Uniwersytet Wrocławski
Wydział Matematyki i Informatyki

Nowoczesne rozwiązania online dla pradawnych problemów grafowych

Autor:
Paweł Schmidt

Rozprawa sporządzona pod opieką
dr. hab. Marcina Bieńkowskiego

2021

University of Wrocław
Faculty of Mathematics and Computer Science

Modern online algorithms for ancient graph problems

Author:
Paweł Schmidt

Doctoral dissertation supervised by
dr. hab. Marcin Bieńkowski

2021

Contents

1 Introduction **1**

1.1 Deterministic vs. randomized algorithms 1

1.2 Why deterministic algorithms are important 3

1.3 Known determinization techniques 4

1.4 Overview of results and thesis outline 5

1.5 Related offline algorithms 9

1.6 Bibliographical notes 10

2 Non-metric facility location **11**

2.1 Introduction 11

2.2 Fractional solution 16

2.3 Deterministic rounding 23

2.4 Handling large aspect ratios 27

2.5 Remarks and applications 29

3 Matching with delays **33**

3.1 Introduction 33

3.2 Algorithm 36

3.3 Analysis 37

3.4 Lower bounds and tightness 45

4 Steiner tree leasing **47**

4.1 Introduction 47

4.2 HST embeddings 50
4.3 Interval model 52
4.4 Algorithm construction 53
4.5 Analysis 55

5 Generalized k-server problem in uniform metrics **61**

5.1 Introduction 61
5.2 Hydra game 63
5.3 Improved algorithm for generalized *k*-server problem 67
5.4 Lower bound 72
5.5 Final remarks 76

6 Afterword **79**

Chapter 1

Introduction

With the spread of broadband internet connections, gaming over the internet has become a primary way of playing competitive games (e.g., real-time strategies or collectible card games). Players typically log in to a service that provides the functionality of finding another person who is willing to compete. If there are many available players, deciding who gets matched first and who has to wait is important for maintaining people's satisfaction with the service. The satisfaction might depend not only on the time spent in a queue but also on a similarity of matched players' skills (losing a game to a much more skilled player or winning a game against a beginner is not fun at all). This is an example of a problem called *matching with delays*.

The process of matching players happens in real-time, that is, future (playing) requests are not known. Standard algorithmic approach, where an *offline* algorithm reads complete input data and outputs a solution is useless here. This motivates the construction of *online* algorithms, which have to satisfy all demands on the fly without prior knowledge of the future. The quality of an online algorithm is measured with *competitive ratio* [BE98], which is a worst-case ratio of the cost of an online algorithm to the cost of the optimal offline solution on the same input.

This dissertation is devoted to the construction of efficient online algorithms for classical graph problems: *facility location*, *Steiner tree* and *perfect matching*, which have been extensively studied in the offline regime and also *k-server*, which is specific to online computation. We put a special emphasis on deterministic algorithms and determinization techniques.

1.1 Deterministic vs. randomized algorithms

In the years after Sleator and Tarjan [ST85] initiated the study on online algorithms, the development of randomized algorithms followed the design of deterministic ones. Most of

this early research involved randomized algorithms that were randomizations of previous deterministic ones. The examples include, but are not limited to, list update [ST85, RWS94], paging [ST85, FKL+91], metrical task systems on uniform metrics [BLS92], TCP acknowledgement [DGS01, KKR03], and ski-rental [KMRS88, KMMO94].

This trend was somehow reversed with the seminal work of Bartal [Bar96], who designed a method of approximating an arbitrary graph (finite metric space) by a *random hierarchically well-separated tree* (HST) with moderate, polylogarithmic distortion. (An HST is a rooted tree in which edge lengths decrease exponentially along any root-to-leaf path.) This approximation was subsequently improved [Bar98, FRT04] and allowed for transforming a competitive algorithm (deterministic or randomized) for trees into a *randomized* algorithm for general graphs with the additional logarithmic factor in the competitive ratio. As a result, analyses of several randomized algorithms were simplified and new algorithms were created. Examples include randomized algorithms for metric labeling, group Steiner tree, buy-at-bulk network design [FRT04], metrical task systems [Bar96, FRT04], constrained page migration [Bar96], distributed paging [ABF98], minimum metric bipartite matching [MNP06], and matching with delays [AGGP17, AJF18].

Another stream of randomized algorithms started with the work of Buchbinder and Naor [BN09]. They provided a framework for designing and analyzing algorithms (mostly for covering or packing problems) using of online primal-dual technique. This framework incorporates the multiplicative weight update method and, in the typical setting, produces a *fractional* solutions worse by a logarithmic (in the size of the linear program) factor than the optimal solution. Usually, transforming a fractional solution into an algorithm requires further efforts as the linear program does not describe a distribution on configurations of the algorithm, but only its marginals. For example, the linear program for caching problem defines probabilities of pages being in the cache [BBN12]. To obtain an algorithm, these probabilities need to be translated into configurations of the cache. While there are multiple randomized algorithms using this framework, it is rarely clear how to deterministically construct an integral solution from a fractional one.

1.1.1 Achievable performance

In the area of online algorithms, it is common that for some problem there is a large gap between competitive ratios achievable by randomized algorithms and deterministic ones. For example, in caching problems [ST85, You02], the gap is exponential: the ratio of deterministic

algorithms is at least linear in the cache size, while the ratio of the best randomized algorithm is logarithmic [FKL+91, BBN12, ACER19].

On the other hand, in some cases the performance of deterministic algorithm asymptotically matches that of randomized solutions. Examples include set cover [AAA+09], metric facility location [Mey01, Fot08]), and also various problems where a constant-competitive solutions exist (e.g., TCP acknowledgement [DGS01, KKR03], or the list-update problem [ST85, RWS94]).

In general, the existence of a randomized algorithm does not give much insight into the existence of deterministic solutions. This is in strong contrast to approximation algorithms, where often randomized algorithms can be derandomized [WS11]. In this thesis, we try to shed some light on this topic.

1.2 Why deterministic algorithms are important

In the previous section, we highlighted that there are limits that deterministic algorithms cannot break. Why then do people keep studying deterministic algorithms? Besides the theoretical importance of the results, there are other, practical reasons.

First, the cost analysis of randomized algorithms ensures low cost only in expectation. This is a somewhat weak guarantee, which might "ignore" extremely unsuccessful executions of the algorithm. This is not acceptable in situations when exceptional safety measures are required, e.g., in banking.

This problem can be usually mitigated in approximation algorithms [WS11], where the probability of bad events can be commonly dampened by running the algorithm repeatedly and taking the best of all output solutions. The bad-event probability can be reduced exponentially with linear number of repetitions. This approach is, however, not applicable in an online regime, where algorithms have to make irrevocable decisions with every piece of input sequence they read.

Second, there is a recent need originating from the machine learning and automated trading systems to design *explainable* algorithms. An explainable algorithm can justify its decisions to humans (as opposed to black-box algorithms) [BH21]. There, the use of randomness can potentially make algorithm's decision unclear to observers.

1.3 Known determinization techniques

While the most of deterministic algorithm constructions are ad hoc, there are some known determinization techniques used in online algorithms. In this section, we briefly review a few of them.

1.3.1 Method of conditional expectations and pessimistic estimators

In the method of conditional expectations [Rag88], one designs a random experiment (algorithm) that solves the optimization problem and then *derandomizes* this algorithm by a sequential rounding of random variables. A variable is assigned the value that optimizes the conditional expected value of the remaining (not yet rounded) random choices. The resulting deterministic solution is as good as the randomized one. The crux is in computing the conditional expected values or finding their viable (pessimistic) estimate [You95]. The method of conditional expectations has proven useful in approximation algorithms. For instance, the approximation algorithms for max-cut and max-sat are obtained this way [WS11].

In principle, the above process can be applied to *online* algorithms. However, bounding the expected value of all future random choices of a randomized algorithm is troublesome or not possible at all. So far, the only successful implementation of this technique is the set cover problem [BN09]. We show a new application of this technique in Chapter 2 for the non-metric facility location problem.

1.3.2 HST approximation

While it is not clear how to obtain a deterministic algorithm from a randomized algorithm based on HST approximation, the *existence of approximation* itself can be helpful to analyze some deterministic algorithms [Umb15]. The key idea is that for some problems, the optimal solution on a tree is well-structured and easily computable. In such a situation, the cost of the optimal solution on a random tree is a good approximate of the optimal solution for the original graph. Given that, it remains to design an algorithm whose cost can be compared to the cost of the optimal solution on *any* random tree approximating the graph. As the graph approximation does not "glue" far vertices, some clustering algorithms can be analyzed this way.

The technique was successfully applied to simplify the analysis of the greedy algorithm solving Steiner tree [IW91] and to design and analyze new algorithms for Steiner trees, forests

and networks, connected facility location, and rent-or-buy problems [Umb15]. We use this technique to obtain a deterministic algorithm for the Steiner tree leasing problem in Chapter 4.

1.3.3 Adaptive adversaries

The standard definition of the competitive ratio for randomized algorithms assumes a so-called *oblivious* adversary, who presents an input sequence, which does not depend on the random actions of the algorithm (it may depend on its probability distribution, though). An overwhelming majority of algorithms uses this definition.

This notion can be strengthened to take into account the random choices of the algorithm. The stronger *adaptive online* adversary [BBK+94] may access past (random) decisions of the randomized algorithm and on this basis construct the input sequence. The adaptive online adversary has to simultaneously construct its own solution to which the algorithm is compared.

Adaptive online adversary and determinization are connected in a surprising way. Namely, proving that a randomized algorithm is c-competitive against the adaptive online adversary, implies the existence of c^2-competitive *deterministic* algorithm [BBK+94]. This property has been used in the past to obtain, among others, better deterministic online algorithms for the page migration problem [Wes94] and the weighted k-server problem [CV13].

1.4 Overview of results and thesis outline

Our primary goal is to give algorithms that *minimize the competitive ratio* defined as follows.

Definition 1.1. *A deterministic algorithm* Alg *is c-competitive if there exists a constant β (independent of the online part of the input) such that, for any input $\mathcal{I}$, it holds that*

$$\textsc{Alg}(\mathcal{I}) \leq c \cdot \textsc{Opt}(\mathcal{I}) + \beta,$$

where $\textsc{Alg}(\mathcal{I})$ is the cost of the solution ouput by Alg *and $\textsc{Opt}(\mathcal{I})$ is the cost of the optimal offline solution to $\mathcal{I}$. When β is equal to 0, we say that* Alg *is strictly c-competitive.*

For randomized algorithms, the cost of the algorithm is replaced by the expected cost (the expectation is taken over random choices of the randomized algorithm).

The running time of an online algorithm is of secondary importance; the only limit imposed on the algorithm is uncertainty about future requests. This is a typical assumption in this area. Some extreme examples include recent breakthrough results on k-server problem [BCL+18,

Lee18], where the authors answer purely information-theoretic question: "Is there enough information in the input sequence to be competitive?". They show the existence of a competitive solution (in their case, it is a curve in a high-dimensional space), which is not necessarily computable in the typical algorithmic sense. Having said that, most of our algorithms run in polynomial time.

Non-metric facility location. In the facility location problem [Shmoo], which have been extensively studied both in operations research and in computer science, there is a set of facilities F (each with a certain opening cost) and clients C. There are also client-facility connections, which have lengths that can either satisfy the triangle inequality (*metric* facility location) or be arbitrary (*non-metric* facility location). The goal is to open a subset of facilities and connect each client to an open facility. The total cost (the sum of opening and connection costs) is subject to minimization.

In the online scenario, we distinguish between potential clients (the above-mentioned set C) and clients $A \subseteq C$ that eventually appear in the input sequence. The best online solution to non-metric facility location is a *randomized* algorithm by Alon et al. [AAA+06] attaining competitive ratio of $O(\log|F| \cdot \log|A|)$. It computes a fractional solution of an LP-relaxation of the problem and then randomly rounds it to achieve a competitive integral solution. To obtain a *deterministic* algorithm, one could reduce the non-metric facility location instance to an instance of the set cover problem and then apply the deterministic algorithm by Alon et al. [AAA+09]. The standard text-book reduction between these two problems requires an exponential blow-up of the input size. However, reduction of Kolen and Tamir [KT90] does not have this disadvantage and leads to an $O((\log|C| + \log|F|) \cdot (\log|C| + \log\log|F|))$-competitive algorithm.

The main result of Chapter 2 is a construction of a historically first published deterministic algorithm for the online non-metric facility location problem. This algorithm attains the competitive ratio of $O(\log|F| \cdot (\log|C| + \log\log|F|))$ and is optimal up to $\log\log$ factors.

Our algorithm relies on online primal-dual framework [BN09] and utilizes a new LP relaxation of non-metric facility location. To service a client, it computes a fractional solution, which is then rounded deterministically. The main difficulty of the rounding lies in maintaining dependencies of purchased facilities and edges: rounding facilities and client-facility connections separately might lead to purchasing an edge to a closed facility. In our approach, the required dependency is not enforced by the LP constraints but follows from the order in which the algorithm increases LP variables.

Matching with delays. Chapter 3 discusses the aforementioned matching with delays (for which we gave the gaming example). The first online algorithm solving the problem of matching with delays was given by Emek et al. [EKW16], who presented a randomized $O(\log^2 n + \log \Delta)$-competitive algorithm. There, n is the number of points in the underlying metric space (a player is characterized by a number of different parameters and corresponds to a point of a metric space) and Δ is its aspect ratio (the ratio between the largest and the smallest distance). The competitive ratio was subsequently improved by Azar et al. [ACK17] to $O(\log n)$.

In Chapter 3, we present the historically first deterministic algorithm for an arbitrary metric space (previous deterministic algorithms were known only for simple spaces [ESW17, ACK17]), whose competitive ratio is $O(m^{\log_2 5.5}) = O(m^{2.46})$, where $2m$ is the number of requests. Our online algorithm uses a simple, local, semi-greedy scheme to find a suitable matching pair. That is, each client independently attempts to find (in a range that increases in time) someone who has waited roughly the same amount of time.

Steiner tree leasing. Many online problems have the common property that the objects purchased by the algorithm are persistent. For example, open facilities in facility location problem and matching edges in matching with delay problem do not disappear with time. In Chapter 4, we focus on a problem that does not have this property. In the Steiner tree leasing problem edges are not bought, but leased for a certain period and disappear when the lease period ends.

In this problem, there is a given graph with a distinguished root vertex. An algorithm has to connect all requested vertices to the root with leased edges. That is, there must exist a path of leased edges between a vertex and the root at the time when this vertex appears in the input sequence.

A randomized $O(\log L \cdot \log n)$-competitive algorithm was given by Meyerson [Mey05] for inputs with L different lease types and graphs with n vertices. This algorithm first approximates a graph by a random tree with distortion of $O(\log n)$ [FRT04] and then, for each edge of the tree runs the randomized $O(\log L)$-competitive algorithm for the parking permit problem.

We construct a deterministic algorithm, which can be seen as a derandomization of Meyerson's algorithm. The algorithm leases a new edge if there were "sufficiently many" requests served "recently" in a "small" neighborhood of this edge. In the analysis, we use techniques presented by Umboh [Umb15]: We exploit the fact that vertices that are far from each other in the graph, are far also in any random tree approximating this graph. This fact allows us

to charge the cost of a deterministic algorithm to the cost of optimal solution on a random tree. The resulting competitive ratio is $O(L \cdot \log k)$ for inputs with L different lease types and k different requested vertices. Our algorithm is competitive even on infinite metric spaces, which is not the case of randomized Meyerson's algorithm [Mey05].

Generalized k-server problem. In all mentioned problems, the algorithm purchases or rents structures in graphs. The last problem discussed in the dissertation is different.

The k-server problem introduced by Manasse et al. [MMS90], is one of the most well-studied and influential cornerstones of online analysis. The problem definition is deceivingly simple: There are k servers, starting at a fixed set of k points of a metric space M. An input is a sequence of requests (points of M) and to service a request, an algorithm needs to move at least one server to the requested position.

In Chapter 5, we study a natural extension of the k-server problem, called the *generalized k-server problem* [KT04, SS06], where each server s_i remains in its own metric space M_i. The request is a k-tuple $(r_1, \ldots, r_k)$, where $r_i \in M_i$, and to service it, an algorithm needs to move servers, so that *at least one* server s_i ends at the requested position r_i. The original k-server problem corresponds to the case where all metric spaces M_i are identical and each request is of the form $(r, \ldots, r)$.

Algorithms attaining competitive ratios that are functions of k exist only in a few special cases of the generalized k-server problem. The case of $k = 2$ has been solved by Sitters and Stougie [SS06, Sit14], who gave constant competitive algorithms for this setting. Results for $k \geq 3$ are known only for simpler metric spaces.

A *uniform metric* case describes a scenario where all metrics M_i are uniform with pairwise distances between different points equal to 1. For this case, Bansal et al. [BEKN18] recently presented an $O(2^k \cdot k)$-competitive deterministic algorithm and an $O(k^3 \cdot \log k)$-competitive randomized one. The deterministic competitive ratio is at least $2^k - 1$ already when metrics M_i have two points [KT04]. Furthermore, using a straightforward reduction to the metrical task system (MTS) problem [BLS92], Bansal et al. show that the randomized competitive ratio is at least $\Omega(k/\log k)$ [BEKN18].

In Chapter 5, we give a randomized $O(k^2 \cdot \log k)$-competitive algorithm for the uniform metric case of the generalized k-server problem improving over the $O(k^3 \cdot \log k)$ bound by Bansal et al. [BEKN18]. Furthermore, we strengthen their lower bound showing that no randomized algorithm can be better than $\Omega(k)$-competitive even when each metric space M_i consist of at least two points.

Our algorithm is phase-based: a single phase lasts until it is certain that requests of this phase cannot be serviced without cost. To track the phases and decide next moves, the algorithm computes a solution to an abstract *Hydra game*, which can be of independent interest.

1.5 Related offline algorithms

Offline counterparts of problems discussed in this dissertation have been studied before. We mention these results for completeness.

The metric variant of the facility location was studied in a series of papers [BA10, CG05, CS03, JMM+03, JMS02, KPR00, Li13, MYZ06, STA97]. Currently, the best known algorithm attains the approximation ratio of 1.488 [BA10, Li13]. The best approximation ratio for the non-metric one is, however, $O(\log |C|)$ [Hoc82]. Lower bounds of 1.463 for metric [GK98] and $\Omega(\log |C|)$ for non-metric variant [Fei98] hold unless $\mathsf{NP} \subseteq \mathsf{DTIME}[n^{O(\log \log n)}]$.

The (non-leasing) Steiner tree problem [WH16] is APX-hard [BP89]. Starting from a trivial 2-approximation, there was a series of improvements [Zel93, KZ97, PS00, RZ05, BGRS10]. The most recent of these results, by Byrka et al. [BGRS10], with approximation ratio of 1.39, has an interesting property: it uses a natural LP relaxation of the problem and the iterated randomized rounding, which yields a solution better than the known integrality gap of this linear program, which is 1.55. The Steiner tree leasing has been studied by Anthony and Gupta. They show a surprising connection to multistage stochastic optimization [AG07], which yields an $O(\min(L, \log n))$-approximation algorithm for inputs with n vertices and L lease types.

The offline variant of the generalized k-server problem problem has not drawn much attention yet. The closest problem studied in an offline regime is k-server, which admits a polynomial time algorithm [CKPV91]. This algorithm reduces finding of an optimal k-server strategy into a flow computation in a tailored graph.

Finally, the offline problem of matching with delays is easily solvable in polynomial time, by an algorithm that finds a min-cost matching [Sch03] in a time-augmented metric space (that is, the distance between two points becomes their distance in the metric space plus the difference in arrival times).

1.6 Bibliographical notes

The results presented in the thesis were previously published in preliminary form in various conference proceedings.

The material in Chapter 2 is an extended form of results that are accepted for publication at the *38th International Symposium on Theoretical Aspects of Computer Science* (STACS2021) [BFS21]. The algorithm for matching with delays from Chapter 3 appeared in the *15th Workshop on Approximation and Online Algorithm* (WAOA2017) [BKS17b]. Chapter 4 on the Steiner tree leasing problem is based on a paper, which appeared in the *15th Algorithms and Data Structures Symposium* (WADS2017) [BKS17a]. Finally, results on generalized k-server problem from Chapter 5 were published in proceedings of the *30th International Symposium on Algorithms and Computation* (ISAAC2019) [BJS19].

Chapter 2

Non-metric facility location

2.1 Introduction

The facility location problem [ABM16] is one of the best-known examples of network design problems, extensively studied both in operations research and in computer science. Its simple definition, NP-hardness, and rich combinatorial structure have led to developments of tools and solutions in key areas of approximation algorithms, combinatorial optimization, and linear programming.

An instance of the facility location problem consists of a set F of facilities, each with a certain opening cost, and a set C of clients. F and C can be seen as two sides of a bipartite graph. The undirected edges between them have lengths that can either satisfy the triangle inequality (*metric* facility location) or be arbitrary (*non-metric* facility location). The goal is to open a subset of facilities and connect each client to an open facility. The total cost (the sum of opening and connection costs) is subject to minimization. In the metric scenario, by taking a metric closure, one can assume that each facility is reachable by each client, but it is not the case for the non-metric variant.

Instances and objectives. In this chapter, we focus on an online variant of the non-metric facility location problem. We first formalize the offline variant in a way that makes a connection to the online variant more apparent.

A *facility-client graph* $G = (F, C, E, \mathsf{cost})$ is a bipartite graph, whose one side is the set F of facilities and another side is the set of clients C. Set $E \subseteq F \times C$ contains available facility-client connections (edges). We use function cost to denote both costs of opening facilities and connection costs (edge lengths). All costs are non-negative.

An instance of the non-metric facility location problem is a pair $(G; A)$, where $G = (F, C, E, \mathsf{cost})$ is a facility-client graph and $A \subseteq C$ is a subset of *active* clients. A feasible solution to such instance is a set of open (purchased) facilities $F' \subseteq F$ and a subset of purchased edges $E' \subseteq E$, such that any active client $c \in A$ is connected by a purchased edge to an open facility. The cost of such solution is equal to $\sum_{f \in F'} \mathsf{cost}(f) + \sum_{e \in E'} \mathsf{cost}(e)$.

For any facility-client graph G, we define its aspect ratio Δ_G as the ratio of the largest to smallest positive cost in G. These costs include both facilities and connection costs. (In the standard definition of the aspect ratio, only distances are taken into account.) Note that the aspect ratio is a property of G and is independent of the set of active clients A.

Online scenario. In an *online variant* of the facility location problem, the facility-client graph G is known in advance, but neither elements of A nor its cardinality are known up-front by an online algorithm ALG. The clients from A appear one by one. Upon seeing a new active client, ALG may purchase additional facilities and edges, with the requirement that facilities and edges purchased so far must constitute a feasible solution to all presented active clients. The total cost of ALG is denoted by $\text{ALG}(G; A)$. (We sometimes use $\text{ALG}(G; A)$ to also denote the solution computed by ALG.) Purchase decisions are final and cannot be revoked later. The goal is to service all requests and minimize the total cost.

2.1.1 Previous work on non-metric facility location

For the online metric facility location, the problem was resolved over ten years ago by Meyerson [Meyo1] and Fotakis [Foto8]: the lower and upper bounds on the competitive ratio are $\Theta(\log |A| / \log \log |A|)$, both for deterministic and randomized algorithms. Simpler deterministic algorithms attaining slightly worse competitive ratio of $O(\log |A|)$ were given by Anagnostopoulos et al. [ABUHo4] and Fotakis [Foto7]. Note that the optimal competitive ratio in the metric case is independent of the set C of potential clients.

2.1.2 Previous work on online non-metric facility location

For the non-metric facility location, the first and currently best online algorithm was a *randomized* algorithm by Alon et al. [AAA+o6]. It achieves the competitive ratio of $O(\log |F| \cdot \log |A|)$. It is based on solving a natural fractional relaxation of the problem: there is a fractional *opening variable* y_f for each facility f, and a *connection variable* $x_{c,f}$ for a client c and a *covering* facility f (facility to which c could be connected). Once a client c arrives, for each covering

facility f independently, their algorithm increases either y_f or $x_{c,f}$, whichever is smaller, using multiplicative update method (see, e.g., [AHK12]). The client c is considered fractionally served once the sum of terms $\min\{x_{c,f}, y_f\}$ over all covering facilities is at least 1. The resulting competitive ratio is $O(\log |F|)$.

The computed fractional solution can be then rounded using a random threshold θ_f common for an opening variable y_f and all connection variables involving facility f. Once any variable exceeds its threshold, it is rounded up to 1 and the corresponding object (facility or connection) is purchased. Dynamically adjusting θ_f to have expectation $\Theta(1/\log |A|)$ guarantees that the resulting integral solution is feasible with high probability and the rounding part incurs a factor of $O(\log |A|)$ in the competitive ratio.

Before our result, first given in [BFS21], no non-trivial deterministic algorithm was published. In particular, the online network design problems (including the non-metric facility location problem) have been listed as unresolved challenges by Buchbinder and Naor [BN09, Section 1.1]. That said, the non-metric facility location can be reduced to a set cover. A usable reduction (not inducing an exponential blow-up of the input size) was given by Kolen and Tamir [KT90]: it preserves the solution costs up to constant factors and creates a set cover instance consisting of $m = \Theta(|F| + |C| \cdot \log \Delta_G)$ sets and $n = \Theta(|C| \cdot \log \Delta_G)$ elements. We present this reduction in details in Section 2.5.1. Using doubling techniques described in Section 2.4, one could assume that $\Delta_G = O(|F| \cdot |C|)$. Applying the deterministic algorithm for the online set cover problem by Alon et al. [AAA+09] yields a solution whose competitive ratio is $O(\log m \cdot \log n) = O((\log |C| + \log |F|) \cdot (\log |C| + \log \log |F|))$.

2.1.3 Our result

In this chapter, we improve the bound above, replacing the first factor of $O(\log |C| + \log |F|)$ by $O(\log |F|)$. This is an asymptotic improvement in a typical scenario where $|F| \ll |C|$.

Theorem 2.1. *There exists a deterministic polynomial-time $O(\log |F| \cdot (\log |C| + \log \log |F|))$-competitive algorithm for the online non-metric facility location problem on set F of facilities and set C of clients.*

Our algorithm attains a nearly optimal competitive ratio, as no deterministic algorithm can have a ratio smaller than $\Omega(\log |F| \cdot \log |C| / (\log \log |F| + \log \log |C|))$. This follows by the lower bound for the online set cover problem [AAA+06, AAA+09] and holds even for uniform facility costs. If we restrict our attention to the *polynomial-time* deterministic solution, then a stricter lower bound of $\Omega(\log |F| \cdot \log |C|)$ holds (assuming $\mathsf{BPP} \neq \mathsf{NP}$) [Koro4].

Challenges. The description of the randomized algorithm by Alon et al. [AAA+06] given above seems deceptively simple, but it hides an important and subtle property, implicitly exploited by the authors. Namely, the threshold θ_f is common for facility f and all connections to it. This ensures the necessary dependency: once $\min\{x_{c,f}, y_f\} \geq \theta_f$, the rounding purchases *both facility f and a connection from c to f*. (Note that the left-hand side of this inequality is the amount that their fractional solution controls.)

It is unclear how to directly extend this property to deterministic rounding. A straightforward attempt would be to focus on facilities only and round them in a deterministic fashion ensuring the necessary coverage of each client. However, neglecting the connection costs in the rounding process easily leads to a situation, where the facilities are rounded "correctly", but the cost of connecting a client to the closest open facility in the integral solution is incomparably larger than the corresponding fractional cost.

We note that all known deterministic schemes that round fractional solutions generated by the multiplicative updates operate in rather limited scenarios, where elements have to be covered or packed and all important interactions between elements are handled at the time of constructing the fractional solution. This is the case for the deterministic rounding for the set cover problem [AAA+09, BN09] and the throughput-competitive virtual circuit routing problem [AAP93, BN09]. These methods are based on derandomization of the method of pessimistic estimators [Rag88] in an online manner, by transforming a pessimistic estimator into a potential function [You95] that can be controlled by the deterministic rounding process.

