\
\iff \quad & b \log_q 2 \geq \alpha' n(\log_q(\alpha' n) - \log_q(q-1) - \log_q(n - \alpha' n)) \\
& \qquad + n(1 + \log_q(n - \alpha' n) - \log_q n) \\
\iff \quad & b \geq \big(1 + \alpha' \log_q(\alpha' n) + (1-\alpha')\log_q(n - \alpha' n) - \log_q n \\
& \qquad -\alpha' \log_q(q-1)\big)\, n \log_2 q.
\end{aligned}$$

We now resubstitute α' and finally obtain

$$\begin{aligned}
b &\geq (1 + (1-\alpha)\log_q((1-\alpha)n) + (1-(1-\alpha))\log_q(n-(1-\alpha)n) - \log_q n \\
&\quad - (1-\alpha)\log_q(q-1))n\log_2 q \\
&\geq (1 + (1-\alpha)\log_q(1-\alpha) + (1-\alpha)\log_q n + \alpha\log_q(\alpha n) - \log_q n \\
&\quad - (1-\alpha)\log_q(q-1))n\log_2 q \\
&\geq \big(1 + (1-\alpha)\log_q(1-\alpha) + \alpha\log_q\alpha - (1-\alpha)\log_q(q-1)\big)\, n\log_2 q \\
&\geq \left(1 + (1-\alpha)\log_q\left(\frac{1-\alpha}{q-1}\right) + \alpha\log_q\alpha\right) n\log_2 q \\
&= (1 - H_q(1-\alpha))\, n\log_2 q.
\end{aligned}$$

Thus, we have established a lower bound on b to guarantee at least αn correct characters or, in other words, a maximal error of $(1-\alpha)n$. $\square$

The above argument heavily relies on the fact that, in case of q-SGUH, the output of a deterministic algorithm is unambiguously determined by the given advice. In the case of q-SGKH, this is no longer true. A deterministic algorithm might base its output on the history and thus might output different strings while reading the same advice. In the following we show that, despite this complication, the same lower bound as in Theorem 5.4 also holds for q-SGKH.

For the analysis, we use the q-ary tree T_n of depth n as a representation of the set Σ^n of all input strings of length n over the alphabet Σ (see Figure 5.1). For $0 \leq i \leq q^n - 1$, the leaf $v_{(0,i)}$ represents the ith string in lexicographic order in Σ^n and every inner vertex $v_{(h,i)}$ represents all 2^h strings of the leaves of the subtree rooted in $v_{(h,i)}$.

Let `A` be an online algorithm for q-SGKH that uses at most b advice bits for any input instance of length n. Due to the pigeonhole principle, at least one advice string is used for at least $\lceil q^n/2^b \rceil$ different input instances. For a given advice string s of length b, we now take a closer look at the set $\mathcal{I}_s$ of input strings for which `A` gets the advice string s. `A` is not able to distinguish between any two strings in $\mathcal{I}_s$ at the beginning of the computation. However, this situation can change during the computation since `A` gets the additional information of what would have been the correct output in every time step.

For the analysis, we investigate how large $\mathcal{I}_s$ can maximally be such that `A` can guarantee a maximal error of r. We can view every computation of an online algorithm as a path in T_n from the root down to a leaf. In every time step, the algorithm decides which subtree to enter. In the following step, it is revealed which direction would have been correct. If instances in more than one subtree of some vertex are represented by the given advice, the algorithm cannot know which subtree is correct.

For any vertex v in T_n, let $F(v)$ denote the maximal number of errors the adversary `Adv` can enforce in the partial input string inside the subtree rooted

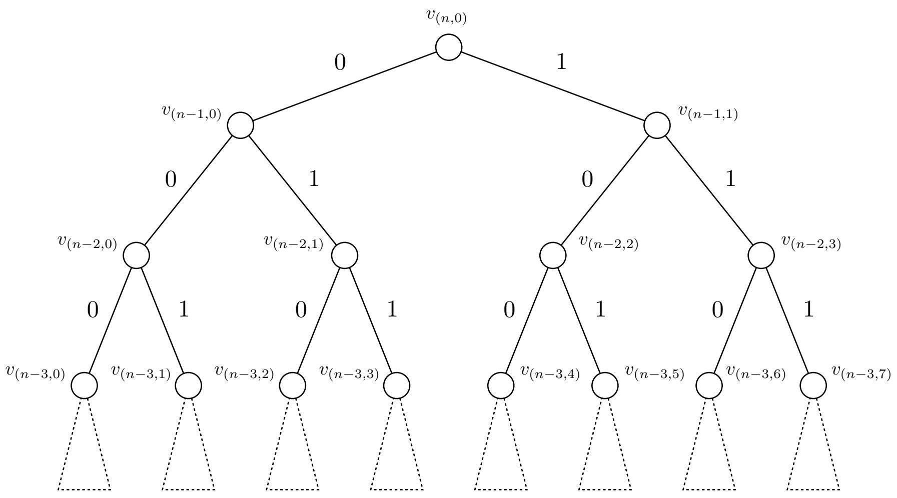


Figure 5.1: Binary tree T_n representing all input instances of size n for $q = 2$.

at v, in addition to the errors already made on the way from the root to v. Moreover, let $\Phi\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be a function such that $\Phi(h, r)$ measures how many strings in $\mathcal{I}_s$ can at most be represented by a vertex at depth h such that the enforceable error is at most r. We are interested in the value $\Phi(n, r)$ which gives us the desired lower bound. The function $\Phi(h, r)$ can be computed as follows.

Lemma 5.2 *For* $0 \leq r \leq h \leq n$, *we have* $\Phi(h,r) = \sum_{i=0}^{r} \binom{h}{i}(q-1)^i = \mathrm{Vol}_q(h,r)$.

From Lemma 5.2 for $h = n$ together with Theorem 5.4, we immediately get the same lower bound on the advice complexity for q-SGKH.

Proof. The function F can be computed recursively as follows. Let $v \in V$ be a vertex in T_n such that the subtree rooted at v contains at least one vertex from $\mathcal{I}_s$ and $\max\{F(u) \mid u \text{ is a child of } v\} = m$. Then,

$$F(v) = \begin{cases} m+1 & \text{if there is another child } w \text{ of } v \text{ with } F(w) = m, \\ m & \text{else.} \end{cases} \tag{5.4}$$

To prove (5.4), we distinguish two cases. In the first case, there are two or more subtrees with the same maximal value of F. In the second case, there exists exactly one subtree with maximal error.

Case 1. There are at least two children u and w of v with $F(u) = F(w) = m$. Thus, it does not matter which subtree the algorithm chooses, because

Adv will choose another subtree with maximal error and thus enforce one error in the current time step as well as the m errors in the corresponding subtree.

Case 2. There is exactly one child u of v with $F(u) = m$ and, for all other children w of v, $F(w) < m$. The algorithm should choose the subtree rooted at u and accept an error of m. Otherwise, Adv would choose the subtree rooted at u and thus enforce one error in the current time step and overall $m + 1$ errors in the subtree rooted in v.

We now show that the function $\Phi(h, r)$ satisfies the recurrence relation

$$\Phi(h, 0) = 1, \tag{5.5}$$

$$\Phi(h, h) = q^h, \text{and} \tag{5.6}$$

$$\Phi(h, r) = \Phi(h - 1, r) + (q - 1) \cdot \Phi(h - 1, r - 1), \text{ for } 0 < r < h. \tag{5.7}$$

To prove (5.5), assume there are two input strings represented by two leaves in T_h. These two leaves have a lowest common ancestor $v_{(g,j)}$, with $1 \leq g \leq h$. When the algorithm comes to $v_{(g,j)}$, it has to choose one of the q successors. It does not matter which subtree the deterministic algorithm takes, Adv can always choose another one, i. e., another possible input string, and hence enforce one error. Thus, $\Phi(h, 0) = 1$.

It is obvious that, if there are h errors allowed, it does not matter what the algorithm does at depth h because, in the worst case, the algorithm makes one error per step and comes to at most h errors. In other words, a subtree at depth h with h allowed mistakes can represent q^h strings, i. e., $\Phi(h, h) = q^h$, proving (5.6).

Additionally, we know from (5.4) that, for a vertex v at depth h with a maximally enforceable error of r, the maximal error in all q subtrees of v cannot be larger than r. Furthermore, we know that no two subtrees can have an error of r. To maximize the number of errors in the subtree rooted at v, one child is assigned an error of r and all others an error of $r - 1$. The maximal number of instances represented by a tree of depth h when r mistakes are allowed is thus $\Phi(h, r) = \Phi(h - 1, r) + (q - 1) \cdot \Phi(h - 1, r - 1)$, which proves (5.7).

Using this recurrence, we are now able to prove the claim of this lemma by induction on h. We already know that $\Phi(1, 0) = 1 = \sum_{i=0}^{0} \binom{1}{i}(q - 1)^i$ and $\Phi(1, 1) = q = \sum_{i=0}^{1} \binom{1}{i}(q - 1)^i$. Now we prove the statement for $h > 1$ and $0 \leq r \leq h$.

As induction hypothesis, assume that

$$\Phi(h, r) = \sum_{i=0}^{r} \binom{h}{i} (q - 1)^i. \tag{5.8}$$

Note that $\Phi(h + 1, r) = \Phi(h, r) + (q - 1) \cdot \Phi(h, r - 1)$ holds due to (5.7) and recall that $\binom{n}{k} + \binom{n}{k-1} = \binom{n+1}{k}$. We get

$$\begin{aligned}
\Phi(h+1,r) &= \Phi(h,r) + (q-1)\cdot\Phi(h,r-1)\\
&= \sum_{i=0}^{r}\binom{h}{i}(q-1)^i + (q-1)\sum_{i=0}^{r-1}\binom{h}{i}(q-1)^i \qquad \text{(by (5.8))}\\
&= \binom{h}{0}(q-1)^0 + \sum_{i=1}^{r}\binom{h}{i}(q-1)^i + \sum_{i=0}^{r-1}\binom{h}{i}(q-1)^{i+1}\\
&= \binom{h}{0}(q-1)^0 + \sum_{i=1}^{r}\binom{h}{i}(q-1)^i + \sum_{i=1}^{r}\binom{h}{i-1}(q-1)^i\\
&= \binom{h}{0}(q-1)^0 + \sum_{i=1}^{r}\left(\binom{h}{i} + \binom{h}{i-1}\right)(q-1)^i\\
&= \binom{h+1}{0}(q-1)^0 + \sum_{i=1}^{r}\binom{h+1}{i}(q-1)^i\\
&= \sum_{i=0}^{r}\binom{h+1}{i}(q-1)^i.
\end{aligned}$$

It follows that

$$\Phi(h+1,h+1) = \sum_{i=0}^{h+1}\binom{h+1}{i}(q-1)^i = q^{h+1},$$

where the last equation holds due to the binomial theorem. With this, the claim follows for all values of $\Phi(h+1,i)$, for $0 \le i \le h+1$. □

Therefore, Lemma 5.2 gives us the same lower bound for q-SGKH as we have already shown for q-SGUH.

