PADERBORN
UNIVERSITY

Online Algorithms for Allocating Heterogeneous Resources

Dissertation

In partial fulfillment of the requirements for the degree of

Doctor rerum naturalium (Dr. rer. nat.)

at the Faculty of Computer Science,
Electrical Engineering and Mathematics
at Paderborn University

submitted by

TILL KNOLLMANN

Reviewers

- Prof. Dr. Friedhelm Meyer auf der Heide,
  Paderborn University
- Prof. Dr. Yann Disser,
  Technical University of Darmstadt
- Prof. Dr. Christian Scheideler,
  Paderborn University

Beauty is the first test:
There is no permanent place in the world for ugly mathematics.

Godfrey Harold Hardy [53, p. 85]

Zusammenfassung

Mit der weltweiten Verbreitung des Cloud Computing sind Cloud-Anbieter täglich mit Fragen konfrontiert wie z.B., wo die virtuellen Server eines Kunden platziert werden sollen und welche Dienste an einem bestimmten Standort angeboten werden sollen. Dienste sollten sich in der Nähe der Kundenanfragen befinden, damit die Netzbelastung gering ist und eine zufriedenstellende Benutzererfahrung geboten wird. Gleichzeitig ist die Änderung der Standorte von Diensten mit Kosten für die Migration/Einrichtung virtueller Maschinen verbunden. Darüber hinaus sind künftige Anfragen von Kunden in der Regel unbekannt, so dass Algorithmen zur kosteneffizienten und dynamischen Verwaltung von Diensten erforderlich sind. Das obige Szenario wurde in der theoretischen Informatik unter dem Begriff *Ressourcenallokation* ausgiebig erforscht. Die klassischen Modelle betrachten jedoch nur eine Art von Dienst, während die Realität heterogen ist, d.h. Cloud-Anbieter verwalten mehrere, *unterschiedliche* Dienste. Daher wird in der folgenden Arbeit der Einfluss heterogener Ressourcen auf die Leistung von Online-Algorithmen für Ressourcenallokationsprobleme untersucht, indem wir *Güter* zum Modellieren von Diensten einführen.

Zunächst erweitern wir das *Page Migration Problem* um verschiedene Güter, die ein Algorithmus bewegen kann. Die gemeinsame Migration von Gütern ist dabei günstiger als eine getrennte. Wir zeigen, dass kein (randomisierter) Algorithmus von kombinierten Bewegungen profitieren kann und, dass ein trivialer Ansatz eine asymptotisch optimale Kompetitivität erreicht.

Zweitens schlagen wir eine Erweiterung des *Facility Location Problems* vor, bei der Facilities eine Menge von Gütern anbieten, die bei der Konstruktion festgelegt werden. Anfragen können mehrere Güter verlangen und müssen von einer Gruppe von Facilities bedient werden, die diese gemeinsam anbieten. Wir konstruieren eine untere Schranke, die die Anzahl der verfügbaren Güter einbezieht, und entwerfen (deterministische und randomisierte) Algorithmen mit begrenzter Kompetitivität, welche der unteren Schranke in vielen Fällen sehr nahe kommt.

Schließlich verallgemeinern wir das *k-Server Problem* durch ein Modell, bei dem jeder Server mehrere Güter anbietet. Jede Anfrage präsentiert eine Menge von Gütern, aus der nur eines für eine Beantwortung benötigt wird. Wir schlagen einen Parameter vor, der Instanzen des klassischen Modells mit dem erweiterten Modell verbindet, und analysieren alle unsere Schranken entsprechend. Bereits bei uniformen Metriken steigt die Kompetitivität an und erfordert nicht-triviale Ansätze. Wir entwerfen Algorithmen mit nahezu optimaler Kompetitivität und zeigen einen unvermeidlichen Trade-off in der Leistung zwischen klassischen k-Server-Instanzen und allgemeinen Instanzen.

Abstract

With the rise of cloud computing worldwide, cloud providers are confronted with questions such as "Where to place virtual servers of a client?" and "Which services to provide at a location?" every day. Services should be close to clients' requests for a low network load and a satisfying experience for the clients. Simultaneously, changing the locations of services implies costs for migrating and deploying virtual machines. Further, future requests of clients are usually unknown, which requires algorithms to manage services cost-efficiently and dynamically. The above scenario has extensively been researched in theoretical computer science under the term *resource allocation*. However, classical models consider only one kind of service, while the reality is heterogeneous, i.e., cloud providers manage multiple *different* services. Therefore, the following thesis studies the influence of heterogeneous resources on the performance of online algorithms for resource allocation problems by introducing *commodities* modeling services.

First, we extend the *page migration problem* by different commodities that an algorithm can move. Migrating commodities together is beneficial to separate management. We show that no (randomized) algorithm can benefit from combined movements in such a model and that a trivial approach achieves an asymptotically optimal competitive ratio.

Second, we propose an extension of the *facility location problem* where facilities offer a set of commodities determined upon construction. Requests can demand several commodities and must be served by a set of facilities jointly offering them. We construct a lower bound incorporating the number of available commodities and design (deterministic and randomized) algorithms with a bounded competitive ratio that comes close to the lower bound in many cases.

Lastly, we generalize the *k-server problem* to a model where each server offers several commodities. Each request presents a commodity set, of which only one is required for serving. We propose a parameter connecting instances of the classical model to the extended model and analyze all our bounds, respectively. Already on uniform metrics, the competitive ratio rises and requires non-trivial approaches. We design algorithms with a close-to-optimal competitive ratio and show an inevitable trade-off in the performance between classical *k*-server instances and general instances.

Preface

At the beginning of my work in research roughly five years ago, I had little knowledge of online computation and resource allocation problems. However, throughout the years, I discovered a passion for these topics. The concept of formally grasping the price of not knowing the future still fascinates me today. As the author of this thesis, I hope my results provide an interesting contribution and inspire others to push the boundaries of what we know.

The results I present in this thesis would not have been possible without the help of many excellent people. I want to take the opportunity here to express my gratitude. First, I would like to thank my advisor Prof. Dr. Friedhelm Meyer auf der Heide, for the possibility of working in his Algorithms and Complexity research group and for his guidance throughout the years. Further, I thank Prof. Dr. Yann Disser and Prof. Dr. Christian Scheideler for agreeing to review my thesis and joining the board of examiners. In general, I thank all members of the latter.

A big thank you goes to all my co-authors and my colleagues at the HNI. Especially, I thank my good friend, colleague, and co-author Jannik Castenow for always supporting me, for the perfect teamwork, and for the many discussions and conversations on professional and personal matters. I consider myself exceptionally lucky to share such a great friendship with you, and I will never forget your support. In addition, I express my gratitude to Dr. Björn Feldkord and Dr. Manuel Malatyali, with whom doing research was a pleasure. I further thank my former office colleagues, Alexander and Johannes, for the many great times in our office and the numerous advice.

Besides the people around me at work, I am grateful for the support I received in my private life. I deeply thank all the people who accompanied me through the ups and downs I experienced, first and foremost, my significant other, who supported me like no one else to stay on my path. Further, I would like to thank my family and my friends.

Till Knollmann
March 2023

Contents

1. Introduction

Undeniably, cloud computing arose as one of the most relevant technologies in the past years. The market growth of public cloud services worldwide had an average annual growth of over 20% from 2012 to 2021 and an expected growth of roughly *490 billion USD in 2022* alone [50]. The cloud offers several advantages over self-managed systems for enterprises, e.g., high scalability, robustness, and flexibility, rendering it a highly attractive business option. One of the biggest advantages is that the *cloud provider* takes care of the technical management of the resources provided by the cloud. Managed resources can be the hardware or virtualization for infrastructure as a service (IaaS), the middleware and runtime environment for a platform as a service (PaaS), or nearly everything for software as a service (SaaS) or serverless computation [39]. Independent of the architecture, cloud computing nowadays relies on virtualization. When a software service is executed in the cloud, the software is usually not installed directly onto a physical machine but in a *virtual machine*. Intuitively, a virtual machine simulates a physical one with software allowing it to be mostly independent of the physical infrastructure. Since virtual machines can run on any physical one, they can, for example, be migrated, copied, paused, and resumed. Further, virtual machines can be reconfigured if, for example, more/less computing power or memory is needed by the running software. These properties enable cloud providers to flexibly manage their clients' services and adapt to changing requirements. For example, if the user base of a service grows rapidly, the cloud provider can scale it by distributing copies of the (virtualized) service in its compute centers. Worldwide, cloud providers manage resources, such as virtual machines, in large-scale networks daily, motivating us to consider resource allocation from a theoretical perspective.