Our techniques. In our solution, we create a new linear relaxation of the problem. We first round the graph distances to powers of 2. For any client, we cluster facilities that have the same distance to this client. (Note that such clusters are client-dependent.) To solve the fractional variant, we run two schemes in parallel: we increase connection variables $x_{c,t}$ corresponding to clusters at distance t, and increase facility variables y_f for all facilities in "reachable" clusters (where the corresponding connection variables are 1). The increases in these variables use two different frameworks: dual fitting for linear increases of connection variables and a primal-dual scheme involving multiplicative updates for facility variables. Ensuring an appropriate balance between these two different types of updates is one of the technical difficulties that we tackle in this chapter.

We stop increasing variables once there exists a collection of clusters that are both "fractionally open" (sum of variables y_f within these clusters is $\Omega(1)$) and "reachable" by the

considered client. To argue about the existence of such a collection, we use both LP inequalities and structural properties of our fractional algorithm.

Finally, we construct a deterministic rounding routine. We focus on facilities only, neglecting whether particular clients are active or not and how far they are from a given facility. However, we strengthen rounding properties, ensuring, for (some) collections of clusters, that if the sum of opening variables in these collections is $\Omega(1)$, then the integral solution contains an open facility in one of these clusters. This ensures that, for a considered client c, the integral solution contains a facility whose distance from c is asymptotically not larger than the cost invested for connecting c in the fractional solution. Ultimately, this yields the desired dependency between facilities and connections.

Note about up-front knowledge of the facility-client graph. Unlike for the randomized variant, obtaining sub-linear guarantees for a deterministic solution requires knowing a priori the set of potential client-facility connections. To see this, consider a graph of $|F|$ facilities with unit opening costs and the set of $|C| = |F|$ clients. The graph edges are constructed dynamically as clients are activated and all revealed possible connections are of cost 0. The first active client can be connected to all facilities. Each subsequent client can be connected to all facilities but the ones already open by an algorithm. This way an online algorithm needs to eventually open all facilities, for a total cost of $|F|$. On the other hand, the offline optimal algorithm can open the last facility opened by an online algorithm and connect all clients to this facility paying just 1. Thus, under the unknown-graph assumption, the competitive ratio of any deterministic algorithm would be at least $|F|$.

2.1.4 Preliminaries and chapter organization

Let T_G contain all powers of two between the largest and the smallest positive distance (inclusively) and also number 0. In particular, T_G contains all distances in G and $|T_G| \leq 2 + \log \Delta_G$. Whenever G is clear from the context, we drop the G subscript.

We may assume that F contains at least two facilities and C contains at least two clients, as otherwise the problem becomes trivial. For a facility $f \in F$, let $\mathsf{set}(f)$ be the set of clients that may be connected to f. For any client $c \in C$ and distance $t \in T$, cluster $F_{c,t}$ contains all facilities that are incident to c using edges of cost t. Note that for a fixed c, clusters $F_{c,t}$ are disjoint (no client has two connections of different costs to the same facility).

Powers-of-two assumption. In the whole chapter, we assume that all facilities and connection costs are either equal to 0 or are powers of 2 and are at least 1. This can be easily achieved by initial scaling of positive costs and distances, so that they are at least 1 and rounding positive ones up to the nearest power of two. This transformation changes the competitive ratio at most by a factor of 2.

Chapter overview. Our core approach is to solve a carefully crafted fractional relaxation of the problem (Section 2.2), and then round it in a deterministic fashion (Section 2.3). This way, we obtain a deterministic online algorithm INT that on any input $(G = (F, C, E, \mathsf{cost}); A)$ computes a feasible solution of cost

$$\textsc{Int}(G; A) \leq O(\log |F| \cdot (\log |C| + \log\log \Delta_G)) \cdot \textsc{Opt}(G; A) + 2 \cdot \max_{f \in F} \mathsf{cost}(f).$$

Moreover INT runs in time $\mathrm{poly}(|G|, |A|, \max_{e \in E} \mathsf{cost}(e), \max_{f \in F} \mathsf{cost}(f))$. In Section 2.4, we apply doubling and edge pruning techniques, to get rid of dependencies on costs in the running time and on Δ_G in the competitive ratio, achieving guarantees of Theorem 2.1.

2.2 Fractional solution

We fix an instance $(G = (F, C, E, \mathsf{cost}); A)$ of the online non-metric facility problem. For each facility f, we introduce an *opening* variable $y_f \geq 0$ (fractional opening of f) and for each client c and each distance $t \in T$ a *connection* variable $x_{c,t} \geq 0$. Intuitively, $x_{c,t}$ denotes how much, fractionally, client c invests into connections to facilities from cluster $F_{c,t}$. For any set F' of facilities we use $y(F')$ as a shorthand for $\sum_{f \in F'} y_f$.

Primal Program. After k clients from A arrive (we denote their set by A_k), we consider the following linear program $\mathcal{P}_k$.

$$\begin{aligned}
\text{minimize} \quad & \sum_{f \in F} \mathsf{cost}(f) \cdot y_f + \sum_{c \in A_k} \sum_{t \in T} t \cdot x_{c,t} & \\
\text{subject to} \quad & x_{c,t} \geq z_{c,t} & \text{for all } c \in A_k,\ t \in T, \\
& y(F_{c,t}) \geq z_{c,t} & \text{for all } c \in A_k,\ t \in T, \\
& \sum_{t \in T} z_{c,t} \geq 1 & \text{for all } c \in A_k,
\end{aligned}$$

and non-negativity of all variables.

Serving Constraints. The LP constraints combined are equivalent to the set of the following (non-linear) requirements

$$\sum_{t \in T} \min\{x_{c,t},\ y(F_{c,t})\} \geq 1 \qquad \text{for all } c \in A_k. \tag{2.1}$$

We call (2.1) for client c the *serving constraint* for client c. In our description, we omit variables $z_{c,t}$ and the original constraints, ensuring only that the serving constraints hold and implicitly setting $z_{c,t} = \min\{x_{c,t},\ y(F_{c,t})\}$.

The LP above is indeed a valid relaxation of the facility location problem. To see this, take any feasible integral solution. For any facility f opened in the integral solution, set variable y_f to 1. For each client c connected to facility f, set variable $x_{c,\tau}$ to 1, where $\tau = \mathsf{cost}(f, c)$. This guarantees that $\min\{x_{c,\tau},\ y(F_{c,\tau})\} = 1$, and thus the serving constraint (2.1) is satisfied for each client c.

Dual Program. The program $\mathcal{D}_k$ dual to $\mathcal{P}_k$ is

$$\begin{aligned}
\text{maximize} \quad & \sum_{c \in A_k} \gamma_c & & \\
\text{subject to} \quad & \gamma_c \leq \alpha_{c,t} + \beta_{c,t} & & \text{for all } c \in A_k,\ t \in T, \\
& \alpha_{c,t} \leq t & & \text{for all } c \in A_k,\ t \in T, \\
& \sum_{c \in \mathsf{set}(f) \cap A_k} \beta_{c,\mathsf{cost}(f,c)} \leq \mathsf{cost}(f) & & \text{for all } f \in F,
\end{aligned}$$

and non-negativity of all variables.

2.2.1 Overview

Our algorithm Frac creates a solution to $\mathcal{P}_k$, ensuring that the serving constraint (2.1) holds for all clients $c \in A_k$. As outlined in the introduction, the computed solution guarantees some additional properties that are useful for the rounding part later.

Whenever a client c arrives, Frac increases connection variables $x_{c,t}$ one by one starting from the smallest t, at the pace proportional to $1/t$. We ensure that $x_{c,t} \in [0, 1]$, i.e., once any of these variables reaches 1, Frac stops increasing them. A distance t, for which $x_{c,t} = 1$, is called *saturated*.

In parallel to manipulating variables $x_{c,t}$, Frac increases all variables y_f for facilities reachable from client c using saturated distances. The variables y_f are increased using the multiplicative update rule [AHK12] (scaled appropriately to take costs of facilities into account).

Together with the solution to $\mathcal{P}_k$, FRAC also constructs an *almost-feasible* solution to $\mathcal{D}_k$. That is, its solution to $\mathcal{D}_k$ is feasible when all dual variables are scaled down by a factor of $O(\log |F|)$. By the weak duality, the scaled-down value of this solution serves as a lower-bound for the optimum. Thus, as typical for the primal-dual type of analysis, the dual variables can be thought of as budgets whose increase balances the increase of primal variables.

2.2.2 Algorithm FRAC

At the very beginning, before any client arrives, FRAC sets all variables y_f to 0 for all positive-cost facilities and to 1 for zero-cost ones. There are no other variables as the set A_0 of active clients is empty. Note that the dual program already contains the last type of constraints, but the sums on their left-hand sides range over empty sets of β variables, and hence these constraints are trivially satisfied.

Whenever a new client c arrives in step k, FRAC updates the primal (dual) programs from $\mathcal{P}_{k-1}$ ($\mathcal{D}_{k-1}$) to $\mathcal{P}_k$ ($\mathcal{D}_k$), and then computes a feasible solution to $\mathcal{P}_k$ (based on the already created solution to $\mathcal{P}_{k-1}$) and a nearly-feasible solution to $\mathcal{D}_k$.

New variables in primal and dual programs: FRAC sets $x_{c,t} \leftarrow 0$ for all $t \in T \setminus \{0\}$ and sets $x_{c,0} \leftarrow 1$. In the dual solution, it sets $\gamma_c \leftarrow 0$, $\alpha_{c,t} \leftarrow 0$ and $\beta_{c,t} \leftarrow 0$ for all $t \in T$.

Update primal program: A new serving constraint $\sum_{t \in T} \min\{x_{c,t}, y(F_{c,t})\} \geq 1$ appears in the primal program (and is violated unless $y(F_{c,0}) \geq 1$). As we never decrease primal variables, the serving constraints (2.1) that existed already in $\mathcal{P}_{k-1}$ are satisfied and will not become violated.

Update dual program: New constraints appear in the dual program and new variables $\beta_{c,t}$ appear on the left-hand side of the already existing inequalities. Since the new variables are initialized to 0, the validity of all dual constraints is unaffected.

Update primal and dual solutions: Let $T_c^1 = \{t \in T : x_{c,t} \geq 1\}$ be the set of saturated distances, i.e., initially FRAC sets $T_c^1 \leftarrow \{0\}$. While the serving constraint for c is violated, FRAC executes the *update operation* consisting of the following steps:

1. Set $\gamma_c \leftarrow \gamma_c + 1$.
2. For each $t \in T$, independently, adjust one dual variable: if $t \in T_c^1$, then set $\beta_{c,t} \leftarrow \beta_{c,t} + 1$ and otherwise set $\alpha_{c,t} \leftarrow \alpha_{c,t} + 1$.

3. If $T_c^1 \subsetneq T$, choose *active distance* $t^* \leftarrow \min(T \setminus T_c^1)$ to be the smallest non-saturated distance, and then set $x_{c,t^*} \leftarrow x_{c,t^*} + 1/t^*$. (Note that $0 \in T_c^1$, and thus $t^* > 0$.)
4. For any facility $f \in \biguplus_{t \in T_c^1} F_{c,t}$, independently, perform *augmentation of* y_f, setting
$$y_f \leftarrow \left(1 + \frac{1}{\mathsf{cost}(f)}\right) \cdot y_f + \frac{1}{|F| \cdot \mathsf{cost}(f)}.$$
5. Update the set of saturated distances, setting $T_c^1 \leftarrow \{t \in T : x_{c,t} \geq 1\}$.

We now argue that if variable y_f is augmented in Step 4, then $\mathsf{cost}(f) > 0$ (i.e., Step 4 is well defined). Let $\tau = \mathsf{cost}(c, f)$. As y_f is augmented, the distance τ is saturated ($x_{c,\tau} = 1$). If $\mathsf{cost}(f) = 0$, then y_f would have been initialized to 1, and then $y(F_{c,\tau}) \geq 1$, in which case the serving constraint for c would be already satisfied.

Side note about T. For the sake of coherence and more streamlined analysis, Frac increases also connection variables $x_{c,t}$ to empty sets $F_{c,t}$, i.e., invests into distances to non-existing facilities. Fixing this overspending would not lead to asymptotic improvement of the performance.

2.2.3 Structural properties

We focus on a single client c processed by Frac. We start with a property of connection variables $x_{c,t}$. The distances from T that are neither saturated nor active are called *inactive*. The following claim follows by an immediate induction on update operations performed by Frac.

Lemma 2.1. *At all times when a client c is considered, $x_{c,t} \in [0,1]$ for any $t \in T$. In particular, $x_{c,t} = 1$ for any saturated distance $t \in T_c^1$. Furthermore,*

1. *either all distances are saturated,*
2. *or there exists an active distance $t^* > 0$, such that (i) all smaller distances are saturated, and (ii) all larger distances are inactive and the corresponding variables $x_{c,t}$ are equal to zero.*

Augmentation is performed on variables y_f corresponding to facilities whose distance from c is saturated.

Lemma 2.2. *On any input $(G = (F, C, E, \mathsf{cost}); A)$,* Frac *returns a feasible solution and runs in time* $\mathrm{poly}(|G|, |A|, \max_{e \in E} \mathsf{cost}(e), \max_{f \in F} \mathsf{cost}(f))$.

Proof. Fix any client $c \in A$. By the definition of Frac, it takes t update operations to increase value $x_{c,t}$ from 0 to 1. Hence, after $\sum_{t \in T} t < 2 \cdot \max_{e \in E} \mathsf{cost}(e)$ update operations, all connection variables are equal to 1. From that point on, all variables y_f for $f \in \biguplus_{t \in T} F_{c,t}$ are augmented

in each update operation. Each variable y_f can be augmented at most $|F| \cdot \mathsf{cost}(f)$ times till it reaches or exceeds 1. That is, after at most $2 \cdot \max_{e \in E} \mathsf{cost}(e) + |F| \cdot \max_{f \in F} \mathsf{cost}(f)$ update operations, the serving constraint is satisfied, i.e., the generated solution is feasible. □

The following lemma shows the crucial property of Frac. Namely for any client c, there exist a "good" distance τ, such that the collection of clusters $F_{c,t}$ at distance $t \leq \tau$ is together fractionally half-open and that Frac invested $\Omega(\tau)$ into connecting client c. For any client c and distance $t \in T$, we define a set $S_{c,t}$ to be a collection of clusters alluded to in the introduction.

$$S_{c,t} = \biguplus_{t' \in T \,:\, t' \leq t} F_{c,t'}$$

Lemma 2.3. *Once* Frac *finishes serving client c, there exists a distance $\tau \in T$, such that $y(S_{c,\tau}) \geq 1/2$ and $\sum_{t \in T} t \cdot x_{c,t} \geq \tau/2$.*

Proof. We consider the state of variables once Frac finishes serving client c. Let $t^* > 0$ be the largest distance from T for which $x_{c,t^*} > 0$. As the serving constraint for client c is satisfied, we have

$$1 \leq \sum_{t \in T} \min\{x_{c,t},\, y(F_{c,t})\} = \min\{x_{c,t^*},\, y(F_{c,t^*})\} + \sum_{t \in T \,:\, t < t^*} \min\{x_{c,t},\, y(F_{c,t})\}. \tag{2.2}$$

We pick τ depending on the value of the last term of (2.2).

If $\min\{x_{c,t^*},\, y(F_{c,t^*})\} \geq 1/2$, we set $\tau = t^*$. Then, $y(S_{c,\tau}) \geq y(F_{c,\tau}) \geq \min\{x_{c,\tau},\, y(F_{c,\tau})\} \geq 1/2$, and the first condition of the lemma follows. Furthermore, $\sum_{t \in T} t \cdot x_{c,t} \geq \tau \cdot x_{c,\tau} \geq \tau/2$.

Otherwise, $\min\{x_{c,t^*},\, y(F_{c,t^*})\} < 1/2$, and then, by (2.2), $\sum_{t \in T \,:\, t < t^*} \min\{x_{c,t},\, y(F_{c,t})\} \geq 1/2$. In such case, we choose τ as the largest distance from T smaller than t^*. Then

$$y(S_{c,\tau}) = \sum_{t \in T \,:\, t \leq \tau} y(F_{c,t}) \geq \sum_{t \in T \,:\, t \leq \tau} \min\{x_{c,t},\, y(F_{c,t})\} \geq 1/2,$$

i.e., the first condition of the lemma holds. By Lemma 2.1, either t^* is active at the end of processing c or all distances become saturated and t^* is the largest distance from T. In either case, $x_{c,t} = 1$ for any distance $t < t^*$, and thus in particular $x_{c,\tau} = 1$. Hence, the second part of the lemma holds as $\sum_{t \in T} t \cdot x_{c,t} \geq \tau \cdot x_{c,\tau} = \tau$. □

2.2.4 Dual solution is almost feasible

Using primal-dual analysis, we may show that the generated dual solution violates each constraint at most by a factor of $O(\log |F|)$.

Lemma 2.4. *For any facility f,* Frac *augments y_f at most $O(\log |F|) \cdot \mathsf{cost}(f)$ times.*

Proof. First, we observe that variable y_f can be augmented only if prior to augmentation it is smaller than 1. To show that, observe that the augmentation of y_f occurs only when Frac processes an active client $c \in \mathsf{set}(f)$. Let $\tau = \mathsf{cost}(f, c)$, i.e., $f \in F_{c,\tau}$. As Frac augments y_f, the distance τ must be saturated, i.e., $x_{c,\tau} = 1$. On the other hand, the serving constraint (2.1) is not satisfied when y_f is augmented, and thus $\min\{x_{c,\tau}, y(F_{c,\tau})\} < 1$ which implies that y_f must be strictly smaller than 1.

In particular, if $\mathsf{cost}(f) = 0$, then y_f is set to 1 immediately at the beginning, and hence no augmentation of y_f is ever performed, and the lemma follows trivially. As all non-zero costs are at least 1, below we assume $\mathsf{cost}(f) \geq 1$.

During the first $\mathsf{cost}(f)$ augmentations, the value of y_f increases from 0 to at least $1/|F|$ (due to additive increases). Next, during the subsequent $\lceil \log_{1+1/\mathsf{cost}(f)} |F| \rceil$ augmentations, the value of y_f reaches at least 1 (due to multiplicative increases), and hence it will not be augmented further. In total, the number of augmentations is upper-bounded by $\mathsf{cost}(f) + \lceil \log_{1+1/\mathsf{cost}(f)} |F| \rceil = O(\log |F|) \cdot \mathsf{cost}(f)$. In the last relation, we used $\mathsf{cost}(f) \geq 1$. □

Lemma 2.5. Frac *violates each dual constraint at most by a factor of* $O(\log |F|)$.

Proof. We show the claim for all types of constraints in the dual program.

1. Each dual constraint $\gamma_c \leq \alpha_{c,t} + \beta_{c,t}$ always holds with equality as together with γ_c, for each $t \in T$, Frac increments either $\alpha_{c,t}$ or $\beta_{c,t}$.

2. Consider a constraint $\alpha_{c,t} \leq t$. Initially $\alpha_{c,t} = 0$ when client c appears, and it is incremented in an update operation only if distance t is not saturated. Distances are processed from the smallest to the largest, and it takes exactly t' update operations for a distance $t' \in T$ to become saturated. Therefore, $\alpha_{c,t}$ can be incremented at most $\sum_{t' \in T : t' \leq t} t'$ times. If $t = 0$, then $\alpha_{c,t} = 0$ trivially. Otherwise, we use the fact that $T \setminus \{0\}$ contains only powers of 2, and hence $\alpha_{c,t} \leq \sum_{t' \in T : t' \leq t} t' < 2 \cdot t$.

3. Finally, fix any facility $f^* \in F$ and consider the constraint $\sum_{c \in \mathsf{set}(f^*) \cap A_k} \beta_{c,\, \mathsf{cost}(f^*,c)} \leq \mathsf{cost}(f^*)$. We want to show that this constraint is violated at most by a factor of $O(\log |F|)$, i.e., that

$$\sum_{c \in \mathsf{set}(f^*) \cap A_k} \beta_{c,\, \mathsf{cost}(f^*,c)} \leq O(\log |F|) \cdot \mathsf{cost}(f^*). \tag{2.3}$$

   The left-hand side of (2.3) is initially 0 and it is incremented only when Frac processes some active client $c^* \in \mathsf{set}(f^*)$. In a single update operation, Frac may increment multiple β variables, but only one of them, namely $\beta_{c^*,\, \mathsf{cost}(f^*,c^*)}$, contributes to the growth

of the left-hand side of (2.3). If variable $\beta_{c^*,\mathsf{cost}(f^*,c^*)}$ is incremented, it means that the distance $\tau = \mathsf{cost}(f^*, c^*)$ is already saturated, i.e., $\tau \in T^1_{c^*}$. Thus, in the same update operation, FRAC augments all variables y_f for $f \in \biguplus_{t \in T^1_{c^*}} F_{c^*,t}$. This set of facilities includes cluster $F_{c^*,\tau}$ and thus also facility f^*. By Lemma 2.4, the augmentation of y_{f^*} may happen at most $O(\log |F|) \cdot \mathsf{cost}(f^*)$ times, which implies our claim. □

2.2.5 Competitive ratio of FRAC

Finally, we show that in each update operation the growth of the primal cost is at most constant times the growth of the dual cost. This will imply the competitive ratio of FRAC.

Lemma 2.6. *For any step k, the value of the solution to $\mathcal{P}_k$ computed by* FRAC *is at most 3 times the value of its solution to $\mathcal{D}_k$.*

Proof. As the values of both solutions are initially zero, it suffices to analyze the growth of the primal and dual objectives for a single update operation. The value of the dual solution grows by 1 as γ_c is incremented only for the requested client c. Thus, it is sufficient to show that the primal solution increases at most by 3.

By y_f, $x_{c,t}$ and T^1_c, we understand the values of these variables before an update operation. Let $F_1 = \biguplus_{t \in T^1_c} F_{c,t}$. As the serving constraint for client c is not satisfied at that point,

$$1 > \sum_{t \in T} \min\{x_{c,t},\, y(F_{c,t})\} \geq \sum_{t \in T^1_c} \min\{x_{c,t},\, y(F_{c,t})\} \geq \sum_{t \in T^1_c} y(F_{c,t}) = y(F_1). \tag{2.4}$$

In the last inequality we used that (by Lemma 2.1), $T^1_c = \{t \in T : x_{c,t} = 1\}$. The last equality follows as sets $F_{c,t}$ are disjoint for different t.

Within a single update operation, let $\Delta x_{c,t}$ and Δy_f be the increases of variables $x_{c,t}$ and y_f, respectively. By Lemma 2.1, FRAC increases one connection variable x_{c,t^*} for an active distance t^* (and no connection variable if there is no active distance) and performs augmentations of y_f for all $f \in F_1$. The increase of the primal value is then

$$\begin{aligned}\Delta\mathcal{P} &= \sum_{t \in T} t \cdot \Delta x_{c,t} + \sum_{f \in F_1} \mathsf{cost}(f) \cdot \Delta y_f \leq 1 + \sum_{f \in F_1} \mathsf{cost}(f) \cdot \left(\frac{y_f}{\mathsf{cost}(f)} + \frac{1}{|F| \cdot \mathsf{cost}(f)} \right) \\ &= 1 + y(F_1) + \frac{|F_1|}{|F|} < 3,\end{aligned}$$

where the last inequality follows by (2.4). □

Lemma 2.7. *For any input $(G = (F, C, E, \mathsf{cost}); A)$, it holds that* $\textsc{Frac}(G; A) \leq O(\log |F|) \cdot \textsc{Opt}(G; A)$.

Proof. Let k be the total number of active clients in A, and let $\mathsf{val}(\mathcal{P}_k)$ and $\mathsf{val}(\mathcal{D}_k)$ be the values of the final primal and dual solutions generated by FRAC. Then,

$$\begin{aligned} \textsc{Frac}(G;A) = \mathsf{val}(\mathcal{P}_k) &\leq 3 \cdot \mathsf{val}(\mathcal{D}_k) && \text{(by Lemma 2.6)} \\ &\leq O(\log |F|) \cdot \textsc{Opt}(G;A) && \text{(by Lemma 2.5 and weak duality).} \end{aligned}$$ □

2.3 Deterministic rounding

Now we define our deterministic algorithm INT, which rounds the fractional solution computed by FRAC. For a client $c \in A$, INT observes the actions of FRAC while processing c and on this basis makes its own decisions. First, INT processes augmentations of variables y_f performed by FRAC, and purchases some facilities. Once FRAC finishes handling client c, INT connects c to the closest open facility. (We show below that such facility exists.)

2.3.1 Purchasing facilities: properties of INTFAC

Purchasing facilities by INT is based solely on graph G and on updates of variables y_f produced by FRAC. In particular, it neglects whether a given client is active or not. We use integral variables $\hat{y}_f \in \{0,1\}$ to denote whether INT opened facility f. Furthermore, for any set F' we use $\hat{y}(F')$ as a shorthand for $\sum_{f \in F'} \hat{y}_f$.

The following lemma is an adaptation of the deterministic rounding routine for the set cover problem by Alon et al. [AAA+09] and its proof is postponed to Section 2.3.3.

Lemma 2.8. *Fix any input $(G = (F, C, E, \mathsf{cost}); A)$. Initially, $\hat{y}_f = y_f = 0$ for any $f \in F$. There exists a deterministic polynomial-time online algorithm* INTFAC *that transforms increments of fractional variables y_f to increments of integral variables $\hat{y}_f \in \{0,1\}$, so that*

- *condition $y(S_{c,t}) \geq 1/2$ implies $\hat{y}(S_{c,t}) \geq 1$ for any client $c \in C$ (active or inactive) and any $t \in T$,*
- $\sum_{f \in F} \mathsf{cost}(f) \cdot \hat{y}_f \leq O(\log |C \times T|) \cdot \sum_{f \in F} \mathsf{cost}(f) \cdot y_f + 2 \cdot \max_{f \in F} \mathsf{cost}(f)$.

2.3.2 Connecting clients

Once INT purchases facilities using deterministic routine INTFAC (cf. Lemma 2.8), it connects client c to the closest open facility. Now we show that such a facility indeed exists and we bound the competitive ratio of INT.

Lemma 2.9. *On any input $(G; A)$, the solution generated by* Int *is feasible and the total cost of connecting clients by* Int *is at most* $2 \cdot \textsc{Frac}(G; A)$.

Proof. Fix any client $c \in A$. By Lemma 2.3, there exists a distance $\tau \in T$ such that $y(S_{c,\tau}) \geq 1/2$ and $\sum_{t \in T} t \cdot x_{c,t} \geq \tau/2$. By Lemma 2.8, once Int purchases facilities, it holds that $\hat{y}(S_{c,\tau}) \geq 1$. It means that at least one facility is opened in set $S_{c,\tau}$, i.e., at distance at most τ from c.

Therefore, Int is feasible and by connecting client c to the closest open facility, it ensures that the connection cost is at most $\tau \leq 2 \cdot \sum_{t \in T} t \cdot x_{c,t}$. The proof is concluded by observing that $\sum_{t \in T} t \cdot x_{c,t}$ is the connection cost of Frac that can be attributed solely to the connection of client c. □

Lemma 2.10. *For any input $(G = (F, C, E, \mathsf{cost}); A)$, it holds that* $\textsc{Int}(G, A) \leq q \cdot \log|F| \cdot (\log|C| + \log\log\Delta_G) \cdot \textsc{Opt}(G, A) + 2 \cdot \max_{f \in F} \mathsf{cost}(f)$, *where q is a universal constant not depending on G or A. Furthermore,* Int *runs in time polynomial in $|G|$, $|A|$, $\max_{e \in E} \mathsf{cost}(e)$, and $\max_{f \in F} \mathsf{cost}(f)$.*

Proof. Let $\rho = \max_{f \in F} \mathsf{cost}(f)$. Then,

$$\begin{aligned}
\textsc{Int}(G; A) &\leq \sum_{f \in F} \mathsf{cost}(f) \cdot \hat{y}_f + 2 \cdot \textsc{Frac}(G; A) && \text{(by Lemma 2.9)} \\
&\leq O(\log|C \times T|) \cdot \textsc{Frac}(G; A) + 2 \cdot \rho && \text{(by Lemma 2.8)} \\
&= O((\log|C| + \log|T|) \cdot \log|F|) \cdot \textsc{Opt}(G; A) + 2 \cdot \rho && \text{(by Lemma 2.7).}
\end{aligned}$$

The bound on the cost of Int is concluded by using $|T| \leq 2 + \log\Delta_G$.