Theorem 5.5 *Consider an input string of length n for q-SGKH, for some $n \in \mathbb{N}$. The minimum number of advice bits for any online algorithm that can guarantee to be correct in more than αncharacters, for $1/q \le \alpha < 1$, is*

$$\left(1 + (1-\alpha)\log_q\left(\frac{1-\alpha}{q-1}\right) + \alpha\log_q\alpha\right) n\log_2 q = (1 - H_q(1-\alpha))\, n\log_2 q. \quad \square$$

Let us give the following useful corollary for the *bit string guessing problem* (i. e., for $q = 2$) which directly follows from Theorem 5.4.

Corollary 5.1 *Consider as input a bit string of length n for 2-SGKH. Every deterministic algorithm that can guarantee to be correct in more than αn bits, for $1/2 \le \alpha < 1$, needs to read at least*

$$(1 + (1-\alpha)\log_2(1-\alpha) + \alpha\log_2\alpha)n$$

many advice bits. □

5.2.3 Upper Bounds

To give an upper bound on the advice complexity of q-SGUH on strings of length n with maximal error r, we analyze the minimal size of a covering code of length n with radius r.

Lemma 5.3 (Moser and Scheder [66]) *Let $n \in \mathbb{N}^{>0}, q \in \mathbb{N}^{>1}, r \in \mathbb{N}$. On any alphabet Σ of size q, there exists a covering code of length n of covering radius r of size at most*

$$\left\lceil \frac{n \cdot \ln q \cdot q^n}{\mathrm{Vol}_q(n,r)} \right\rceil = \left\lceil \frac{n \cdot \ln q \cdot q^n}{\sum_{i=0}^{r} \binom{n}{i}(q-1)^i} \right\rceil .$$

Proof. For completeness, we repeat the probabilistic proof here. Let $m := \lceil (n \cdot \ln q \cdot q^n)/(\sum_{i=0}^{r} \binom{n}{i}(q-1)^i) \rceil$. We show that a set of m points chosen uniformly at random and independently from $\{1, \dots, q\}^n$ builds a covering code C of radius at most r with a probability greater than zero. Let s be a fixed string from $\{1, \dots, q\}^n$. The event $s \in \mathit{Ball}(t, r)$ that the string s belongs to a Hamming ball $\mathit{Ball}(t, r)$ of radius r around a randomly chosen element $t \in \{1, \dots, q\}^n$ has a probability of $\frac{\mathit{Vol}_q(n,r)}{q^n}$. The probability of the complementary event is therefore

$$P(s \notin \bigcup_{t \in C} \mathit{Ball}(t, r)) = \left(1 - \frac{\mathit{Vol}_q(n,r)}{q^n}\right)^{|C|} < \mathrm{e}^{-|C| \cdot \mathit{Vol}_q(n,r)/q^n}.$$

For $|C| = \left\lceil \frac{n \cdot \ln q \cdot q^n}{\mathit{Vol}_q(n,r)} \right\rceil$, this yields

$$P(s \notin \bigcup_{t \in C} \mathit{Ball}(t, r)) < \mathrm{e}^{-|C| \cdot \mathit{Vol}_q(n,r)/q^n} \le \mathrm{e}^{-n \ln q} = q^{-n}.$$

By the union bound, the probability that there is any $u \in \{1, \dots, q\}^n$ not covered by the covering code $|C|$ with radius r is at most q^n times $P(s \notin \bigcup_{t \in C} \mathit{Ball}(t, r))$ and thus smaller than 1. Hence, there is a positive probability that C is a covering code with radius r. □

To estimate the upper bound on the advice to guarantee a certain number of correct characters, we need a lower bound on the volume of the Hamming ball of a given radius r.

Lemma 5.4 *Let $p \in \mathbb{R}$, $0 \le p \le 1 - 1/q$, such that $pn \in \mathbb{N}$. For sufficiently large n, $\mathrm{Vol}_q(n, pn) \ge q^{H_q(p) \cdot n - \frac{1}{2} \log_q(2n)}$, where*

$$H_q(p) = p \log_q(q-1) - p \log_q p - (1-p) \log_q(1-p)$$

is the q-ary entropy function.

Proof. We know from [65] that

$$\binom{n}{pn} \geq \frac{1}{\sqrt{8np(1-p)}} \cdot 2^{H_2(p)\cdot n}.$$

It follows that

$$\begin{aligned}\mathrm{Vol}_q(n,pn) &= \sum_{i=0}^{pn} \binom{n}{i} (q-1)^i \\ &\geq \binom{n}{pn} \cdot (q-1)^{pn} \geq \frac{1}{\sqrt{8np(1-p)}} \cdot 2^{H_2(p)\cdot n} \cdot (q-1)^{pn}.\end{aligned}$$

Together with the simple fact that $2^{H_2(p)\cdot n} \cdot (q-1)^{pn} = q^{H_q(p)\cdot n}$, we get

$$\mathrm{Vol}_q(n,pn) \geq \frac{q^{H_q(p)\cdot n}}{\sqrt{8np(1-p)}} \geq \frac{q^{H_q(p)\cdot n}}{\sqrt{2n}} = q^{H_q(p)\cdot n - \frac{1}{2}\log_q(2n)}. \qquad \square$$

Now we are ready to prove an upper bound on the number of advice bits sufficient to guarantee αn correctly guessed characters.

Theorem 5.6 *Consider all inputs of length n for q-*SGUH*, for some $n \in \mathbb{N}$. There is an online algorithm that is correct in more than αn characters for each input of q-*SGUH *of length n, for $1/q \leq \alpha < 1$, and needs at most*

$$\lceil (1 - H_q(1-\alpha))\, n \log_2 q + (3\log_2 n)/2 + \log_2(\ln q) + 1/2 \rceil$$

many advice bits.

Proof. Guessing at least αn characters correctly means there can be at most $\alpha' n$ errors for $\alpha' = 1 - \alpha$. To guarantee that there are at most $\alpha' n$ errors, we need to cover the strings of Σ^n with Hamming balls of radius at most $\alpha' n$. We know from Lemma 5.3 that there exists such a covering with at most

$$\left\lceil \frac{n \cdot \ln q \cdot q^n}{\mathrm{Vol}_q(n, \alpha' n)} \right\rceil$$

balls.

Such a covering leads to an algorithm that can guarantee that there are at most $\alpha' n$ errors and that uses b advice bits such that b is the smallest integer satisfying

$$2^b \geq \left\lceil \frac{n \cdot \ln q \cdot q^n}{\mathrm{Vol}_q(n, \alpha' n)} \right\rceil. \tag{5.9}$$

From Lemma 5.4, we know that

$$\mathrm{Vol}_q(n, \alpha' n) \geq q^{H_q(\alpha')\cdot n - \frac{1}{2}\log_q(2n)} \overset{(5.3)}{=} \left(\frac{q-1}{\alpha'}\right)^{\alpha' n} \left(\frac{1}{1-\alpha'}\right)^{(1-\alpha')n} \cdot \frac{1}{\sqrt{2n}}.$$

Thus, it is sufficient for b to satisfy

$$\frac{n \cdot \ln q \cdot q^n}{2^b} \leq \left(\frac{q-1}{\alpha'}\right)^{\alpha' n} \left(\frac{1}{1-\alpha'}\right)^{(1-\alpha')n} \cdot \frac{1}{\sqrt{2n}}.$$

After taking the logarithm to the base q on both sides, we get

$$\begin{aligned}\log_q n + n + \log_q(\ln q) - \log_q\left(2^b\right) \leq{} & \alpha' n \log_q\left(\frac{(q-1)n}{\alpha' n}\right) \\ & + (n - \alpha' n)\log_q\left(\frac{n}{n-\alpha' n}\right) \\ & - \frac{1}{2}\log_q(2n)\end{aligned}$$

which is equivalent to

$$\begin{aligned}-b\log_q 2 \leq{} & -\alpha' n \log_q(\alpha' n) + \alpha' n \log_q(q-1) + \alpha' n \log_q(n-\alpha' n) - n \\ & - n\log_q(n-\alpha' n) + n\log_q n - \frac{1}{2}\log_q(2n) - \log_q(\ln q) - \log_q n.\end{aligned}$$

Dividing by $-\log_q 2$ yields

$$\begin{aligned}b \geq{} & \alpha' n \log_2(\alpha' n) - \alpha' n \log_2(q-1) - \alpha' n \log_2(n-\alpha' n) + n \\ & + n\log_2(n-\alpha' n) - n\log_2 n + \frac{1}{2}\log_2(2n) + \log_2(\ln q) + \log_2 n \\ \geq{} & (1 + \alpha' \log_q(\alpha' n) + (1-\alpha')\log_q(n-\alpha' n) - \log_q n \\ & - \alpha' \log_q(q-1))\, n\log_2 q + \frac{1}{2}\log_2(2n) + \log_2(\ln q) + \log_2 n \\ ={} & \big(1 + \alpha' \log_q \alpha' + (1-\alpha')\log_q(1-\alpha') - \alpha' \log_q(q-1)\big)\, n\log_2 q \\ & + \frac{1}{2}\log_2(n) + \log_2(\ln q) + \frac{1}{2} + \log_2 n \\ ={} & \big(1 + \alpha' \log_q \alpha' + (1-\alpha')\log_q(1-\alpha') - \alpha' \log_q(q-1)\big)\, n\log_2 q \\ & + \frac{3}{2}\log_2 n + \log_2(\ln q) + \frac{1}{2} \\ ={} & \left(1 + \alpha' \log_q\left(\frac{\alpha'}{q-1}\right) + (1-\alpha')\log_q(1-\alpha')\right) n\log_2 q + \frac{3}{2}\log_2 n \\ & + \log_2(\ln q) + \frac{1}{2} \\ ={} & (1 - H_q(\alpha'))\, n\log_2 q + \frac{3}{2}\log_2 n + \log_2(\ln q) + \frac{1}{2}.\end{aligned}$$

We now resubstitute α' by $1-\alpha$ and finally get

$$b \geq (1 - H_q(1-\alpha))\, n\log_2 q + \frac{3}{2}\log_2 n + \log_2(\ln q) + \frac{1}{2}.$$

Because we wanted to find the minimum value for b such that (5.9) is satisfied, we choose

$$b = \left\lceil (1 - H_q(1-\alpha))\, n \log_2 q + (3 \log_2 n)/2 + \log_2(\ln q) + \frac{1}{2} \right\rceil . \qquad \square$$

Hence, we now have an upper bound on the number b of advice bits necessary for an algorithm to guarantee at least αn correctly guessed characters.