Managing resources in cloud computing. A fundamental problem for the cloud provider is determining the locations of the managed virtual machines in the network. In addition, these locations might need to be adapted over time as the clients access the provided services. Consider a cloud provider with a network of computing nodes hosting virtual machines running services as sketched in Figure 1.1 (on the next page). The clients naturally wish to access the provided virtual machines and their services and pose requests for them at multiple network locations. The requests' locations depend on many factors, including how the clients connect to the overall system. Hence, the cloud provider has limited to no influence on these locations. Each time a client's request appears, the provider establishes a route through the network to an eligible machine so that the client gets served. The route is then used to transfer data between the client and its service. Since each route adds a load to the network, migrating or copying a virtual machine closer to the requesting clients becomes worthwhile at some point. However, an operation like virtual machine migration also comes with an overhead, e.g., the allocation of local resources at the destination node, the transfer of data, and the re-initialization. Thus, a significant responsibility of the cloud provider is *balancing* the effort of routing requests through the system and adapting the locations where resources are deployed. In the best case, the management should be invisible to the user, i.e., downtime should be avoided, rendering it a non-trivial task.

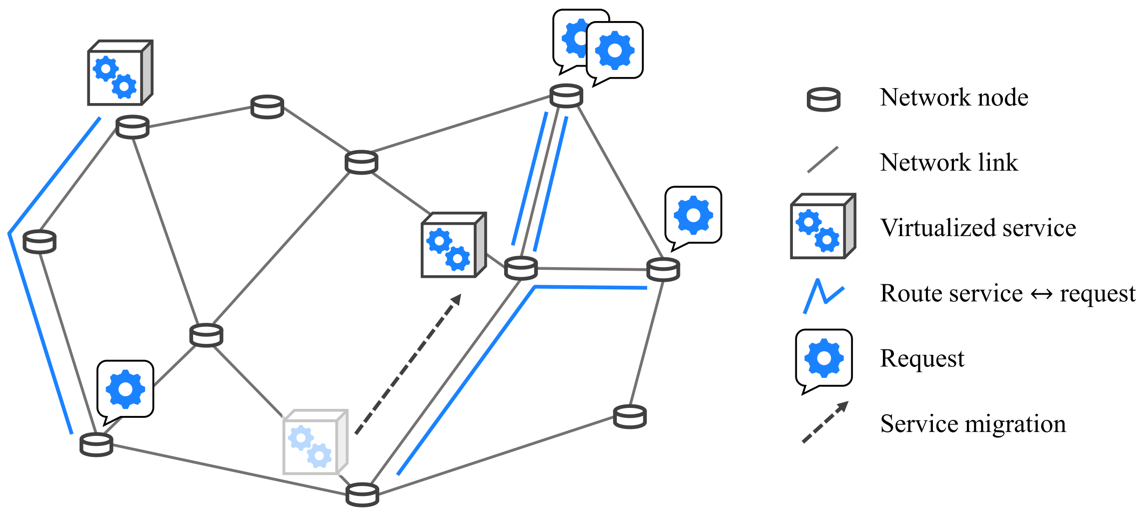

Figure 1.1: A fundamental challenge for a cloud provider is the following. In the distributed system controlled by the provider, services running in virtual machines are provided to customers. Incoming requests of clients for these services require a route from the client to a running instance of the service (left part of the network). When many requests appear in the same area (upper right part of the figure), migrating the virtualized service to the requests becomes worthwhile.

Besides being a fundamental challenge in the context of cloud computing, the above setting also appears in the large research field considered by the Collaborative Research Center (CRC) 901 *On-The-Fly Computing* [56]. Here, the research vision aims at a distributed system that offers the automatic on-the-fly configuration and provision of IT services out of existing base services. Ideally, a new client describes its needs and requirements for the desired software service. The system automatically composes the software from appropriate existing base services, deploys the final service, and gives the client (paid) access. After a service is composed, it is executed in the computing centers of the system. Thereby, there can be multiple instances of each service at different locations. Then, the scenario above applies as stakeholders interested in the services appear in the network to use them. The research presented in this thesis was motivated by the CRC and produced within it during the second and third funding periods.

Resource allocation problems. From the perspective of theoretical computer science, the above scenario falls into models that we collectively refer to as *resource allocation problems*. Here, we abstract from the scenario above to frame a general model capturing the problem of cost-efficiently manipulating the locations of resources in a network. Formally, we define a resource allocation problem as in Definition 1.1.

Definition 1.1 — Resource Allocation Problem. A *resource allocation* problem is given by:

(a) A *metric space* (M,d) with a set of locations M and a distance function d.
(b) A set of *resources* that are at any time placed at locations of M.
(c) A sequence R of *requests* along with a definition of how a request is *served* and a cost associated with the serving.
(d) A set of *actions* an algorithm can execute to manipulate the resources, where each action has a defined cost.

The *task* of an algorithm is to serve all requests while its *goal* is to *minimize the total cost* given by the sum of all costs for serving and executed actions. In the *online* case, the requests are given one by one in the order of R, and the algorithm has to serve each request before the next arrives.

The network is abstracted away by a metric space, while the setting is reduced to resources (e.g., virtual machines, services, data) demanded by requests (e.g., of clients or other services). An algorithm (executed by a cloud provider, for example) is given the task of serving all requests while optimizing the costs incurred by serving requests (e.g., managing routes in the network) and manipulating resources (e.g., migrating machines). Resource allocation problems can also be studied without assuming a metric space, e.g., where the distance function does not fulfill the triangle inequality or symmetry. Usually, the techniques used for non-metric resource allocation problems are vastly different, and the performance of online algorithms is worse than when considering metrics. In this thesis, we only consider metric problems and, thus, define resource allocation problems for a metric.

The above definition captures various models considered in theoretical computer science, usually differing in the actions the algorithm can execute and the requirements for a request to get served. Examples of such problems are the *page migration problem*, the *facility location problem*, and the *k-server problem*. In the *page migration problem*, there is only one resource, called *page*. A request is served by connecting it to the resource with a cost of the distance between them. The allowed actions of an algorithm are solely a movement of the resource for a cost dependent on the moved distance. Contrarily, in the *facility location problem*, resources (called *facilities*) cannot be moved at all. While a connection to the nearest resource still serves requests (where the distance between the resource and request is the cost), an algorithm cannot migrate resources but only instantiate them anew. Deploying a resource to an empty location is achieved by creating a new copy of the desired resource that cannot be relocated afterward. The deployment incurs a cost defined by a function dependent on the target location. The *k-server problem* shifts the focus towards deciding which resource to move. Here, there is a fixed number of k identical resources (called *servers*). A request can only be served when one of the resources is on its location, and the actions allowed by an algorithm are only a movement of a resource with a cost dependent on the moved distance.

Unknown future requests. Resource allocation problems allow for a theoretical view of the challenge of a cloud provider. However, they alone do not capture the real difficulty of the scenario we described. The client's requests translate to the input of an algorithm that manages the resources. Inherently, this input is not known entirely before the algorithm takes action, but the requests arrive step by step as time goes on. Still, the cloud provider must serve the requests immediately upon arrival to offer the clients a satisfying experience. More specifically, the provider has to act based on incomplete data without knowing future requests, while the actions produce immediate and irrevocable costs. Such a problem, where the full input is *not* known in advance, and the algorithm makes irrevocable decisions, is commonly called an *online problem*. The solution to the problem is constructed over time as an algorithm reacts to each arriving request. Complementary, when the full input is known *before* a solution is calculated, the problem is called *offline*. Resource allocation problems pose a new challenge when considered online: How should the quality of a solution be measured? The de-facto standard for measuring the performance of online algorithms is to compare the online cost to the optimal cost that could have been achieved when the entire input was known upfront. We formally introduce this measure, called the *competitive ratio*, in Section 1.1 below. For the research presented in this thesis, we are motivated by the scenario of a cloud provider again to consider resource allocation problems in their online version and design algorithms with a small competitive ratio.