By Lemma 2.2, Frac running time is $\mathrm{poly}(|G|, |A|, \max_{e \in E} \mathsf{cost}(e), \max_{f \in F} \mathsf{cost}(f))$. On top of that, Int adds its own computations (in particular the rounding scheme of IntFac), whose runtime is polynomial in $|G|$ and $|A|$. This implies the second part of the lemma (the running time of Int). □

2.3.3 Purchasing facilities: algorithm INTFAC

We start with a technical claim and later we define our rounding procedure IntFac.

Lemma 2.11. *Fix any $q \in [0, 1/2]$ and any $r \geq 0$. Let X be a binary variable being 0 with probability $p > 0$. Then, $\mathbf{E}[\exp(q \cdot X)] \leq \exp(-(3/2) \cdot q \cdot \ln p)$.*

Proof. Using the definition of X, we have

$$\begin{aligned}\mathbf{E}[\exp(q \cdot X)] &= p \cdot \mathrm{e}^0 + (1-p) \cdot \mathrm{e}^q = \exp(\ln p) + (1 - \exp(\ln p)) \cdot \mathrm{e}^q \\ &\leq 1 + \ln p - e^q \cdot \ln p = 1 - \ln p \cdot (\mathrm{e}^q - 1) \\ &\leq 1 - (3/2) \cdot q \cdot \ln p \\ &\leq \exp(-(3/2) \cdot q \cdot \ln p).\end{aligned}$$

In the first inequality, we used that $e^x \cdot 1 + (1 - e^x) \cdot z \leq (1 + x) \cdot 1 + (-x) \cdot z$ for any $x \leq 0$ and $z \geq 1$ and in the second one, we used that $e^x - 1 \leq 3x/2$ for any $x \in [0, 1/2]$. □

Algorithm Description. As we mentioned earlier, our routine IntFac for rounding facilities is an adaptation of the deterministic rounding procedure for the set cover problem by Alon et al. [AAA+09]. On the basis of the facility-client graph G, we define the set $C \times T$ of *elements*. Intuitively, our solution Frac "covers" an element $(c, t) \in C \times T$ by fractionally opening facilities from $S_{c,t}$. The routine IntFac deterministically rounds these covering choices.

Let $\ell = |C \times T|$, $\rho = \max_{f \in F} \mathsf{cost}(F)$ and $b = 6 \cdot \ln \ell = O(\log |C \times T|)$. We consider the potential function $\Phi = \Phi_1 + \Phi_2$, where

$$\Phi_1 = \sum_{(c,t)\,:\,\hat{y}(S_{c,t})=0} \ell^{4 \cdot y(S_{c,t})} \qquad \text{and} \qquad \Phi_2 = \ell \cdot \exp\left(\sum_{f \in F} \frac{\mathsf{cost}(f)}{2\rho} \cdot (\hat{y}_f - b \cdot y_f) \right).$$

Assume that Frac augmented variable y_f. Then our algorithm IntFac chooses whether to set $\hat{y}_f$ to 1 or not (purchase f or not), so that the potential Φ does not increase. (We again emphasize that this choice neglects the current set of active clients.)

Correctness and Performance. In the lemma below, we show that IntFac is well defined, i.e., it is possible to fix variable $\hat{y}_f$, so that the potential Φ does not increase. This implies that both Φ_1 and Φ_2 remain upper-bounded, which can be in turn used to show properties of Lemma 2.8.

Lemma 2.12. *Assume y_{f^*} is increased by δ. If $\hat{y}_{f^*} = 1$, then Φ does not increase. Otherwise, there is a choice to either set $\hat{y}_{f^*}$ to 1 or not, so that Φ does not increase.*

Proof. By y_f and $\hat{y}_f$, we mean the values of these variables before an update operation of Frac.

First, we assume $\hat{y}_{f^*} = 1$. Increasing variable y_{f^*} affects values of $y(S_{c,t})$ for $f^* \in S_{c,t}$: all such $y(S_{c,t})$ increase by δ. However, for any element (c, t), such that $f^* \in S_{c,t}$, it holds that $\hat{y}(S_{c,t}) \geq \hat{y}_{f*} = 1$, i.e., element (c, t) is not counted in the sum occurring in Φ_1. Thus, increasing

variable y_{f^*} does not affect Φ_1. Furthermore, increasing y_{f^*} and keeping $\hat{y}_{f^*}$ unchanged can only decrease Φ_2. Thus, $\Phi = \Phi_1 + \Phi_2$ does not increase when $\hat{y}_{f^*} = 1$.

Second, we consider the case $\hat{y}_{f^*} = 0$. To show that either setting $\hat{y}_{f^*}$ to 1 or leaving it at 0 does not increase the potential, we use the probabilistic method and show that if we pick such action randomly (setting $\hat{y}_{f^*} = 1$ with probability $1 - \ell^{-4\cdot\delta}$), then, in expectation, neither Φ_1 nor Φ_2 increases.

- As observed above, only elements (c,t) for which $S_{c,t}$ contain f^* are affected by the increase of y_{f^*} and possible change of $\hat{y}_{f^*}$. Let $Q = \{(c,t) : f^* \in S_{c,t} \text{ and } \hat{y}(S_{c,t}) = 0\}$ be the set of such elements contributing to Φ_1.

  Fix any element $(c,t) \in Q$. Its initial contribution towards Φ_1 is $\ell^{4\cdot y(S_{c,t})}$ and when y_{f^*} increases, the contribution grows to $\ell^{4\cdot(y(S_{c,t})+\delta)}$. However, with probability $1 - \ell^{-4\cdot\delta}$, variable $\hat{y}_{f^*}$ is set to 1, thus $\hat{y}(S_{c,t})$ grows from 0 to 1, and in effect element (c,t) stops contributing to Φ_1. Hence, the expected final contribution of element (c,t) towards Φ_1 is $\ell^{4\cdot(y(S_{c,t})+\delta)} \cdot \ell^{-4\cdot\delta} + 0 \cdot (1 - \ell^{-4\cdot\delta}) = \ell^{4\cdot y(S_{c,t})}$, i.e., is equal to its initial contribution. Therefore, in expectation, the value of Φ_1 is unchanged.

- It remains to bound the expected value of Φ_2. Let $\hat{Y}$ be the random variable equal to the value of $\hat{y}_{f^*}$ after the random choice (i.e., $\hat{Y} = 1$ with probability $1 - \ell^{-4\cdot\delta}$) and Φ_2' denote the value of Φ_2 after increasing y_{f^*} and after the random choice. Using $y_{f^*} = 0$, we obtain

$$\begin{aligned}\Phi_2' &= \ell \cdot \exp\left(\sum_{f\in F} \frac{\mathsf{cost}(f)}{2\rho} \cdot (\hat{y}_f - b\cdot y_f) + \frac{\mathsf{cost}(f^*)}{2\rho}\cdot\hat{Y} - \frac{b\cdot\mathsf{cost}(f^*)}{2\rho}\cdot\delta\right)\\ &= \Phi_2 \cdot \exp\left(\frac{\mathsf{cost}(f^*)}{2\rho}\cdot\hat{Y}\right)\cdot\exp\left(-\frac{b\cdot\mathsf{cost}(f^*)}{2\rho}\cdot\delta\right).\end{aligned}$$

  To estimate $\mathbf{E}[\Phi_2']$, we upper-bound the expected value of $\exp(\hat{Y}\cdot\mathsf{cost}(f^*)/(2\rho))$, using Lemma 2.11 with $q = \mathsf{cost}(f^*)/(2\rho) \le 1/2$ and $p = \ell^{-4\cdot\delta}$, obtaining that

$$\mathbf{E}\left[\exp\left(\frac{\mathsf{cost}(f^*)}{2\rho}\cdot\hat{Y}\right)\right] \le \exp\left(-\frac{(3/2)\cdot\mathsf{cost}(f^*)}{2\rho}\cdot\ln p\right) = \exp\left(\frac{6\cdot\ln\ell\cdot\mathsf{cost}(f^*)}{2\rho}\cdot\delta\right).$$

  Therefore, $\mathbf{E}[\Phi_2'] \le \Phi_2$ and the lemma follows. □

We conclude this section with proof of Lemma 2.8 (restated below).

Lemma 2.8. *Fix any input $(G = (F, C, E, \mathsf{cost}); A)$. Initially, $\hat{y}_f = y_f = 0$ for any $f \in F$. There exists a deterministic polynomial-time online algorithm* IntFac *that transforms increments of fractional variables y_f to increments of integral variables $\hat{y}_f \in \{0,1\}$, so that*

- *condition* $y(S_{c,t}) \geq 1/2$ *implies* $\hat{y}(S_{c,t}) \geq 1$ *for any client* $c \in C$ *(active or inactive) and any* $t \in T$,
- $\sum_{f \in F} \mathsf{cost}(f) \cdot \hat{y}_f \leq O(\log |C \times T|) \cdot \sum_{f \in F} \mathsf{cost}(f) \cdot y_f + 2 \cdot \max_{f \in F} \mathsf{cost}(f)$.

Proof. Initially, all variables y_f and $\hat{y}_f$ are zero, and thus $\Phi = \sum_{(c,t) \in C \times T} \ell^0 + \ell \cdot \exp(0) = 2 \cdot \ell$. By Lemma 2.12, the potential never increases. Since Φ_2 is non-negative, any summand of Φ_1 is always at most $2 \cdot \ell \leq \ell^2$. Therefore, $4 \cdot y(S_{c,t}) \geq 2$ always implies $\hat{y}(S_{c,t}) > 0$, i.e., the first part of the lemma follows.

To show the second part, we again use that $\Phi = \Phi_1 + \Phi_2 \leq 2 \cdot \ell$ at any time. As Φ_1 is non-negative, $\Phi_2 \leq 2 \cdot \ell$. Substituting the definition of Φ_2, dividing by ℓ, and taking natural logarithm of both sides yields

$$\frac{1}{2\rho} \cdot \sum_{f \in F} \left(\hat{y}_f \cdot \mathsf{cost}(f) - b \cdot y_f \cdot \mathsf{cost}(f) \right) \leq \ln(2) < 1.$$

Therefore, $\sum_{f \in F} \hat{y}_f \cdot \mathsf{cost}(f) \leq 2\rho + b \cdot \sum_{f \in F} y_f \cdot \mathsf{cost}(f)$. □

2.4 Handling large aspect ratios

The guarantee of Lemma 2.10 has two deficiencies: (i) the bound on the competitive ratio of Int depends on the aspect ratio of G and on the cost of the most expensive facility, (ii) the running time of Int depends on the maximal cost in graph G (which can be exponentially large in the input description). We show how to use cost doubling and edge pruning to handle these issues, creating our final deterministic solution Det and proving the main theorem (restated below).

Theorem 2.1. *There exists a deterministic polynomial-time* $O(\log |F| \cdot (\log |C| + \log \log |F|))$*-competitive algorithm for the online non-metric facility location problem on set* F *of facilities and set* C *of clients.*

Proof. Fix facility-client graph $G = (F, C, E, \mathsf{cost})$ for the non-metric facility location problem. Recall that we assumed that all non-zero costs and distances in G are powers of 2 and are at least 1. Let $R = \log |F| \cdot (\log |C| + \log \log(|F| \cdot |C|))$.

We now construct a deterministic algorithm Det which is $O(R)$-competitive on an input $(G; A)$. Let q be the constant from Lemma 2.10. Det operates in phases, numbered from 0. In phase j, it executes the following operations.

1. DET *pre-purchases* all facilities and edges of G whose cost is smaller than $2^j/(|F| \cdot |C|)$.

2. DET creates an auxiliary facility-client graph $\tilde{G}_j$ applying the following modifications to G.

   - First, DET creates graph G_j containing only edges and facilities from G whose individual cost is at most 2^j. It also removes connections to facilities that have been removed in this process.
   - Second, the costs of all facilities and edges that have been pre-purchased by DET are set to zero in G_j. In a result, G_j is a sub-graph of G with adjusted distances and costs of facilities, has the same set of clients, its set of facilities is a subset of F, and $\Delta_{G_j} \leq |F| \cdot |C|$.
   - Third, $\tilde{G}_j$ is the modified version of G_j, where all costs have been scaled down, so that the smallest positive cost is equal to 1. We denote the scaling factor by $h_j \leq 1$.

3. DET simulates algorithm INT on input $(\tilde{G}_j; A)$. That is, for a client $c \in A$, DET verifies whether the overall cost of INT (including serving c) remains at most $h_j \cdot (q \cdot R + 2) \cdot 2^j$. In such case, DET outputs the choices of INT for client c as its own. We emphasize that INT is run also on clients that have been already served in the previous phases; in effect, DET may purchase the same facilities or connections multiple times.

4. Eventually, either the sequence A of active clients ends and the total cost of INT on $(\tilde{G}_j; A)$ is at most $h_j \cdot (q \cdot R + 2) \cdot 2^j$ (in which case DET terminates as well) or the purchases made by INT, while handling a client $c \in A$, caused its cost to exceed $h_j \cdot (q \cdot R + 2) \cdot 2^j$. (This includes the special case where c is disconnected from all facilities in $\tilde{G}_j$, because all edges incident to c in G were either more expensive than 2^j or were leading to facilities more expensive than 2^j.) In the case of exceeded cost, DET disregards the decisions of INT for client c, terminates INT, and starts phase $j + 1$, processing also all clients that were already served in phase j.

We now analyze the performance of DET. Let $k = \lceil \log(\text{OPT}(G; A)) \rceil \geq 0$. We show that DET terminates latest in phase k. Assume that DET has not finished within phases $0, 1, \ldots, k-1$. In phase k, DET creates auxiliary graphs G_k and $\tilde{G}_k$, and runs INT on graph $\tilde{G}_k$. Graph G_k contains all edges of G of cost at most 2^k; their cost in G_k is the same or reset to zero. As $\text{OPT}(G; A) \leq 2^k$, $\text{OPT}(G; A)$ purchases only edges that are in G_k, and thus $\text{OPT}(G; A)$ is also a feasible solution to instance $(G_k; A)$. Thus, $\text{OPT}(G_k; A) \leq \text{OPT}(G; A) \leq 2^k$. As $\tilde{G}_k$ is the scaled-down copy of G_k, $\text{OPT}(\tilde{G}_k; A) = h_k \cdot \text{OPT}(G_k; A) \leq h_k \cdot 2^k$.

Let $\tilde{F}_k$ be the set of facilities of graph $\tilde{G}_k$ and $\tilde{\text{cost}}_k(f)$ is the cost of opening facility f in graph $\tilde{G}_k$. Clearly, $|\tilde{F}_k| \leq |F|$ and $\tilde{\text{cost}}_k(f) \leq h_k \cdot \text{cost}(f)$ for any $f \in F$. By our construction, $\Delta_{\tilde{G}_k} = \Delta_{G_k} \leq |F| \cdot |C|$. Hence, Lemma 2.10 implies that

$$\begin{aligned}
\textsc{Int}(\tilde{G}_k; A) &\leq q \cdot \log |F_k| \cdot \left(\log |C| + \log\log \Delta_{\tilde{G}_k}\right) \cdot \textsc{Opt}(\tilde{G}_k; A) + 2 \cdot \max_{f \in \tilde{F}_k} \tilde{\text{cost}}_k(f) \\
&\leq h_k \cdot q \cdot \log |F| \cdot (\log |C| + \log\log(|F| \cdot |C|)) \cdot 2^k + 2 \cdot h_k \cdot 2^k \\
&= h_k \cdot (q \cdot R + 2) \cdot 2^k.
\end{aligned}$$

Therefore, Int is not terminated prematurely within phase k because of high cost and it finishes the entire sequence A. This implies the feasibility of Int: it serves all clients latest in phase k.

To bound the total cost of Det, recall that at the beginning of phase j, Det purchases at most $|F| \cdot |C|$ edges and at most $|F|$ facilities, each of cost at most $2^j/(|F| \cdot |C|)$. The associated overall cost is at most $2 \cdot 2^j$. The cost of the subsequent execution of algorithm Int on $\tilde{G}_j$ is, by our termination rule, at most $h_j \cdot (q \cdot R + 2) \cdot 2^j$, and thus the cost incurred by repeating Int's actions on G is at most $(q \cdot R + 2) \cdot 2^j$. The overall cost is then $\textsc{Det}(G; A) \leq \sum_{j=0}^{k} (q \cdot R + 4) \cdot 2^j = O(R) \cdot 2^k = O(R) \cdot \textsc{Opt}(G; A) = O(\log |F| \cdot (\log |C| + \log\log |F|)) \cdot \textsc{Opt}(G; A)$.

For the running time of Det, we note that in phase j, Int is run on a graph $\tilde{G}_j$ whose smallest cost is 1, and hence the largest cost is at most $\Delta_{\tilde{G}_j} = \Delta_{G_j} \leq |F| \cdot |C|$. Thus, by Lemma 2.10, the running time of Int in a single phase is polynomial in $|G|$ and $|A|$, and the number of phases is logarithmic in the maximum cost occurring in G, and thus also polynomial in $|G|$. □

2.5 Remarks and applications

2.5.1 Previous (unpublished) algorithm for non-metric facility location

As already mentioned in the introduction, one can obtain an algorithm with a slightly worse competitive ratio by using an *online* reduction to the set cover problem by Kolen and Tamir [KT90], which we present in this section.

We will show their reduction in three steps. First, for a given non-metric facility location input $\mathcal{I}_{\text{NFL}}$, we construct an instance of set cover problem $\mathcal{I}_{\text{SC}}$. Second, we show how to translate an online set cover algorithm into an online algorithm for non-metric facility location. To this end, we show what facilities and connections are purchased as the algorithm buys more sets. The cost of the resulting solution to $\mathcal{I}_{\text{NFL}}$ will be bounded by the cost of solution to $\mathcal{I}_{\text{SC}}$. Third, from any solution to $\mathcal{I}_{\text{NFL}}$, we create a solution to $\mathcal{I}_{\text{SC}}$ of comparable cost. We use this to relate the costs of optimal solutions to both problems.

Inputs. Fix $\mathcal{I}_{\text{NFL}} = ((F, C, E, \text{cost}); A)$. Let $T = \{\text{cost}(f, c) : f \in F, c \in C\}$ be the set of all distances appearing in $\mathcal{I}_{\text{NFL}}$ (recall that we assumed that all distances are either 0 or a power of two). We define $\mathcal{I}_{\text{SC}} = (S, U, \text{cost}; V)$ to be an input to the set cover problem, where $V \subseteq U$ is the sequence of elements to be covered. The universe U contains two types of elements: For each client $c \in C$, we include a *real* element e_c in U and for each client $c \in C$ and distance $t \in T$ we have a *virtual* element $e_{c,t}$ in U. Similarly, the set family S contains two types of sets. For each facility $f \in F$ we have a *real* set s_f in S with $\text{cost}(s_f) = \text{cost}(f)$ and for each client $c \in C$ and distance $t \in T$ we have a *virtual* set $s_{c,t}$ in U with $\text{cost}(s_{c,t}) = t$.

The idea of this constructions is to use real sets and elements for satisfying the coverage requirement of facility location. The virtual elements and sets will ensure that the connection cost is properly paid by the set cover solution.

An element e_c for client $c \in C$ is in set s_f if facility f can cover c. For client $c \in C$ and distance $t \in T$, an element $e_{c,t}$ can be covered by set $s_{c,t}$ and by all sets s_f for facilities f that are closer than t from c, i.e., if $\text{cost}(f, c) < t$. When client c appears in the input sequence $\mathcal{I}_{\text{NFL}}$ (i.e., $c \in A$), we add to the set V of active elements e_c and $e_{c,t}$ for all $t \in T$. That is, $V = \{e_c : c \in A\} \uplus \{e_{c,t} : c \in A, t \in T\}$.

Algorithm. Suppose we have a deterministic online algorithm $\textsc{Alg}_{\text{SC}}$ for the set cover problem. We will define a deterministic online algorithm $\textsc{Alg}_{\text{NFL}}$ for the non-metric facility location problem, such that

$$\textsc{Alg}_{\text{NFL}}(\mathcal{I}_{\text{NFL}}) \leq 2 \cdot \textsc{Alg}_{\text{SC}}(\mathcal{I}_{\text{SC}}). \tag{2.5}$$

When $\textsc{Alg}_{\text{SC}}$ buys set s_f corresponding to a facility $f \in F$, $\textsc{Alg}_{\text{NFL}}$ simply purchases facility f. Algorithm $\textsc{Alg}_{\text{NFL}}$ connects an active client $c \in A$ to the closest open facility (such facility exists because e_c has to be covered by a real set, which corresponds to some facility).

It remains to bound the cost. The opening cost of $\textsc{Alg}_{\text{NFL}}$ is clearly equal to the cost of real sets purchased by $\textsc{Alg}_{\text{SC}}$. To bound the connection cost, observe that if $\textsc{Alg}_{\text{NFL}}$ connects client c to facility f at distance $t = \text{cost}(f, c)$, then in the solution of $\textsc{Alg}_{\text{SC}}$, all elements $e_{c,t'}$ for $t' \leq t$ are covered by virtual sets $s_{c,t'}$. This incurs a cost of at most $\sum_{t' \in T : t' \leq t} t' < 2 \cdot t = 2 \cdot \text{cost}(f, c)$. Summing over all clients and facilities yields (2.5).

Optima. To lower bound the cost of optimal solution to $\mathcal{I}_{\text{NFL}}$, we define an offline algorithm $\textsc{Off}$ for the set cover problem. The algorithm $\textsc{Off}$ buys all real sets that correspond to facilities opened in $\textsc{Opt}(\mathcal{I}_{\text{NFL}})$. When an active client c is connected to facility f with $\text{cost}(f, c) = t$,

all virtual sets $s_{c,t'}$ for $t' \leq t$ are purchased. The cost of purchased real sets equals the cost of opened facilities, while the cost of all virtual sets is at most twice the connection cost of $\textsc{Opt}(\mathcal{I}_{\text{NFL}})$.

To show the feasibility of solution $\textsc{Off}(\mathcal{I}_{\text{SC}})$, fix $c \in A$ be an active client. Let f be the facility it is connected to in the solution $\textsc{Opt}(\mathcal{I}_{\text{SC}})$. Active element e_c, as well as elements $e_{c,t'}$ for $t' > \text{dist}(f,c)$ are covered by set s_f. For the remaining distances $t' \leq \text{dist}(f,c)$, clients $e_{c,t'}$ are covered by the virtual sets $s_{c,t'}$. Therefore, the cost of optimal solution to $\mathcal{I}_{\text{SC}}$ does not exceed the cost of solution $\textsc{Off}(\mathcal{I}_{\text{SC}})$ and thus

$$2 \cdot \textsc{Opt}(\mathcal{I}_{\text{NFL}}) \geq \textsc{Off}(\mathcal{I}_{\text{SC}}) \geq \textsc{Opt}(\mathcal{I}_{\text{SC}}). \tag{2.6}$$

Wrapping up. In this section, we presented a reduction, which transforms an instance of the non-metric facility location problem into an instance of the set cover problem with $|F| + |C| \cdot \log \Delta_G$ sets and $|C| + |C| \cdot \log \Delta_G$ elements. By, (2.5) and (2.6), c-competitive algorithm for this instance of the set-cover problem can be transformed into $4 \cdot c$-competitive algorithm for the non-metric facility location problem.

Using this reduction together with doubling techniques described in Section 2.4 and the deterministic algorithm for the online set cover problem by Alon et al. [AAA+09] yields a solution to the non-metric facility location problem whose competitive ratio is $O((\log |C| + \log |F|) \cdot (\log |C| + \log \log |F|))$.

2.5.2 Application to online node-weighted Steiner tree

Our result for the non-metric facility location problem has an immediate application for the online node-weighted Steiner tree (NWST) problem, where the graph consists of ℓ nodes and an online algorithm is given k terminals to be connected. Namely, the randomized solution for the online NWST problem by Naor et al. [NPS11] is in fact a deterministic polynomial-time "wrapper" around randomized routine solving the non-metric facility location problem. To solve an instance of the NWST problem, their algorithm constructs a sub-instance of non-metric facility location with $O(\ell)$ facilities, $O(\ell)$ potential clients, and $O(k)$ active clients. Such instance can be solved by the randomized algorithm of Alon et al. [AAA+06] with the competitive ratio of $O(\log k \cdot \log \ell)$. The wrapper adds another $O(\log k)$ factor in the ratio, resulting in an $O(\log^2 k \cdot \log \ell)$-competitive algorithm.

Our deterministic algorithm, when applied to this setting would be $O(\log^2 \ell)$-competitive on the constructed non-metric facility location sub-instance. Therefore, by replacing the

randomized algorithm by Alon et al. [AAA+06] with our deterministic one, we immediately obtain the first online deterministic solution for online NWST.

Corollary 2.1. *There exists a polynomial-time deterministic online algorithm for the node-weighted Steiner tree problem, which is $O(\log k \cdot \log^2 \ell)$-competitive on graphs with ℓ nodes and k terminals.*

We note that the currently best solution for the node-weighted Steiner tree is randomized and achieves the ratio of $O(\log^2 \ell)$ [HLP17, HLP14] and the best known lower bound for deterministic algorithms is $\Omega(\log \ell \cdot \log k / (\log \log \ell + \log \log k))$ [NPS11, AAA+09].

Chapter 3

Matching with delays

3.1 Introduction

In this chapter, we give a deterministic online algorithm for the problem of matching with delays [EKW16, ACK17]. For an informal description, imagine that there are players who are logging in real time into a gaming service, each wanting to compete with another human player. The system pairs the players according to their known capabilities, such as playing strength, and a decision with whom to match a given player can be delayed until a reasonable match is found. That is, the website tries to simultaneously minimize two objectives: the waiting times of players and their dissimilarity, i.e., each player would like to play with another one with comparable capabilities. An algorithm running the website has to work online, without the knowledge about future player arrivals and make its decision irrevocably: once two players are paired, they remain paired forever.

3.1.1 Problem definition

More formally, in the problem of matching with delays there is a metric space M with a distance function $\mathsf{dist} : M \times M \to \mathbb{R}$, both known from the beginning to an online algorithm. The online part of the input is a sequence of $2m$ requests $\{(p_i, t_i)\}_{i=1}^{2m}$, where point $p_i \in M$ corresponds to a player in our informal description above and t_i is the time of its arrival satisfying $t_1 \leq t_2 \leq \ldots \leq t_{2m}$. The integer m is not known a priori to the online algorithm. At any time τ, the online algorithm may decide to match any pair of requests (p_i, t_i) and (p_j, t_j) that have already arrived ($\tau \geq t_i$ and $\tau \geq t_j$) and have not been matched yet. The cost incurred by such *matching edge* is $\mathsf{dist}(p_i, p_j) + (\tau - t_i) + (\tau - t_j)$, i.e., the sum of the *connection*

cost and the *waiting costs* of these two requests. The goal is to eventually match all requests and minimize the total cost.

3.1.2 Previous work

The problem of matching with delays was introduced by Emek et al. [EKW16], who presented a randomized $O(\log^2 n + \log \Delta)$-competitive algorithm. There, n is the number of points in the metric space M and Δ is its aspect ratio (the ratio between the largest and the smallest distance in M). The competitive ratio was subsequently improved by Azar et al. [ACK17] to $O(\log n)$. They showed that the competitive ratio of any randomized algorithm is at least $\Omega(\sqrt{\log n})$. The currently best lower bound of $\Omega(\log n / \log \log n)$ for randomized solutions was given by Ashlagi et al. [AAC+17].

Until then, the construction of a competitive *deterministic* algorithm for general metric spaces remained an open problem. It was hypothesized that the competitive ratio achievable by deterministic algorithms might be superpolynomial in n (cf. Sect. 5 of [ACK17]). Deterministic algorithms were known only for simple spaces: Azar et al. [ACK17] gave an $O(\text{height})$-competitive algorithm for trees and Emek et al. [ESW17] constructed a 3-competitive deterministic solution for two-point metrics (the latter competitive ratio is best possible).

3.1.3 Our contribution

In this chapter, we give the historically first deterministic algorithm for an arbitrary metric space, whose competitive ratio is $O(m^{\log_2 5.5}) = O(m^{2.46})$, where $2m$ is the number of requests. While previous solutions to the problem of matching with delays [EKW16, ACK17] required M to be finite and known a priori (to approximate it first by a random HST tree [FRT04] or a random HST tree with reduced height [BBMN15]), our solution works even when M is revealed in online manner. That is, we require only that, together with any request r, the online algorithm learns the distances from r to all previous, not yet matched requests.