Corollary 5.2 *Consider as input a bit string of length n for* 2-SGUH. *There is an online algorithm reading at most*

$$\left\lceil (1 + (1-\alpha) \log_2 (1-\alpha) + \alpha \log_2 \alpha)\, n + (3 \log_2 n)/2 + \log_2(\ln 2) + \frac{1}{2} \right\rceil$$

advice bits and producing outputs that are correct in more than αn bits, for $1/2 \leq \alpha < 1$. $\square$

Chapter 6

String Guessing as a Method to Prove Lower Bounds on the Advice Complexity

6.1 Introduction

Employing the result from Chapter 5, we use the string guessing problem as a technique to prove lower bounds for other well-studied online problems. It seems to be a promising approach to use string guessing this way to show the hardness of further online problems.

Our first application, discussed in Section 6.2, deals with an online version of the maximum clique problem where the vertices of the underlying graph arrive consecutively. In every time step, the online algorithm has to decide whether the current vertex belongs to the solution or not. We give a lower bound on the number of advice bits necessary for optimality or reaching a given competitive ratio. These lower bounds are linear in the number of vertices. In Section 6.3, we give a formal definition of an advice-preserving reduction and use this reduction in Section 6.4 to give a lower bound on the online version of the set cover problem as introduced in [1]. We show how to use the results on the string guessing problem to give a lower bound that closes an exponential gap between the lower and upper bounds given in [62] and prove a linear lower bound for arbitrary competitive ratios.

6.2 The Online Maximum Clique Problem

In the following section, we analyze the *online maximum clique problem* (MaxClique, see [38]). In MaxClique, in every time step, a vertex is given together with all edges to vertices that were already revealed in previous steps, and an online algorithm $\mathtt{A}$ has to decide whether the newly revealed vertex becomes part of the solution or not.

For designing a reasonable cost function, we briefly give some considerations. First, assume there is a maximum clique of size n in the input graph G and the algorithm finds a clique of size $n-1$. Then, intuitively, the cost of the solution should be $n-1$, irrespective of how many vertices of the found clique are also part of the largest clique in G. On the other hand, assume the algorithm selects a vertex that is not connected to any vertex revealed afterwards. Unless this is the only vertex $\mathtt{A}$ takes, $\mathtt{A}$ does not output a clique (i. e., its solution is not feasible). Never the less, we do not want to put such hard restriction onto the output of an algorithm, it should be allowed to give an output, different to the situation in [38], where not all selected vertices are part of a clique. Even if $\mathtt{A}$ gives an output in which many vertices form a large or even a maximum clique, but one additional vertex is selected, the output is indeed no clique, but very close to a relatively good or even optimal solution. Thus, this solution should have almost optimal cost. Then again, we should clearly prevent the algorithm from simply selecting all vertices that are given.

Therefore, we consider, for an output $\mathtt{A}(I)$, the maximum clique $C_{\mathtt{A}(I)}$ in the graph $G_{\mathtt{A}(I)}$ restricted to the selected vertices $\mathtt{A}(I)$. Then, the solution becomes better the larger the maximum clique in $G_{\mathtt{A}(I)}$ is, and it becomes worse the more vertices are selected that are not part of $C_{\mathtt{A}(I)}$. All in all, we propose the cost function given in the following definition.

Definition 6.1 (MaxClique) *The* ***online maximum clique problem,*** **MaxClique** *for short, is the following online problem. The input is a graph $G = (V, E)$ and the goal is to find a clique $C \subseteq V$ in G of maximum size. In each time step i, one vertex $v_i \in V$ is revealed together with all edges $\{\{v_i, v_j\} \in E \mid j < i\}$, and the online algorithm $\mathtt{A}$ has to decide whether $v_i \in C$ or not. Let $\mathtt{A}(I)$ be the set of vertices selected by $\mathtt{A}$ and let $C_{\mathtt{A}(I)}$ be a maximum clique in the graph $G_{\mathtt{A}(I)}$. The cost function is defined by $cost(\mathtt{A}(I)) = \left|C_{\mathtt{A}(I)}\right|^2 / |\mathtt{A}(I)|$.*

Clearly, for the optimal solution $\mathtt{Opt}(I)$ of a graph with a maximum clique C_{opt}, we have $cost(\mathtt{Opt}(I)) = \frac{|C_{\mathrm{opt}}|}{|\mathtt{Opt}(I)|} \cdot |C_{\mathrm{opt}}| = |C_{\mathrm{opt}}|$, thus, the competitive ratio c of $\mathtt{A}$ on I can be computed as $c = \frac{cost(\mathtt{Opt}(I))}{cost(\mathtt{A}(I))} = \frac{|\mathtt{A}(I)|}{|C_{\mathtt{A}(I)}|} \cdot \frac{|C_{\mathrm{opt}}|}{|C_{\mathtt{A}(I)}|}$. In other words, the quality of the algorithm is given by the product of the two ratios $|\mathtt{A}(I)|/|C_{\mathtt{A}(I)}|$ and $|C_{\mathrm{opt}}|/|C_{\mathtt{A}(I)}|$. The first ratio measures how many useless vertices the algorithm has taken and the second ratio measures how many correct vertices the algorithm did not take.

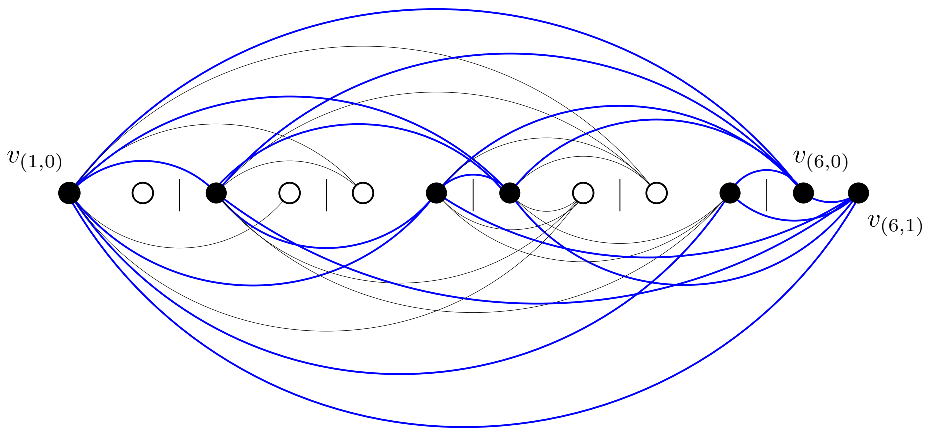


Figure 6.1: The graph G_{00101} constructed from the string guessing instance $s = 00101$.

In order to give a lower bound on the advice complexity of MaxClique, we use our results for 2-SGKH. To this end, we investigate the following subclass of instances, where every instance corresponds to a particular bit string. Let $s = s_1 s_2 \ldots s_{n'}$ be a bit string of length n', for some $n' \in \mathbb{N}$. We construct an input instance I_s for MaxClique corresponding to s as follows. Consider the graph $G_{I_s} = (V(I_s), E(I_s))$ with $n = 2n' + 2$ vertices. Let $V(I_s) = \{v_{(1,0)}, v_{(1,1)}, v_{(2,0)}, v_{(2,1)}, \ldots, v_{(n'+1,0)}, v_{(n'+1,1)}\}$ and let $V'(I_s) = \{v_{(i,s_i)} \mid 1 \leq i \leq n'\}$ be the set of the n' vertices that correspond to the string s. Moreover, let

$$\begin{aligned} E(I_s) = \{&\{v_{(i,s_i)}, v_{(j,k)}\} \mid 1 \leq i < j \leq n', k \in \{0,1\}\} \\ &\cup \{\{v, v_{(n'+1,0)}\}, \{v, v_{(n'+1,1)}\} \mid v \in V'(I_s)\} \\ &\cup \{\{v_{(n'+1,0)}, v_{(n'+1,1)}\}\}. \end{aligned}$$

Clearly, the vertices from $V'(I_s)$ plus the vertices $v_{(n'+1,0)}$ and $v_{(n'+1,1)}$ form a unique optimal solution for I_s of size $n' + 2$. Although the vertices $v_{(i,0)}$ and $v_{(i,1)}$ are revealed separately, for the analysis, we combine them to one pair. After the first pair is revealed, the vertices of the second pair $(v_{(2,0)}, v_{(2,1)})$ are given and so on. An example for the string $s = 00101$ of length 5 is given in Figure 6.1.

Assume that `A` knows that one vertex of each pair is part of the optimal solution. Additionally, we assume that `A` knows how long the instance is. Hence, it knows when the last two vertices $v_{(n'+1,1)}, v_{(n'+1,2)}$ are revealed and selects both of them.

Then, we can see MaxClique as guessing one vertex per pair. Similar to guessing a string s, also when trying to find the correct vertex in a pair

$(v_{(i,0)}, v_{(i,1)})$ in G_{I_s}, the correct decision can only depend on the input known so far, the history, and the given advice. However, in general, an algorithm has four options in every time step. These are to take the first vertex, the second one, both, or none. As a next step, we show that, for any online algorithm with advice, it is the best strategy to take both vertices of any pair for which no advice is used. In this way, we derive an upper bound on the cost of solutions of online algorithms with a restricted advice.

Lemma 6.1 *Let s be an instance of length n' for* 2-SGKH *and let* $\mathtt{B}$ *be the best online algorithm for* 2-SGKH *that reads b advice bits. Let the number of bits that* $\mathtt{B}$ *guesses correctly be at most $\alpha n'$, where $0 \leq \alpha \leq 1$. Then, for all online algorithms* $\mathtt{A}$ *for a corresponding* MaxClique *instance I_s that read b advice bits, we have*

$$cost(\mathtt{A}(I_s)) \leq \frac{(\alpha n + 2 + (1-\alpha)n')^2}{\alpha n' + 2 + 2 \cdot (1-\alpha)n'}.$$

Furthermore, for any online algorithm A for I_s, an online algorithm $\mathtt{A}^*$ *that correctly guesses the same pairs as* $\mathtt{A}$ *and takes both vertices for all remaining pairs satisfies* $((\alpha n + 2 + (1-\alpha)n')^2)/(\alpha n' + 2 + 2 \cdot (1-\alpha)n') \geq cost(\mathtt{A}^*(I_s)) \geq cost(\mathtt{A}(I_s))$.