Heterogeneous requests and resources. While past research on online resource allocation has successfully shown algorithms with a guaranteed competitive ratio, the common drawback of most existing models is the following assumption. Usually, the considered resources are of the same kind, meaning, in the example of the cloud provider, there is only one service in the system. In reality, not all virtual machines managed by a cloud provider offer the same service, but they are *heterogeneous*. There are *different* services, and arriving requests demand a subset of these. One client might want a service offering to encrypt data, while another might be interested in a service for storing data or offering high computational power. Managing different services together usually offers a benefit for the cloud provider by a reduced cost for its actions. When migrating separate machines with different services from the same source to the same destination, migrating them *together* can be less costly than migrating them separately. A lot of the introduced management overhead for the migration that occurs before the data transfer (e.g., requesting the migration at the destination and setting up a channel) can be paid only once instead of multiple times. Similarly, instantiating services together in a single virtual machine spares workload compared to instantiating them separately. We introduce the ideas above to resource allocation problems by considering a set of *commodities*. A commodity corresponds to a service in the example of a cloud provider. The resources offer subsets of the commodity set, and each request can express preferences or restrictions for the commodities that can serve it. The costs of an algorithm's actions are influenced by the commodities involved, allowing to model reduced costs when managing commodities together.

Topic of the thesis. The research presented in this thesis aims to develop an understanding of the influence of heterogeneity in online resource allocation problems. To this end, we present how established online resource allocation problems can be generalized by heterogeneity using multiple commodities. By considering three fundamental problems – the page migration problem, the facility location problem, and the k-server problem – we show different effects of multi-commodity resources and requests that pose new challenges for the design of competitive online algorithms. Further, we study the introduced generalizations by lower bounds and design algorithms with provably good competitive ratios. To gradually show the influence of heterogeneity, we present competitive ratios parameterized by key properties of the extended models.

1.1 Technical Preliminaries

Next, we present the main fundamental concepts used throughout this thesis before outlining the content in Section 1.2. We give a brief introduction to the *competitive ratio*, *Yao's minimax principle*, and *linear programs and duality*. We refer to the standard literature on online algorithms for further details, e.g., [2, 21].

The competitive ratio. Introduced as *amortized efficiency* by Sleator and Tarjan in [87], the *competitive ratio* quickly evolved to a standard analysis technique for the performance of online algorithms. It measures the worst ratio achievable over all instances of a problem of the cost of an online algorithm $\mathrm{C}_{\mathrm{ALG}}$ and the cost of an optimal offline algorithm $\mathrm{C}_{\mathrm{OPT}}$. The formal definition is captured in Definition 1.2.

Definition 1.2 — Competitive Ratio. An online algorithm ALG achieves a *competitive ratio* of c for a problem P, if for all instances of P it holds that

$$\mathrm{C}_{\mathrm{ALG}} \leq c \cdot \mathrm{C}_{\mathrm{OPT}} + a, \text{ where } a \text{ is a constant independent of the instance.}$$

Online computation can be seen as a game between the online algorithm and an *adversary*. The adversary is the controller of the input (the request sequence in our case). It generates the input sequence that the online algorithm has to solve while presenting a good (offline) solution to the input such that the competitive ratio is as large as possible. Thereby, the adversary has a description of the online algorithm. The above definition fits deterministic online algorithms. Here, the adversary precisely knows how the algorithm will behave and can generate the worst-case input sequence. For randomized algorithms, the competitive ratio potentially improves because the adversary is weakened when an online algorithm behaves based on random experiments, and its behavior is (possibly) not entirely known. How much the competitive ratio can improve depends on the adversary's capabilities. Ben-David et al. [15] present three different adversary types with different capabilities and relate their strength, i.e., the limits of randomized online algorithms against them. The commonly used adversary for the problems we consider is the *oblivious adversary*. The oblivious adversary knows the description of the online algorithm but not the outcome of random choices and must construct the entire input sequence before the online algorithm solves it. Since it does not know the outcomes of the random experiments done by the algorithm while constructing the input, the oblivious adversary often cannot create as difficult inputs as against deterministic algorithms. For randomized algorithms against the oblivious adversary, the expected cost $\mathrm{E}[\mathrm{C}_{\mathrm{ALG}}]$ of the online algorithm is compared to the optimal solution. Thus, we adapt the definition of the competitive ratio as in Definition 1.3.

Definition 1.3 — Randomized Competitive Ratio. An *randomized* online algorithm ALG achieves against the *oblivious adversary* a *competitive ratio* of c for a problem P, if for all instances of P it holds that

$$\mathrm{E}[\mathrm{C}_{\mathrm{ALG}}] \leq c \cdot \mathrm{C}_{\mathrm{OPT}} + a, \text{ where } a \text{ is a constant independent of the instance.}$$

Besides the oblivious adversary, other adversary types are the *adaptive online adversary* and the *adaptive offline adversary*. Both of them can adapt to the outcome of the random experiments of the algorithms and create the input sequence online, i.e., after observing a step of the online algorithm, the next request is generated. Thereby, the adversary learns the outcomes of random experiments of the algorithm while constructing the online sequence. The adaptive online adversary has to solve the input sequence online while constructing it. So, the adversary generates a request, has to serve it, sees the algorithm's action (and the outcome of random experiments), and generates the next request. Contrarily, the adaptive offline adversary can solve the sequence optimally after it is determined completely. The paper of Ben-David et al. [15] relates these adversaries and finds a strong relation between them: An algorithm with a competitive ratio of c for the adaptive offline adversary achieves the same competitive ratio against the adaptive online algorithm. It implies a competitive ratio of c against the oblivious adversary. Intuitively, the oblivious adversary can be viewed as the weakest, while the adaptive offline adversary is so strong that randomization poses no advantage over determinism. All results mentioned in this thesis are considered against the oblivious adversary if not stated differently.

Note that the competitive ratio is a worst-case ratio, i.e., a competitive algorithm achieves the respective performance on *all* input sequences. While this enables *guarantees*, the obvious drawback is that the quality of algorithms performing significantly better besides the worst case remains hidden. Several attempts have been made to tackle this shortcoming. For an overview, we refer to [69]. Noteworthy is the application of *smoothed analysis* [88] to the competitive ratio yielding the term *smoothed competitive ratio* [13]. Here, the input sequence generated by the adversary is randomly perturbed according to a probability distribution before the online algorithm has to solve it. This way, the adversary can no longer enforce very specific worst-case inputs. For example, an application to metrical task systems [84] reveals that the smoothed competitive ratio of the famous work function algorithm is significantly better than the classical competitive ratio.

Yao's principle. Besides designing algorithms with a bounded competitive ratio, we are interested in *lower bounds* stating limits on the best achievable competitive ratio. A lower bound states that no algorithm can do better than the declared bound for the competitive ratio. Since the statement is against any online algorithm, especially for randomized algorithms, designing lower bounds includes many possible random choices. Yao's principle [91] simplifies the design of such bounds against the oblivious adversary. Since the oblivious adversary is the weakest of the adversaries presented above, a lower bound against it immediately implies lower bounds for the other adversaries. Note that a lower bound for randomized algorithms also holds for deterministic algorithms. Theorem 1.1 restates Yao's principle for lower bounds on online algorithms.

Theorem 1.1 — Yao's Principle for Online Algorithms. The (expected) competitive ratio of *any* randomized online algorithm against its worst-case input sequence is at least as high as the (expected) competitive ratio against *any* random distribution of inputs of the best deterministic algorithm for the distribution.