We note that m is uncomparable with n (the number of different points in the metric space M) and their relation depends on the application. For instance, in the traditional player ranking systems, such as Elo rating [Elo78] used for chess, M is just a finite set of integers (ratings) and typically there are many players with the same rating (multiple requests appear at the same point of M). For such setting, our algorithm is admittedly outperformed by the deterministic algorithm for trees by Azar et al. [ACK17] (which is $O(n)$-competitive for such metric). On the other hand, there is an emerging trend for representing players in online games

as multi-dimensional vectors [DCT+12, ASvZ13, CSEN17]: some real-time strategy games and first-person shooters started to characterize players by their rank, reflex, planning, offensive and defensive skills, number of actions per minute, network latency, etc. For such cases, M is a subset of $\mathbb{R}^k$ with ℓ^1 metric, and thus $m \ll n$.

Our online algorithm ALG uses a simple, local, semi-greedy scheme to find a suitable matching pair. In the analysis, we fix a final perfect matching of OPT and observe what happens when we gradually add matching edges that ALG creates during its execution. That is, we trace the evolution of alternating paths and cycles in time. To bound the cost of ALG, we charge the cost of an edge that ALG is adding against the cost of already existing matching edges from the same alternating path. Interestingly, our charging argument on alternating cycles bears some resemblance to the analyses of algorithms for the problems that are not directly related to matching with delays: online metric (bipartite) matching on line metrics [ABN+14] and offline greedy matching [RT81].

3.1.4 Related work

Originally, matching problems have been studied in variants where delaying decisions was not permitted. The setting most similar to the problem of matching with delays is called online *metric bipartite matching*. It involves m *offline points* given to an algorithm at the beginning and m *requests* presented in online manner that need to be matched (immediately after their arrival) to offline points. Both the points and the requests lie in a common metric space and the goal is to minimize the weight of a perfect matching created by the algorithm. For general metric spaces, the best randomized solution is $O(\log m)$-competitive [BBGN14, GL12, MNP06], and the deterministic algorithms achieve the optimal competitive ratio of $2m - 1$ [KP93, KMV94]. Interestingly, even for line metrics [ABN+14, FHK05, KN03], the best known deterministic algorithm attains a competitive ratio that is polynomial in m [ABN+14].

In comparison, in the problem of matching with delays considered in this chapter, all $2m$ requests appear in online manner (the requests need not be matched immediately after arrival), m is not known to an algorithm, and we allow to match any pair of them. That said, there is also a bipartite variant of the problem of matching with delays, in which all requests appear online, but m of them are negative and m are positive. An algorithm may then only match pairs of requests of different polarities [AAC+17, ACK17].

The problem of matching with delays can be cast as augmenting min-cost perfect matching with a time axis, allowing the algorithm to delay its decisions, but penalizing the delays. There

are many other problems that use this paradigm: the ski-rental problem and its continuous counterpart, the spin-block problem [KMMO94], various problems in aggregating messages in computer networks [AB05, BBC+13, DGS01, KKR03, KNR02, PSW10], and, more recently, service with delay [AGGP17, BKS18], set cover with delay [ACKT20], and network design (Steiner trees and facility location) with delay [AT20].

Finally, there is a vast amount of work devoted to other online matching variants, where offline points and online requests are connected by graph edges and the goal is to maximize the weight or the cardinality of the produced matching. These types of matching problems have been studied since the seminal work of Karp et al. [KVV90] and are motivated by applications to online auctions, see, e.g., a survey by Mehta [Meh13].

3.2 Algorithm

We will identify requests with the points at which they arrive. To this end, we assume that all requested points are different, but we allow distances between different metric points to be zero. For any request p, we denote the time of its arrival by $\mathsf{atime}(p)$.

Our algorithm is parameterized with real numbers $\alpha > 0$ and $\beta > 1$, whose exact values will be optimized later. For any request p, we define its waiting time at time $\tau \geq \mathsf{atime}(p)$ as

$$\mathsf{wait}_\tau(p) = \tau - \mathsf{atime}(p)$$

and its budget at time τ as

$$\mathsf{budget}_\tau(p) = \alpha \cdot \mathsf{wait}_\tau(p).$$

Our online algorithm Alg matches two requests p and q at time τ as soon as the following two conditions are satisfied:

budget sufficiency: $\mathsf{budget}_\tau(p) + \mathsf{budget}_\tau(q) \geq \mathsf{dist}(p, q)$,

budget balance: $\mathsf{budget}_\tau(p) \leq \beta \cdot \mathsf{budget}_\tau(q)$ and $\mathsf{budget}_\tau(q) \leq \beta \cdot \mathsf{budget}_\tau(p)$.

Note that the budget balance condition is equivalent to relations on waiting times, i.e., $\mathsf{wait}_\tau(p) \leq \beta \cdot \mathsf{wait}_\tau(q)$ and $\mathsf{wait}_\tau(q) \leq \beta \cdot \mathsf{wait}_\tau(p)$.

If the conditions above are met simultaneously for many point pairs, we break ties arbitrarily, and process them in any order. Note that at the time when p and q become matched, the sum of their budgets may exceed $\mathsf{dist}(p, q)$. For example, this occurs when q appears at time strictly

larger than $\mathsf{atime}(p) + \mathsf{dist}(p,q)$: they are then matched by ALG as soon as the budget balance condition becomes true.

The observation below follows immediately by the definition of ALG.

Observation 3.1. *Fix time τ and two requests p and q, such that $\mathsf{atime}(p) \leq \tau$ and $\mathsf{atime}(q) \leq \tau$. Assume that neither p nor q has been matched by* ALG *strictly before time τ. Then exactly one of the following conditions holds:*

- $\alpha \cdot (\mathit{wait}_\tau(p) + \mathit{wait}_\tau(q)) \leq \mathit{dist}(p,q)$,
- $\alpha \cdot (\mathit{wait}_\tau(p) + \mathit{wait}_\tau(q)) > \mathit{dist}(p,q)$ *and* $\mathit{wait}_\tau(p) \geq \beta \cdot \mathit{wait}_\tau(q)$,
- $\alpha \cdot (\mathit{wait}_\tau(p) + \mathit{wait}_\tau(q)) > \mathit{dist}(p,q)$ *and* $\mathit{wait}_\tau(q) \geq \beta \cdot \mathit{wait}_\tau(p)$.

3.3 Analysis

To analyze the performance of ALG, we look at matchings generated by ALG and by an optimal offline algorithm OPT. If points p and q were matched at time τ by ALG, then we say that ALG creates a (matching) edge $e = (p,q)$. Its cost is

$$\textsc{Alg}(e) = \textsc{Alg}(p,q) = \mathsf{dist}(p,q) + \mathsf{wait}_\tau(p) + \mathsf{wait}_\tau(q).$$

We call e an ALG-edge. The cost of an edge in the solution of OPT (an OPT-edge) is defined analogously. In an optimal solution, however, the matching time is always equal to the arrival time of the later of two matched requests.

We consider a dynamically changing graph consisting of requested points, OPT-edges and ALG-edges. For the analysis, we assume that it changes in the following way: all requested points and all OPT-edges are present in the graph from the beginning, but the ALG-edges are added to the graph in m steps, in the order they are created by ALG.

At all times, the matching edges present in the graph form alternating paths or cycles (i.e., paths or cycles whose edges are interleaved ALG-edges and OPT-edges). Furthermore, any node-maximal alternating path starts and ends with OPT-edges. Assume now that a matching edge e created by ALG is added to the graph. It may either connect the ends of two different alternating paths, thus creating a single longer alternating path or connect the ends of one alternating path, generating an alternating cycle. In the former case, we call edge e *non-final*, in the latter case — *final*. Note that at the end of the ALG execution, when m ALG-edges are added, the graph contains only alternating cycles.

We extend the notion of cost to alternating path and cycles. For any cycle C, $\mathsf{cost}(C)$ is simply the sum of costs of its edges: the cost of an Opt-edge on such cycle is the cost paid by Opt and the cost of an Alg-edge is that of Alg. We also define $\textsc{Opt}(C)$, $\textsc{Alg}(C)$ and $\textsc{Alg}_{\text{NF}}(C)$ as the costs of Opt-edges, Alg-edges and non-final Alg-edges on cycle C, respectively. Clearly, $\textsc{Alg}(C) + \textsc{Opt}(C) = \mathsf{cost}(C)$. We define the same notions for alternating paths; as a path P does not contain final Alg-edges, $\textsc{Alg}_{\text{NF}}(P) = \textsc{Alg}(P)$.

An alternating path is called *κ-step maximal alternating path* if it exists in the graph after Alg matched κ pairs and it cannot be extended, i.e., it ends with two requests that are not yet matched by the first κ Alg-edges.

3.3.1 Tree construction

To facilitate the analysis, along with the graph, we create a dynamically changing forest F of binary trees, where each leaf of F corresponds to an Opt-edge and each internal (non-leaf) node of F to a non-final Alg-edge (and vice versa). After Alg matched κ pairs, each subtree of F corresponds to a κ-step maximal alternating path or to an alternating cycle. More precisely, at the beginning, F consists of m single nodes representing Opt-edges. Afterwards, whenever an Alg-edge is created, we perform the following operation on F.

- When a non-final Alg-edge $e = (p, q)$ is added to the graph, we look at the two alternating paths P and Q that end with p and q, respectively. We take the corresponding trees $T(P)$ and $T(Q)$ of F. We add a node $v(e)$ (representing edge e) to F and make $T(P)$ and $T(Q)$ its subtrees.

- When a final Alg-edge $e = (p, q)$ is added to the graph, it turns an alternating path P into an alternating cycle C. We then simply say that the tree $T(P)$ that corresponded to P, now corresponds to C.

An example of the graph and the associated forest F is presented in Figure 3.1.

For any tree node w, we define its weight $\mathsf{weight}(w)$ as the cost of the corresponding matching edge, i.e., the cost of an Opt-edge for a leaf and the cost of a non-final Alg-edge for a non-leaf node. For any node w, by T_w we denote the tree rooted at w. We extend the notion of weight in a natural manner to all subtrees of F. In these terms, the weight of a tree T in F is equal to the total cost of the corresponding alternating path. (If T represents an alternating cycle C, then its weight is equal to the cost of C minus the cost of the final Alg-edge from C.)

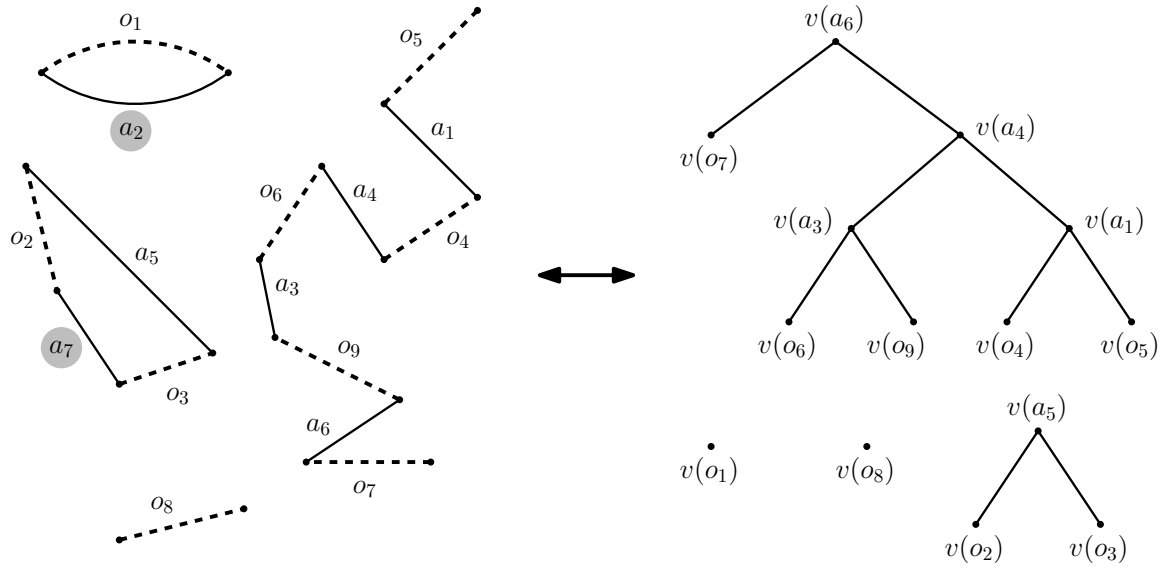

Figure 3.1: The left side contains an example graph consisting of all Opt-edges $o_1, o_2, \ldots, o_9$ (dashed lines) and the first $\kappa = 7$ Alg-edges $a_1, a_2, \ldots, a_7$ (solid lines). Alg-edges are numbered in the order they were created and added to the graph. Shaded Alg-edges (a_2 and a_7) are final, the remaining ones are non-final. The right side depicts the corresponding forest F: leaves of F represent Opt-edges and non-leaf nodes of F correspond to non-final Alg-edges. Trees rooted at nodes $v(o_1)$ and $v(a_5)$ represent alternating cycles and those rooted at nodes $v(a_6)$ and $v(o_8)$ represent alternating paths in the graph.

Note that we consistently used terms "points" and "edges" for objects that Alg and Opt are operating on in the metric space M. On the other hand, the term "nodes" will always refer to tree nodes in F and we will not use the term "edge" to denote an edge in F.

3.3.2 Outline of the analysis

Our approach to bounding the cost of Alg is now as follows. We look at the forest F at the end of Alg execution. The corresponding graph contains only alternating cycles. The cost of non-final Alg-edges is then, by the definition, equal to the total weight of internal (non-leaf) nodes of F, while the cost of Opt-edges is equal to the total weight of leaves of F. Hence, our goal is to relate the total weight of any tree to the weight of its leaves.

The central piece of our analysis is showing that for any internal node w with children u and v, it holds that $\mathsf{weight}(w) \leq \xi \cdot \min\{\mathsf{weight}(T_u), \mathsf{weight}(T_v)\}$, where ξ is a constant depending on parameters α and β (see Corollary 3.1). Using this relation, we will bound the total weight of any tree by $O(m^{\log_2(\xi+2)-1})$ times the total weight of its leaves. This implies the same bound on the ratio between non-final Alg-edges and Opt-edges on each alternating cycle.

Finally, we show that the cost of final ALG-edges incurs at most an additional constant factor in the total cost of ALG.

3.3.3 Cost of non-final ALG-edges

As described in Section 3.3.1, when ALG adds a κ-th ALG-edge e to the graph, and this edge is non-final, e joins two $(\kappa-1)$-step maximal alternating paths P and Q. We will bound $\text{ALG}(e)$ by a constant (depending on α and β) times $\min\{\mathsf{cost}(P), \mathsf{cost}(Q)\}$. We start with bounding the waiting cost of ALG related to one endpoint of e.

Lemma 3.1. *Let $e = (p, q)$ be the κ-th ALG-edge added at time τ, such that e is non-final. Let $P = (a_1, a_2, \ldots, a_\ell)$ be the $(\kappa-1)$-step maximal alternating path ending at $p = a_1$. Then, $\mathsf{wait}_\tau(p) \leq \max\{\alpha^{-1}, \beta/(\beta-1)\} \cdot \mathsf{cost}(P)$.*

Proof. First we lower-bound the cost of an alternating path P. We look at any edge (a_i, a_{i+1}) from P. Its cost (no matter whether paid by ALG or OPT) is certainly larger than $\mathsf{dist}(a_i, a_{i+1}) + |\mathsf{atime}(a_i) - \mathsf{atime}(a_{i+1})|$. Therefore, using triangle inequality (on distances and times), we obtain

$$\begin{aligned}\mathsf{cost}(P) &\geq \sum_{i=1}^{\ell-1} \left(\mathsf{dist}(a_i, a_{i+1}) + |\mathsf{atime}(a_i) - \mathsf{atime}(a_{i+1})|\right) \\ &\geq \mathsf{dist}(a_1, a_\ell) + |\mathsf{atime}(a_1) - \mathsf{atime}(a_\ell)|. \end{aligned} \tag{3.1}$$

Therefore, in our proof we will simply bound $\mathsf{wait}_\tau(p) = \mathsf{wait}_\tau(a_1)$ using either $\mathsf{dist}(a_1, a_\ell)$ or $|\mathsf{atime}(a_1) - \mathsf{atime}(a_\ell)|$.

Recall that ALG matches a_1 at time τ. Consider the state of a_ℓ at time τ. If a_ℓ has not been presented to ALG yet ($\mathsf{atime}(a_\ell) > \tau$), then $\mathsf{wait}_\tau(a_1) = \tau - \mathsf{atime}(a_1) < \mathsf{atime}(a_\ell) - \mathsf{atime}(a_1) < \beta/(\beta-1) \cdot (\mathsf{atime}(a_\ell) - \mathsf{atime}(a_1))$, and the lemma follows.

In the remaining part of the proof, we assume that a_ℓ was already presented to the algorithm ($\mathsf{atime}(a_\ell) \leq \tau$). As P is a $(\kappa-1)$-step maximal alternating path, a_ℓ is not matched by ALG right after ALG creates $(\kappa-1)$-th matching edge. The earliest time when a_ℓ may become matched is when ALG creates the next, κ-th matching edge, i.e., at time τ. Therefore a_ℓ is not matched before time τ.

Now observe that there must be a reason for which requests a_1 and a_ℓ have not been matched with each other before time τ. Roughly speaking, either the sum of budgets of requests a_1 and a_ℓ does not suffice to cover the cost of $\mathsf{dist}(a_1, a_\ell)$ or one of them waits

significantly longer than the other. Formally, we apply Observation 3.1 to the pair (a_1, a_ℓ) obtaining three possible cases. In each of the cases we bound $\mathsf{wait}_\tau(a_1)$ appropriately.

Insufficient budgets: If $\alpha \cdot (\mathsf{wait}_\tau(a_1) + \mathsf{wait}_\tau(a_\ell)) \leq \mathsf{dist}(a_1, a_\ell)$, then by non-negativity of $\mathsf{wait}_\tau(a_\ell)$, it follows that $\mathsf{wait}_\tau(a_1) \leq \alpha^{-1} \cdot \mathsf{dist}(a_1, a_\ell)$.

a_1 waited much longer than a_ℓ: If $\alpha \cdot (\mathsf{wait}_\tau(a_1) + \mathsf{wait}_\tau(a_\ell)) > \mathsf{dist}(a_1, a_\ell)$ and $\mathsf{wait}_\tau(a_1) \geq \beta \cdot \mathsf{wait}_\tau(a_\ell)$, then $\mathsf{atime}(a_\ell) - \mathsf{atime}(a_1) = \mathsf{wait}_\tau(a_1) - \mathsf{wait}_\tau(a_\ell) \geq (1 - 1/\beta) \cdot \mathsf{wait}_\tau(a_1)$. Therefore, $\mathsf{wait}_\tau(a_1) \leq \beta/(\beta - 1) \cdot |\mathsf{atime}(a_1) - \mathsf{atime}(a_\ell)|$.

a_ℓ waited much longer than a_1: If $\alpha \cdot (\mathsf{wait}_\tau(a_1) + \mathsf{wait}_\tau(a_\ell)) > \mathsf{dist}(a_1, a_\ell)$ and $\mathsf{wait}_\tau(a_\ell) \geq \beta \cdot \mathsf{wait}_\tau(a_1)$, then $\mathsf{atime}(a_1) - \mathsf{atime}(a_\ell) = \mathsf{wait}_\tau(a_\ell) - \mathsf{wait}_\tau(a_1) \geq (\beta - 1) \cdot \mathsf{wait}_\tau(a_1)$. Thus, $\mathsf{wait}_\tau(a_1) \leq 1/(\beta - 1) \cdot |\mathsf{atime}(a_1) - \mathsf{atime}(a_\ell)| < \beta/(\beta - 1) \cdot |\mathsf{atime}(a_1) - \mathsf{atime}(a_\ell)|$.

□

Lemma 3.2. *Let $e = (p, q)$ be the κ-th* Alg*-edge, such that e is non-final. Let $P = (a_1, a_2, \ldots, a_\ell)$ and $Q = (b_1, b_2, \ldots, b_{\ell'})$ be the $(\kappa - 1)$-step maximal alternating path ending at $p = a_1$ and $q = b_1$, respectively. Then,*

$$\textsc{Alg}(e) \leq (1 + \alpha) \cdot (\beta + 1) \cdot \max\{\alpha^{-1}, \beta/(\beta - 1)\} \cdot \min\{\mathsf{cost}(P), \mathsf{cost}(Q)\}.$$

Proof. Let τ be the time when p is matched with q by Alg. Using the definition of Alg, we obtain

$$\begin{aligned}
\textsc{Alg}(p, q) &= \mathsf{dist}(p, q) + \mathsf{wait}_\tau(p) + \mathsf{wait}_\tau(q) \\
&\leq \mathsf{budget}_\tau(p) + \mathsf{budget}_\tau(q) + \mathsf{wait}_\tau(p) + \mathsf{wait}_\tau(q) \\
&= (1 + \alpha) \cdot (\mathsf{wait}_\tau(p) + \mathsf{wait}_\tau(q)) \\
&\leq (1 + \alpha) \cdot (\beta + 1) \cdot \min\{\mathsf{wait}_\tau(p), \mathsf{wait}_\tau(q)\}. \qquad (3.2)
\end{aligned}$$

The first inequality follows by the budget sufficiency condition of Alg and the second one by the budget balance condition.

By Lemma 3.1, we have $\mathsf{wait}_\tau(p) \leq \max\{\alpha^{-1}, \beta/(\beta - 1)\} \cdot \mathsf{cost}(P)$ and $\mathsf{wait}_\tau(q) \leq \max\{\alpha^{-1}, \beta/(\beta - 1)\} \cdot \mathsf{cost}(Q)$, which combined with (3.2) immediately yield the lemma. □

Recall now the iterative construction of the forest F from Section 3.3.1: whenever a non-final matching edge e created by Alg joins two alternating paths P and Q, we add a new node w to F, such that $\mathsf{weight}(w) = \textsc{Alg}(e)$ and make trees $T(P)$ and $T(Q)$ its children. These trees

correspond to paths P and Q, and satisfy $\mathsf{weight}(T(P)) = \mathsf{cost}(P)$ and $\mathsf{weight}(T(Q)) = \mathsf{cost}(Q)$. Therefore, Lemma 3.2 immediately implies the following equivalent relation on tree weights.

Corollary 3.1. *Let w be an internal node of the forest F whose children are u and v. Then,*

$$\mathit{weight}(w) \leq (1+\alpha)\cdot(\beta+1)\cdot\max\left\{\alpha^{-1}, \beta/(\beta-1)\right\}\cdot\min\left\{\mathit{weight}(T_u), \mathit{weight}(T_v)\right\}.$$

This relation can be used to express the total weight of a tree of F in terms of the total weight of its leaves. Before we bound the cost of ALG on non-final edges of a single alternating cycle, we need a further technical claim that will facilitate the inductive proof.

Lemma 3.3. *Fix any constant $\xi \geq 0$ and let $f(a) = a^{\log_2(\xi+2)}$. Then,*

$$\xi\cdot\min\{f(x), f(y)\} + f(x) + f(y) \leq f(x+y)$$

for all $x, y \geq 0$.

Proof. Fix any $z \geq 0$ and let $g_z(a) = (\xi+1)\cdot f(a) + f(z-a)$. We observe that $g_z(0) = f(z)$ and $g_z(z/2) = (\xi+1)\cdot f(z/2) + f(z/2) = (\xi+2)\cdot(z/2)^{\log_2(\xi+2)} = z^{\log_2(\xi+2)} = f(z)$. Moreover, the function g_z is convex in the interval $[0, z]$ as it is a sum of two convex functions. As $g_z(0) = g_z(z/2) = f(z)$, by convexity, $g_z(a) \leq f(z)$ for any $a \in [0, z/2]$.

To prove the lemma, assume without loss of generality that $x \leq y$. By the monotonicity, $f(x) \leq f(y)$, and therefore

$$\begin{aligned}\xi\cdot\min\{f(x), f(y)\} + f(x) + f(y) &= (\xi+1)\cdot f(x) + f((x+y)-x)\\ &= g_{x+y}(x)\\ &\leq f(x+y).\end{aligned}$$

The last inequality follows as $x \leq (x+y)/2$. □

Lemma 3.4. *Let T be a weighted full binary tree and $\xi \geq 0$ be any constant. Assume that for each internal node w with children u and v, their weights satisfy $\mathit{weight}(w) \leq \xi\cdot\min\{\mathit{weight}(T_u), \mathit{weight}(T_v)\}$. Then,*

$$\mathit{weight}(T) \leq (\xi+2)\cdot|L_T(T)|^{\log_2(\xi/2+1)}\cdot\mathit{weight}(L_T(T)),$$

where $L_T(T)$ is the set of leaves of T and $\mathit{weight}(L_T(T))$ is their total weight.

Proof. We scale weights of all nodes, so that the average weight of each leaf is 1, i.e., we define a scaled weight function ws as

$$\mathsf{ws}(w) = \mathsf{weight}(w)\cdot\frac{|L_T(T)|}{\mathsf{weight}(L_T(T))}.$$

Note that ws also satisfies $\mathsf{ws}(w) \leq \xi \cdot \min\{\mathsf{ws}(T_u), \mathsf{ws}(T_v)\}$. Moreover, since we scaled all weights in the similar way, $\mathsf{ws}(T)/\mathsf{ws}(L_T(T)) = \mathsf{weight}(T)/\mathsf{weight}(L_T(T))$, and hence to show the lemma, it suffices to bound the term $\mathsf{ws}(T)/\mathsf{ws}(L_T(T))$.

For any node $w \in T$ and the corresponding subtree T_w rooted at w, we define $\mathsf{size}(T_w) = \mathsf{ws}(L_T(T_w)) + |L_T(T_w)|$. We inductively show that for any node of $w \in T$, it holds that

$$\mathsf{ws}(T_w) \leq \mathsf{size}(T_w)^{\log_2(\xi+2)}. \tag{3.3}$$

For the induction basis, assume that w is a leaf of T. Then,

$$\mathsf{ws}(T_w) = \mathsf{ws}(L_T(T_w)) \leq \mathsf{size}(T_w) \leq \mathsf{size}(T_w)^{\log_2(\xi+2)},$$

where the last inequality follows as $\mathsf{size}(T_w) \geq |L(T_w)| = 1$ and $\xi \geq 0$.

For the inductive step, let w be a non-leaf node of T and let u and v be its children. Then,

$$\begin{aligned}
\mathsf{ws}(T_w) &= \mathsf{ws}(T_u) + \mathsf{ws}(T_v) + \mathsf{ws}(w) \\
&\leq \mathsf{ws}(T_u) + \mathsf{ws}(T_v) + \xi \cdot \min\{\,\mathsf{ws}(T_u),\, \mathsf{ws}(T_v)\,\} \\
&\leq \mathsf{size}(T_u)^{\log_2(\xi+2)} + \mathsf{size}(T_v)^{\log_2(\xi+2)} \\
&\quad + \xi \cdot \min\left\{\mathsf{size}(T_u)^{\log_2(\xi+2)},\, \mathsf{size}(T_v)^{\log_2(\xi+2)}\right\} \\
&\leq (\mathsf{size}(T_u) + \mathsf{size}(T_v))^{\log_2(\xi+2)} \\
&= \mathsf{size}(T_w)^{\log_2(\xi+2)}.
\end{aligned}$$

The first inequality follows by the lemma assumption and the second one by the inductive assumptions for T_u and T_v. The last inequality is a consequence of Lemma 3.3 and the final equality follows by the additivity of function size.

Recall that we scaled weights so that $\mathsf{ws}(L_T(T)) = |L_T(T)|$. Therefore, applying (3.3) to the whole tree T yields $\mathsf{ws}(T) \leq (\mathsf{ws}(L_T(T)) + |L_T(T)|)^{\log_2(\xi+2)} = (2 \cdot |L_T(T)|)^{\log_2(\xi+2)} = (\xi+2) \cdot |L_T(T)|^{\log_2(\xi+2)}$. Hence,

$$\frac{\mathsf{weight}(T)}{\mathsf{weight}(L_T(T))} = \frac{\mathsf{ws}(T)}{\mathsf{ws}(L_T(T))} \leq \frac{(\xi+2) \cdot |L_T(T)|^{\log_2(\xi+2)}}{|L_T(T)|} = (\xi+2) \cdot |L_T(T)|^{\log_2(\xi/2+1)},$$

which concludes the proof. □

Lemma 3.5. *Let C be an alternating cycle obtained from combining matchings of* Alg *and* Opt. *Then* $\textsc{Alg}_{\textsc{NF}}(C) \leq (\xi+2) \cdot m^{\log_2(\xi/2+1)} \cdot \textsc{Opt}(C)$, *where* $\xi = (1+\alpha) \cdot (\beta+1) \cdot \max\{\alpha^{-1}, \beta/(\beta-1)\}$.