Proof. First we prove, by a reduction from the string guessing problem, that no algorithm for MaxClique can correctly guess more than $\alpha n'$ pairs when using at most b advice bits. Consider any algorithm $\mathtt{A}$ for MaxClique such that $\tilde{\alpha} n'$ is the number of pairs $(v_{(i,0)}, v_{(i,1)})$ of vertices in G_{I_s} that $\mathtt{A}$ guessed correctly. For the sake of contradiction, suppose $\tilde{\alpha} > \alpha$ and consider the following reduction to solve 2-SGKH with $\tilde{\alpha} n'$ correctly guessed bits. Every time step in 2-SGKH can be transformed into two time steps of MaxClique by the above transformation. We then create an online algorithm $\mathtt{A}'$ for 2-SGKH as follows. According to the output of $\mathtt{A}$ in the two time steps that are associated with one pair, $\mathtt{A}'$ gives the output 0 if $\mathtt{A}$ takes the first vertex and 1 otherwise. Thus, $\mathtt{A}'$ is an online algorithm with advice for 2-SGKH that guesses more than $\alpha n'$ bits correctly while using b advice bits and that is hence strictly better than $\mathtt{B}$, which is a contradiction to our assumption.

It follows that $\tilde{\alpha} \leq \alpha$. We may thus assume that $\mathtt{A}$ guesses exactly $\alpha n'$ pairs correctly for MaxClique, which is, by the above reasoning, the best $\mathtt{A}$ can do. Additionally, we may assume that $\mathtt{A}$ also knows where these $\alpha n'$ pairs lie in the instance I_s. For the rest of the $(1-\alpha)n'$ requests, suppose that $\mathtt{A}$ takes, for a fraction of β, both vertices of the corresponding pair, for a fraction of γ the wrong one, and, for the remainder, no vertex at all. Thus, $\mathtt{A}$ outputs a solution of size $\alpha n' + 2 + 2(1-\alpha)\beta n' + (1-\alpha)\gamma n'$ while there is a clique $C_{\mathtt{A}(I_s)}$ in $G_{\mathtt{A}(I_s)}$ of size $\alpha n' + 2 + (1-\alpha)\beta n'$ yielding

$$cost(\mathtt{A}(I_s)) = \frac{(\alpha n' + 2 + (1-\alpha)\beta n')^2}{\alpha n' + 2 + (1-\alpha)(2\beta + \gamma)n'}.$$

We immediately observe that this term does not depend on the number of pairs for which A chooses no vertex and that it decreases with increasing γ. We therefore set γ to 0 and verify that the remaining term increases with β. To this end, let us substitute $X = \alpha n' + 2$ and $Y = (1-\alpha)n'$ and consider the function

$$f(\beta) = \frac{(\alpha n' + 2 + (1-\alpha)\beta n')^2}{\alpha n' + 2 + (1-\alpha)2\beta n'} = \frac{(X+Y\beta)^2}{X+2Y\beta}.$$

Since the derivative

$$f'(\beta) = \frac{2Y(X+Y\beta)(X+2Y\beta) - 2Y(X+Y\beta)^2}{(X+2Y\beta)^2}$$

is positive for all values of α and β between 0 and 1, we may set β to 1. In other words, the best algorithm $\mathtt{A}^*$ is the one that makes the right decisions on exactly the same set of pairs as A takes both vertices for all remaining pairs. □

In order to give a lower bound on the advice complexity, we analyze an online algorithm with advice that gets a sufficiently large number of advice bits to know αn pairs and, following Lemma 6.1, takes both vertices for all unknown positions. Using our results from Section 5.2, we can prove the following theorem.

Theorem 6.1 *For any* $1 \leq c \leq 1.5$ *and for any* $\varepsilon > 0$*, any* $(c-\varepsilon)$*-competitive online algorithm* A *for* MaxClique *needs at least*

$$(1 + (c-1)\log_2(c-1) + (2-c)\log_2(2-c))\,\frac{n-2}{2}$$

advice bits.

Proof. Let $n' = (n-2)/2$. As above, assume that A reads a sufficiently large number of advice bits to correctly guess $\alpha n'$ pairs. In order to give a lower bound, we again assume that A also knows where these $\alpha n'$ pairs lie in the instance and that, according to Lemma 6.1, A takes both vertices for all pairs where the corresponding bit is unknown. Thus,

$$\mathit{cost}(\mathtt{A}(I)) = \frac{(\alpha n' + 2 + (1-\alpha)n')^2}{\alpha n' + 2 + 2(1-\alpha)n'} = \frac{n'^2 + 2n' + 4}{2n' - \alpha n' + 2} = \frac{n' + 2 + \frac{4}{n'}}{2 - \alpha + \frac{2}{n'}}.$$

For the competitive ratio, we therefore get

$$c = \frac{\mathit{cost}(\mathtt{Opt}(I))}{\mathit{cost}(\mathtt{A}(I))} = \frac{(n'+2)(2-\alpha+\frac{2}{n'})}{n'+2+\frac{4}{n'}} > \frac{n'(2-\alpha)}{n'+2+\frac{4}{n'}} = \frac{(2-\alpha)}{1+\frac{2}{n'}+\frac{4}{n'^2}}.$$

For any α and any ε, there exists an n_0 such that

$$\frac{(2-\alpha)}{1+\frac{2}{n'}+\frac{4}{n'^2}} \geq (2-\alpha-\varepsilon),$$

for all $n' \geq n_0$. In other words, `A` has to guess at least $\alpha n'$ characters correctly to reach a competitive ratio of $2 - \alpha - \varepsilon$. Using Corollary 5.1, we get a lower bound on the number of advice bits necessary to reach a competitive ratio of c of

$$\begin{aligned}
&(1 + (1 - (2 - c)) \log_2(1 - (2 - c)) + (2 - c) \log_2(2 - c))\, n' \\
&= (1 + (c - 1) \log_2(c - 1) + (2 - c) \log_2(2 - c))\, n' \\
&= (1 + (c - 1) \log_2(c - 1) + (2 - c) \log_2(2 - c))\, \frac{n - 2}{2}
\end{aligned}$$

as we claimed. □

Note that, without advice, an online algorithm for MaxClique can reach a competitive ratio of $(n' + 2)/((n' + 2)^2/(2n' + 2)) = (2n' + 2)/(n' + 2) \approx 2$ in these instances by just taking every vertex.

6.3 Advice-Preserving Reductions

In order to use Theorem 5.4 for giving lower bounds on the number of advice bits necessary to achieve a specific competitive ratio, we give a definition of an advice-preserving reduction of online minimization problems.

Definition 6.2 (A-Reduction) *Let U_1 and U_2 be two online problems and let $\mathcal{I}_1$ and $\mathcal{I}_2$ be the sets of possible input instances for U_1 and U_2, respectively. Furthermore, let each $I \in \mathcal{I}_1$ and each $I' \in \mathcal{I}_2$ have the form $I = (x_1, \ldots, x_n)$ and $I' = (x'_1, \ldots, x'_m)$, respectively, for some $n, m \in \mathbb{N}$ and some requests x_i and x'_j. Let $\mathcal{O}_1(I)$ be the set of feasible outputs for some input I of U_1. $\mathcal{O}_1(I)$ has the form $(y_1, \ldots, y_n)$. The set $\mathcal{O}_1 = \bigcup_{I \in \mathcal{I}_1} \mathcal{O}_1(I)$ is the set of all possible outputs for U_1. For U_2, we use an analogous notation. Additionally, we define, for every instance $I = (x_1, \ldots, x_n)$, the set of all prefixes $\mathrm{Pref}(I) = \{(x_1, \ldots, x_j) \mid 1 \leq j \leq n)\}$. Furthermore, we denote by $\mathrm{Pref}(\mathcal{I}) = \bigcup_{I \in \mathcal{I}} \mathrm{Pref}(I)$ the set of all prefixes of all instances in $\mathcal{I} \in \{\mathcal{I}_1, \mathcal{I}_2\}$. The set $\mathrm{Pref}(\mathcal{O})$ is defined analogously, for $\mathcal{O} \in \{\mathcal{O}_1, \mathcal{O}_2\}$.*

*We say that **U_1 is A-reducible to U_2, $U_1 \leq_{\mathrm{A}} U_2$** for short, **with parameters u, v, w** for $u \in \mathbb{N}^+$ and $v, w \in \mathbb{Q}^+$, if there exist two functions*

$$\begin{aligned}
F\colon\ &\mathrm{Pref}(\mathcal{I}_1) \to \mathrm{Pref}(\mathcal{I}_2) \text{ and} \\
H\colon\ &\mathrm{Pref}(\mathcal{O}_2) \to \mathrm{Pref}(\mathcal{O}_1)
\end{aligned}$$

such that, for any $I = (x_1, \ldots, x_n) \in \mathcal{I}_1$,

1. - *$F(x_1, \ldots, x_n) = (x'_1, \ldots, x'_{un}) = I' \in \mathcal{I}_2$ and*
   - *$F(x_1, \ldots, x_k) = (x'_1, \ldots, x'_{uk}) \in \mathrm{Pref}(I')$, for all $1 \leq k \leq n$;*

   and such that, for any $Y_2 = (y'_1, \ldots, y'_{un}) \in \mathcal{O}_2$, *2.* • $H(Y_2) = H(y'_1, \ldots, y'_{un}) = Y_1 = (y_1, \ldots, y_n) \in \mathcal{O}_1$ *and*

• $H(y'_1, \ldots, y'_{uk}) = (y_1, \ldots, y_k) \in \mathrm{Pref}(Y_1)$*, for all* $1 \leq k \leq n$*;*

Furthermore, F and H have to satisfy the following properties:

3. • *F and H are computable deterministically,*

• *$H(F(I)) \in \mathcal{O}_1(I)$ is a feasible output, for all $I \in \mathcal{I}_1$,*

• *for every instance $I \in \mathcal{I}_1$, $cost(Opt(I)) = cost(H(Opt(F(I))))$, and*

• *if Y_2 is an output for $F(I) = I'$ with $cost(Y_2) = c$, computed by an algorithm $\mathtt{A}_2^{\phi}$ using b bits of advice, then there is an algorithm $\mathtt{A}_1^{\phi}$ that, also using b advice bits, computes the solution $H(\mathtt{A}_2(F(I)))$ for I with $cost(H(\mathtt{A}_2(F(I)))) = v \cdot c + w$.*

The scheme of this reduction is shown in Fig.6.2.

$$\begin{tikzcd}
\begin{array}{c} I \in U_1 \\ |I| = n \end{array} \arrow[r, "{F,u,v,w}"] \arrow[d, dashed, "\mathtt{A}_1(I) = H(\mathtt{A}_2(F(I))"] & \begin{array}{c} I' = F(I) \in U_2, \\ |I'| = un \end{array} \arrow[d, "\mathtt{A}_2(F(I))"] \\
\begin{array}{c} Y \in \mathcal{O}_1 \\ (|Y| = n) \end{array} & \begin{array}{c} Y' \in \mathcal{O}_2 \\ (|Y'| = un) \end{array} \arrow[l, "{H,u,v,w}"']
\end{tikzcd}$$

Figure 6.2: Construction of an algorithm $\mathtt{A}_1$ by using the reduction to $\mathtt{A}_2$

If $U_1 \leq_{\mathrm{A}} U_2$ with parameters u, v, w, then we know that, if there exists an algorithm A_2 that solves U_2 with cost c_2 there also exists an algorithm that solves U_1 with cost $c_1 = v \cdot c_2 + w$ using the same number of advice bits.