So, to design lower bounds for randomized algorithms, it is sufficient to create an appropriate distribution of input sequences and show a bound on the expected competitive ratio for the best deterministic algorithm against it. Since we only need to consider a deterministic algorithm again, the lower bound design simplifies significantly. Yao's principle evolved into a standard technique for designing lower bounds. We apply it for our lower bounds for the multi-commodity online page migration and the multi-commodity online facility location problem (see Chapters 2 and 3). For further details on Yao's principle, we refer to [21, Chapter 8].

Linear programs and duality. (Offline) resource allocation problems are *optimization problems*, i.e., the goal is to determine the *best* of all feasible solutions. Many optimization problems can be framed as *linear programs*, a compact common representation. Linear programs and the concept of duality offer an elegant approach to analyzing certain online problems. Below, we briefly overview the most important terms and results relevant to us. A more detailed overview of the topic can be found in the survey on primal-dual online algorithms by Buchbinder and Naor [29].

A linear program consists of *variables*, an *objective function* that is to be minimized (or maximized), and *constraints* that limit the values variables can take. The objective function and the constraints are defined via linear functions on the variables. When the occurring variables are integral, the problem is an *integer linear program*. A solution to such a problem is an assignment of values to all variables. It is *feasible* if all constraints are fulfilled. For any problem that can be framed as a linear program – called *primal*, there is an equivalent linear program – called *dual*. For a primal that minimizes (maximizes), the dual maximizes (minimizes) the objective function. A core result useful for analyzing online problems is *weak duality* stating that the value of the objective function of *any* feasible solution to the dual is a lower bound to the value of the objective function of the optimal primal.

The variables and constraints are not known upfront for online problems but are revealed to the algorithm over time. Nevertheless, the primal-dual approach allows the design of online algorithms that can be analyzed using weak duality. Suppose the problem is a minimization problem, which is usually the case. A primal-dual algorithm maintains feasible solutions to the primal/dual defined for the requests seen so far. Arriving requests require that the algorithm extends its solution. Thereby, it is not allowed to reduce previously set variables. The latter reflects that the online algorithm cannot undo previous actions. Weak duality offers an elegant way to compare an online algorithm's cost with the optimal solution required for the competitive ratio. For any sequence of requests and the linear program defined over them, when the ratio between the cost of the primal solution and the (feasible) dual solution maintained by the online algorithm is bounded, weak duality ensures the same bound on the competitive ratio. With the same strategy, we show the competitive ratio of our deterministic algorithm for the multi-commodity online facility location problem in Chapter 3.

1.2 Thesis Outline & Main Results

This thesis considers three classical online resource allocation problems – the page migration problem, the facility location problem, and the k-server problem – and extends their models by heterogeneity. More specifically, we introduce a set of commodities S that are offered by resources and demanded by requests. We analyze and prove lower and upper bounds for the competitive ratio. For the upper bounds, we design and analyze respective online algorithms. On a technical level, our results look beyond the classical competitive ratio as a worst-case measure by considering parameterizations that show the influence of heterogeneity. Thereby, we connect the extended models to the classical ones, which can always be instantiated by assuming a single commodity in our models. The thesis is split into three chapters by the three online problems.

Multi-commodity page migration. As a starting point, we extend the page migration problem with multiple commodities in Chapter 2. In the page migration problem, there is only one resource – called page. A request is served by connecting it to the page with a cost of the distance between them. The algorithm can move the page for a cost of D times the moved distance. $D \geq 1$ is a constant that can be interpreted as the page size. For the online problem, deterministic algorithms that achieve constant competitive ratios are known, e.g., with a competitive ratio of 4 for general graphs [18].

Adding commodities. We extend the classical model by a set S of commodities to the *multi-commodity page migration problem*. Instead of one page, there are $|S|$ many pages, each of them offering one unique commodity. A request demands a subset of the commodities and is served by connecting it to each page offering a desired commodity. The costs are the distances between the request and all connected pages. As in the classical model, an algorithm can move pages. A single page of commodity $s \in S$ can be moved by a cost of D_s times the moved distance, where $D_s \geq 1$ is a constant for s. Moving a set of pages $\sigma \subseteq S$ from a source location i to a destination location j has a cost dictated by a function $f_{i,j}^{\sigma}$. The movement cost function allows modeling reduced costs when moving multiple pages together compared to moving them separately. With the model extension, we can study if online algorithms can benefit from moving multiple commodities together. Regarding the example of a cloud provider, situations in which multiple services (commodities) can be migrated at once are captured.

Thesis results. We present a lower bound on the competitive ratio of any (randomized) online algorithm against an oblivious adversary of $\Omega(\max_{i,j \in M, S' \subseteq S}\{\sum_{s \in S'} f_{i,j}^{\{s\}} / f_{i,j}^{S'}\})$. Intuitively, the lower bound states that any algorithm can be forced to move all pages of the worst possible subset S' between the worst possible locations i and j separately for a cost of $\sum_{s \in S'} f_{i,j}^{\{s\}}$. At the same time, the optimal solution can take full advantage of a combined movement of the pages for a cost of $f_{i,j}^{S'}$. Consequently, no online algorithm can benefit from managing the pages together. Consider a deterministic classical page migration algorithm A with a competitive ratio of c (not necessarily constant). We show the trivial approach that runs A for every commodity separately achieves a competitive ratio of $\mathcal{O}(c \cdot \max_{i,j \in M, S' \subseteq S}\{\sum_{s \in S'} f_{i,j}^{\{s\}} / f_{i,j}^{S'}\})$. Since there are deterministic algorithms with a constant competitive ratio, our bounds are asymptotically tight. Therefore, our results show that the extended model introduces too much power for the adversary, and online algorithms cannot take advantage of the joint management of resources.

Multi-commodity facility location. In Chapter 3, we extend the facility location problem by commodities to the *multi-commodity facility location problem*. The facility location problem considers facilities as resources that are opened at locations of the metric space. A request is served by connecting it to a facility with a cost of the distance between them. An algorithm can open a new facility at any location $m \in M$ for a cost determined by a function f_m. In comparison to the page migration problem, facilities cannot be moved. For the online case, the competitive ratio is in $\Theta(\log n / \log\log n)$ (n is the number of requests) by a lower bound against the oblivious adversary, a randomized, and a deterministic algorithm [48].

Adding commodities. We generalize the classical model by a set S of commodities. Each facility has a fixed set of commodities (called *configuration*) it offers that is determined upon construction. An arriving request demands a set of commodities and can be served by connecting it to a set of facilities jointly offering them. The serving costs are the distances between the request and all connected facilities. The algorithm can construct a facility offering commodities $\sigma \subseteq S$ at a location $m \in M$ for a cost determined by a function f_m^σ. Here, the function can model situations where offering multiple commodities in a single facility is less costly than offering the same set of commodities in several facilities. Formally, the function is sub-additive for a fixed location, i.e., for a fixed $m \in M$ for all $\sigma, \tau \subseteq S$ it holds that $f_m^{\sigma \cup \tau} \leq f_m^\sigma + f_m^\tau$. Thereby, we can model situations as in the introductory example, where a cloud provider deploys several services (commodities) in the same virtual machine (facility).

Chapter basis. We base our results on the following publication. To the best of our knowledge, we were the first to consider the multi-commodity facility location problem in its online version. Below, we outline the results of our publication.

The Online Multi-Commodity Facility Location Problem
Jannik Castenow, Björn Feldkord, Till Knollmann, Manuel Malatyali, and Friedhelm Meyer auf der Heide
In: Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2020), July 2020, Pages 129–139
Cf. [31]

In the paper, we present a lower bound of $\Omega(\sqrt{|S|} + {}^{\log n}/_{\log \log n})$ for (randomized) online algorithms against the oblivious adversary. The lower bound shows that the number of commodities significantly influences the competitive ratio and indicates that randomized approaches – as in the classical problem – have no advantage over deterministic algorithms. Furthermore, the lower bound reveals that a well-performing algorithm needs to include commodities in facilities that were not yet requested. Intuitively, it must *predict* which commodities will be needed to benefit from a combined construction.