Proof. As described in Section 3.3.1, C is associated with a tree T from forest F, such that Opt-edges of C correspond to the set of leaves of T (denoted $L(T)$) and non-final Alg-edges

of C correspond to internal (non-leaf) nodes of T. Hence, $\textsc{Opt}(C) = \mathsf{weight}(L_T(T))$ and $\textsc{Alg}_{\text{NF}}(C) + \textsc{Opt}(C) = \mathsf{weight}(T)$.

By Corollary 3.1, the weight of any internal tree node w with children u, v satisfies $\mathsf{weight}(w) \leq \xi \cdot \min\{\mathsf{weight}(T_u), \mathsf{weight}(T_v)\}$. Therefore, we may apply Lemma 3.4 to tree T, obtaining $\mathsf{weight}(T) \leq (\xi+2) \cdot |L_T(T)|^{\log_2(\xi/2+1)} \cdot \mathsf{weight}(L(T))$, and thus

$$\begin{aligned}\textsc{Alg}_{\text{NF}}(C) \leq \mathsf{weight}(T) &\leq (\xi+2) \cdot |L(T)|^{\log_2(\xi/2+1)} \cdot \mathsf{weight}(L(T)) \\ &\leq (\xi+2) \cdot m^{\log_2(\xi/2+1)} \cdot \mathsf{weight}(L(T)) \\ &= (\xi+2) \cdot m^{\log_2(\xi/2+1)} \cdot \textsc{Opt}(C).\end{aligned}$$

The last inequality follows as $|L_T(T)|$, the number of T leaves, is equal to the number of $\textsc{Opt}$-edges on cycle C, which is clearly at most m. □

3.3.4 Cost of final ALG-edges

In the previous section, we derived a bound on the cost of all non-final $\textsc{Alg}$-edges. The following lemma shows that the cost of final $\textsc{Alg}$-edges contribute at most a constant factor to the competitive ratio.

Lemma 3.6. *Let e be a final $\textsc{Alg}$-edge matched at time τ and C be the alternating cycle containing e. Then $\textsc{Alg}(e) \leq (1+\alpha) \cdot \max\{\alpha^{-1}, (\beta+1)/(\beta-1)\} \cdot (\textsc{Alg}_{\text{NF}}(C) + \textsc{Opt}(C))$.*

Proof. Fix a final $\textsc{Alg}$-edge $e = (p, q)$, where $\mathsf{atime}(q) \geq \mathsf{atime}(p)$. By the budget sufficiency condition of $\textsc{Alg}$,

$$\textsc{Alg}(e) \leq (1+\alpha) \cdot (\mathsf{wait}_\tau(p) + \mathsf{wait}_\tau(q)). \tag{3.4}$$

Our goal now is to bound $\mathsf{wait}_\tau(p) + \mathsf{wait}_\tau(q)$ in terms of $\mathsf{dist}(p, q)$ or $\mathsf{atime}(q) - \mathsf{atime}(p)$. Observe that whenever $\textsc{Alg}$ matches two requests, the budget sufficiency condition of $\textsc{Alg}$ or one of the inequalities of the budget balance condition is satisfied with equality. We apply this observation to the pair (p, q).

- If the budget sufficiency condition holds with equality, then $\alpha \cdot (\mathsf{wait}_\tau(p) + \mathsf{wait}_\tau(q)) = \mathsf{dist}(p, q)$, and therefore $\mathsf{wait}_\tau(p) + \mathsf{wait}_\tau(q) = \alpha^{-1} \cdot \mathsf{dist}(p, q)$.
- If the budget balance condition holds with equality, $\beta \cdot \mathsf{wait}_\tau(q) = \mathsf{wait}_\tau(p)$. Then,

$$\begin{aligned}(\beta-1) \cdot (\mathsf{wait}_\tau(p) + \mathsf{wait}_\tau(q)) &= (\beta-1) \cdot (\beta+1) \cdot \mathsf{wait}_\tau(q) \\ &= (\beta+1) \cdot (\mathsf{wait}_\tau(p) - \mathsf{wait}_\tau(q)) \\ &= (\beta+1) \cdot (\mathsf{atime}(q) - \mathsf{atime}(p)).\end{aligned}$$

Hence, in either case it holds that

$$\mathsf{wait}_\tau(p) + \mathsf{wait}_\tau(q) \leq \max\left\{\alpha^{-1}, \frac{\beta+1}{\beta-1}\right\} \cdot (\mathsf{dist}(p,q) + |\mathsf{atime}(q) - \mathsf{atime}(p)|). \tag{3.5}$$

Finally, we bound $\mathsf{dist}(p,q) + |\mathsf{atime}(q) - \mathsf{atime}(p)|$ in terms of costs of other edges of C. These edges form a path $P = (a_1, a_2, \ldots, a_\ell)$, where $a_1 = p$ and $a_\ell = q$. By the triangle inequality applied to distances and time differences (in the same way as in (3.1)), we obtain that

$$\mathsf{dist}(p,q) + |\mathsf{atime}(q) - \mathsf{atime}(p)| \leq \mathsf{cost}(P) = \textsc{Alg}_{\text{NF}}(C) + \textsc{Opt}(C). \tag{3.6}$$

The lemma follows immediately by combining (3.4), (3.5) and (3.6). □

3.3.5 The competitive ratio

Finally, we optimize constants α and β used throughout the previous sections and bound the competitiveness of Alg.

Theorem 3.1. *For $\beta = 2$ and $\alpha = 1/2$, the competitive ratio of algorithm* Alg *is $O(m^{\log_2 5.5}) = O(m^{2.46})$, where $2m$ is the number of requests in the input sequence.*

Proof. The union of matchings constructed by Alg and Opt can be split into a set $\mathcal{C}$ of disjoint cycles. It is sufficient to show that we have the desired performance guarantee on each cycle from $\mathcal{C}$.

Fix a cycle $C \in \mathcal{C}$. Let $e = (p,q)$ be the final Alg-edge of C. By Lemma 3.6, $\textsc{Alg}(e) \leq 4.5 \cdot (\textsc{Alg}_{\text{NF}}(C) + \textsc{Opt}(C))$. Therefore, the competitive ratio of Alg is at most

$$\begin{aligned}\frac{\textsc{Alg}(C)}{\textsc{Opt}(C)} &\leq \frac{5.5 \cdot \textsc{Alg}_{\text{NF}}(C) + 4.5 \cdot \textsc{Opt}(C)}{\textsc{Opt}(C)} \\ &\leq O\left(m^{\log_2 5.5}\right) = O\left(m^{2.46}\right),\end{aligned}$$

where the second inequality follows by Lemma 3.5. □

3.4 Lower bounds and tightness

It is not known whether the analysis of our algorithm is tight. However, one can show that its competitive ratio is at least $\Omega(m^{\log_2 1.5}) = \Omega(m^{0.58})$. To this end, assume that all requests arrive at the same time. For such input, Opt does not pay for delays and simply returns the min-cost perfect matching. On the other hand, Alg computes the same matching as a greedy routine (i.e., it greedily connects two nearest, not yet matched requests). Hence, even if we

neglect the delay costs of ALG, its competitive ratio would be at least the approximation ratio of the greedy algorithm for min-cost perfect matching. The latter was shown to be $\Theta(m^{\log_2 1.5})$ by Reingold and Tarjan [RT81].

The reasoning above indicates an inherent difficulty of the problem. In order to beat the $\Omega(m^{\log_2 1.5})$ barrier, an online algorithm has to handle settings when all requests are given simultaneously more effectively. In particular, for such and similar input instances it has to employ a non-local and non-greedy policy of choosing requests to match.

Chapter 4

Steiner tree leasing

4.1 Introduction

The traditional network design [GK11] focuses on graph optimization problems, in which an algorithm purchases bandwidth on links to maintain certain graph properties, such as connectivity or throughput. A standard feature of most considered models is the permanence of bandwidth allocations. For example, in the Steiner tree problem [WH16], the goal is to buy a subset of edges connecting a given set of terminals to the chosen root node r. Even the online flavor of this problem [AA92, Ang09, IW91, Mat16, WY95] has this feature: the terminals arrive in online manner, and an algorithm irrevocably buys additional links, so that the terminals seen so far are connected to the root r. In this setting, each purchase is everlasting, i.e., the problem should be rather termed *incremental* Steiner tree.

Rent-or-buy variants. A well-studied modification of the online Steiner tree scenario is to relax the need of upfront commitment and additionally allow an algorithm to *rent* edges (at a fraction of the edge purchase price). Each terminal must be connected to the root r using either type of edges, but rented edges are valid only for a single terminal and cannot be reused by subsequent ones. This variant is called *online single-source rent-or-buy* [AK98, ABF93, BFR95, BS89, FGS04, LRWY99, Umb15] and is also equivalent to the *file replication* problem.

Note that the rent-or-buy variant is still incremental: bought edges persist in the graph till the end and — if a request to connect a specific terminal appears sufficiently many times in the input — any reasonable algorithm finally buys a path connecting this terminal to the root r.

Leasing variants. Many markets give an option of temporary leasing of resources. In particular, the advent of digital services in cloud computing changed the business model from buying physical servers to leasing virtual ones. This allowed companies to adapt quickly to varying requirements of their customers [AFG+09]. Furthermore, the software-defined networking enabled similar mechanisms on the network level, allowing companies to lease network links on the fly [CB10]. Typically, possible leases have different lengths and costs, and obey economies of scale, e.g., leasing a link for a week is more expensive than leasing it for a day, but not more than seven times. One can view rent-or-buy variants as an extreme case of leasing, where only two leases are available: a lifetime one (buying) and a lease of infinitesimal duration (renting).

These trends motivate the study of algorithmic leasing variants of popular network design mechanisms [AKM+15, AMM14, AG07, Mey05, NW13]. Note that from the algorithmic standpoint, leasing variants have a truly online nature: leases have finite duration and expire after some time. An online algorithm has to adapt itself to varying access patterns, e.g., by acquiring longer leases in response to increased demand.

4.1.1 The model

In this chapter, we study the online variant of the Steiner tree leasing problem introduced by Meyerson [Mey05]. The problem is defined in a weighted undirected graph G with a distinguished root node $r \in V(G)$. For each pair of nodes u and v, by $d_G(u, v)$ we denote the length (with respect to edge weights) of the shortest path between u and v. There is a known set $\mathcal{L}$ of available lease types, where each type $\ell \in \mathcal{L}$ is characterized by its duration D_ℓ and cost ratio C_ℓ. We denote the number of leases by $L = |\mathcal{L}|$.

An input to the problem consists of a sequence σ of requests, each being a terminal (a node of G), arriving sequentially in an online manner. We treat the root r also as a terminal. We assume that each request arrives at a different time (arrival times are real non-negative numbers). In response to a requested terminal σ_t, which appears at time t, an online algorithm has to connect σ_t to r using leased edges in G, leasing additional ones if necessary. (We note that a terminal may occur multiple times in sequence σ.) If an algorithm acquires a lease type $\ell \in \mathcal{L}$ of an edge $e = (u, v)$, it pays $C_\ell \cdot d_G(u, v)$. The edge leased at time t remains available for the period $[t, t + D_\ell)$; afterwards the lease expires.

To recap, an input instance $\mathcal{I}$ is a tuple $(G, d_G, r, \mathcal{L}; \sigma)$, where G, d_G, r, and $\mathcal{L}$ are known

a priori, and σ is presented in an online fashion to an algorithm. The total cost of edges leased by an algorithm is subject to minimization.

4.1.2 Previous results

The Steiner tree leasing problem on a single edge is known as the *parking permit problem* for which optimally competitive algorithms were given by Meyerson [Mey05]: a deterministic $O(L)$-competitive one and a randomized $O(\log L)$-competitive one. (Note that the rent-or-buy variant on a single edge is equivalent to the classic ski rental problem with a trivially achievable constant competitive ratio.)

The randomized algorithm can be extended to trees [Mey05]. As any n-node graph can be approximated by a random tree with expected distortion of $O(\log n)$ [FRT04], this approach yields a randomized $O(\log L \cdot \log n)$-competitive solution for graphs.

Better or non-randomized algorithms were known only for specific variants of Steiner tree leasing. In particular, for a rent-or-buy variant (recall that it corresponds to a special 2-lease variant, where the cheaper lease suffices only for serving a single request, and the more expensive lease lasts forever) Awerbuch, Azar and Bartal gave a randomized $O(\log k)$-competitive algorithm and $O(\log^2 k)$-deterministic one [AAB04]. The latter result was improved to $O(\log k)$ only recently by Umboh [Umb15]. In these results, k denotes the number of different terminals in an input.

Other network design problems were also studied in leasing context. In particular, a randomized $O(\log L \cdot \log n)$-competitive algorithm was given for the Online Steiner Forest Leasing by Meyerson [Mey05]. Deterministic algorithms for this problem are known only for the rent-or-buy subcase, for which an optimal competitive ratio of $O(\log k)$ was achieved by Umboh [Umb15]. Other problems include the facility location [AKM+15, NW13] and the set cover [AMM14].

4.1.3 Our contribution

In this chapter, we present the first deterministic online algorithm for Steiner tree leasing. Our algorithm is $O(L \cdot \log k)$-competitive. It outperforms the randomized $O(\log L \cdot \log n)$-competitive solution by Meyerson [Mey05] when k (the number of different terminals in the input) is small.

While the result might not be optimal, neither $O(L)$ nor $O(\log k)$ can be beaten by a deterministic solution: $\Omega(L)$ bound follows by the lower bound on the parking permit problem

(which is equivalent to Steiner tree leasing on a single edge) and $\Omega(\log k)$ bound follows by the online Steiner tree problem (which is a specific case of Steiner tree leasing with a single lease of the infinite duration).

In our solution (presented in Section 4.4), a path that connects a requested terminal to an already existing Steiner tree is chosen greedily. However, we still have to decide which lease type to use for such path. To this end, we check how many requests were "recently" served in a "neighborhood" of the currently requested terminal; once certain thresholds are met, more expensive leases are acquired. While such approach is natural, the main difficulty stems from the dynamics of the leased edges. Namely, while in the rent-or-buy scenario the Steiner tree maintained by an algorithm may only grow, in the leasing variant it may also shrink as edge leases expire. As a result, the already aggregated serving cost may cease to be sufficient to cover a more expensive lease for the new connection. Coping with this issue is the main challenge we tackle in this chapter.

We use a recent analysis technique by Umboh [Umb15]: the online algorithm is run on a graph G, but its cost is compared to the cost of Opt run on a tree T. This tree T is a hierarchically separated tree (HST), whose leaves are requested terminals and whose distances dominate graph distances. By showing that our algorithm is $O(L)$-competitive against Opt on T, for *any* choice of T, we obtain that it is $O(L \cdot \log k)$-competitive against Opt on the original graph G. The details of this reduction are presented in Section 4.2.

We emphasize that the competitive ratio of our algorithm is a function of the number of different terminals, k, and not the number of nodes in the graph, n, as it is the case for the randomized algorithm of [Meyo5]. In fact, our algorithm and its analysis work without changes also in any (infinite) metric space, e.g., on the Euclidean plane; in the chapter, we use the graph terminology for simplicity.

4.2 HST embeddings

In this section, we show how to use hierarchically separated trees (HSTs) for the analysis of an algorithm for the Steiner tree leasing problem. Unlike many online constructions for network design problems (see, e.g., [AA97, Meyo5]), here HSTs are not used for an algorithm construction. Moreover, in our analysis, an HST will approximate not the whole graph, but only the subgraph spanned by terminals.

Definition 4.1 (Dominating HST embedding of terminals)**.** *Fix any instance $\mathcal{I} = (G, d_G, r, \mathcal{L}; \sigma)$ of Steiner tree leasing. Let $X \subseteq V(G)$ be the set of terminals requested in σ (including the root r).*

Assume that the minimum distance between any pair of nodes from X is at least 1. *(For analysis, we may always scale the instance, so that this property holds.) A* dominating HST embedding *of terminals of $\mathcal{I}$ is a rooted tree T with pairwise distances given by metric d_T, satisfying the following properties.*

1. *The leaves of T are exactly the nodes of X and they are on the same level.*
2. *The distance from any leaf of T to its parent is* 1.
3. *The edge lengths increase by a factor of* 2 *on any leaf-to-root path.*
4. *d_T* dominates *d_G, i.e., $d_T(u,v) \geq d_G(u,v)$ for any pair of nodes $u, v \in X$.*

Fix now any instance $\mathcal{I} = (G, d_G, r, \mathcal{L}; \sigma)$ of Steiner tree leasing and let (T, d_T) be any dominating HST embedding of terminals of $\mathcal{I}$. Let $\mathcal{I}_T = (T, d_T, r, \mathcal{L}; \sigma)$ be the instance $\mathcal{I}$, where graph G was replaced by tree T with distances given by d_T.

While estimating $\textsc{Opt}(\mathcal{I})$ directly may be quite involved, lower-bounding $\textsc{Opt}(\mathcal{I}_T)$ is much easier. In particular, for each request σ_t there is a unique path in T connecting σ_t with r. As d_T dominates d_G, it is also feasible to compare the cost of an online algorithm on $\mathcal{I}$ to $\textsc{Opt}(\mathcal{I}_T)$. Finally, it is possible to relate $\textsc{Opt}(\mathcal{I}_T)$ to $\textsc{Opt}(\mathcal{I})$ as stated in the following lemma, due to Umboh [Umb15].

Lemma 4.1. *Let $\mathcal{I} = (G, d_G, r, \mathcal{L}; \sigma)$ be an instance of Steiner tree leasing and let X be the set of terminals of $\mathcal{I}$. There exists a dominating HST embedding (T^*, d_{T^*}) of terminals X, such that $\textsc{Opt}(\mathcal{I}_{T^*}) \leq O(\log |X|) \cdot \textsc{Opt}(\mathcal{I})$, where $\mathcal{I}_{T^*} = (T^*, d_{T^*}, r, \mathcal{L}; \sigma)$.*

Proof. Fix any dominating HST embedding (T, d_T) of terminals X. The solution $\textsc{Opt}(\mathcal{I})$ is a schedule that leases particular edges of G at particular times. Let $\textsc{Off}(\mathcal{I}_T)$ be an offline solution that, for any leased edge $e = (u, v)$ in $\textsc{Opt}(\mathcal{I})$, leases all edges on the unique path in T from u to v, using the same lease type. While it is not necessary for the proof, it is worth observing that, by the domination property (cf. Definition 4.1), $\textsc{Opt}(\mathcal{I}) \leq \textsc{Off}(\mathcal{I}_T)$.

By the FRT approximation [FRT04], there exists a probability distribution $\mathcal{D}$ over dominating HST embeddings (T, d_T) of X, such that $\mathbf{E}_{T \sim \mathcal{D}}[d_T(u,v)] \leq O(\log |X|) \cdot d_G(u,v)$ for all $u, v \in X$. This relation summed over all edges (u, v) used in the solution of $\textsc{Opt}(\mathcal{I})$ yields that

$$\mathbf{E}_{T \sim \mathcal{D}}[\textsc{Off}(\mathcal{I}_T)] \leq O(\log |X|) \cdot \textsc{Opt}(\mathcal{I}).$$

By the average argument, there exists a dominating HST embedding (T^*, d_{T^*}), such that $\textsc{Off}(\mathcal{I}_{T^*}) \leq O(\log |X|) \cdot \textsc{Opt}(\mathcal{I})$, and the proof follows by observing that $\textsc{Opt}(\mathcal{I}_{T^*})$ is at most $\textsc{Off}(\mathcal{I}_{T^*})$. □

The lemma can be generalized to any network design problem whose objective function is a linear combination of edge lengths. In Section 4.4, we will construct an algorithm for Steiner tree leasing which is $O(L)$-competitive against the cost of Opt on *any* HST embedding. By Lemma 4.1, this algorithm is $O(L \cdot \log k)$-competitive.

4.3 Interval model

In this section, we make several assumptions on the available leases. At the expense of a constant increase of the competitive ratio, they will make the construction of our algorithm easier. Similar assumptions were also made for the parking permit problem [Mey05].

Definition 4.2. *In the* interval model, *the following conditions hold for the input instance.*

- *Costs factors and durations of all leases are powers of two.*
- *Lease types are sorted both by their costs and durations, i.e., if* $\ell' < \ell$, *then* $D_{\ell'} < D_\ell$ *and* $C_{\ell'} < C_\ell$.
- *Fix any lease type* ℓ *and let* $J_\ell^m = [m \cdot D_\ell, (m+1) \cdot D_\ell)$ *for any* $m \in \mathbb{N}$. *For any time* t *and any edge, there is a unique lease of type* ℓ *that can be acquired by an algorithm: it is the lease for period* J_ℓ^m *containing* t.

The last property of the interval model means that, unlike the standard leasing model outlined in Section 4.1.1, if an algorithm leases an edge at a time t using a lease type $\ell \in \mathcal{L}$, such transaction may occur within the lease duration. Hence, the acquired lease may expire earlier than at time $t + D_\ell$. We also define $J_\ell[t]$ to be the period J_ℓ^m containing time t.

Observation 4.1. *In the interval model, when lease of type* ℓ *expires, all leases of smaller types expire as well.*

Lemma 4.2. *Any (online or offline) algorithm for the original leasing model can be transformed into an algorithm for the interval model (and back) without changing its cost by more than a constant factor.*

Proof. We introduce the changes in the model in three stages, ensuring that the cost of an algorithm in each stage changes at most by a constant factor.

1. We round down costs of leases to powers of two. This changes the cost of any algorithm at most by a factor of two.

2. We round down durations of leases to powers of two and limit the possibility of their purchase only to intervals J_ℓ^m. Any algorithm that is feasible in the modified model is clearly feasible in the original model (it purchases the corresponding, non-shortened leases at the same times). Furthermore, any lease in the original model is covered by at most three (disjoint and time-adjacent) leases in the modified model. Therefore, any algorithm feasible in the original model can be modified to work in the modified model and this increases its cost at most by a factor of three.

3. Finally, if we have two lease types $\ell, \ell' \in \mathcal{L}$, such that $D_\ell \leq D_{\ell'}$ and $C_\ell \geq C_{\ell'}$, no reasonable algorithm uses ℓ in its schedule, and hence we may remove such lease types from $\mathcal{L}$.

□

The lemma above shows that R-competitive algorithm for the interval model, is $O(R)$-competitive for the original leasing model. Therefore, we will assume the interval model in the remaining part of the chapter.

4.4 Algorithm construction

We present our algorithm Accumulate-and-Lease-Greedily (Alg). For simplicity of the description, we assume that a given graph G is complete (with the metric given by d_G). Such assumption is without loss of generality, as leasing the edge (u, v) can be always replaced by leasing a shortest path connecting u and v.

We will say that an edge e is *ℓ-leased* at time t, if an algorithm leased e for period $J_\ell[t]$ using lease type ℓ. Additionally, a request σ_t is *ℓ-leased* if at time t an algorithm ℓ-leases an edge $e = (\sigma_t, u)$ for some u.

By $F_\ell[t]$ and F_ℓ^m we denote the set of all requests that arrived during $J_\ell[t]$ and J_ℓ^m, respectively. Furthermore, $T_{\geq \ell}[t]$ denotes the set of requests that are connected, at time t, to the root r using edges of lease types at least ℓ.

High-level idea. In the execution of Alg, at any time t, the set of all currently leased edges will be a single (possibly empty) tree, called the Steiner tree of Alg. Furthermore, on any path from the root r that consists of leased edges, the closer we are to the root, the longer leases

we have. In effect, $T_{\geq \ell}[t]$ always forms a tree. Moreover, when leases expire, the set of leased edges shrinks, but it remains connected.

When a request σ_t arrives, we check whether we can afford a lease ℓ for σ_t, starting from the longest (and the most expensive) available lease: We compute the distance d from σ_t to $T_{\geq \ell}[t]$. Then, we check if there were "sufficiently many" requests served "recently" in a "small" (compared to d) neighborhood of σ_t. If so, then we connect σ_t to $T_{\geq \ell}[t]$ using lease type ℓ.

Algorithm description. More precisely, fix any time t when a request σ_t is presented to ALG. ALG checks, for each lease type ℓ starting from the most expensive (the L-th one), what the cost of connecting σ_t to the tree $T_{\geq \ell}[t]$ would be. That is, among all the nodes of $T_{\geq \ell}[t]$, it finds the node x_ℓ closest to σ_t. If we ℓ-lease the edge (σ_t, x_ℓ) at the cost $C_\ell \cdot d_G(\sigma_t, x_\ell)$, then σ_t becomes connected to the root r via a path of leased edges (of lease type at least ℓ). We round the distance $d_G(\sigma_t, x_\ell)$ up to the smallest power of two, denoted 2^j. Then, we look at the set N_ℓ^t (cf. Line 7 in Algorithm 1) of requests that

- arrived at any time in $J_\ell[t]$,
- are at distance at most $2^j/4$ from σ_t,
- are of class j (i.e., upon their arrival, ALG connected them to its Steiner tree, using an edge of length from $(2^{j-1}, 2^j]$, i.e., roughly $d_G(\sigma_t, x_\ell)$).

Note that the terminals of N_ℓ^t are not necessarily connected to the Steiner tree of ALG at time t. If the number of requests in N_ℓ^t is at least C_ℓ/C_1, then ALG ℓ-leases the edge (σ_t, x_ℓ) and sets the class of σ_t to j. Otherwise, ALG proceeds to cheaper lease types. If no lease type ℓ satisfies the condition $|N_\ell^t| \geq C_\ell/C_1$, ALG eventually 1-leases the edge (σ_t, x_1). Note that ALG leases exactly one edge for each terminal that is not connected to the tree at the time of its arrival.

Pseudocode of ALG is given in Algorithm 1. We recall the property of ALG stated earlier in its informal description.

Observation 4.2. *For any time t and lease type $\ell \in \mathcal{L}$, $T_{\geq \ell}[t]$ is a single tree.*

Algorithm 1 ACCUMULATE-AND-LEASE-GREEDILY for the Steiner tree leasing problem

```
1: while request σ_t arrives do
2:     for ℓ ← L ... 1 do
3:         if σ_t is not connected to the root r with a path of leased edges then
4:             /* Check whether we can afford lease ℓ for σ_t */
5:             let x_ℓ be the node of T_{≥ℓ}[t] closest to σ_t
6:             j ← ⌈log d_G(σ_t, x_ℓ)⌉
7:             N_ℓ^t ← {f ∈ F_ℓ[t] : d_G(σ_t, f) ≤ 2^{j−2} and class(f) = j}
8:             if |N_ℓ^t| · C_1 ≥ C_ℓ or ℓ = 1 then
9:                 ℓ-lease the edge (σ_t, x_ℓ)
10:                class(σ_t) ← j
11:            end if
12:        end if
13:    end for
14: end while
```

4.5 Analysis

Throughout this section, we fix an input instance $\mathcal{I} = (G, d_G, r, \mathcal{L}; \sigma)$ and a corresponding "tree instance" $\mathcal{I}_T = (T, d_T, r, \mathcal{L}; \sigma)$, where (T, d_T) is a dominating HST embedding of terminals of $\mathcal{I}$ (cf. Section 4.2).

Without loss of generality, we assume that when a request σ_t arrives, it is not yet connected to the Steiner tree of ALG (otherwise σ_t would be ignored by ALG). If σ_t was ℓ-leased, then we call N_ℓ^t (computed in Line 7 of Algorithm 1) its *neighbor set*. We denote the set of all requests of class j by W_j. Additionally, let W_j^ℓ consist of all requests from W_j that were ℓ-leased.

A brief idea of the proof is as follows. Suppose each request $\sigma_t \in W_j$ receives a credit of $C_1 \cdot 2^j$ when it arrives. If σ_t was ℓ-leased, the actual cost paid by ALG was $C_\ell \cdot 2^j$. While the latter amount can be much larger for an individual request σ_t, in Section 4.5.1, we show that, for any fixed lease type ℓ, the total cost paid for ℓ-leased edges is bounded by the sum of all requests' credits. In Section 4.5.2, we exploit properties of dominating HST embeddings to show how all credits can be charged (up to constant factors) to the leasing costs OPT pays for particular edges of the tree T. Altogether, this will show that $\text{ALG}(\mathcal{I}) \leq O(L) \cdot \text{OPT}(\mathcal{I}_T)$. Along with Lemma 4.1, this will bound the competitive ratio of ALG (see Section 4.5.3).