Thus, such a reduction is a powerful instrument to give lower bounds on the number of advice bits that are necessary to reach a certain competitive ratio. If we can show that the string guessing problem is A-reducible to a problem U_2 (i.e., if q-SGKH $\leq_{\mathrm{A}} U_2$) with parameters u, v, w, this implies for the cost function

$$cost(H(\mathtt{A}_2(I'))) = u \cdot cost(\mathtt{A}_2(I')) + w.$$

We want to employ this reduction for proving lower bounds. More precisely, we want to show that there does not exist any algorithm $\mathtt{A}_2$ for U_2 using b advice bits that can guarantee an output with $cost(\mathtt{A}_2(I'))$, where $u \cdot cost(\mathtt{A}_2(I')) + w$ is smaller than any known lower bound on the $cost(\mathtt{A}_1(I))$ with b advice bits.

It is easy to see that this reduction is transitive. In other words, from $U_1 \leq_{\mathrm{A}} U_2$ with parameters u, v, w and $U_2 \leq_{\mathrm{A}} U_3$ with parameters u', v', w', it follows, that $U_1 \leq_{\mathrm{A}} U_3$ with parameters $u'' = u' \cdot u, v'' = v' \cdot v$, and $w'' = v \cdot w' + w$.

6.4 The Online Set Cover Problem

In this section, we study the advice complexity of the unweighted online set cover problem (SetCover). This problem was introduced in [1] and is defined as follows.

Definition 6.3 (Online Set Cover Problem) *Given a* ground set $X = \{1, 2, \ldots, n\}$ *of size n, a sequence of* requests X' *that are elements from X, and a set family $\mathcal{S} \subseteq Pot(X)$ of size m, without loss of generality $\emptyset \notin \mathcal{S}$, a feasible solution for the* ***online set cover problem*** **(SetCover)** *is any subset $\{S_1, \ldots, S_k\}$ of $\mathcal{S}$ such that*

$$\bigcup_{i=1}^{k} S_i \supseteq X'.$$

The aim is to minimize k, i. e., to use as few sets as possible. In the online version of this problem, the set X and the family $\mathcal{S}$ are known beforehand, but the elements of X' arrive successively one by one in consecutive time steps. An online algorithm $\mathtt{A}$ *solves* SetCover *if, immediately after each yet uncovered request j, it specifies a set $S_i \in \mathcal{S}$ such that $j \in S_i$.*

We may assume, without loss of generality, that no set in $\mathcal{S}$ is the subset of another set in $\mathcal{S}$.

First, we use Theorem 5.4 to give a lower bound on the number of advice bits necessary to achieve a specific competitive ratio that improves over the best known lower bound of c. For this, we first show that the string guessing problem can be reduced to SetCover.

Lemma 6.2 *For any $q \geq 2$, q-SGKH $\leq_{\mathrm{A}}$ SetCover, with parameters $u = 1, v = 1, w = 0$.*

Proof. Let $q \geq 2$. For proving q-SGKH $\leq_{\mathrm{A}}$ SetCover, we construct an algorithm $\mathtt{A}_1$ solving q-SGKH that uses an algorithm $\mathtt{A}_2$ for solving SetCover. For an easier notation, we denote the set of q-SGKH instances of length k by (k, q)-SGKH .

For the reduction, we set the parameters $u = 1, v = 1, w = 0$ and construct a function F that transforms an instance I_B from Σ^* with $|\Sigma| = q$ for (k, q)-SGKH into an instance I' for SetCover. For the first request k, i.e., the length of the string, we construct the sets X and S, i.e., the ground set and the set family for our set cover instance, depending on k and q as follows:

$$X = \{x_s \mid \ s \text{ is a possibly empty string over } \Sigma \text{ of length at most } k\}$$

Thus, X contains $|X| = \sum_{i=0}^{k} q^i = \frac{q^{k+1}-1}{q-1}$ elements. Furthermore,

$$\begin{aligned} \mathcal{S} &= \{\mathrm{tr}(s) \mid s \in \Sigma^k\}, \\ \text{where} \quad \mathrm{tr}(s) &= \{X_{s'} \mid \ s' \text{ is a prefix of } s\} \\ &= \{x_\lambda, x_{s_1}, x_{s_1 s_2}, \ldots, x_{s_1 \ldots s_k}\}, \quad \text{for all } s = s_1 \ldots s_k \in \Sigma^k, \end{aligned}$$

is the *transformation* of the string s. Here, λ denotes the empty string. Each set $\mathrm{tr}(s)$ contains $|\,\mathrm{tr}(s)| = k + 1$ elements, and $\mathcal{S}$ consists of $|\mathcal{S}| = q^k$ sets.

An instance $I_{SG} = (k, e_1, e_2, \ldots, e_k)$, for (k, q)-SGKH , where $k \in \mathbb{N}$, and $e_i \in \Sigma$, $1 \leq i \leq k$, is transformed into an instance $I_{SC} = (x_\lambda, x_{e_1}, \ldots, x_{e_k})$ for SetCover by

$$\begin{aligned} F(k) &= X, \mathcal{S}, x_\lambda, \\ F(k, e_1, \ldots, e_j) &= F(k, e_1, \ldots, e_{j-1}), x_{e_j}. \end{aligned}$$

For each request x_{e_i}, the algorithm A_2 has to select one set of $\mathcal{S}$ that covers the element x_{e_i}. Each element of $\mathcal{S}$ that contains x_{e_i} corresponds to one possible continuation of the input. As long as the further elements of the continuation cover the following requests, no new set has to be selected. As soon one request was answered incorrectly, all other elements of the selected sets cannot be requested, and in the next step a new set has to be taken. As long as the last selected set $\mathrm{tr}(s')$ covers the new element x_{e_i}, the answer of the next character request e_{i+1} corresponds to the next element $x_{e_{i+1}}$ in $\mathrm{tr}(s')$.

Now, we present the function H. For every request x_{e_j}, we have to distinguish two cases.

First, we consider the case where the request has already been solved by the set that was selected in the step before. Assume that the last set output by $\mathtt{A}_2$ on the input $F(k, e_1, \ldots, e_{j-2}), x_{e_{j-1}}$ was $\mathrm{tr}(s')$, for

$$\begin{aligned} s' &= s'_1 s'_2 \ldots s'_{j-1} s'_j s'_{j+1} \ldots s'_k \\ &= e_1 e_2 \ldots e_{j-1} e_j e'_{j+1} \ldots e'_k, \ \text{for some } e'_{j+1}, \ldots, e'_k \end{aligned}$$

Here, the output for I_{SG} is

$$H(\mathtt{A}_2(F(k, e_1, \ldots, e_{j-1}), x_{e_j})) = e'_{j+1}.$$

In the second case, the output $\mathtt{A}_2(F(k, e_1, \ldots, e_{j-2}), x_{e_{j-1}})$ was $\mathrm{tr}(s')$ for

$$\begin{aligned} s' &= s'_1 s'_2 \dots s'_{j-1} s'_j s'_{j+1} \dots s'_k \\ &= e_1 e_2 \dots e_{j-1} e'_j e'_{j+1} \dots e'_k, \end{aligned}$$

for some $e'_j, \dots, e'_k$ where $e'_j \neq e_j$. Note that from $s_j \neq e_j$ follows, by construction, $s_l \neq e_l$, for all $j \leq l \leq k$.

In this case, the output in step j is

$$\mathtt{A}_2(F(k, e_1, \dots, e_{j-1}), x_{e_j}) = \mathrm{tr}(s''),$$

where

$$s'' = e_1 \dots e_j e''_{j+1} \dots e''_k.$$

Here, the output for I_{SG} is $H(\mathtt{A}_2(F(k, e_1, \dots, e_{j-1}), x_{e_j})) = e''_{j+1}$.

For each request for $\mathtt{A}_1$, the function F computes one request for $\mathtt{A}_2$, this corresponds to the parameter $u = 1$.

It remains to show that this is an A-reduction with the parameters $u = 1$, $v = 1$, and $w = 0$. The length of the input for $\mathtt{A}_1$ has the same length as the input for $\mathtt{A}_2$. Additionally, $\mathtt{A}_1$ will make as many errors on the original instance as $\mathtt{A}_2$ makes on the transformed instance. $\mathtt{A}_2$ has to select at least one set. An optimal solution for an instance s' for q-SGKH has cost 0 and corresponds to the selection of the set $\mathrm{tr}(s') \in \mathcal{S}$ with cost 1. It follows that $\mathit{cost}(\mathtt{A}_2(F(I))) = 1 \cdot \mathit{cost}(H(\mathtt{A}_2(F(I)))) + 0$. With this, the claim follows. □

The reduction above helps us to establish an improved lower bound on the advice complexity of SetCover. But first, we have to show that it is equally hard to guess a percentage of α characters over one string of length rk, as to guess the same percentage over r strings of length k over an alphabet of the same size. First, we formally define the problem of guessing r strings of size k. To this end, we use the notion $\circ$ for the concatenation of two instances with same length. Concatenating the two instances $I_1 = (k, d_{(1,1)} \dots, d_{(1,k)})$ and $I_2 = (k, d_{(2,1)} \dots, d_{(2,k)})$ of length $k+1$ yields the instance $I_1 \circ I_2 = ((2,k), d_{(1,1)} \dots, d_{(1,k)}, d_{(2,1)} \dots, d_{(2,k)})$ of length $2k+1$.

Definition 6.4 (*(r,k,q)*-multiple string guessing) *We define the* ***(r,k,q)-multiple string guessing problem with known history over an alphabet Σ of size $q \geq 2$, (r,k,q)*-MultSGKH** *for short, as a model of the* q-SGKH *on r strings $I_1, \dots, I_r$ of length k over an alphabet of size q.*

The input is $I = I_1 \circ \dots \circ I_r$, where $I_i = (k, d_{(i,1)}, \dots, d_{(i,n)})$. Thus, $I = ((r,k), d_{(1,1)}, \dots, d_{(1,k)}, d_{(2,1)}, \dots, d_{(r,k)})$ is an instance of length $r \cdot k + 1$. The last request $d_{(i,k)}$ of the string i has to be answered by the first bit of string $i+1$, for all $i \in \{1, \dots, r-1\}$ The cost of a solution $\mathtt{A}(I)$ is sum of the costs of the r

strings analogous to q-SGKH*,*

$$cost(\mathtt{A}(I)) = \sum_{i=1}^{r} cost(\mathtt{A}(I_i)).$$

The following lemma states that as many advice bits are necessary to guess αrk characters of a string of length rk over an alphabet of size q correctly as for correctly guessing αrk characters of an instance for (r, k, q)-MultSGKH.