On the positive side, we present two algorithms – deterministic and randomized – with bounded competitive ratios under an assumption on the construction cost function. Recapitulate that f_m^σ is the cost for opening a facility offering the configuration $\sigma \subseteq S$ at location $m \in M$. The assumption is denoted by Condition 1 in the paper and requires that for all $\sigma \subseteq S$, it holds that ${}^{f_m^\sigma}/_{|\sigma|} \geq {}^{f_m^S}/_{|S|}$. Intuitively, the cost per commodity must scale and be minimal when offering all commodities. Condition 1 holds for all functions that solely depend on the location and the size of the offered commodity set. It especially holds for the construction cost function used in the lower bound. The deterministic algorithm is based on Fotakis' primal-dual algorithm for online facility location [47] and achieves a competitive ratio of $\mathcal{O}(\sqrt{|S|} \log n)$ when Condition 1 holds. The randomized algorithm is based on Meyerson's algorithm [76] and achieves a competitive ratio of $\mathcal{O}(\sqrt{|S|} \cdot ({}^{\log n}/_{\log \log n}))$ under Condition 1. The lower bound incurs an additive term of $\sqrt{|S|}$, while the upper bounds have a multiplicative dependence on $\sqrt{|S|}$. Such a situation, for example, also appears in the facility leasing domain [78], and we strongly believe that the real dependency is multiplicative. The main difficulty for the algorithm design is determining a set of configurations in which facilities are constructed. Naively allowing all $2^{|S|} - 1$ many possible commodity sets is not feasible for a small competitive ratio, as it is too difficult for an online algorithm to decide on which configuration to choose. Both algorithms use the key observation that under Condition 1, it is sufficient to construct facilities offering a single commodity or all commodities. Intuitively, at some point, the algorithms decide that all commodities may be needed and construct a facility offering all. In the paper, we further showed that the competitive ratio of the deterministic algorithm improves for more restricted construction cost functions.

Thesis results. Next, we present how we generalize the publication results in this thesis. We first state the lower bound of the paper. After that, we generalize both our algorithms as follows. We frame them in a unified way and adapt their descriptions accordingly, allowing us to parameterize their analyses. The parameter h that we introduce captures a key property of the construction cost function and is determined as follows based on two parameters h_1 and h_2. We first explain h_2 for a better presentation. When an online algorithm constructs a facility, it has to decide on one of $2^{|S|}-1$ many possible configurations. To reduce the number of choices, we design algorithms that restrict themselves to a fixed set of configurations they offer. The construction of a facility at a location is motivated by commodities that were requested around. So, for each fixed commodity, the algorithm selects one from all considered configurations containing the commodity. We limit the choice by a parameter h_2. The number of configurations the algorithm considers that contain any fixed commodity is bounded by h_2 for every commodity. However, limiting the number of possible configurations to choose from when constructing a facility has a major drawback. Some configurations can no longer be offered as the algorithm decided not to consider them a possibility. On the other hand, such a configuration may be the best choice, i.e., have the lowest cost and is chosen by an optimal solution. Here, the commodities of such a configuration must be covered by configurations that the algorithm allows. We use a parameter h_1 to grasp how cost-efficient such a covering is. The cost of covering all commodities of any configuration with the algorithm's configurations is at most h_1 times the optimal cost. Intuitively, there is a trade-off between h_1 and h_2. Reducing the number of considered configurations reduces h_2 and increases h_1 as the fewer configurations are available, the more it potentially costs to cover any set of commodities with them. The parameter h is then defined as $h = h_1 \cdot h_2$, and balances the trade-off above. Intuitively, h captures how well a fixed set of configurations can be determined, which allows a cost-efficient covering of every possible configuration. Lower values for h are preferable as they enable a lower competitive ratio. However, h strongly depends on the given construction cost function.

Parameterizing our algorithms by h allows them to act more flexibly. They are no longer restricted to facilities offering either one or all commodities. Dependent on the structure of the construction cost function, they determine a fixed set of configurations that are considered. Each such configuration is a set of commodities that can be offered relatively cheaply in one facility.

We prove that our deterministic algorithm achieves a competitive ratio of $\mathcal{O}(h \log n)$, and our randomized algorithm one of $\mathcal{O}(h \cdot (\log n/\log\log n))$. We show that Condition 1 of our publication assures $h \in \mathcal{O}(\sqrt{|S|})$ such that the publication results can be seen as a special case of our generalized results. As mentioned above, we showed in the paper that a more restricted class of functions allows a better competitive ratio for the deterministic algorithm. In this thesis, we prove a bound on the parameter h for the same class of functions. Thus, the improved bounds in the paper are a consequence of the underlying property of the construction cost function. Our technique in this thesis allows us to generalize the improved bounds for *both* our algorithms. Noteworthy, the resulting dependence on the number of commodities is often $h \ll \sqrt{|S|}$. Our improved approach generally allows deriving bounds respecting the construction cost function without analyzing the algorithms anew. It is sufficient to analyze the construction cost function alone. We further design a function where $h \notin \mathcal{O}(\sqrt{|S|})$ and discuss the implied limits of our approach. It is unclear what algorithmic approaches are required to circumvent the function. However, the bound relies on a construction cost function that varies greatly between locations. For construction cost functions independent of the location, we conjecture that $h \in \mathcal{O}(\sqrt{|S|})$, i.e., the dependence on $|S|$ meets the lower bound.

Future work may require models that allow a facility to be closed again. Therefore, we consider models where facilities are not open forever after construction but must be *leased*. An established model of the literature [78] combines online facility location with the parking permit problem by Meyerson [77]. Here, there is a set of leases L. A lease of L must be chosen when constructing a facility. The lease determines how long the constructed facility remains open and influences the construction cost. The lower bound for deterministic algorithms is $\Omega(|L| + \log n/\log\log n)$, and the upper bound by [78] is $\mathcal{O}(|L| \log n)$. We present how the facility leasing model and our multi-commodity model can be united. Here, we show how our deterministic algorithm can be combined with the algorithm of [78] and prove that it achieves a competitive ratio of $\mathcal{O}(|L|\, h \log n)$, where h is the same parameter as in the model without leasing. The combination is possible as both algorithms are based on Fotakis primal-dual algorithm [47]. The leasing model above may still lack flexibility. Thus, we consider an additional model where a facility is constructed with a cost of f_m^σ. Afterward, to remain open, a maintenance cost of c_m^σ must be paid in every consecutive time step. We prove that if the construction costs are independent of the location and maintenance costs are equal for all locations and configurations the problem can be solved with a competitive ratio of $\mathcal{O}(h \log n)$, i.e., as good as without leasing. The proof utilizes a relationship between the maintenance cost model and the previous leasing model.

Our results show how heterogeneity introduces new parameters (the more complex construction cost function) that have a relevant impact on the achievable competitive ratio.

Multi-commodity k-server. We extend the k-server problem to the *multi-commodity k-server problem* in Chapter 4. In the k-server problem, there are k identical resources (called *servers*). Serving a request requires a server to be at the request's location. Then, the serving incurs zero cost, but an algorithm may need to move a server to the request. To this end, the algorithm can move each server at any time for a cost of the moved distance. The necessity to place a server on each request makes the problem difficult in the online case. We focus on deterministic algorithms, where the competitive ratio is at least k [71]. The famous k-server conjecture claiming that there is a deterministic algorithm with a competitive ratio of k for every metric space is, until today, not settled. The currently best algorithm for general metrics is the work function algorithm with a competitive ratio of $2k-1$ [61].

Adding commodities. As before, we introduce heterogeneity by a set of commodities S. Every server offers a fixed subset of the commodities. A request demands a set of commodities and can be served (with zero cost) by a server offering *one* of the demanded commodities at its location. As in the classical model, each server can be moved with a cost equal to the moved distance. Unlike our extensions of the page migration problem and the facility location problem, an algorithm cannot reduce costs by managing commodities together. Heterogeneity appears as a restriction of requests and introduces more potential costs for the online algorithm based on which server serves a request. Our extension allows modeling situations in which a cloud provider has k virtual machines (servers) that offer different services (commodities).

Chapter basis. We base our results on the following publication. Below, we briefly outline its results before presenting the main results of this thesis.