4.5.1 Upper bound on ALG

The core of this section is Lemma 4.4, which essentially states that for any lease type ℓ, all requests' credits can cover all leases of type ℓ. Before proceeding to its proof, we first show the following structural property.

Lemma 4.3. *Fix a class j, a lease type ℓ, and a pair of distinct requests $\sigma_s, \sigma_t \in W_j^\ell$. Their neighbor sets, N_ℓ^t and N_ℓ^s, are disjoint.*

Proof. Without loss of generality, $s < t$. We will prove the lemma by contradiction. Assume there exists a request $\sigma_u \in N_\ell^s \cap N_\ell^t$.

By the definition of neighbor sets, $\sigma_u \in F_\ell[s] \cap F_\ell[t]$. In the interval model, there are only two possibilities: either periods $J_\ell[s]$ and $J_\ell[t]$ are equal or they are disjoint. As in the latter case the corresponding sets $F_\ell[s]$ and $F_\ell[t]$ would be disjoint as well, it holds that $J_\ell[s] = J_\ell[t]$.

As the leases of type ℓ that ALG bought for σ_t and σ_s started and expired at the same time, σ_s was in the tree $T_{\geq\ell}[t]$ when σ_t arrived. Thus, the distance between σ_t and the tree $T_{\geq\ell}[t]$ was at most $d_G(\sigma_t, \sigma_s)$. From the triangle inequality and diameters of sets N_ℓ^s and N_ℓ^t, it follows that $d_G(\sigma_t, \sigma_s) \leq d_G(\sigma_t, \sigma_u) + d_G(\sigma_u, \sigma_s) \leq 2^{j-2} + 2^{j-2} = 2^{j-1}$. Hence, the request σ_t would be of class $j-1$ or lower, which would contradict its choice. □

Lemma 4.4. *For any class j and a lease type $\ell \in \mathcal{L}$, $C_\ell \cdot |W_j^\ell| \leq C_1 \cdot |W_j|$.*

Proof. The lemma follows trivially for $\ell = 1$, and therefore we assume that $\ell \geq 2$.

We look at any request $\sigma_t \in W_j^\ell$ and its neighbor set N_ℓ^t. As σ_t is of class j, N_ℓ^t contains requests only of class j, i.e., $N_\ell^t \subseteq W_j$. By Lemma 4.3, the neighbor sets of all requests from W_j^ℓ are disjoint, and hence $\sum_{\sigma_t \in W_j^\ell} |N_\ell^t| \leq |W_j|$.

As ALG ℓ-leases request σ_t, its neighbor set N_ℓ^t contains at least C_ℓ / C_1 requests. Therefore, $|W_j| \geq \sum_{\sigma_t \in W_j^\ell} |N_\ell^t| \geq \sum_{\sigma_t \in W_j^\ell} C_\ell / C_1 = |W_j^\ell| \cdot C_\ell / C_1$. □

Lemma 4.5. *For any input $\mathcal{I}$, it holds that $\text{ALG}(\mathcal{I}) \leq L \cdot \sum_j |W_j| \cdot C_1 \cdot 2^j$.*

Proof. The cost of serving any request $\sigma_t \in W_j^\ell$ is at most $C_\ell \cdot 2^j$. Using Lemma 4.4, we obtain $\text{ALG}(\mathcal{I}) \leq \sum_{\ell \in \mathcal{L}} \sum_j |W_j^\ell| \cdot C_\ell \cdot 2^j \leq L \cdot \sum_j |W_j| \cdot C_1 \cdot 2^j$. □

4.5.2 Lower bound on OPT

In this part, we bound the sum of all requests' credits by $O(1) \cdot \text{OPT}(\mathcal{I}_T)$.

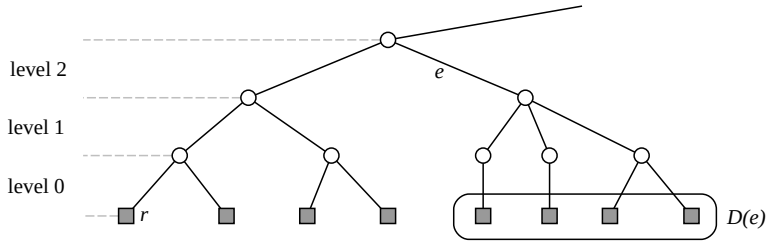

Figure 4.1: An example of an HST embedding. Square nodes (leaves of the HST) represent terminals from an input sequence (including root r). Edge e is a 2-level edge (of length 2^2). $D(e)$ is the set of leaves below edge e.

We number edge levels of T starting from bottom ones and counting from 0. That is, a j-level edge is of length 2^j. Moreover, for an edge e, we denote the set of all leaves below it by $D(e)$, see Figure 4.1. We denote the set of all j-level edges by E_j. The next observation follows immediately by Definition 4.1.

Observation 4.3. *Fix any j-level edge e. For any two leaves $u, v \in D(e)$, it holds that $d_G(u,v) \leq d_T(u,v) \leq 2^{j+1}$.*

Lemma 4.6. *For any request σ_t of class $j \geq 3$, there exists an edge $e \in E_{j-3}$, such that $\sigma_t \in D(e)$ and e lies on the unique path from σ_t to the root r in T.*

Proof. When ALG ℓ-leases request σ_t and assigns class j to it, the distance between σ_t and the tree $T_{\geq \ell}[t]$ is larger than 2^{j-1}, and thus $d_G(\sigma_t, r) > 2^{j-1}$. By Observation 4.3, the unique path in T between r and σ_t must cross two $(j-2)$-level edges, and hence also two $(j-3)$-level edges. We pick e to be the $(j-3)$-level edge that is closer to σ_t (see Figure 4.1). □

The lemma above implicitly creates a mapping φ from the set of all requests of class $j \geq 3$ to edges in tree T: a request of class j (i.e., connected by ALG with an edge of length at most 2^j) is mapped to a tree edge of level $j-3$ (of length 2^{j-3}). Note that a request of class j could be mapped to an edge in E_{j-2}. However, the next lemma requires that all the requests mapped to an edge e are close to each other. We extend φ to include also the requests of classes $j \leq 2$, by mapping any such request σ_t to the edge $e \in E_0$ adjacent to σ_t in tree T. In these terms, $\varphi^{-1}(e)$ is a set of requests mapped to e. For an edge e of level $j \geq 1$, $\varphi^{-1}(e) = D(e) \cap W_{j+3}$, and for $e \in E_0$, we have $\varphi^{-1}(e) = D(e) \cap \bigcup_{j=0}^{3} W_j$.

Let OPT(e) be the total leasing cost of e in the optimal solution for $\mathcal{I}_T$. Our goal now is to show that the sum of credits of requests in $\varphi^{-1}(e)$ is at most $O(\text{OPT}(e))$. To do so, we first prove a general bound on the amount of credit that holds for all possible periods J_ℓ^m. Later on, we will apply it to periods J_ℓ^m when OPT leased edge e.

Lemma 4.7. *Fix a lease type $\ell > 1$ and a j-level edge e of T. Then, for any $m \in \mathbb{N}$, $|\varphi^{-1}(e) \cap F_\ell^m| \leq 8 \cdot C_\ell / C_1$.*

Proof. We first assume that $j \geq 1$ and we will show that $|\varphi^{-1}(e) \cap F_\ell^m| \leq C_\ell/C_1 + 1 \leq 2 \cdot C_\ell/C_1$.

For a contradiction, assume that $\varphi^{-1}(e) \cap F_\ell^m$ contains more than $b = C_\ell/C_1 + 1$ requests. Let σ_s and σ_t be the b-th and the $(b+1)$-th of them, respectively. By Lemma 4.6, all requests of $\varphi^{-1}(e)$ are of class $j+3$ and are contained in $D(e)$. By Observation 4.3, they are all within a distance of 2^{j+1} from σ_s in the graph. Therefore, N_ℓ^s, the neighbor set of σ_s considered by ALG, contains all previous requests from $\varphi^{-1}(e) \cap F_\ell^m$ (there are $b - 1 = C_\ell/C_1$ many of them). The condition at Line 8 of Algorithm 1 is thus fulfilled, and therefore ALG buys a lease of type at least ℓ for σ_s.

In effect, when σ_t arrives, σ_s is in $T_{\geq \ell}[t]$. Hence, the distance from σ_t to the tree $T_{\geq \ell}[t]$ in the graph is at most $d_G(\sigma_t, \sigma_s) \leq d_T(\sigma_t, \sigma_s) \leq 2^{j+1}$. Therefore, the class of σ_t is at most $j+1$, which contradicts the choice of σ_t.

The analysis above can be extended to any 0-level edge e. Because $D(e)$ for $e \in E_0$ contains exactly one terminal, all requests from $\varphi^{-1}(e) \cap F_\ell^m$ are always contained in the appropriate neighbor set. This implies that $|\varphi^{-1}(e) \cap F_\ell^m \cap W_i| \leq 2 \cdot C_\ell/C_1$ for any class $i \in \{0,1,2,3\}$. As $\varphi^{-1}(e) = D(e) \cap \bigcup_{i=0}^{3} W_i$, we obtain $|\varphi^{-1}(e) \cap F_\ell^m| \leq 4 \cdot 2 \cdot C_\ell/C_1$. □

Lemma 4.8. *Fix a j-level edge e of T. Then, $|\varphi^{-1}(e)| \cdot C_1 \cdot 2^j \leq 8 \cdot \text{OPT}(e)$.*

Proof. By Lemma 4.6, for each request σ_t in $\varphi^{-1}(e)$, OPT has to have edge e leased at time t, as edge e lies on the only path between σ_t and the root r (see also Figure 4.1).

Let $P(e)$ be the set of all pairs (ℓ, m), such that OPT ℓ-leases e for period J_ℓ^m. That is, $\text{OPT}(e) = \sum_{(\ell,m)\in P(e)} C_\ell \cdot 2^j$. In the optimal solution J_ℓ^m periods are pairwise disjoint for all pairs (ℓ, m) in $P(e)$, and hence so are sets F_ℓ^m. Thus,

$$\begin{aligned} |\varphi^{-1}(e)| \cdot C_1 \cdot 2^j &= \sum_{(\ell,m)\in P(e)} |\varphi^{-1}(e) \cap F_\ell^m| \cdot C_1 \cdot 2^j \\ &\leq \sum_{(\ell,m)\in P(e)} 8 \cdot C_\ell \cdot 2^j = 8 \cdot \text{OPT}(e), \end{aligned}$$

where the inequality follows by Lemma 4.7. □

Lemma 4.9. *For any input $\mathcal{I}$ and any dominating HST embedding (T, d_T) of terminals of $\mathcal{I}$, it holds that $\sum_j |W_j| \cdot C_1 \cdot 2^j \leq O(1) \cdot \text{OPT}(\mathcal{I}_T)$.*

Proof. Fix any level $j \geq 1$. Recall that all requests of class $j+3$ (and only them) are mapped by φ to edges from E_j. Hence, we obtain

$$|W_{j+3}| = \sum_{e \in E_j} |\varphi^{-1}(e)|. \tag{4.1}$$

On the other hand, all requests of class $j \in \{0,1,2,3\}$ (and only them) are mapped by φ to edges from E_0. Therefore, $\sum_{j \leq 3} |W_j| = \sum_{e \in E_0} |\varphi^{-1}(e)|$, and consequently

$$\sum_{j \leq 3} |W_j| \cdot 2^j \leq 8 \sum_{e \in E_0} |\varphi^{-1}(e)|. \tag{4.2}$$

We use (4.1) and (4.2) to bound $\sum_j |W_j| \cdot 2^j$:

$$\begin{aligned}
\sum_{j \geq 0} |W_j| \cdot C_1 \cdot 2^j &\leq \sum_{e \in E_0} 8 \cdot |\varphi^{-1}(e)| \cdot C_1 + \sum_{j \geq 1} \sum_{e \in E_j} 2^{j+3} \cdot |\varphi^{-1}(e)| \cdot C_1 \\
&= \sum_{j \geq 0} \sum_{e \in E_j} 8 \cdot 2^j \cdot |\varphi^{-1}(e)| \cdot C_1 \\
&\leq \sum_{j \geq 0} \sum_{e \in E_j} 8 \cdot 8 \cdot \textsc{Opt}(e) = O(1) \cdot \textsc{Opt}(\mathcal{I}_T).
\end{aligned}$$

The second inequality is a consequence of Lemma 4.8. □

4.5.3 The competitive ratio

Theorem 4.1. Accumulate-and-Lease-Greedily *is $O(L \cdot \log k)$-competitive.*

Proof. Fix an instance $\mathcal{I} = (G, d_G, r, \mathcal{L}; \sigma)$. By Lemma 4.1, there exists a dominating HST embedding (T, d_T), such that $\textsc{Opt}(\mathcal{I}_T) \leq O(\log k) \cdot \textsc{Opt}(\mathcal{I})$, where $\mathcal{I}_T = (T, d_T, r, \mathcal{L}; \sigma)$. By Lemma 4.5, the total cost of Alg is at most L times the sum of all requests' credits, $\sum_j |W_j| \cdot C_1 \cdot 2^j$. By Lemma 4.9, the latter amount is at most $O(1) \cdot \textsc{Opt}(\mathcal{I}_T)$, and hence $\textsc{Alg}(\mathcal{I}) \leq O(L) \cdot \textsc{Opt}(\mathcal{I}_T) \leq O(L \cdot \log k) \cdot \textsc{Opt}(\mathcal{I})$, which concludes the proof. □

Chapter 5

Generalized k-server problem in uniform metrics

5.1 Introduction

The k-server problem, introduced by Manasse et al. [MMS90], is one of the most well-studied and influential cornerstones of online analysis. The problem definition is deceivingly simple: There are k servers, starting at a fixed set of k points of a metric space M. An input is a sequence of requests (points of M) and to service a request, an algorithm needs to move servers, so that at least one server ends at the request position. As typical for online problems, the k-server problem is sequential in nature: an online algorithm ALG learns a new request only after it services the current one. The cost of ALG is defined as the total distance traveled by all its servers. The goal is to service all requests and minimize the total cost.

In a natural extension of the k-server problem, called the *generalized k-server problem* [KT04, SS06], each server s_i remains in its own metric space M_i. The request is a k-tuple $(r_1, \ldots, r_k)$, where $r_i \in M_i$, and to service it, an algorithm needs to move servers, so that *at least one* server s_i ends at the request position r_i. The original k-server problem corresponds to the case where all metric spaces M_i are identical and each request is of the form $(r, \ldots, r)$. The generalized k-server problem contains many known online problems, such as the weighted k-server problem [BEK17, CV13, CS04, FR94] or the CNN problem [Chr03, KT04, Sit14, SS06] as special cases.

So far, the existence of an $f(k)$-competitive algorithm for the generalized k-server problem in arbitrary metric spaces remains open. Furthermore, even for specific spaces, such as the line [KT04] or uniform metrics [BEK17, BEKN18, CFK20, KT04], the generalized k-server

problem requires techniques substantially different from those used to tackle the classic k-server problems. For these reasons, studying this problem could lead to new techniques for designing online algorithms.

5.1.1 Previous work

After almost three decades of extensive research counted in dozens of publications (see, e.g., a slightly dated survey by Koutsoupias [Kou09]), we are closer to understanding the nature of the classic k-server problem. The competitive ratio achievable by deterministic algorithms is between k [MMS90] and $2k-1$ [KP95] with k-competitive algorithms known for special cases, such as uniform metrics [ST85], lines and trees [CKPV91, CL91], or metrics of $k+1$ points [MMS90]. Less is known about competitive ratios for randomized algorithms: the best known lower bound holding for an arbitrary metric space is $\Omega(\log k / \log\log k)$ [BLMN03] and the currently best upper bound of $O(\log^6 k)$ has been recently obtained in a breakthrough result [BCL+18, Lee18].

In comparison, little is known about the generalized k-server problem. In particular, algorithms attaining competitive ratios that are functions of k exist only in a few special cases. The case of $k = 2$ has been solved by Sitters and Stougie [SS06, Sit14], who gave constant competitive algorithms for this setting. Results for $k \geq 3$ are known only for simpler metric spaces, as described below.

A *uniform metric* case describes a scenario where all metrics M_i are uniform with pairwise distances between different points equal to 1. For this case, Bansal et al. [BEKN18] recently presented an $O(k \cdot 2^k)$-competitive deterministic algorithm and an $O(k^3 \cdot \log k)$-competitive randomized one. The deterministic competitive ratio is at least $2^k - 1$ already when metrics M_i have two points [KT04]. Furthermore, using a straightforward reduction to the metrical task system (MTS) problem [BLS92], they show that the randomized competitive ratio is at least $\Omega(k / \log k)$ [BEKN18].

A *weighted uniform metric* case describes a scenario where each metric M_i is uniform, but they have different scales, i.e., the pairwise distances between points of M_i are equal to some values $w_i > 0$. For this setting, Bansal et al. [BEKN18] gave an $2^{2^{O(k)}}$-competitive deterministic algorithm extending an $2^{2^{O(k)}}$-competitive algorithm for the weighted k-server problem in uniform metrics [FR94]. (The latter problem corresponds to the case where all requests are of the form $(r, \ldots, r)$.) This matches a lower bound of $2^{2^{\Omega(k)}}$ [BEK17] (which also holds already for the weighted k-server problem).

5.1.2 Our results and chapter organization

In this chapter, we study the uniform metric case of the generalized k-server problem. We give a randomized $O(k^2 \cdot \log k)$-competitive algorithm improving over the $O(k^3 \cdot \log k)$ bound by Bansal et al. [BEKN18].

To this end, we first define an elegant abstract online problem: a *Hydra game* played by an online algorithm against an adversary on an unweighted tree. We present the problem along with a randomized, low-cost online algorithm Herc in Section 5.2. We defer a formal definition of the generalized k-server problem to Section 5.3.1. Later, in Section 5.3.2 and Section 5.3.3, we briefly sketch the structural claims concerning the generalized k-server problem given by Bansal et al. [BEKN18]. Using this structural information, in Section 5.3.4, we link the generalized k-server problem to the Hydra game: we show that a (randomized) algorithm of total cost R for the Hydra game on a specific tree (called *factorial tree*) implies a (randomized) $(R+1)$-competitive solution for the generalized k-server problem. This, along with the performance guarantees of Herc given in Section 5.2, yields the desired competitiveness bound. We remark that while the explicit definition of the Hydra game is new, the algorithm of Bansal et al. [BEKN18] easily extends to its framework.

Finally, in Section 5.4, we give an explicit lower bound construction for the generalized k-server problem, which does not use a reduction to the metrical task system problem, hereby improving the bound from $\Omega(k/\log k)$ to $\Omega(k)$.

5.2 Hydra game

The Hydra game is played between an online algorithm and an adversary on a fixed unweighted tree T, known to the algorithm in advance. The nodes of T have states which change throughout the game: Each node can be either *asleep*, *alive* or *dead*. Initially, the root r_T is alive and all other nodes are asleep. At all times, the following invariant is preserved: all ancestors of alive nodes are dead and all their descendants are asleep. In a single step, the adversary picks a single alive node w, kills it (changes its state to dead) and makes all its (asleep) children alive. Note that such adversarial move preserves the invariant above.

An algorithm must remain at some alive node (initially, it is at the root r_T). If an algorithm is at a node w that has just been killed, it has to move to any still alive node w' of its choice. For such movement it pays $\mathrm{dist}(w, w')$, the length of the shortest path between w and w' in the tree T. The game ends when all nodes except one (due to the invariant, it has to be an alive leaf)

are dead. Unlike many online problems, here our sole goal is to minimize the total (movement) cost of an online algorithm (i.e., without comparing it to the cost of the offline optimum).

This game is not particularly interesting in the deterministic setting: As an adversary can always kill the node where a deterministic algorithm resides, the algorithm has to visit all but one nodes of tree T, thus paying $\Omega(|T|)$. On the other hand, a trivial DFS traversal of tree T has the cost of $O(|T|)$. Therefore, we focus on randomized algorithms and assume that the adversary is oblivious: it knows an online algorithm, but not the random choices made by it thus far.

5.2.1 Randomized algorithm definition

It is convenient to describe our randomized algorithm Herc as maintaining a probability distribution η over set of nodes, where for any node u, $\eta(u)$ denotes the probability that Herc is at u. We require that $\eta(u) = 0$ for any non-alive node u. Whenever Herc decreases the probability at a given node u by p and increases it at another node w by the same amount, we charge cost $p \cdot \mathsf{dist}(u, w)$ to Herc. By a straightforward argument, one can convert such description into a more standard, "behavioral" one, which describes randomized actions conditioned on the current state of an algorithm, and show that the costs of both descriptions coincide.

Lemma 5.1. *Let η be a probability distribution describing the position of* Herc *in the tree. Fix two tree nodes, u and w. Suppose η' is a probability distribution obtained from η by decreasing $\eta(u)$ by p and increasing $\eta(w)$ by p. Then,* Herc *can change its random position, so that it will be described by η', and the expected cost of such change is $p \cdot \mathsf{dist}(u, w)$.*

Proof. We define Herc's action as follows: if Herc is at node u, then with probability $p/\eta(u)$ it moves to node w. If Herc is at some other node it does not change its position.

We observe that the new distribution of Herc is exactly η'. Indeed, the probability of being at node u decreases by $\eta(u) \cdot p/\eta(u) = p$, while the probability of being at node w increases by the same amount. The probabilities for all nodes different than u or w remain unchanged.

Furthermore, the probability that Herc moves is $\eta(u) \cdot (p/\eta(u)) = p$ and the traveled distance is $\mathsf{dist}(u, w)$. The expected cost of the move is then $p \cdot \mathsf{dist}(u, w)$, as desired. □

To define Herc, we introduce the notion of nodes' ranks. At any time during the game, for any node u from tree T, $\mathsf{rank}(u)$ denotes the number of non-dead (i.e., alive or asleep) leaves in the subtree rooted at u. As Herc knows tree T in advance, it knows node ranks as well.

Algorithm Herc maintains η that is distributed over all alive nodes proportionally to their ranks. As all ancestors of an alive node are dead and all its descendants are asleep, we have $\eta(u) = \mathsf{rank}(u)/\mathsf{rank}(r_T)$ if u is alive and $\eta(u) = 0$ otherwise. In particular, at the beginning η is 1 at the root and 0 everywhere else.

While this already defines the algorithm, we still discuss its behavior when an alive node u is killed by the adversary. By Herc definition, we can think of the new probability distribution η' as obtained from η in the following way. First, Herc sets $\eta'(u) = 0$. Next, the probability distribution at other nodes is modified as follows.

Case 1. Node u is not a leaf. Herc distributes the probability of u among all (now alive) children of u proportionally to their ranks, i.e., sets $\eta'(w) = (\mathsf{rank}(w)/\mathsf{rank}(u)) \cdot \eta(u)$ for each child w of u.

Case 2. Node u is a leaf. Note that there were some other non-dead leaves, as otherwise the game would have ended before this step, and therefore $\eta(u) \leq 1/2$. Herc distributes $\eta(u)$ among all other nodes, scaling the probabilities of the remaining nodes up by a factor of $1/(1-\eta(u))$. That is, it sets $\eta'(w) = \eta(w)/(1-\eta(u))$ for any node w.

Note that in either case, η' is a valid probability distribution, i.e., all probabilities are non-negative and sum to 1. Moreover, η' is distributed over alive nodes proportionally to their new ranks, and is equal to zero at non-alive nodes.

Observation 5.1. *At any time, the probability of an alive leaf u is exactly $\eta(u) = 1/\mathsf{rank}(r_T)$.*

5.2.2 Analysis

For the analysis, we need a few more definitions. We denote the height and the number of the leaves of tree T by h_T and L_T, respectively. Let $\mathsf{level}(u)$ denote the height of the subtree rooted at u, where leaves are at level 0. Note that $h_T = \mathsf{level}(r_T)$.

To bound the cost of Herc, we define a potential Φ, which is a function of the current state of all nodes of T and the current probability distribution η of Herc. We show that Φ is initially $O(h_T \cdot (1 + \log L_T))$, is always non-negative, and the cost of each Herc's action can be covered by the decrease of Φ. This will show that the total cost of Herc is at most the initial value of Φ, i.e., $O(h_T \cdot (1 + \log L_T))$.

Recall that $\eta(w) = 0$ for any non-alive node w and that $\mathsf{rank}(u)$ is the number of non-dead leaves in the subtree rooted at u. Specifically, $\mathsf{rank}(r_T)$ is the total number of non-dead leaves

in T. The potential is defined as

$$\Phi = 4 \cdot h_T \cdot H(\mathsf{rank}(r_T)) + \sum_{w \in T} \eta(w) \cdot \mathsf{level}(w)\,, \tag{5.1}$$

where $H(n) = \sum_{i=1}^{n} 1/i$ is the n-th harmonic number.

Lemma 5.2. *At any time,* $\Phi = O(h_T \cdot (1 + \log L_T))$.

Proof. Since $\mathsf{rank}(r_T) \leq L_T$ at all times, the first summand of Φ is $O(h_T \cdot \log L_T)$. The second summand of Φ is a convex combination of node levels, which range from 0 to h_T, and is thus bounded by h_T. □

Lemma 5.3. *Fix any step in which an adversary kills a node u and in result* Herc *changes the probability distribution from η to η'. Let Δ*Herc *be the cost incurred in this step by* Herc *and let $\Delta\Phi$ be the resulting change in the potential Φ. Then, $\Delta\Phi \leq -\Delta$*Herc.

Proof. We denote the ranks before and after the adversarial event by rank and rank', respectively. We consider two cases depending on the type of u.

Case 1. The killed node u is an internal node. In this case, ΔHerc $= \eta(u)$ as Herc simply moves the total probability of $\eta(u)$ along a distance of one (from u to its children). As $\mathsf{rank}'(r_T) = \mathsf{rank}(r_T)$, the first summand of Φ remains unchanged. Let $C(u)$ be the set of children of u. Then,

$$\begin{aligned}
\Delta\Phi &= \sum_{w \in T} (\eta'(w) - \eta(w)) \cdot \mathsf{level}(w) = -\eta(u) \cdot \mathsf{level}(u) + \sum_{w \in C(u)} \eta'(w) \cdot \mathsf{level}(w) \\
&\leq -\eta(u) \cdot \mathsf{level}(u) + \sum_{w \in C(u)} \eta'(w) \cdot (\mathsf{level}(u) - 1) \\
&= -\eta(u) \cdot \mathsf{level}(u) + \eta(u) \cdot (\mathsf{level}(u) - 1) = -\Delta\text{Herc}\,,
\end{aligned}$$

where the inequality holds as level of a node is smaller than the level of its parent and the penultimate equality follows as the whole probability mass at u is distributed to its children.

Case 2. The killed node u is a leaf. It is not the last alive node, as in such case the game would have ended before, i.e., it holds that $\mathsf{rank}(r_T) \geq 2$. Herc moves the probability of $\eta(u) = 1/\mathsf{rank}(r_T)$ (cf. Observation 5.1) along a distance of at most $2 \cdot h_T$, and thus ΔHerc $\leq 2 \cdot h_T / \mathsf{rank}(r_T)$.

Furthermore, for any $w \neq u$, $\eta'(w) = \eta(w)/(1-\eta(u))$. Using $\eta(u) = 1/\mathsf{rank}(r_T)$, we infer that the probability at a node $w \neq u$ increases by

$$\begin{aligned}\eta'(w) - \eta(w) &= \left(\frac{1}{1-\eta(u)} - 1\right) \cdot \eta(w) = \frac{\eta(u)}{1-\eta(u)} \cdot \eta(w) \\ &= \frac{1}{\mathsf{rank}(r_T) - 1} \cdot \eta(w) \leq \frac{2}{\mathsf{rank}(r_T)} \cdot \eta(w)\,, \end{aligned} \tag{5.2}$$

where the last inequality follows as $\mathsf{rank}(r_T) \geq 2$.