Lemma 6.3 *Let* (rk, q)-SGKH *be the* q-SGKH *on a string of length* rk*. Then*

$$(rk, q)\text{-SGKH} \leq_{\text{A}} (r, k, q)\text{-MultSGKH}.$$

Proof. The reduction is straightforward. Let $I = (rk, d_{(1,1)}, \ldots, d_{(1,k)}, d_{(2,1)}, \ldots, d_{(r,k)})$ be an instance for the (rk, q)-SGKH. The first request rk gets mapped onto the first request (r, k) of the instance I_1, and the following requests $d_{(i,j)}$ get mapped on $d_{(i,j)}$ of (r, k, q)-MultSGKH for all $i \in \{1, \ldots, r\}$ and $j \in \{1, \ldots, k\}$. It follows

$$(rk, q)\text{-SGKH} \leq_{\text{A}} (r, k, q)\text{-MultSGKH}$$

with parameters $u = 1, v = 1, w = 0$. □

Now we are ready for the main theorem of this section.

Theorem 6.2 *For any* $k \in \mathbb{N}^{>0}$*, any constant* $c < k+1$*, and any* $q \in \mathbb{N}_{\geq 2}$*, any online algorithm with advice for* SetCover *that is c-competitive needs to read at least*

$$G(c, k, q) \cdot \frac{k \cdot \log_2 q}{q^k} \cdot m$$

or

$$G(c, k, q) \cdot \frac{k(q-1)\log_2 q}{q^{k+1} - 1} n$$

advice bits, where $n = |X|, m = |\mathcal{S}|$ *and*

$$G(c, k, q) = 1 + \left(\frac{c-1}{k}\right) \log_q\left(\frac{c-1}{k(q-1)}\right) + \left(\frac{k-c+1}{k}\right) \log_q\left(\frac{k-c+1}{k}\right).$$

Proof. In order to prove the claim, for every instance of (r, k, q)-MultSGKH, we give a corresponding instance of SetCover and a reduction from the multiple string guessing problem to this ((r, k, q)-MultSGKH $\leq_{\text{A}}$ SetCover). Together with the reduction from Lemma 6.3, this yields

$$(rk, q)\text{-SGKH} \leq_{\text{A}} \text{SetCover}.$$

The given instance is a set $\{s_1, \ldots, s_r\}$ of r strings of length $|s_i| = k$ for $i \in \{1, \ldots, r\}$ over an alphabet of size q. This corresponds to a set $\{I_{s_1}, \ldots, I_{s_r}\}$ of

(k,q)-SGKH instances and thus to an instance $I_{s_1} \circ \cdots \circ I_{s_r}$ for the (r,k,q)-MultSGKH. Analogous to the construction in the proof of Theorem 6.2, we construct r sub-instances of SetCover, where all sets of variables are pairwise disjoint and join the corresponding sets to one SetCover instance.

We now start with the formal construction of the instance. For each string s_i, we build an instance $(X_i, \mathcal{S}_i, I_i)$ for SetCover, such that the sets X_i are pairwise disjoint.

For the whole instance, we build the disjoint union of the subsets of the sub-instances. This gives

$$\begin{aligned} X &= \bigcup_{i=1}^{r} X_i, \\ \mathcal{S} &= \bigcup_{i=1}^{r} \mathcal{S}_i, \text{ and} \\ I &= I_1 \circ I_2 \circ \cdots \circ I_r. \end{aligned}$$

In this instance, there is an optimal solution of size r, because, for each of the sub-instances $(X_i, \mathcal{S}_i, I_i)$, there exist an optimal solution of size 1. All the sets are disjoint, so the optimal solution for the whole instance consists of all optimal solutions of the sub-instances. For the size of the set, we have $|X| = \frac{q^{k+1}-1}{q-1} \cdot r = n$ and $|\mathcal{S}| = q^k \cdot r = m$.

For an algorithm A that reads as many advice bits as are necessary to guess $\alpha \cdot rk$ bits correctly, it follows that

$$cost(\mathtt{A}) = r + (1-\alpha) \cdot rk.$$

The competitive ratio that A reaches is

$$c = \frac{r + (1-\alpha) \cdot rk}{r} = 1 + (1-\alpha) \cdot k$$

and thus,

$$\alpha = 1 - \frac{c-1}{k}.$$

Therefore, we can directly apply Theorem 5.4 for $\alpha = 1 - \frac{c-1}{k}$ yielding that at least

$$\begin{aligned}
&\left(1+\left(1-\left(1-\frac{c-1}{k}\right)\right)\log_q\left(\frac{1-\left(1-\frac{c-1}{k}\right)}{q-1}\right)\right.\\
&\left.\quad+\left(1-\frac{c-1}{k}\right)\log_q\left(1-\frac{c-1}{k}\right)\right)\cdot rk\log_2 q\\
=&\left(1+\left(\frac{c-1}{k}\right)\log_q\left(\frac{c-1}{k(q-1)}\right)\right.\\
&\left.\quad+\left(\frac{k-c+1}{k}\right)\log_q\left(\frac{k-c+1}{k}\right)\right)\cdot rk\log_2 q\\
=&G(c,k,q)\cdot rk\log_2 q
\end{aligned}$$

advice bits are necessary to reach a competitive ratio of c.

To measure the competitive ratio in $|\mathcal{S}| = m$ and in $|X| = n$, we calculate $r = \frac{k}{q^k}\cdot m$, and $r = \frac{k\cdot(q-1)}{q^{k+1}-1}\cdot n$, and, finally, this leads to

$$G(c,k,q)\cdot\frac{k\cdot\log_2 q}{q^k}\cdot m$$

or

$$G(c,k,q)\cdot\frac{k\cdot(q-1)\cdot\log_2 q}{q^{k+1}-1}\cdot n,$$

which concludes the proof. □

Note that, in [62], a lower bound on achieving a competitive ratio of c was shown that is merely logarithmic in m. Thus, for constant values of c (i. e., for $c \leq 2$), Theorem 6.2 exponentially improves over the best known result. Figure 6.3 and 6.4 illustrate the lower bound for $k = 1 (c \leq 2)$ and $k = 2 (c \leq 2)$.

6.5 Discussion

We used string guessing as a generic online problem and gave two reductions from it to the online maximum clique problem (MaxClique) as a maximization problem and to the the online set cover problem (SetCover) as a minimization problem, in order to transfer a lower bound on the advice complexity. For SetCover, we introduced a formal model of advice-preserving reduction that works for online minimization problems.

We gave a first formal reduction among online problems in order to prove lower bounds on the advice complexity. It remains as an open problem to extend the reduction to work also for for maximization problems. The string guessing problem might be a starting point for a sequence of reductions among online problems with respect to their advice complexity.

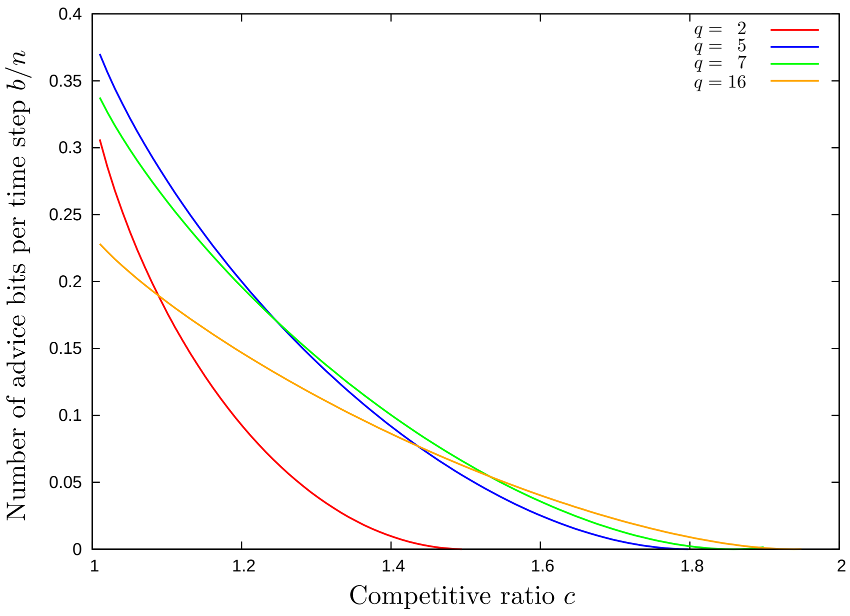


Figure 6.3: The number of advice bits per time step that is necessary for the online set cover problem for $k = 1$.

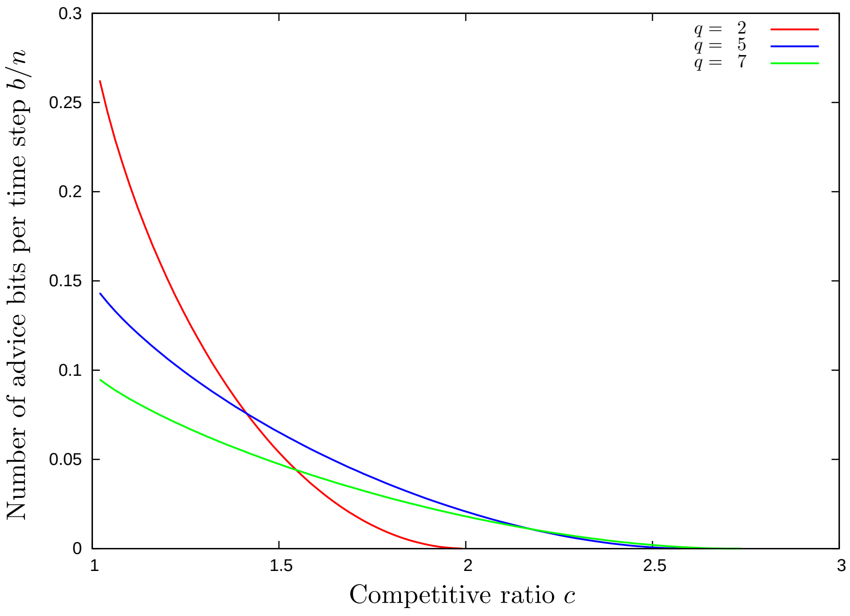


Figure 6.4: The number of advice bits per time step that is necessary for the online set cover problem for $k = 2$.