The k-Server with Preferences Problem

Jannik Castenow, Björn Feldkord, Till Knollmann, Manuel Malatyali, and Friedhelm Meyer auf der Heide
In: Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2022), July 2022, Pages 345–356
Cf. [32]

In our publication, we considered a special case of the multi-commodity k-server problem, the any-or-one case. In that case, every server offers exactly one unique commodity, i.e., there are k commodities. Also, every request can either be served by *any* server (called general request) or a specific *one* (called specific request). Note that a general request is as in the classical k-server problem and that a specific request leaves no choice on which server to move. A naive approach would use a classical k-server algorithm for general requests and move the specified server whenever a specific request appears. We show that already for seemingly simple instances (uniform metric, and $k = 3$ servers on 3 locations), such approaches can end up with an unbounded competitive ratio. Further, we show that introducing specific requests generally raises the lower bound to $2k-1$.

Independently, the lower bound was shown earlier in [52] with a different structure. Using both structures allows us to extend the bound, as discussed later.

As the lower bound already holds on uniform metrics, we design algorithms for them. We present a deterministic algorithm (CONF-MCOKSP) with a competitive ratio of $3k-2$ and show that there is an instance in which it has a competitive ratio of at least $3k-2$. Since the general lower bound is by $k-1$ lower, we consider a parameter s which is the ratio between the number of relevant specific requests and the total number of requests in the input sequence. For $s=0$, there are no specific requests, and the instance is of the classical k-server problem (with a lower bound of k). For $s=1$, there are only specific requests, and the instance is trivial to solve optimally because there is no freedom of choice. The lower bound of $2k-1$ is at roughly $s \approx 1/2$. For CONF-MCOKSP, we analyze the competitive ratio respecting s and show that it is optimal for $s=0$, raises smoothly to $3k-2$ at $s \approx 1/3$, decreases to $2k-1$ at $s \approx 1/2$ (meets the general lower bound), and decreases further to 1 at $s=1$. So, the competitive ratio of the algorithm roughly follows the lower bounds but rises too high. In the paper, we analyze the reason and determine a crucial behavioral rule – acting *defensively* – that has to be embedded to get closer to the bound of $2k-1$. We present a second deterministic algorithm (DEF-MCOKSP) following the rule and prove a competitive ratio of $2k+14$. However, DEF-MCOKSP has a lower bound of provably $2k-1$ on the competitive ratio for $s=0$ (k-server instances), while the first algorithm is optimal here. We analyze that difference by showing bounds on the competitive ratio in general and for $s=0$ that depend on how many servers act defensively or not. Our results show a trade-off between performing well on k-server instances and instances with specific requests that no algorithm can avoid, i.e., no algorithm can perform best for all values of s.

Thesis results. In this thesis, we strengthen the results of our publication as follows. The paper lacks a lower bound parameterized entirely in s. We present a smooth lower bound *for any value s* can take between k-server instances ($s=0$) and purely specific instances ($s=1$). The lower bound thereby relates the hardness of our model extension in the online case to the number of specific requests. It connects the previously known lower bounds and illustrates the adversary's power.

We further improve the adaptive bounds of CONF-MCOKSP and, in particular, show that for all $s \geq \frac{k}{2k-1} \approx \frac{1}{2}$, the competitive ratio meets the lower bound. We sharpen the bounds of DEF-MCOKSP and show a tight competitive ratio for $s \geq \frac{k}{2k-1}$ as well. Thus, the power of the adversary lies clearly in instances where $s < \frac{k}{2k-1}$. Interestingly, for larger s, the competitive ratio decreases independently of k. Regarding the bounds for algorithms that follow our behavioral rule, we show that the lower bound not only increases for $s=0$, but the smallest achievable competitive ratio for $s < \frac{k}{2k-1}$ inevitably increases by the number of servers acting defensively. Our two algorithms differ only in the per-server decision of whether or not to act defensively. Therefore, they can be combined flexibly. It suffices to determine how many servers shall follow CONF-MCOKSP, and how many shall follow DEF-MCOKSP. We formally analyze the competitive ratio of such a combination in the number of servers following CONF-MCOKSP. As before, the competitive ratio of the resulting algorithm MIXED-MCOKSP is framed parameterized in s. MIXED-MCOKSP generalizes CONF-MCOKSP and DEF-MCOKSP and can be tailored at will following the trade-off in the competitive ratio we determined. Our results of this thesis show a detailed picture of the performance of deterministic online algorithms on uniform metrics for the any-or-one case.

Beyond uniform metrics, we present a deterministic algorithm for the any-or-one case for line metrics based on the double coverage algorithm [34] that achieves a competitive ratio of 6 for $k=2$. Extending the algorithm for more servers poses surprising difficulties that we discuss respectively. Our algorithm above can be seen as a first step towards more complex metrics.

Our results reveal how heterogeneity introduces a situation where no algorithm is *best*, but a trade-off appears that requires different algorithmic approaches.

The big picture. Our results demonstrate how heterogeneity introduces several effects when applied to online resource allocation requiring new techniques for well-performing algorithms. Heterogeneity can make the adversary too strong (as in the page migration problem), introduce a heavy dependence on new parameters (as in the facility location problem), or create an inevitable trade-off (as in the k-server problem). We believe there is far more research possible to enrich existing models and develop a deeper understanding of online algorithms. Chapter 5 presents an outlook on promising research directions.

2. Multi-Commodity Online Page Migration

The upcoming chapter considers the *multi-commodity online page migration problem* (abbreviated MCOPMP), a generalization of the *page migration* problem. Page migration considers one resource (called *page*) that can be moved around in the metric space. A request is served by connecting it to the page with a cost of the connection distance. An algorithm can migrate the page to another location for a cost of D times the moved distance, where $D \geq 1$ is a constant that can be interpreted as the page size. The model captures situations where the resource is fixed and can (but need not) be migrated to areas where many requests appear.

We generalize the model by introducing commodities. Thereby, we assume multiple pages, each belonging to a different commodity. An arriving request specifies the desired commodities, and an algorithm has to serve each commodity to the request. Respecting the introductory example (see Chapter 1), each page can be interpreted as a virtual machine with a unique service. Arriving requests might require access to not only one but multiple services and hence, need to be served by multiple pages. Naively, one could manage each page separately. However, overhead costs can be spared when migrating several machines from a common source to a common destination in one go. For example, connecting to the destination, exchanging cryptographic keys, and initializing a tunnel for the data transmission must only be done once. Therefore, an algorithm can reduce costs when migrating pages together instead of separately.

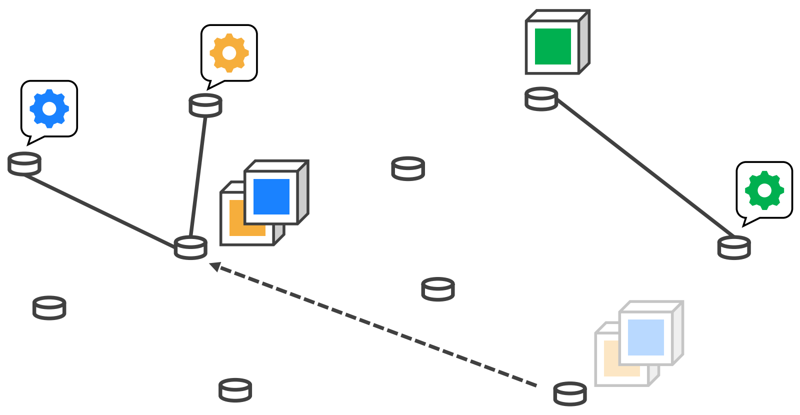

Figure 2.1: In *multi-commodity online page migration*, there is one page (box) per commodity (commodities are given as colors). Requests (speech bubbles) may require multiple pages to be served. While serving a request by a connection to the page is sufficient (solid lines), an algorithm might move pages to reduce future costs (a dashed arrow is a movement pointing to the destination). Here, jointly moving pages from a source to a destination (like blue and gold) improves upon separate movements.

Next, we formalize our model in Section 2.1 before briefly reviewing related work in Section 2.2. Afterward, we outline our results in Section 2.3. Mainly, the adversary in our model is too strong such that, for the competitive ratio, no beneficial usage of joint movements is possible. We present a lower bound against randomized online algorithms in Section 2.4. A trivial deterministic algorithm matching the lower bound is shown in Section 2.5.