Using (5.2) and the relation $\mathsf{rank}'(r_T) = \mathsf{rank}(r_T) - 1$ (the number of non-dead leaves decreases by 1), we compute the change of the potential:

$$\begin{aligned}\Delta\Phi &= 4 \cdot h_T \cdot \big(H(\mathsf{rank}'(r_T)) - H(\mathsf{rank}(r_T))\big) + \sum_{w \in T} (\eta'(w) - \eta(w)) \cdot \mathsf{level}(w) \\ &= -\frac{4 \cdot h_T}{\mathsf{rank}(r_T)} + (\eta'(u) - \eta(u)) \cdot \mathsf{level}(u) + \sum_{w \neq u} (\eta'(w) - \eta(w)) \cdot \mathsf{level}(w) \\ &\leq -\frac{4 \cdot h_T}{\mathsf{rank}(r_T)} + \sum_{w \neq u} \frac{2}{\mathsf{rank}(r_T)} \cdot \eta(w) \cdot h_T \leq -\frac{2 \cdot h_T}{\mathsf{rank}(r_T)} \leq -\Delta\textsc{Herc}\,. \end{aligned}$$

In the first inequality, we used that $\mathsf{level}(u) = 0$ and $\mathsf{level}(w) \leq h_T$ for any w.

Summing up, we showed that $\Delta\Phi \leq -\Delta\textsc{Herc}$ in both cases. □

Theorem 5.1. *For the Hydra game played on any tree T of height h_T and L_T leaves, the total cost of* Herc *is at most $O(h_T \cdot (1 + \log L_T))$.*

Proof. Let Φ_B denote the initial value of Φ. By non-negativity of Φ and Lemma 5.3, it holds that the total cost of Herc is at most Φ_B. The latter amount is at most $O(h_T \cdot (1 + \log L_T))$ by Lemma 5.2. □

Although Herc and Theorem 5.1 may seem simple, when applied to appropriate trees, they yield improved bounds for the generalized k-server problem in uniform metrics, as shown in the next section.

5.3 Improved algorithm for generalized k-server problem

In this part, we show how any solution for the Hydra game on a specific tree (defined later) implies a solution to the generalized k-server problem in uniform metrics. This will yield an $O(k^2 \log k)$-competitive randomized algorithm for the generalized k-server problem, improving the previous bound of $O(k^3 \cdot \log k)$ [BEKN18]. We note that this reduction is implicit in the paper of Bansal et al. [BEKN18], so our contribution is in formalizing the Hydra game and solving it more efficiently.

5.3.1 Preliminaries

The generalized k-server problem in uniform metrics is formally defined as follows. The offline part of the input comprises k uniform metric spaces $M_1, \ldots, M_k$. The metric M_i has $n_i \geq 2$ points, the distance between each pair of its points is 1. There are k servers denoted $s_1, \ldots, s_k$, the server s_i starts at some fixed point in M_i and always remains at some point of M_i.

The online part of the input is a sequence of requests, each request being a k-tuple $(r_1, \ldots, r_k) \in \prod_{i=1}^{k} M_i$. To service a request, an algorithm needs to move its servers, so that at least one server s_i ends at the request position r_i. Only after the current request is serviced, an online algorithm is given the next one. The cost of an algorithm is the total distance traveled by all its k servers.

5.3.2 Phase-based approach

We start by showing how to split the sequence of requests into phases. To this end, we need a few more definitions. A (server) *configuration* is a k-tuple $c = (c_1, \ldots, c_k) \in \prod_{i=1}^{k} M_i$, denoting positions of respective servers. For a request $r = (r_1, \ldots, r_k) \in \prod_{i=1}^{k} M_i$, we define the set of *compatible* configurations $\mathsf{comp}(r) = \{(c_1, \ldots, c_k) : \exists_i\, c_i = r_i\}$, i.e., the set of all configurations that can service the request r without moving a server. Other configurations we call *incompatible* with r.

An input is split into phases, with the first phase starting with the beginning of an input. The phase division process described below is constructed to ensure that Opt pays at least 1 in any phase, perhaps except the last one. At the beginning of a phase, all configurations are *phase-feasible*. Within a phase, upon a request r, all configurations incompatible with r become *phase-infeasible*. The phase ends once all configurations are phase-infeasible; if this is not the end of the input, the next phase starts immediately, i.e., all configurations are restored to the phase-feasible state before the next request. Note that the description above is merely a way of splitting an input into phases and marking configurations as phase-feasible and phase-infeasible. The actual description of an online algorithm will be given later.

Fix any finished phase and any configuration c and consider an algorithm that starts the phase with its servers at configuration c. When configuration c becomes phase-infeasible, such algorithm is forced to move and pay at least 1. As each configuration eventually becomes phase-infeasible in a finished phase, any algorithm (even Opt) must pay at least 1 in any finished phase. Hence, if the cost of a phase-based algorithm for servicing requests of a single phase can be bounded by $f(k)$, the competitive ratio of this algorithm is then at most $f(k)$.

5.3.3 Configuration spaces

Phase-based algorithms that we construct will not only track the set of phase-feasible configurations, but they will also group these configurations in certain sets, called *configuration spaces*.

To this end, we introduce a special *wildcard* character $\star$. Following [BEKN18], for any k-tuple $q = (q_1, \ldots, q_k) \in \prod_{i=1}^{k}(M_i \cup \{\star\})$, we define a *(configuration) space* $S[q] = \{(c_1, \ldots, c_k) \in \prod_{i=1}^{k} M_i : \forall_i\, c_i = q_i \vee q_i = \star\}$. A coordinate with $q_i = \star$ is called *free* for the configuration space $S[q]$. That is, $S[q]$ contains all configurations that agree with q on all non-free coordinates.

The number of free coordinates in q defines the *dimension* of $S[q]$ denoted $\dim(S[q])$. Observe that the k-dimensional space $S[(\star, \ldots, \star)]$ contains all configurations. If tuple q has no $\star$ at any position, then $S[q]$ is 0-dimensional and contains only (configuration) q. The following lemma, proven by Bansal et al. [BEKN18], follows immediately from the definition of configuration spaces.

Lemma 5.4 (Lemma 3.1 of [BEKN18])**.** *Let $S[q]$ be a d-dimensional configuration space (for some $d \geq 0$) whose all configurations are phase-feasible. Fix a request r. If there exists a configuration in $S[q]$ that is not compatible with r, then there exist d (not necessarily disjoint) subspaces $S[q_1], \ldots, S[q_d]$, each of dimension $d-1$, such that $\bigcup_i S[q_i] = S[q] \cap \mathsf{comp}(r)$. Furthermore, for all i, the k-tuples q_i and q differ exactly at one position.*

Using the lemma above, we may describe a way for an online algorithm to keep track of all phase-feasible configurations. To this end, it maintains a set $\mathcal{A}$ of (not necessarily disjoint) configuration spaces, such that their union is exactly the set of all phase-feasible configurations. We call spaces from $\mathcal{A}$ *alive*.

At the beginning, $\mathcal{A} = \{S[(\star, \ldots, \star)]\}$. Assume now that a request r makes some configurations from a d-dimensional space $S[q] \in \mathcal{A}$ phase-infeasible. (A request may affect many spaces from $\mathcal{A}$; we apply the described operations to each of them sequentially in an arbitrary order.) In such case, $S[q]$ stops to be alive, it is removed from $\mathcal{A}$ and till the end of the phase it will be called *dead*. Next, we apply Lemma 5.4 to $S[q]$, obtaining d configuration spaces $S[q_1], \ldots, S[q_d]$, such that their union is $S[q] \cap \mathsf{comp}(r)$, i.e., contains all those configurations from $S[q]$ that remain phase-feasible. We make all spaces $S[q_1], \ldots, S[q_d]$ alive and we insert them into $\mathcal{A}$. (Note that when $d = 0$, set $S[q]$ is removed from $\mathcal{A}$, but no space is added to it.) This way we ensure that the union of spaces from $\mathcal{A}$ remains equal to the set of all phase-feasible configurations. Note that when a phase ends, $\mathcal{A}$ becomes empty. We emphasize

that the evolution of set $\mathcal{A}$ within a phase depends only on the sequence of requests and not on the particular behavior of an online algorithm.

5.3.4 Factorial trees: from Hydra game to generalized k-server

Given the framework above, an online algorithm may keep track of the set of alive spaces $\mathcal{A}$, and at all times try to be in a configuration from some alive space. If this space becomes dead, an algorithm changes its configuration to any configuration from some other alive space from $\mathcal{A}$.

The crux is to choose an appropriate next alive space. To this end, our algorithm for the generalized k-server problem will internally run an instance of the Hydra game (a new instance for each phase) on a special tree, and maintain a mapping from alive and dead spaces to alive and dead nodes in the tree. Moreover, spaces that are created during the algorithm runtime, as described in Section 5.3.3, have to be dynamically mapped to tree nodes that were so far asleep.

In our reduction, we use a *k-factorial* tree. It has height k (the root is on level k and leaves on level 0). Any node on level d has exactly d children, i.e., the subtree rooted at a d-level node has $d!$ leaves, hence the tree name. On the k-factorial tree, the total cost of Herc is $O(k \cdot (1 + \log k!)) = O(k^2 \cdot \log k)$. We now show that this implies an improved algorithm for the generalized k-server problem.

Theorem 5.2. *If there exists a (randomized) online algorithm* H *for the Hydra game on the k-factorial tree of total (expected) cost R, then there exists a (randomized) $(R+1)$-competitive online algorithm* G *for the generalized k-server problem in uniform metrics.*

Proof. Let $\mathcal{I}$ be an input for the generalized k-server problem in uniform metric spaces. G splits $\mathcal{I}$ into phases as described in Section 5.3.2 and, in each phase, it tracks the phase-feasible nodes using set $\mathcal{A}$ of alive spaces as described in Section 5.3.3. For each phase, G runs a new instance I_H of the Hydra game on a k-factorial tree T, translates requests from $\mathcal{I}$ to adversarial actions in I_H, and reads the answers of H executed on I_H. At all times, G maintains a (bijective) mapping from alive (respectively, dead) d-dimensional configuration spaces to alive (respectively, dead) nodes on the d-th level of the tree T. In particular, at the beginning, the only alive space is the k-dimensional space $S[(\star, \ldots, \star)]$, which corresponds to the tree root (on level k). The configuration of G will always be an element of the space corresponding to the tree node containing H. More precisely, within each phase, a request r is processed in the following way by G.

- Suppose that request r does not make any configuration phase-infeasible. In this case, G services r from its current configuration and no changes are made to $\mathcal{A}$. Also no adversarial actions are executed in the Hydra game.

- Suppose that request r makes some (but not all) configurations phase-infeasible. We assume that this kills only one d-dimensional configuration space $S[q]$. (If r causes multiple configuration spaces to become dead, G processes each such killing event separately, in an arbitrary order.)

  By the description given in Section 5.3.3, $S[q]$ is then removed from $\mathcal{A}$ and d new $(d-1)$-dimensional spaces $S[q_1], \ldots, S[q_d]$ are added to $\mathcal{A}$. G executes appropriate adversarial actions in the Hydra game: a node v corresponding to $S[q]$ is killed and its d children on level $d-1$ change state from asleep to alive. G modifies the mapping to track the change of $\mathcal{A}$: (new and now alive) spaces $S[q_1], \ldots, S[q_d]$ become mapped to (formerly asleep and now alive) d children of v. Afterwards, G observes the answer of algorithm H on the factorial tree and replays it. Suppose H moves from (now dead) node v to an alive node v', whose corresponding space is $S[q'] \in \mathcal{A}$. In this case, G changes its configuration to the closest configuration (requiring minimal number of server moves) from $S[q']$. It remains to relate its cost to the cost of H. By Lemma 5.4 (applied to spaces corresponding to all nodes on the tree path from v to v'), the corresponding k-tuples q, q' differ on at most $\mathrm{dist}(v, v')$ positions. Therefore, adjusting the configuration of G, so that it becomes an element of $S[q']$, requires at most $\mathrm{dist}(v, v')$ server moves, which is exactly the cost of H.

  Finally, note that when G processes all killing events, it ends in a configuration of an alive space, and hence it can service the request r from its new configuration.

- Suppose that request r makes all remaining configurations phase-infeasible. In such case, G moves an arbitrary server to service this request, which incurs a cost of 1. In this case, the current phase ends, a new one begins, and G initializes a new instance of the Hydra game.

Let $f \geq 1$ be the number of all phases for input $\mathcal{I}$ (the last one may be not finished). The cost of Opt in a single finished phase is at least 1. By the reasoning above, the (expected) cost of G in a single phase is at most $R+1$. Therefore, $\mathbf{E}[\mathrm{G}(I)] \leq (R+1) \cdot f \leq (R+1) \cdot \textsc{Opt}(I) + (R+1)$, which completes the proof. □

Using our algorithm Herc for the Hydra game along with the reduction given by Theorem 5.2 immediately implies the following result.

Corollary 5.1. *There exists a randomized $O(k^2 \cdot \log k)$-competitive online algorithm for the generalized k-server problem in uniform metrics.*

5.4 Lower bound

Next, we show that that competitive ratio of any (even randomized) online algorithm for the generalized k-server problem in uniform metrics is at least $\Omega(k)$, as long as each metric space M_i contains at least two points. For each M_i, we choose two distinct points, the initial position of the i-th server, which we denote 0 and any other point, which we denote 1. The adversary is going to issue only requests satisfying $r_i \in \{0, 1\}$ for all i, hence without loss of generality any algorithm will restrict its server's position in each M_i to 0 and 1. (To see this, assume without loss of generality that the algorithm is lazy, i.e., it is only allowed to move when a request is not covered by any of its server, and is then allowed only to move a single server to cover that request.) For this reason, from now on we assume that $M_i = \{0, 1\}$ for all i, ignoring superfluous points of the metrics.

The configuration of any algorithm can be then encoded using a binary word of length k. It is convenient to view all these 2^k words (configurations) as nodes of the k-dimensional hypercube: two words are connected by a hypercube edge if they differ at exactly one position. Observe that a cost of changing configuration c to c', denoted $\mathrm{dist}(c, c')$ is exactly the distance between c and c' in the hypercube, equal to the number of positions on which the corresponding binary strings differ.

In our construction, we compare the cost of an online algorithm to the cost of an algorithm provided by the adversary. Since Opt's cost can be only lower than the latter, such approach yields a lower bound on the performance of the online algorithm.

For each word w, there is exactly one word at distance k, which we call its *antipode* and denote $\bar{w}$. Clearly, $\bar{w}_i = 1 - w_i$ for all i. Whenever we say that an adversary *penalizes* configuration c, it issues a request at $\bar{c}$. An algorithm that has servers at configuration c needs to move at least one of them. On the other hand, any algorithm with servers at configuration $c' \neq c$ need not move its servers; this property will be heavily used by an adversary's algorithm.

5.4.1 A warm-up: deterministic algorithms

To illustrate our general framework, we start with a description of an $\Omega(2^k/k)$ lower bound that holds for any deterministic algorithm Det [BEKN18]. (A more refined analysis yields

a better lower bound of $2^k - 1$ [KT04].) The adversarial strategy consists of a sequence of independent identical phases. Whenever DET is in some configuration, the adversary penalizes this configuration. The phase ends when $2^k - 1$ *different* configurations have been penalized. This means that DET was forced to move at least $2^k - 1$ times, at a total cost of at least $2^k - 1$. In the same phase, the adversary's algorithm makes only a single move (of cost at most k) at the very beginning of the phase: it moves to the only configuration that is not going to be penalized in the current phase. This shows that the DET-to-OPT ratio in each phase is at least $(2^k - 1)/k$.

5.4.2 Extension to randomized algorithms

Adopting the idea above to a randomized algorithm RAND is not straightforward. Again, we focus on a single phase and the adversary wants to leave (at least) one configuration non-penalized in this phase. However, now the adversary only knows RAND's probability distribution μ over configurations and not its actual configuration. (At any time, for any configuration c, $\mu(c)$ is the probability that RAND's configuration is equal to c.) We focus on a greedy adversarial strategy that always penalizes the configuration with maximum probability. However, arguing that RAND incurs a significant cost is not as easy as for DET.

First, the support of μ can also include configurations that have been already penalized by the adversary in the current phase. This is but a nuisance, easily overcome by penalizing such configurations repeatedly if RAND keeps using them, until their probability becomes negligible. Therefore, in this informal discussion, we assume that once a configuration c is penalized in a given phase, $\mu(c)$ remains equal to zero.

Second, a straightforward analysis of the greedy adversarial strategy fails to give a non-trivial lower bound. Assume that $i \in \{0, \ldots, 2^k - 2\}$ configurations have already been penalized in a given phase, and the support of μ contains the remaining $2^k - i$ configurations. The maximum probability assigned to one of these configurations is at least $1/(2^k - i)$. When such configuration is penalized, RAND needs to move at least one server with probability at least $1/(2^k - i)$. With such bounds, we would then prove that the algorithm's expected cost is at least $\sum_{i=0}^{2^k-2} 1/(2^k - i) = \Omega(\log 2^k) = \Omega(k)$. Since we bounded the adversary's cost per phase by k, this gives only a constant lower bound.

What we failed to account is that the actual distance traveled by RAND in a single step is either larger than 1 or RAND would not be able to maintain a uniform distribution over

non-penalized configurations. However, actually exploiting this property seems quite complex, and therefore we modify the adversarial strategy instead.

The crux of our actual construction is choosing a subset Q of the configurations, such that Q is sufficiently large (we still have $\log(|Q|) = \Omega(k)$), but the minimum distance between any two points of Q is $\Omega(k)$. Initially, the adversary forces the support of μ to be contained in Q. Afterwards, the adversarial strategy is almost as described above, but reduced to set Q only. This way, in each step the support of μ is a set $S \subseteq Q$, and the adversary forces RAND to move with probability at least $1/|S|$ *over a distance at least* $\Omega(k)$, which is the extra $\Theta(k)$ factor. We begin by proving the existence of such a set Q for sufficiently large k. The proof is standard (see, e.g., Chapter 17 of [Juk11]); we give it below for completeness.

Lemma 5.5. *For any $k \geq 16$, there exists a set $Q \subseteq \{0,1\}^k$ of binary words of length k, satisfying the following two properties:*

size property: $|Q| \geq 2^{k/2}/k$,

distance property: $\mathit{dist}(v,w) \geq k/16$ *for any $v, w \in Q$.*

Proof. Let $\ell = \lfloor k/16 \rfloor \geq k/32$. For any word q, we define its ℓ-neighborhood $B_\ell(q) = \{w : \mathsf{dist}(q,w) \leq \ell\}$.

We construct set Q greedily. We maintain set Q and set $\Gamma(Q) = \bigcup_{q \in Q} B_\ell(q)$. We start with $Q = \emptyset$ (and thus with $\Gamma(Q) = \emptyset$). In each step, we extend Q with an arbitrary word $w \in \{0,1\}^k \setminus \Gamma(Q)$ and update $\Gamma(Q)$ accordingly. We proceed until set $\Gamma(Q)$ contains all possible length-k words. Clearly, the resulting set Q satisfies the distance property.

It remains to show that $|Q| \geq 2^{k/2}/k$. For a word q, the size of $B_\ell(q)$ is

$$\begin{aligned} |B_\ell(q)| &= \sum_{i=0}^{\lfloor k/16 \rfloor} \binom{k}{i} < k \cdot \binom{k}{\lfloor k/16 \rfloor} \leq k \cdot \left(\frac{k \cdot \mathrm{e}}{\lfloor k/16 \rfloor}\right)^{\lfloor k/16 \rfloor} \\ &\leq k \cdot \left(\frac{k \cdot \mathrm{e}}{k/32}\right)^{k/16} = k \cdot \left((32 \cdot \mathrm{e})^{1/8}\right)^{k/2} < k \cdot 2^{k/2}. \end{aligned}$$

That is, in a single step, $\Gamma(Q)$ increases by at most $k \cdot 2^{k/2}$ elements. Therefore, the process continues for at least $2^k/(k \cdot 2^{k/2}) = 2^{k/2}/k$ steps, and thus the size of Q is at least $2^{k/2}/k$. □

Theorem 5.3. *The competitive ratio of every (randomized) online algorithm solving the generalized k-server problem in uniform metrics is at least $\Omega(k)$.*

Proof. In the following we assume that $k \geq 16$, otherwise the theorem follows trivially. We fix any randomized online algorithm RAND. The lower bound strategy consists of a sequence of

independent phases. Requests of each phase can be (optimally) serviced with cost at most k and we show that RAND's expected cost for a single phase is $\Omega(k^2)$, i.e., the ratio between these costs is $\Omega(k)$. As the adversary may present an arbitrary number of phases to the algorithm, this shows that the competitive ratio of RAND is $\Omega(k)$, i.e., by making the cost of RAND arbitrarily high, the additive constant in the definition of the competitive ratio (cf. Section 5.3.1) becomes negligible.

As in our informal introduction, $\mu(c)$ denotes the probability that RAND has its servers in configuration c (at time specified in the context). We extend the notion μ to sets, i.e., $\mu(X) = \sum_{c \in X} \mu(x)$ where X is a set of configurations. We denote the complement of X (to $\prod_{i=1}^{k} M_i$) by X^C. We use $\varepsilon = 2^{-(2k+2)}$ throughout the proof.

To make the description concise, we define an auxiliary routine CONFINE(X) for the adversary (for some configuration set X). In this routine, the adversary repeatedly checks whether there exists a configuration $c \notin X$, such that $\mu(x) > \varepsilon$. In such case, it penalizes c; if no such configuration exists, the routine terminates. We may assume that the procedure always terminates after finite number of steps, as otherwise RAND's competitive ratio would be unbounded. (RAND pays at least ε in each step of the routine while an adversary's algorithm may move its servers to any configuration from set X, and from that time service all requests of CONFINE(X) with no cost.)

The adversarial strategy for a single phase is as follows. First, it constructs Q_1 as the configuration set fulfilling the properties of Lemma 5.5; let m denote its cardinality. The phase consists then of m executions of CONFINE routine: CONFINE(Q_1), CONFINE(Q_2), ..., CONFINE(Q_m). For $i \in \{2, \ldots, m\}$, set Q_i is defined in the following way. The adversary observes RAND's distribution μ right after routine CONFINE(Q_{i-1}) terminates; at this point this distribution is denoted μ_{i-1}. Then, the adversary picks configuration c_{i-1} to be the element of Q_{i-1} that maximizes the probability μ_{i-1}, and sets $Q_i = Q_{i-1} \setminus \{c_{i-1}\}$.

We begin by describing the way that the adversary services the requests. Observe that set Q_m contains a single configuration, henceforth denoted c^*. The configuration c^* is contained in all sets $Q_1, \ldots, Q_m$, and thus c^* is never penalized in the current phase. Hence, by moving to c^* at the beginning of the phase, which costs at most k, and remaining there till the phase ends, the adversary's algorithm services all phase requests at no further cost.

It remains to lower-bound the cost of RAND. CONFINE(Q_1) may incur no cost; its sole goal is to confine the support of μ to Q_1. Now, we fix any $i \in \{2, \ldots, m\}$ and estimate the cost incurred by CONFINE(Q_i). Recall that the probability distribution right before CONFINE(Q_i)

starts (and right after $\textsc{Confine}(Q_{i-1})$ terminates) is denoted μ_{i-1} and the distribution right after $\textsc{Confine}(Q_i)$ terminates is denoted μ_i.

During $\textsc{Confine}(Q_i)$ a probability mass $\mu_{i-1}(c_{i-1})$, is moved from c_{i-1} to nodes of set Q_i (recall that $Q_i \uplus \{c_{i-1}\} = Q_{i-1}$). Some negligible amounts (at most $\mu_i(Q_i^C)$) of this probability may however remain outside of Q_i after $\textsc{Confine}(Q_i)$ terminates. That is, $\textsc{Rand}$ moves at least the probability mass of $\mu_{i-1}(c_{i-1}) - \mu_i(Q_i^C)$ from configuration c_{i-1} to configurations from Q_i (i.e., along a distance of at least $\mathsf{dist}(c_{i-1}, Q_i)$), Therefore, its expected cost due to $\textsc{Confine}(Q_i)$ is at least $(\mu_{i-1}(c_{i-1}) - \mu_i(Q_i^C)) \cdot \mathsf{dist}(c_{i-1}, Q_i)$.

First, using the properties of $\textsc{Confine}(Q_{i-1})$ and the definition of c_{i-1}, we obtain

$$\mu_{i-1}(c_{i-1}) \geq \frac{\mu_{i-1}(Q_{i-1})}{|Q_{i-1}|} = \frac{1 - \mu_{i-1}(Q_{i-1}^C)}{|Q_{i-1}|} \geq \frac{1 - |Q_{i-1}^C| \cdot \varepsilon}{|Q_{i-1}|} > \frac{1 - 2^{-(k+2)}}{|Q_{i-1}|}\,. \tag{5.3}$$

Second, using the properties of $\textsc{Confine}(Q_i)$ yields

$$\mu_i(Q_i^C) \leq |Q_i^C| \cdot \varepsilon < 2^{-(k+2)} = \frac{2^{-2}}{2^k} < \frac{2^{-2}}{|Q_{i-1}|}\,. \tag{5.4}$$

Using (5.3) and (5.4), we bound the expected cost of $\textsc{Rand}$ due to routine $\textsc{Confine}(Q_i)$ as

$$\begin{aligned}
\mathbf{E}[\textsc{Rand}(\textsc{Confine}(Q_i))] &\geq \left(\mu_{i-1}(c_{i-1}) - \mu_i(Q_i^C)\right) \cdot \mathsf{dist}(c_{i-1}, Q_i) \\
&\geq \left(\frac{1 - 2^{-(k+2)}}{|Q_i|} - \frac{2^{-2}}{|Q_i|}\right) \cdot \frac{k}{16} \geq \frac{1}{2 \cdot |Q_i|} \cdot \frac{k}{16} \\
&= k/(32 \cdot (m - i + 1))
\end{aligned} \tag{5.5}$$

The second inequality above follows as all configurations from $\{c_{i-1}\} \uplus Q_i$ are distinct elements of Q_1, and hence their mutual distance is at least $k/16$ by the distance property of Q_1 (cf. Lemma 5.5). By summing (5.5) over $i \in \{2, \ldots, m\}$, we obtain that the total cost of $\textsc{Rand}$ in a single phase is $\mathbf{E}[\textsc{Rand}] \geq \sum_{i=2}^{m} \mathbf{E}[\textsc{Rand}(\textsc{Confine}(Q_i))] \geq \frac{k}{32} \cdot \sum_{i=2}^{m} \frac{1}{m-i+1} = \Omega(k \cdot \log m) = \Omega(k^2)$. The last equality holds as $m \geq 2^{k/2}/k$ by the size property of Q_1. (cf. Lemma 5.5). □

5.5 Final remarks

In Chapter 5, we presented an abstract Hydra game whose solution we applied to create an algorithm for the generalized k-server problem. Any improvement of our $\textsc{Herc}$ strategy for the Hydra game would yield an improvement for the generalized k-server problem. However, we may show that on a wide class of trees (that includes factorial trees used in our reduction), $\textsc{Herc}$ is optimal up to a constant factor. Thus, further improving our upper bound of $O(k^2 \log k)$ for the generalized k-server problem will require another approach.

A lower bound for the cost of any randomized strategy for the Hydra game is essentially the same as our single-phase construction from Section 5.4.2 for the generalized k-server problem. That is, the adversary fixes a subset Q of tree leaves, makes only nodes of Q alive (this forces the algorithm to be inside set Q), and then iteratively kills nodes of Q where the algorithm is most likely to be. As in the proof from Section 5.4.2, such adversarial strategy incurs the cost of $\Omega(\mathsf{mindist}(Q) \cdot \log |Q|)$, where $\mathsf{mindist}(Q) = \min_{u \neq v \in Q} \mathsf{dist}(u, v)$.

The construction of appropriate Q for a tree T of depth $k = h_T$ (be either the k-factorial tree or the complete k-ary tree) is as follows. Let Z be the set of all nodes of T at level $\lfloor k/2 \rfloor$; for such trees, $\log |Z| = \Omega(\log L_T)$. Let Q consist of $|Z|$ leaves of the tree, one per node of Z chosen arbitrarily from its subtree. Then, $\mathsf{mindist}(Q) = \Omega(h_T)$ and $\log |Q| = \Omega(\log L_T)$, and thus the resulting lower bound $\Omega(h_T \cdot \log L_T)$ on the cost asymptotically matches the performance of HERC from Theorem 5.1.