Here, e. g., a more detailed analysis of MaxClique should be possible. The extension to other online problems and a partition of online problems into advice complexity classes would also be very interesting.

Chapter 7

Conclusion

In this thesis, we investigated two theoretical approaches to integrate several kinds of extra knowledge into standard computational models. In the first approach, the extra information was given in the special form of one or more optimal solutions to a similar instance. In Chapter 2, we presented a reoptimization algorithm for the minimum Steiner tree problem on graphs with sharpened triangle inequality, with the local modification of adding a vertex as a non-terminal or as a terminal. Here, we were able to improve the known approximation ratio for the general problem and to reach an approximation ratio of $\frac{1}{2} + \beta$.

Even though the given information of an optimal solution to a similar instance seems to be a strong aid, we have seen in Chapter 3 that, for some problems, information about one or more solutions does not help at all. Moreover, if exponentially many solutions for a similar instance are given, the approximation ratio cannot be significantly improved. Furthermore, for the general TSP, all this strong extra information is not helpful in order to achieve a better local search algorithm.

In the second approach devoted to online algorithms, the information is given in a more generic way as a bit string on an extra input tape. Here, we gave first results for online coloring algorithms with advice for 3-colorable graphs. We presented nearly matching lower and upper bounds on the number of advice bits for the 3-coloring of planar chordal and general 3-colorable graphs. Additionally, we gave 4-coloring online algorithms with advice for those graph classes (see Table 4.1). One remarkable point is that the lower and upper bounds of 3-coloring for chordal graphs are equal to the bounds for 3-coloring of general 3-colorable graphs, even though the first problem lies in $\mathcal{P}$ and 3- coloring in general is known to be $\mathcal{NP}$-hard. The first difference in the advice complexity of the two problems of color chordal graphs and general 3-colorable graphs arises for online algorithms for finding a 4-coloring for these graphs. Unfortunately, no lower bounds on the advice complexity are known in this cases, leaving room for further research.

Furthermore, we used string guessing as a generic online problem and gave two reductions to the online maximum clique problem (MaxClique) and to the online set cover problem (SetCover), in order to transfer a lower bound on the advice complexity. A formal model of advice-preserving reductions that works for online minimization problems was given. Here, a generalization to maximization problems would be interesting. Furthermore, it would be very helpful to have a more precise analysis of the advice needed for string guessing to achieve a given quality. Here, it would be interesting to analyze the difference between the two cases with and without knowing the history.

With this, also some of the first research results for reductions among online problems in order to prove lower bounds on the advice complexity could be embedded into a formal framework. In the end, the string guessing problem might be a starting point for a sequence of reductions among online problems with respect to their advice complexity. The extension to other online problems and a partition of online problems into advice complexity classes would also be very interesting.

Bibliography

[1] N. Alon, B. Awerbuch, Y. Azar, N. Buchbinder, and J. Naor. The online set cover problem. *SIAM Journal on Computing*, 39(2):361–370, 2009.

[2] C. Archetti, L. Bertazzi, and M. G. Speranza. Reoptimizing the traveling salesman problem. *Networks*, 42(3):154–159, 2003.

[3] C. Archetti, L. Bertazzi, and M. G. Speranza. Reoptimizing the 0-1 knapsack problem. Technical Report 267, University of Brescia, 2006.

[4] G. Ausiello, V. Bonifaci, and B. Escoffier. Complexity and approximation in reoptimization. In A. S. B. Cooper, editor, *Computability in Context: Computation and Logic in the Real World*. Imperial College Press/World Scientific, 2011.

[5] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation*. Springer-Verlag, 1999.

[6] G. Ausiello, B. Escoffier, J. Monnot, and V. T. Paschos. Reoptimization of minimum and maximum traveling salesman's tours. In L. Arge and R. V. Freivalds, editors, *Proc. of the 10th Scandinavian Workshop on Algorithm Theory (SWAT 2006)*, volume 4059 of *Lecture Notes in Computer Science*, pages 196–207. Springer-Verlag, 2006.

[7] T. Berg and H. Hempel. Reoptimization of traveling salesperson problems: Changing single edge-weights. In A. H. Dediu, A.-M. Ionescu, and C. Martín-Vide, editors, *Proc. of the 3rd International Conference on Language and Automata Theory and Applications (LATA 2009)*, volume 5457 of *Lecture Notes in Computer Science*, pages 141–151. Springer-Verlag, 2009.

[8] M. W. Bern and P. E. Plassmann. The Steiner problem with edge lengths 1 and 2. *Information Processing Letters*, 32(4):171–176, 1989.

[9] M. P. Bianchi, H.-J. Böckenhauer, J. Hromkovič, and L. Keller. Online coloring of bipartite graphs with and without advice. In *Proc. of the 18th*

Annual International Conference on Computing and Combinatorics (COCOON 2012), volume 7434 of *Lecture Notes in Computer Science*, pages 519–530, 2012.

[10] D. Bilò, H.-J. Böckenhauer, J. Hromkovič, R. Královič, T. Mömke, P. Widmayer, and A. Zych. Reoptimization of Steiner trees. In J. Gudmundsson, editor, *Proc. of the 11th Scandinavian Workshop on Algorithm Theory (SWAT 2008)*, volume 5124 of *Lecture Notes in Computer Science*, pages 258–269. Springer-Verlag, 2008.

[11] D. Bilò, H.-J. Böckenhauer, D. Komm, R. Královič, T. Mömke, S. Seibert, and A. Zych. Reoptimization of the shortest common superstring problem. In G. Kucherov and E. Ukkonen, editors, *Proc. of the 20th Annual Symposium on Combinatorial Pattern Matching (CPM 2009)*, volume 5577 of *Lecture Notes in Computer Science*, pages 78–91. Springer-Verlag, 2009.

[12] D. Bilò, H.-J. Böckenhauer, D. Komm, R. Královič, T. Mömke, S. Seibert, and A. Zych. Reoptimization of the shortest common superstring problem. *Algorithmica*, 61(2):227–251, 2011.

[13] D. Bilò, P. Widmayer, and A. Zych. Reoptimization of weighted graph and covering problems. In E. Bampis and M. Skutella, editors, *Proc. of the 6th International Workshop on Approximation and Online Algorithms (WAOA 2008)*, volume 5426 of *Lecture Notes in Computer Science*, pages 201–213. Springer-Verlag, 2009.

[14] D. Bilò and A. Zych. New reoptimization techniques employed to Steiner tree problem. In *Proc. of the 6th Latin-American Algorithms, Graphs and Optimization Symposium (LAGOS 11)*, 2011.

[15] H.-J. Böckenhauer, L. Forlizzi, J. Hromkovič, J. Kneis, J. Kupke, G. Proietti, and P. Widmayer. Reusing optimal TSP solutions for locally modified input instances (extended abstract). In G. Navarro, L. E. Bertossi, and Y. Kohayakawa, editors, *Proc. of the 4th IFIP International Conference on Theoretical Computer Science (TCS 2006)*, volume 209 of *IFIP*, pages 251–270. Springer-Verlag, 2006.

[16] H.-J. Böckenhauer, L. Forlizzi, J. Hromkovič, J. Kneis, J. Kupke, G. Proietti, and P. Widmayer. On the approximability of TSP on local modifications of optimally solved instances. *Algorithmic Operations Research*, 2(2):83–93, 2007.

[17] H.-J. Böckenhauer, K. Freiermuth, J. Hromkovič, T. Mömke, A. Sprock, and B. Steffen. The Steiner tree problem with sharpened triangle inequality: hardness and reoptimization. In T. Calamoneri and J. Díaz, editors, *Proc. of the 7th International Conference on Algorithms and Complexity*

(CIAC 2010), volume 6078 of *Lecture Notes in Computer Science*, pages 180–191. Springer-Verlag, 2010.

[18] H.-J. Böckenhauer, K. Freiermuth, J. Hromkovič, T. Mömke, A. Sprock, and B. Steffen. Steiner tree reoptimization in graphs with sharpened triangle inequality. *Journal of Discrete Algorithms*, 11:73–86, 2012.

[19] H.-J. Böckenhauer, J. Hromkovič, R. Klasing, S. Seibert, and W. Unger. Approximation algorithms for TSP with sharpened triangle inequality. *Information Processing Letters*, 75:133–138, 2000.

[20] H.-J. Böckenhauer, J. Hromkovič, R. Klasing, S. Seibert, and W. Unger. An improved lower bound on the approximability of metric TSP and approximation algorithms for the TSP with sharpened triangle inequality. In H. Reichel and S. Tison, editors, *Proc. of the 17th Annual Symposium on Theoretical Aspects of Computer Science (STACS 2000)*, volume 1770 of *Lecture Notes in Computer Science*, pages 382–394. Springer-Verlag, 2000.

[21] H.-J. Böckenhauer, J. Hromkovič, D. Komm, S. Krug, J. Smula, and A. Sprock. The string guessing problem as a method to prove lower bounds on the advice complexity. Technical Report TR12-162, Electronic Colloquium on Computational Complexity (ECCC), 2012.

[22] H.-J. Böckenhauer, J. Hromkovič, D. Komm, S. Krug, J. Smula, and A. Sprock. The string guessing problem as a method to prove lower bounds on the advice complexity. In *Proc. of the 19th Annual International Conference on Computing and Combinatorics (COCOON 2013)*, 2013. to appear.

[23] H.-J. Böckenhauer, J. Hromkovič, R. Královič, T. Mömke, and P. Rossmanith. Reoptimization of Steiner trees: Changing the terminal set. *Theoretical Computer Science*, 410(36):3428–3435, 2009.

[24] H.-J. Böckenhauer, J. Hromkovič, T. Mömke, and P. Widmayer. On the hardness of reoptimization. In V. Geffert, J. Karhumäki, A. Bertoni, B. Preneel, P. Návrat, and M. Bieliková, editors, *Proc. of the 34th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2008)*, volume 4910 of *Lecture Notes in Computer Science*, pages 50–65. Springer-Verlag, 2008.

[25] H.-J. Böckenhauer, J. Hromkovič, and A. Sprock. Knowing all optimal solutions does not help for TSP reoptimization. In J. Kelemen and A. Kelemenová, editors, *Models of Computation, Cooperation and Life*, pages 7–15. Springer-Verlag, 2011.

[26] H.-J. Böckenhauer, J. Hromkovič, and A. Sprock. On the hardness of reoptimization with multiple given solutions. *Fundamenta Informaticae*, 110(1-4):59–76, 2011.

[27] H.-J. Böckenhauer and D. Komm. Reoptimization of the metric deadline TSP. In E. Ochmanski and J. Tyszkiewicz, editors, *Proc. of the 33th International Symposium on Mathematical Foundations of Computer Science (MFCS 2008)*, volume 5162 of *Lecture Notes in Computer Science*, pages 156–167. Springer-Verlag, 2008.