2.1 Problem Definition & Model

We define the multi-commodity page migration problem as a resource allocation problem following Definition 1.1. Recapitulate that the task of an algorithm is to serve every request while the goal is to minimize the total cost.

We consider any metric space (M,d) and assume a set S of commodities. The set of resources is given by $|S|$ pages. Each page offers one unique commodity. A request $r \in R$ consists of a location of M and the subset $s_r \subseteq S$ of the commodities it demands. We abuse the notation here and refer by r also to the location of a request $r \in R$. Let $p_1, \dots, p_{|S|}$ be the locations of the pages. Request r is served when connected to all pages it requests. The cost for serving request r is given by $\sum_{i \in s_r} d(r, p_i)$. The actions of an algorithm are movements of pages. When an algorithm moves a subset $\sigma \subseteq S$ of pages from location i to location j, the cost is defined as $f_{i,j}^{\sigma} \geq 0$. Here f is a cost function such that for fixed $i, j \in M$ and for all $\sigma, \tau \subseteq S$ it holds that $f_{i,j}^{\sigma \cup \tau} \leq f_{i,j}^{\sigma} + f_{i,j}^{\tau}$ (*sub-additivity*) and for all $\sigma \subseteq S, \tau \subseteq \sigma$ it holds that $f_{i,j}^{\sigma} \geq f_{i,j}^{\tau}$ (*monotonicity*). Sub-additivity and monotonicity can be enforced by defining $f_{i,j}^{\sigma}$ as the cheapest way to move the commodities of σ from i to j. Assuming then that sub-additivity is violated for some $\sigma, \tau \subseteq S$ and $i, j \in M$, it holds that $f_{i,j}^{\sigma \cup \tau} > f_{i,j}^{\sigma} + f_{i,j}^{\tau}$ and it is cheaper to move $\sigma \cup \tau$ from i to j separately than combined. By definition, $f_{i,j}^{\sigma \cup \tau} \leq f_{i,j}^{\sigma} + f_{i,j}^{\tau}$ contradicting the assumption that sub-additivity is violated. By similar arguments, monotonicity holds. Additionally, we assume that f fulfills the triangle inequality and symmetry for fixed σ, i.e., for all $i, j, k \in M$ it holds that $f_{i,k}^{\sigma} \leq f_{i,j}^{\sigma} + f_{j,k}^{\sigma}$ and $f_{i,j}^{\sigma} = f_{j,i}^{\sigma}$. Further, for any $s \in S$ it holds that $f_{i,j}^{\{s\}} = D_s \cdot d(i,j)$, where D_s is a constant for s. D_s can be interpreted as the size of the page offering s. Of course, one could consider a non-linear dependence or independence between the functions f and d. As a starting point, we focus on linear dependencies.

Comparison to classical page migration. Our model generalizes the classical page migration problem, where $|S| = 1$, i.e., there is only a single commodity (and thus, resource) that can be moved. By the function f, we can model situations where moving resources together between the same locations is beneficial compared to moving them separately. On the algorithmic side, an algorithm not only has to decide on *when* and *where* to move a page but also *which* pages to move jointly.

2.2 Related Work

The page migration problem was originally intended for shared memory systems, where a set of processors, each with local memory, work on shared data. A page in such a setting is restricted to one local memory at a time. Processors can access the page even when it is in another processor's memory, but a page can also be moved (for a higher cost) to a processor accessing it frequently. The classical problem considers the problem on a graph, where the movement cost is D times the moved distance while the serving cost is simply the distance between the request and the page. A dynamic program can solve the offline version of the problem. Hence, research on the problem concentrated on the online version.

Next, we give a rough overview of the research in that direction. For more results, we refer to the survey by Bienkowski [16]. Black and Sleator started in [20] to consider the online page migration problem. They showed that no deterministic online algorithm could achieve a competitive ratio lower than 3, even for systems with two processors. In addition, they presented optimal deterministic algorithms for uniform and tree metrics. For other metrics, the lower bound is slightly larger than 3 for any D [74]. The currently best deterministic algorithm for general graphs achieves a competitive ratio of 4 [18]. For randomized solutions, there is the simple *coin-flip algorithm* that achieves a competitive ratio of 3 even against the adaptive online adversary [90]. Further, the results by Westbrook [90] give a competitive ratio of roughly 2.62 against the oblivious adversary. The page migration problem falls into the class of *relaxed task systems*, i.e., metrical task systems that can be related to other metrical task systems, in this case, the k-server problem. For an explanation of metrical task systems, see Section 3.2 in Chapter 4. The relationship between both problems enabled solutions to the *k page migration problem*, a generalization with k identical pages instead of one. Bartal et al. [11] present a randomized and a deterministic algorithm for this problem with competitive ratios $\mathcal{O}(k)$ and $\mathcal{O}(k^2)$, respectively. Note that the MCOPMP can also be seen as a generalization to the k page migration problem.

2.3 Our Results

Observe that a trivial solution to the MCOPMP is to execute an algorithm for the classical single-commodity case for each commodity separately. In the worst case, the optimal solution combines all movements and pays for each only a $1/|S|$ fraction of the cost of the algorithm for the same movements. Simultaneously, the optimal solution cannot be better than by a factor of $|S|$ because the cost of a combined movement must be at least as high as the cost of moving any of the $|S|$ commodities on its own. Thus, a trivial algorithm has a factor of $|S|$ in the competitive ratio.

In our model, an online algorithm can profit by moving several pages together. One first observation is that moving only pages for which requests were observed in the past is – in the worst case – equivalent to the trivial solution explained above. The reason is that the adversary could reorder any input sequence so that requests for the same page appear consecutively such that the algorithm moves the pages one by one. An algorithm that takes advantage of joint movements thus – at some point – has to move pages to a location where they were not yet requested. In other words, a *prediction* on the future locations of pages is needed.

Unfortunately, we show Theorem 2.1 in Section 2.4: i.e., any kind of prediction is not beneficial, even if it is done randomly. For any two locations $i, j \in M$ and any commodity set $S' \subseteq S$, no online algorithm can efficiently guess the correct set of pages to place on i and j. Thus, it has to pay a cost for each page of S' to converge toward the correct partitioning. At the same time, the optimal solution can take the full benefit from a combined movement and pays $f_{i,j}^{S'}$.

Theorem 2.1 — Lower Bound. No randomized online algorithm for the multi-commodity online page migration problem can achieve a competitive ratio against the oblivious adversary better than $\Omega(\max_{i,j \in M, S' \subseteq S}\{\sum_{s \in S'} f_{i,j}^{\{s\}} / f_{i,j}^{S'}\})$.

Note that the bound above applies against the oblivious adversary. Therefore, it applies to all other adversary types mentioned in Section 1.1 and especially to deterministic algorithms. As implied by our lower bound above, the trivial algorithm that runs a classical page migration algorithm for each page separately performs asymptotically optimal since there are constant competitive algorithms for classical page migration. The proof of the theorem is presented in Section 2.5

Theorem 2.2 — Deterministic Upper Bound. Consider a deterministic algorithm for the multi-commodity online page migration problem that uses a page migration algorithm with a competitive ratio of c for each page separately, i.e., that never moves pages combined. The algorithm achieves a competitive ratio of at most $\mathcal{O}(c \max_{i,j \in M, S' \subseteq S}\{\sum_{s \in S'} f_{i,j}^{\{s\}} / f_{i,j}^{S'}\})$.

We can see here how the adversary in our model is so strong that joint movements are not beneficial, and thus, a trivial algorithm already achieves an optimal competitive ratio. Any adversary can heavily exploit that whenever an algorithm uses a joint movement, it has to move pages for which there is no hint of the best future location yet. Then, the movement of these pages can immediately be made a mistake by the adversary. The inherent problem for the algorithm is that a predictive movement of a page can be as costly as not moving it.