Chapter 6

Afterword

In this chapter, we discuss possible further research directions on the problems studied in this dissertation.

Non-metric facility location. In Chapter 2, we presented a deterministic solution to the non-metric facility location problem, whose performance nearly matches that of the best randomized one. By clustering facilities, we encoded dependencies between facilities and clients, which allowed us later to apply the rounding scheme to facilities only, neglecting the actual active clients. It would be however interesting and useful to have an online deterministic rounding routine able to handle such dependencies internally (e.g., by creating a pessimistic estimator that can be computed and handled in an online manner), as it is the case for the set cover problem or throughput-competitive virtual circuit routing [BN09].

A natural research direction would be extending our distance clustering techniques to other network design problems for which only randomized algorithms existed so far, e.g., online multicast problems on trees [AAA+06], online group Steiner problem on trees [AAA+06], or variants of the facility location problem that are used as building blocks for solutions to other node-weighted Steiner problems [HLP14, HLP17]. (For these problems there are no known direct reductions to the set cover problem).

Matching with delays. In Chapter 3, we showed a deterministic algorithm Alg for the problem of matching with delays whose competitive ratio is $O(m^{\log_2 5.5})$ for inputs, with $2m$ requested elements. This result was improved by Bienkowski et al. [BKLS18], who presented a $O(m)$-competitive dual-fitting algorithm, and afterwards by Azar and Fanani [AF18], whose greedy algorithm attains competitive ratio of $O(m^{\log_2 1.5})$. The last value (as we ar-

gued in Section 3.4) is a lower bound on a competitive ratio of a class of natural greedy algorithms. The currently best lower bound (holding even for randomized solutions) is $\Omega(\log n / \log\log n)$ [AAC+17], where n is the size of the input graph.

Steiner tree leasing. In Chapter 4, we showed that the technique of analyzing greedy algorithms using HSTs can be also applied to the leasing variant of the online Steiner tree problem. A natural research direction is to employ it for other leasing variants of graph problems, such as Steiner forest or facility location.

Closing the gap between the current upper and lower bounds for the deterministic algorithms solving the Steiner tree leasing problem ($O(L \cdot \log k)$ and $\Omega(L + \log k)$, respectively, for instances with k requested elements and L different lease types) is an intriguing open problem. In particular, it seems that improving the competitive ratio requires a very careful interplay between path-choosing and lease-upgrade routines. We remark that analogous gaps exist also for randomized algorithms for the Steiner tree leasing problem [Mey05] and for a leasing variant of the facility location problem [NW13].

Generalized k-server problem. In Chapter 5, we presented an abstract Hydra game whose solution we applied to create an algorithm for the generalized k-server problem. Since our algorithm Herc for Hydra game is optimal up to a constant factor, further improving our upper bound of $O(k^2 \log k)$ for the generalized k-server problem is likely to require a completely different approach. In our phase-based analysis, we used a simple lower bound on the cost of the optimal solution. A crucial for improving the competitive ratio for the generalized k-server problem is a better understanding of the optimal schedule. In particular, it would be useful to characterize situations when it is forced to move more than one of its servers.

A good starting point for further research is the generalized k-server problem problem with each metric space containing exactly 2 points. (This variant is equivalent to metrical task system on the k-dimensional hypercube.) It is not difficult to construct a phase-based $O(k^2)$-competitive algorithm for such inputs and the lower bound of $\Omega(k)$ on competitiveness of any algorithm holds.

Finally, it would be particularly interesting to apply projection techniques developed recently for k-server [BCL+18, Lee18] and metrical task systems [BCLL19] to the generalized k-server problem problem.

Bibliography

[AA92] Noga Alon and Yossi Azar. On-line Steiner trees in the Euclidean plane. In *Proc. 8th ACM Symp. on Computational Geometry (SoCG)*, pages 337–343, 1992.

[AA97] Baruch Awerbuch and Yossi Azar. Buy-at-bulk network design. In *Proc. 38th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 542–547, 1997.

[AAA+06] Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Naor. A general approach to online network optimization problems. *ACM Trans. Algorithms*, 2(4):640–660, 2006.

[AAA+09] Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Naor. The online set cover problem. *SIAM Journal on Computing*, 39(2):361–370, 2009.

[AAB04] Baruch Awerbuch, Yossi Azar, and Yair Bartal. On-line generalized Steiner problem. *Theoretical Computer Science*, 324(2–3):313–324, 2004.

[AAC+17] Itai Ashlagi, Yossi Azar, Moses Charikar, Ashish Chiplunkar, Ofir Geri, Haim Kaplan, Rahul M. Makhijani, Yuyi Wang, and Roger Wattenhofer. Min-cost bipartite perfect matching with delays. In *Proc. 20th Int. Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX)*, pages 1:1–1:20, 2017.

[AAP93] Baruch Awerbuch, Yossi Azar, and Serge A. Plotkin. Throughput-competitive on-line routing. In *Proc. 34th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 32–40, 1993.

[AB05] Susanne Albers and Helge Bals. Dynamic TCP acknowledgment: Penalizing long delays. *SIAM Journal on Discrete Mathematics*, 19(4):938–951, 2005.

[ABF93] Baruch Awerbuch, Yair Bartal, and Amos Fiat. Competitive distributed file allocation. In *Proc. 25th ACM Symp. on Theory of Computing (STOC)*, pages 164–173, 1993.

[ABF98] Baruch Awerbuch, Yair Bartal, and Amos Fiat. Distributed paging for general networks. *Journal of Algorithms*, 28(1):67–104, 1998.

[ABM16] Karen Aardal, Jaroslaw Byrka, and Mohammad Mahdian. Facility location. In *Encyclopedia of Algorithms*, pages 717–724. Springer, 2016.

[ABN+14] Antonios Antoniadis, Neal Barcelo, Michael Nugent, Kirk Pruhs, and Michele Scquizzato. A o(n)-competitive deterministic algorithm for online matching on a line. In *Proc. 12th Workshop on Approximation and Online Algorithms (WAOA)*, pages 11–22, 2014.

[ABUH04] Aris Anagnostopoulos, Russell Bent, Eli Upfal, and Pascal Van Hentenryck. A simple and deterministic competitive algorithm for online facility location. *Inf. Comput.*, 194(2):175–202, 2004.

[ACER19] Anna Adamaszek, Artur Czumaj, Matthias Englert, and Harald Räcke. An $O(\log k)$-competitive algorithm for generalized caching. *ACM Transactions on Algorithms*, 15(1):6:1–6:18, 2019.

[ACK17] Yossi Azar, Ashish Chiplunkar, and Haim Kaplan. Polylogarithmic bounds on the competitiveness of min-cost perfect matching with delays. In *Proc. 28th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 1051–1061, 2017.

[ACKT20] Yossi Azar, Ashish Chiplunkar, Shay Kutten, and Noam Touitou. Set cover with delay - clairvoyance is not required. In *Proc. 28th European Symp. on Algorithms (ESA)*, pages 8:1–8:21, 2020.

[AF18] Yossi Azar and Amit Jacob Fanani. Deterministic min-cost matching with delays. In *Proc. 16th Workshop on Approximation and Online Algorithms (WAOA)*, pages 21–35, 2018.

[AFG+09] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the clouds: A Berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, 2009.

[AG07] Barbara M. Anthony and Anupam Gupta. Infrastructure leasing problems. In *Proc. 12th Int. Conf. on Integer Programming and Combinatorial Optimization (IPCO)*, pages 424–438, 2007.

[AGGP17] Yossi Azar, Arun Ganesh, Rong Ge, and Debmalya Panigrahi. Online service with delay. In *Proc. 49th ACM Symp. on Theory of Computing (STOC)*, pages 551–563, 2017.

[AHK12] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing Systems*, 8(1):121–164, 2012.

[AJF18] Yossi Azar and Amit Jacob-Fanani. Deterministic min-cost matching with delays. In *Proc. 16th Workshop on Approximation and Online Algorithms (WAOA)*, 2018.

[AK98] Susanne Albers and Hisashi Koga. New on-line algorithms for the page replication problem. *Journal of Algorithms*, 27(1):75–96, 1998.

[AKM+15] Sebastian Abshoff, Peter Kling, Christine Markarian, Friedhelm Meyer auf der Heide, and Peter Pietrzyk. Towards the price of leasing online. *Journal of Combinatorial Optimization*, pages 1–20, 2015.

[AMM14] Sebastian Abshoff, Christine Markarian, and Friedhelm Meyer auf der Heide. Randomized online algorithms for set cover leasing problems. In *Proc. 8th Int. Conf. on Combinatorial Optimization and Applications (COCOA)*, pages 25–34, 2014.

[Ang09] Spyros Angelopoulos. On the competitiveness of the online asymmetric and Euclidean Steiner tree problems. In *Proc. 7th Workshop on Approximation and Online Algorithms (WAOA)*, pages 1–12, 2009.

[ASvZ13] Tetske Avontuur, Pieter Spronck, and Menno van Zaanen. Player skill modeling in Starcraft II. In *Proc. 9th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE-13*, 2013.

[AT20] Yossi Azar and Noam Touitou. Beyond tree embeddings - a deterministic framework for network design with deadlines or delay. In *Proc. 61st IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 1368–1379, 2020.

[BA10] Jaroslaw Byrka and Karen Aardal. An optimal bifactor approximation algorithm for the metric uncapacitated facility location problem. *SIAM Journal on Computing*, 39(6):2212–2231, 2010.

[Bar96] Yair Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *Proc. 37th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 184–193, 1996.

[Bar98] Yair Bartal. On approximating arbitrary metrics by tree metrics. In *Proc. 30th ACM Symp. on Theory of Computing (STOC)*, pages 161–168, 1998.

[BBC+13] Marcin Bienkowski, Jaroslaw Byrka, Marek Chrobak, Łukasz Jeż, Jiři Sgall, and Grzegorz Stachowiak. Online control message aggregation in chain networks. In *Proc. 13th Algorithms and Data Structures Symposium (WADS)*, pages 133–145, 2013.

[BBGN14] Nikhil Bansal, Niv Buchbinder, Anupam Gupta, and Joseph Naor. A randomized $O(\log^2 k)$-competitive algorithm for metric bipartite matching. *Algorithmica*, 68(2):390–403, 2014.

[BBK+94] Shai Ben-David, Allan Borodin, Richard M. Karp, Gabor Tardos, and Avi Wigderson. On the power of randomization in online algorithms. *Algorithmica*, 11(1):2–14, 1994.

[BBMN15] Nikhil Bansal, Niv Buchbinder, Aleksander Mądry, and Joseph Naor. A polylogarithmic-competitive algorithm for the k-server problem. *Journal of the ACM*, 62(5):40:1–40:49, 2015.

[BBN12] Nikhil Bansal, Niv Buchbinder, and Joseph Naor. Randomized competitive algorithms for generalized caching. *SIAM Journal on Computing*, 41(2):391–414, 2012.

[BCL+18] Sébastien Bubeck, Michael B. Cohen, Yin Tat Lee, James R. Lee, and Aleksander Madry. k-server via multiscale entropic regularization. In *Proc. 50th ACM Symp. on Theory of Computing (STOC)*, pages 3–16, 2018.

[BCLL19] Sébastien Bubeck, Michael B. Cohen, James R. Lee, and Yin Tat Lee. Metrical task systems on trees via mirror descent and unfair gluing. In *Proc. 30th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 89–97, 2019.

[BE98] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[BEK17] Nikhil Bansal, Marek Eliás, and Grigorios Koumoutsos. Weighted k-server bounds via combinatorial dichotomies. In *Proc. 58th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 493–504. IEEE Computer Society, 2017.

[BEKN18] Nikhil Bansal, Marek Eliás, Grigorios Koumoutsos, and Jesper Nederlof. Competitive algorithms for generalized k-server in uniform metrics. In *Proc. 29th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 992–1001, 2018.

[BFR95] Yair Bartal, Amos Fiat, and Yuval Rabani. Competitive algorithms for distributed data management. *Journal of Computer and System Sciences*, 51(3):341–358, 1995.

[BFS21] Marcin Bienkowski, Björn Feldkord, and Paweł Schmidt. A nearly optimal deterministic online algorithm for non-metric facility location. In *Proc. 38th Symp. on Theoretical Aspects of Computer Science (STACS)*, 2021.

[BGRS10] Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. An improved lp-based approximation for steiner tree. In *Proc. 42nd ACM Symp. on Theory of Computing (STOC)*, pages 583–592, 2010.

[BH21] Nadia Burkart and Marco Huber. A survey on the explainability of supervised machine learning. *Journal of Artificial Intelligence Research*, 70, 2021.

[BJS19] Marcin Bienkowski, Łukasz Jeż, and Paweł Schmidt. Slaying hydrae: Improved bounds for generalized k-server in uniform metrics. In *Proc. 30th Int. Symp. on Algorithms and Computation (ISAAC)*, pages 14:1–14:14, 2019.

[BKLS18] Marcin Bienkowski, Artur Kraska, Hsiang-Hsuan Liu, and Paweł Schmidt. A primal-dual online deterministic algorithm for matching with delays. In *Proc. 16th Workshop on Approximation and Online Algorithms (WAOA)*, pages 51–68, 2018.

[BKS17a] Marcin Bienkowski, Artur Kraska, and Paweł Schmidt. A deterministic algorithm for online steiner tree leasing. In *Proc. 15th Algorithms and Data Structures Symposium (WADS)*, pages 169–180, 2017.

[BKS17b] Marcin Bienkowski, Artur Kraska, and Paweł Schmidt. A match in time saves nine: Deterministic online matching with delays. In *Proc. 15th Workshop on Approximation and Online Algorithms (WAOA)*, pages 132–146, 2017.

[BKS18] Marcin Bienkowski, Artur Kraska, and Paweł Schmidt. Online service with delay on a line. In *Proc. 25th Int. Colloq. on Structural Information and Communication Complexity (SIROCCO)*, pages 237–248, 2018.

[BLMN03] Yair Bartal, Nathan Linial, Manor Mendel, and Assaf Naor. On metric Ramsey-type phenomena. In *Proc. 35th ACM Symp. on Theory of Computing (STOC)*, pages 463–472, 2003.

[BLS92] Alan Borodin, Nati Linial, and Michael E. Saks. An optimal on-line algorithm for metrical task system. *Journal of the ACM*, 39(4):745–763, 1992.

[BN09] Niv Buchbinder and Joseph Naor. Online primal-dual algorithms for covering and packing. *Mathematics of Operations Research*, 34(2):270–286, 2009.

[BP89] Marshall W. Bern and Paul E. Plassmann. The steiner problem with edge lengths 1 and 2. *Information Processing Letters*, 32(4):171–176, 1989.

[BS89] David L. Black and Daniel D. Sleator. Competitive algorithms for replication and migration problems. Technical Report CMU-CS-89-201, Department of Computer Science, Carnegie-Mellon University, 1989.

[CB10] N.M. Mosharaf Kabir Chowdhury and Raouf Boutaba. A survey of network virtualization. *Computer Networks*, 54(5):862–876, 2010.

[CFK20] Dimitris Christou, Dimitris Fotakis, and Grigorios Koumoutsos. Memoryless algorithms for the generalized k-server problem on uniform metrics. *CoRR*, abs/2007.08669, 2020.

[CG05] Moses Charikar and Sudipto Guha. Improved combinatorial algorithms for facility location problems. *SIAM Journal on Computing*, 34(4):803–824, 2005.

[Chr03] Marek Chrobak. SIGACT news online algorithms column 1. *SIGACT News*, 34(4):68–77, 2003.

[CKPV91] Marek Chrobak, Howard J. Karloff, Thomas H. Payne, and Sundar Vishwanathan. New results on server problems. *SIAM Journal on Discrete Mathematics*, 4(2):172–181, 1991.

[CL91] Marek Chrobak and Lawrence L. Larmore. An optimal on-line algorithm for k-servers on trees. *SIAM Journal on Computing*, 20(1):144–148, 1991.

[CS03] Fabián A. Chudak and David B. Shmoys. Improved approximation algorithms for the uncapacitated facility location problem. *SIAM Journal on Computing*, 33(1):1–25, 2003.

[CS04] Marek Chrobak and Jirí Sgall. The weighted 2-server problem. *Theoretical Computer Science*, 324(2-3):289–312, 2004.

[CSEN17] Zhengxing Chen, Yizhou Sun, Magy Seif El-Nasr, and Truong-Huy D. Nguyen. Player skill decomposition in multiplayer online battle arenas. 2017.

[CV13] Ashish Chiplunkar and Sundar Vishwanathan. On randomized memoryless algorithms for the weighted k-server problem. In *Proc. 54th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 11–19, 2013.

[DCT+12] Olivier Delalleau, Emile Contal, Eric Thibodeau-Laufer, Raul Chandias Ferrari, Yoshua Bengio, and Frank Zhang. Beyond skill rating: Advanced matchmaking in Ghost Recon Online. *IEEE Trans. Comput. Intellig. and AI in Games*, 4(3):167–177, 2012.

[DGS01] Daniel R. Dooly, Sally A. Goldman, and Stephen D. Scott. On-line analysis of the TCP acknowledgment delay problem. *Journal of the ACM*, 48(2):243–273, 2001.

[EKW16] Yuval Emek, Shay Kutten, and Roger Wattenhofer. Online matching: haste makes waste! In *Proc. 48th ACM Symp. on Theory of Computing (STOC)*, pages 333–344, 2016.

[Elo78] Arpad E. Elo. *The rating of chessplayers, past and present*. Arco Publishing, 1978.

[ESW17] Yuval Emek, Yaacov Shapiro, and Yuyi Wang. Minimum cost perfect matching with delays for two sources. In *Proc. 10th Int. Conf. on Algorithms and Complexity (CIAC)*, pages 209–221, 2017.

[Fei98] Uriel Feige. A threshold of ln n for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.

[FGS04] Rudolf Fleischer, Włodzimierz Głazek, and Steve S. Seiden. New results for online page replication. *Theoretical Computer Science*, 324(2–3):219–251, 2004.

[FHK05] Bernhard Fuchs, Winfried Hochstättler, and Walter Kern. Online matching on a line. *Theoretical Computer Science*, 332(1–3):251–264, 2005.

[FKL+91] Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel D. Sleator, and Neal E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, 1991.

[Fot07] Dimitris Fotakis. A primal-dual algorithm for online non-uniform facility location. *Journal of Discrete Algorithms*, 5(1):141–148, 2007.

[Fot08] Dimitris Fotakis. On the competitive ratio for online facility location. *Algorithmica*, 50(1):1–57, 2008.

[FR94] Amos Fiat and Moty Ricklin. Competitive algorithms for the weighted server problem. *Theoretical Computer Science*, 130(1):85–99, 1994.

[FRT04] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *Journal of Computer and System Sciences*, 69(3):485–497, 2004.

[GK98] Sudipto Guha and Samir Khuller. Greedy strikes back: Improved facility location algorithms. In *Proc. 9th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 649–657, 1998.

[GK11] Anupam Gupta and Jochen Könemann. Approximation algorithms for network design: A survey. *Surveys in Operations Research and Management Science*, 16(1):3–20, 2011.

[GL12] Anupam Gupta and Kevin Lewi. The online metric matching problem for doubling metrics. In *Proc. 39th Int. Colloq. on Automata, Languages and Programming (ICALP)*, pages 424–435, 2012.

[HLP14] MohammadTaghi Hajiaghayi, Vahid Liaghat, and Debmalya Panigrahi. Near-optimal online algorithms for prize-collecting steiner problems. In *Proc. 41st Int. Colloq. on Automata, Languages and Programming (ICALP)*, pages 576–587, 2014.

[HLP17] MohammadTaghi Hajiaghayi, Vahid Liaghat, and Debmalya Panigrahi. Online node-weighted steiner forest and extensions via disk paintings. *SIAM Journal on Computing*, 46(3):911–935, 2017.

[Hoc82] Dorit S. Hochbaum. Heuristics for the fixed cost median problem. *Mathematical Programming*, 22(1):148–162, 1982.

[IW91] Makoto Imase and Bernard M. Waxman. Dynamic Steiner tree problem. *SIAM Journal on Discrete Mathematics*, 4(3):369–384, 1991.

[JMM+03] Kamal Jain, Mohammad Mahdian, Evangelos Markakis, Amin Saberi, and Vijay V. Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP. *Journal of the ACM*, 50(6):795–824, 2003.

[JMS02] Kamal Jain, Mohammad Mahdian, and Amin Saberi. A new greedy approach for facility location problems. In *Proc. 34th ACM Symp. on Theory of Computing (STOC)*, pages 731–740, 2002.

[Juk11] Stasys Jukna. *Extremal Combinatorics - With Applications in Computer Science*. Springer, 2011.

[KKR03] Anna R. Karlin, Claire Kenyon, and Dana Randall. Dynamic TCP acknowledgement and other stories about e/(e - 1). *Algorithmica*, 36(3):209–224, 2003.

[KMMO94] Anna R. Karlin, Mark S. Manasse, Lyle A. McGeoch, and Susan Owicki. Competitive randomized algorithms for non-uniform problems. *Algorithmica*, 11(6):542–571, 1994.

[KMRS88] Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel D. Sleator. Competitive snoopy caching. *Algorithmica*, 3(1):77–119, 1988.

[KMV94] Samir Khuller, Stephen G. Mitchell, and Vijay V. Vazirani. On-line algorithms for weighted bipartite matching and stable marriages. *Theoretical Computer Science*, 127(2):255–267, 1994.

[KN03] Elias Koutsoupias and Akash Nanavati. The online matching problem on a line. In *Proc. 1st Workshop on Approximation and Online Algorithms (WAOA)*, pages 179–191, 2003.

[KNR02] Sanjeev Khanna, Joseph Naor, and Danny Raz. Control message aggregation in group communication protocols. In *Proc. 29th Int. Colloq. on Automata, Languages and Programming (ICALP)*, pages 135–146, 2002.

[Kor04] Simon Korman. On the use of randomization in the online set cover problem. Master's thesis, The Weizmann Institute of Science, 2004.

[Kou09] Elias Koutsoupias. The k-server problem. *Computer Science Review*, 3(2):105–118, 2009.

[KP93] Bala Kalyanasundaram and Kirk Pruhs. Online weighted matching. *Journal of Algorithms*, 14(3):478–488, 1993.

[KP95] Elias Koutsoupias and Christos H. Papadimitriou. On the k-server conjecture. *Journal of the ACM*, 42(5):971–983, 1995.

[KPR00] Madhukar R. Korupolu, C. Greg Plaxton, and Rajmohan Rajaraman. Analysis of a local search heuristic for facility location problems. *Journal of Algorithms*, 37(1):146–188, 2000.

[KT90] Antoon Kolen and Arie Tamir. Covering problems. In P.B. Mirchandani and R.L. Francis, editors, *Discrete Location Theory*, Wiley Series in Discrete Mathematics and Optimization. Wiley, 1990.

[KT04] Elias Koutsoupias and David Scot Taylor. The CNN problem and other k-server variants. *Theoretical Computer Science*, 324(2-3):347–359, 2004.

[KVV90] Richard M. Karp, Umesh V. Vazirani, and Vijay V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proc. 22nd ACM Symp. on Theory of Computing (STOC)*, pages 352–358, 1990.

[KZ97] Marek Karpinski and Alexander Zelikovsky. New approximation algorithms for the steiner tree problems. *Journal of Combinatorial Optimization*, 1(1):47–65, 1997.

[Lee18] James R. Lee. Fusible HSTs and the randomized k-server conjecture. In *Proc. 59th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 438–449, 2018.

[Li13] Shi Li. A 1.488 approximation algorithm for the uncapacitated facility location problem. In *Proc. 38th Int. Colloq. on Automata, Languages and Programming (ICALP)*, pages 77–88, 2013.

[LRWY99] Carsten Lund, Nick Reingold, Jeffery Westbrook, and Dicky C. K. Yan. Competitive on-line algorithms for distributed data management. *SIAM Journal on Computing*, 28(3):1086–1111, 1999.

[Mat16] Akira Matsubayashi. Non-greedy online Steiner trees on outerplanar graphs. In *Proc. 14th Workshop on Approximation and Online Algorithms (WAOA)*, pages 129–141, 2016.

[Meh13] Aranyak Mehta. Online matching and ad allocation. *Foundations and Trends in Theoretical Computer Science*, 8(4):265–368, 2013.

[Mey01] Adam Meyerson. Online facility location. In *Proc. 42nd IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 426–431, 2001.

[Mey05] Adam Meyerson. The parking permit problem. In *Proc. 46th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 274–284, 2005.

[MMS90] Mark S. Manasse, Lyle A. McGeoch, and Daniel D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11(2):208–230, 1990.

[MNP06] Adam Meyerson, Akash Nanavati, and Laura J. Poplawski. Randomized online algorithms for minimum metric bipartite matching. In *Proc. 7th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 954–959, 2006.

[MYZ06] Mohammad Mahdian, Yinyu Ye, and Jiawei Zhang. Approximation algorithms for metric facility location problems. *SIAM Journal on Computing*, 36(2):411–432, 2006.

[NPS11] Joseph Naor, Debmalya Panigrahi, and Mohit Singh. Online node-weighted steiner tree and related problems. In *Proc. 52nd IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 210–219, 2011.

[NW13] Chandrashekhar Nagarajan and David P. Williamson. Offline and online facility leasing. *Discrete Optimization*, 10(4):361–370, 2013.

[PS00] Hans Jürgen Prömel and Angelika Steger. A new approximation algorithm for the steiner tree problem with performance ratio 5/3. *Journal of Algorithms*, 36(1):89–101, 2000.

[PSW10] Yvonne Anne Pignolet, Stefan Schmid, and Roger Wattenhofer. Tight bounds for delay-sensitive aggregation. *Discrete Mathematics & Theoretical Computer Science*, 12(1):39–58, 2010.

[Rag88] Prabhakar Raghavan. Probabilistic construction of deterministic algorithms: Approximating packing integer programs. *Journal of Computer and System Sciences*, 37(2):130–143, 1988.

[RT81] Edward M. Reingold and Robert Endre Tarjan. On a greedy heuristic for complete matching. *SIAM Journal on Computing*, 10(4):676–681, 1981.

[RWS94] Nick Reingold, Jeffery Westbrook, and Daniel Dominic Sleator. Randomized competitive algorithms for the list update problem. *Algorithmica*, 11(1):15–32, 1994.

[RZ05] Gabriel Robins and Alexander Zelikovsky. Tighter bounds for graph steiner tree approximation. *SIAM Journal on Discrete Mathematics*, 19(1):122–134, 2005.

[Sch03] Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Algorithms and combinatorics. Springer, 2003.

[Shm00] David B. Shmoys. Approximation algorithms for facility location problems. In Klaus Jansen and Samir Khuller, editors, *Proc. 3rd Approximation, Randomization,*

and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM), pages 27–33. Springer, 2000.

[Sit14] René Sitters. The generalized work function algorithm is competitive for the generalized 2-server problem. *SIAM Journal on Computing*, 43(1):96–125, 2014.

[SS06] René A. Sitters and Leen Stougie. The generalized two-server problem. *Journal of the ACM*, 53(3):437–458, 2006.

[ST85] Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[STA97] David B. Shmoys, Éva Tardos, and Karen Aardal. Approximation algorithms for facility location problems (extended abstract). In *Proc. 29th ACM Symp. on Theory of Computing (STOC)*, pages 265–274, 1997.

[Umb15] Seeun Umboh. Online network design algorithms via hierarchical decompositions. In *Proc. 26th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 1373–1387, 2015.

[Wes94] Jeffery Westbrook. Randomized algorithms for the multiprocessor page migration. *SIAM Journal on Computing*, 23:951–965, 1994.

[WH16] Weili Wu and Yaochun Huang. Steiner trees. In *Encyclopedia of Algorithms*, pages 2102–2107. 2016.

[WS11] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

[WY95] Jeffery Westbrook and Dicky C. K. Yan. The performance of greedy algorithms for the on-line Steiner tree and related problems. *Mathematical Systems Theory*, 28(5):451–468, 1995.

[You95] Neal E. Young. Randomized rounding without solving the linear program. In *Proc. 6th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 170–178, 1995.

[You02] Neal E. Young. On-line file caching. *Algorithmica*, 33(3):371–383, 2002.

[Zel93] Alexander Zelikovsky. An 11/6-approximation algorithm for the network steiner problem. *Algorithmica*, 9(5):463–470, 1993.