[28] H.-J. Böckenhauer, D. Komm, R. Královič, and R. Královič. On the advice complexity of the k-server problem. In L. Aceto, M. Henzinger, and J. Sgall, editors, *Proc. of the 38th International Colloquium on Automata, Languages and Programming (ICALP 2011)*, volume 6755 of *Lecture Notes in Computer Science*, pages 207–218. Springer-Verlag, 2011.

[29] H.-J. Böckenhauer, D. Komm, R. Královič, R. Královič, and T. Mömke. On the advice complexity of online problems. In Y. Dong, D.-Z. Du, and O. H. Ibarra, editors, *Proc. of the 20th International Symposium on Algorithms and Computation (ISAAC 2009)*, volume 5878 of *Lecture Notes in Computer Science*, pages 331–340. Springer-Verlag, 2009.

[30] H.-J. Böckenhauer, D. Komm, R. Královič, and P. Rossmanith. On the advice complexity of the knapsack problem. In D. Fernández-Baca, editor, *Proc. of the 10th Latin American Symposium on Theoretical Informatics (LATIN 2012)*, volume 7256 of *Lecture Notes in Computer Science*, pages 61–72. Springer-Verlag, 2012.

[31] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.

[32] A. Brandstädt, V. B. Le, and J. P. Spinrad. *Graph Classes: A Survey.* Society for Industrial and Applied Mathematics, 1999.

[33] J. Byrka, F. Grandoni, T. Rothvoß, and L. Sanità. An improved LP-based approximation for Steiner tree. In *Proc. of the 42st Annual ACM Symposium on Theory of Computing (STOC 2010)*, pages 583–592. ACM, 2010.

[34] A. Cayley. A theorem on trees. In *Quart. J. Math. 23*, pages 376–378, 1889.

[35] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, 1976.

[36] G. Cohnen, I. Honkala, S. Litsyn, and A. Lobstein. *Covering Codes.* Elsevier Science Publishers Ltd., 1997.

[37] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, 3rd edition, 2009.

[38] M. Demange, X. Paradon, and V. T. Paschos. On-line maximum-order induced hereditary subgraph problems. In V. Hlavác, K. G. Jeffery, and J. Wiedermann, editors, *Proc. of the 27th Annual Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2000)*, volume 1963 of *Lecture Notes in Computer Science*, pages 327–335. Springer-Verlag, 2000.

[39] S. Dobrev, R. Královič, and D. Pardubská. How much information about the future is needed? In V. Geffert, J. Karhumäki, A. Bertoni, B. Preneel, P. Návrat, and M. Bieliková, editors, *Proc. of the 34th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2008)*, volume 4910 of *Lecture Notes in Computer Science*, pages 247–258. Springer-Verlag, 2008.

[40] S. E. Dreyfus and R. A. Wagner. The Steiner problem in graphs. *Networks*, 1:195–207, 1971/72.

[41] Y. Emek, P. Fraigniaud, A. Korman, and A. Rosén. Online computation with advice. In S. Albers, A. Marchetti-Spaccamela, Y. Matias, S. E. Nikoletseas, and W. Thomas, editors, *Proc. of the 36th International Colloquium on Automata, Languages and Programming (ICALP 2009)*, volume 5555 of *Lecture Notes in Computer Science*, pages 427–438. Springer-Verlag, 2009.

[42] Y. Emek, P. Fraigniaud, A. Korman, and A. Rosén. Online computation with advice. *Theoretical Computer Science*, 412(24):2642–2656, 2011.

[43] B. Escoffier, M. Milanič, and V. T. Paschos. Simple and fast reoptimizations for the Steiner tree problem. *Algorithmic Operations Research*, 4(2):86–94, 2009.

[44] U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.

[45] A. Fiat and G. J. Woeginger, editors. *Online Algorithms, The State of the Art*, volume 1442 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[46] M. Forišek, L. Keller, and M. Steinová. Advice complexity of online coloring for paths. In *Proc. of the 6rd International Conference on Language and Automata Theory and Applications (LATA 2012)*, pages 228–239, 2012.

[47] K. Freiermuth and H. Stewénius. Multi-dimensional diamond graph constructions. Unpublished manuscript, 2009.

[48] V. Guruswami and S. Khanna. On the hardness of 4-coloring a 3-colorable graph. In *Computational Complexity, 2000. Proceedings. 15th Annual IEEE Conference on*, pages 188 –197, 2000.

[49] V. Guruswami, A. Rudra, and M. Sudan. Essential coding theory. http://www.cse.buffalo.edu/~atri/courses/coding-theory/book/.

[50] M. M. Halldórsson, S. Ueno, H. Nakao, and Y. Kajitani. Approximating Steiner trees in graphs with restricted weights. *Networks*, 31(4):283–292, 1998.

[51] J. Hromkovič. *Algorithmics for Hard Problems. Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics.* Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, 2003.

[52] J. Hromkovič. *Design and Analysis of Randomized Algorithms: Introduction to Design Paradigms (Texts in Theoretical Computer Science. An EATCS Series).* Springer-Verlag, 2005.

[53] J. Hromkovič, R. Královič, and R. Královič. Information complexity of online problems. In *Proc. of the 35th International Symposium on Mathematical Foundations of Computer Science (MFCS 2010)*, volume 6281 of *Lecture Notes in Computer Science*, pages 24–36. Springer-Verlag, 2010.

[54] F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner Tree Problems*, volume 53 of *Annals of Discrete Mathematics.* North-Holland, 1992.

[55] S. Irani and A. R. Karlin. On online computation. In D. S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, volume 521, chapter 13, pages 521–564. PWS Publishing Company, 1997.

[56] A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator. Competitive snoopy caching. *Algorithmica*, 3(1):79–119, 1988.

[57] S. Khanna, N. Linial, and S. Safra. On the hardness of approximating the chromatic number. In *Proc. of the 2nd Israel Symposium on Theory and Computing Systems (ISTCS 1993)*, pages 250 –260, 1993.

[58] H. Kierstead. Recursive and on-line graph coloring. In Y. L. Ershov, S. Goncharov, A. Nerode, J. Remmel, and V. Marek, editors, *Handbook of Recursive Mathematics Volume 2: Recursive Algebra, Analysis and Combinatorics*, volume 139 of *Studies in Logic and the Foundations of Mathematics*, pages 1233–1269. Elsevier Science Publishers Ltd., 1998.

[59] H. Kierstead and W. Trotter. On-line graph coloring. In L. A. McGeoch and D. D. Sleator, editors, *On-line Algorithms*, volume 7 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 85–92. AMS—DIMACS—ACM, 1992.

[60] D. Komm. *Advice and Randomization in Online Computation.* Doctoral dissertation, ETH Zurich, No. 20164, 2012.

[61] D. Komm and R. Královič. Advice complexity and barely random algorithms. In I. Černá, T. Gyimóthy, J. Hromkovič, K. G. Jeffery, R. Královič, M. Vukolic, and S. Wolf, editors, *Proc. of the 37th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2011)*, volume 6543 of *Lecture Notes in Computer Science*, pages 332–343. Springer-Verlag, 2011.

[62] D. Komm, R. Královič, and T. Mömke. On the advice complexity of the set cover problem. In E. A. Hirsch, J. Karhumäki, A. Lepistö, and M. Prilutskii, editors, *Proc. of the 7th Symposium on Computer Science in Russia (CSR 2012)*, volume 7353 of *Lecture Notes in Computer Science*, pages 241–252. Springer-Verlag, 2012.

[63] R. Laskar, H. Mulder, and B. Novick. Maximal outerplanar graphs as chordal graphs, path-neighborhood graphs, and triangle graphs. Econometric Institute Report EI 2011-16, Erasmus University Rotterdam, Econometric Institute, 2011.

[64] L. Lovász, M. E. Saks, and W. T. Trotter. An on-line graph coloring algorithm with sublinear performance ratio. *Discrete Mathematics*, 75(1–3):319–325, 1989.

[65] F. MacWilliams and N. Sloane. *The Theory of Error-Correcting Codes.* North-Holland, 2nd edition, 1978.

[66] R. A. Moser and D. Scheder. A full derandomization of Schöning's k-SAT algorithm. In *Proc. of the 43st Annual ACM Symposium on Theory of Computing (STOC 2011)*, pages 245–252. ACM, 2011.

[67] C. H. Papadimitriou and K. Steiglitz. Some examples of difficult traveling salesman problems. *Operations Research*, 26:434–443, 1978.

[68] C. H. Papadimitriou and M. Yannakakis. The traveling salesman problem with distances one and two. *Mathematics of Operations Research*, 18(1):1–11, 1993.

[69] H. J. Prömel and A. Steger. *The Steiner Tree Problem.* Advanced Lectures in Mathematics. Friedr. Vieweg & Sohn, 2002.

[70] M. Renault and A. Rosén. On online algorithms with advice for the k-server problem. In R. Solis-Oba and G. Persiano, editors, *Approximation and Online Algorithms*, volume 7164 of *Lecture Notes in Computer Science*, pages 198–210. Springer-Verlag, 2012.

[71] G. Robins and A. Z. Zelikovsky. Improved Steiner tree approximation in graphs. In *Proc. of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2000)*, pages 770–779. ACM/SIAM, 2000.

[72] S. Sahni and T. F. Gonzalez. P-complete approximation problems. *Journal of the ACM*, 23(3):555–565, 1976.

[73] M. W. Schäffter. Scheduling with forbidden sets. *Discrete Applied Mathematics*, 72(1-2):155–166, 1997.

[74] S. Seibert, A. Sprock, and W. Unger. Advice complexity of the online vertex coloring problem. Technical Report 765, ETH Zürich, 2012.

[75] S. Seibert, A. Sprock, and W. Unger. Advice complexity of the online coloring problem. In *Proc. of the 8th International Conference on Algorithms and Complexity (CIAC 2013)*, volume 7878 of *Lecture Notes in Computer Science*, pages 345–357. Springer-Verlag, 2013.

[76] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[77] S. van Hoesel and A. Wagelmans. On the complexity of postoptimality analysis of 0/1 programs. *Discrete Applied Mathematics*, 91(1-3):251–263, 1999.

[78] S. Vishwanathan. Randomized online graph coloring. *Journal of Algorithms*, 13(4):657–669, 1992.

[79] D. B. West. *Introduction to Graph Theory (2nd Edition)*. Prentice Hall, 2 edition, 2000.

[80] A. C.-C. Yao. New algorithms for bin packing. *Journal of the ACM*, 27(2):207–227, 1980.

[81] A. Zych. *Reoptimization of NP-Hard problems*. Doctoral dissertation, ETH Zurich, No. 20257, 2012.