R The statement of Theorem 2.2 also holds for randomized algorithms. As our arguments do not rely on any property of the used page migration algorithm other than its competitive ratio, even the adversary type immediately adapts to the adversary type considered for it.

2.4 The Lower Bound

For the proof of Theorem 2.1, we use that any advantage in cost by jointly moving several pages requires pages to move that were not yet requested at the destination. In a sense, an algorithm has to predict the future location of a page to take advantage of joint movements. As we show below, no such prediction is beneficial.

Theorem 2.1 — Lower Bound. No randomized online algorithm for the multi-commodity online page migration problem can achieve a competitive ratio against the oblivious adversary better than $\Omega(\max_{i,j\in M, S'\subseteq S}\{\Sigma_{s\in S'} f_{i,j}^{\{s\}}/f_{i,j}^{S'}\})$.

Proof. Let $i, j \in M$ and $S' \subseteq S$ be the locations and the largest commodity set that maximize $\Sigma_{s\in S'} f_{i,j}^{\{s\}}/f_{i,j}^{S'}$. Note that $|S'| \geq 2$ as if not, the ratio is at its minimum value of 1. Assume that all pages of S' are initially located at i (if not, we can force the online algorithm to move the pages by repeatedly requesting every page at i). Next, we construct a probability distribution over request sequences for which the expected competitive ratio of the best-performing deterministic online algorithm is at least $\Omega(\Sigma_{s\in S'} f_{i,j}^{\{s\}}/f_{i,j}^{S'})$. Then, according to Yao's Principle (see Theorem 1.1), the lower bound follows.

Let F be a sufficiently large number such that $F \geq f_{i,j}^{\{s\}}$ for all $s \in S'$. Let $x \geq \frac{F}{d(i,j)}$. We construct the input sequence as follows: For each commodity of S' uniformly and independently at random, assign it to one of the two sets S_i or S_j. In arbitrary order, consider each commodity of $s \in S'$ and request it x times on i (if $s \in S_i$) or on j (if $s \in S_j$). By requesting the element x many times, the algorithm has no advantage by not moving the page if it is not at the location of the respective requests. See Figure 2.2 for an explanation of why the online algorithm performs badly on the given input sequence.

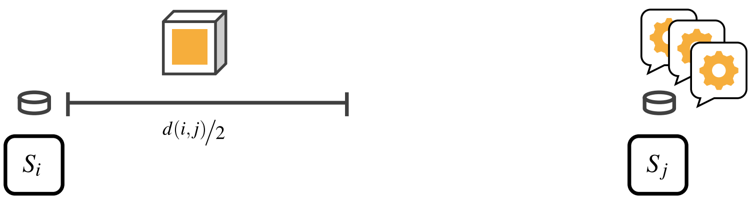

Figure 2.2: In the set-up of the lower bound, initially, all commodities of S' are at i. The optimal solution can move all commodities of S_j together to j. As the algorithm does not know the partition S_i, S_j, for each commodity, with a probability of 0.5, the algorithm's page (*gold*) is at least $d(i,j)/2$ from the location of the associated requests when they arrive. Either the page must be moved, or the algorithm pays a high cost for serving the incoming requests.

Note that S' contains S_j entirely and thus, $f_{i,j}^{S_j} \leq f_{i,j}^{S'}$ by the definition of f. The optimal solution moves all pages in S_j to j for a cost of $f_{i,j}^{S_j} \leq f_{i,j}^{S'}$. The algorithm has the following cost: Consider the page of commodity $s \in S'$. The algorithm might have moved s from i due to previous requests (although s was not requested yet). Let ℓ be the location of s in the algorithm's solution just before the requests for s arrive. If $d(\ell,i) \leq \frac{d(i,j)}{2}$, with a probability of $\frac{1}{2}$ the requests for s appear at j. By the triangle inequality, it holds that

$$d(i,j) \leq d(i,\ell) + d(\ell,j) \leq \frac{d(i,j)}{2} + d(\ell,j) \Leftrightarrow d(\ell,j) \geq \frac{d(i,j)}{2}.$$

If $d(\ell,j) \leq \frac{d(i,j)}{2}$, with a probability of $\frac{1}{2}$ the requests for s appear at i. Then, it holds that

$$d(i,j) \leq d(i,\ell) + d(\ell,j) \leq d(i,\ell) + \frac{d(i,j)}{2} \Leftrightarrow d(\ell,i) \geq \frac{d(i,j)}{2}.$$

Therefore, in general, with a probability of at least $\frac{1}{2}$, ℓ is in a distance of at least $\frac{d(i,j)}{2}$ from the location where the requests for s appear. Assume this case occurs and let without loss of generality be $d(\ell,i) \leq \frac{d(i,j)}{2}$. Consider the location ℓ' of s after x many requests for s. If the total movement of the algorithm during the x requests is at least $\frac{d(i,j)}{4}$, due to the triangle inequality, the total movement cost is at least $D_s \frac{d(i,j)}{4} = \frac{f_{i,j}^{\{s\}}}{4}$. Else, for any of the x requests, the algorithm's server is at least $\frac{d(i,j)}{4}$ apart of j and the cost for all x requests is at least $x\frac{d(i,j)}{4} \geq \frac{F}{d(i,j)} \frac{d(i,j)}{4} = \frac{F}{4} \geq \frac{f_{i,j}^{\{s\}}}{4}$.

Thus, in any case where ℓ is in a distance at least $\frac{d(i,j)}{2}$ from the location where the requests for s appear, the algorithm has a cost of at least $f_{i,j}^{\{s\}}/4$. The probability for this to happen is at least $\frac{1}{2}$, as explained earlier. Summed up over all $s \in S'$ the expected total cost of the algorithm is at least $\sum_{s\in S'} \frac{1}{2} \cdot \frac{f_{i,j}^{\{s\}}}{4} = \frac{1}{8} \sum_{s\in S} f_{i,j}^{\{s\}}$ and the theorem follows. ■

2.5 A Deterministic Algorithm

Next, we show how the trivial algorithm that considers each page separately meets the lower bound. Here, we compare the optimal solution with a derivation that never combines pages but moves them along the same paths.

Theorem 2.2 — Deterministic Upper Bound. Consider a deterministic algorithm for the multi-commodity online page migration problem that uses a page migration algorithm with a competitive ratio of c for each page separately, i.e., that never moves pages combined. The algorithm achieves a competitive ratio of at most $\mathcal{O}(c \max_{i,j\in M, S'\subseteq S}\{\sum_{s\in S'} f_{i,j}^{\{s\}}/f_{i,j}^{S'}\})$.

Proof. Consider an optimal solution $\textsc{Opt}$ and all its movements of pages. Let $\overline{\textsc{Opt}}$ be the solution that moves pages exactly as $\textsc{Opt}$, but never combines pages, i.e., every time, $\textsc{Opt}$ moves a set of pages S' from i to j, $\overline{\textsc{Opt}}$ moves each page of S' separately from i to j. For any such movement from i to j, $\textsc{Opt}$ pays at least $f_{i,j}^{S'}$ while $\overline{\textsc{Opt}}$ pays at most $\sum_{s\in S'} f_{i,j}^{\{s\}}$. The serving costs of $\textsc{Opt}$ and $\overline{\textsc{Opt}}$ are the same and thus $C_{\overline{\textsc{Opt}}}/C_{\textsc{Opt}} \leq \max_{i,j\in M, S'\subseteq S}\{\sum_{s\in S'} f_{i,j}^{\{s\}}/f_{i,j}^{S'}\}$.

Since $\overline{\textsc{Opt}}$ never combines pages and the proposed online algorithm $\textsc{Alg}$ achieves a competitive ratio of c for each commodity, $\textsc{Alg}$ has a competitive ratio of c against $\overline{\textsc{Opt}}$. Thus, $C_{\textsc{Alg}} \leq c \cdot C_{\overline{\textsc{Opt}}} \leq c \cdot \max_{i,j\in M, S'\subseteq S}\{\sum_{s\in S'} f_{i,j}^{\{s\}}/f_{i,j}^{S'}\} \cdot C_{\textsc{Opt}}$. ■

3. Multi-Commodity Online Facility Location

The following chapter considers the *multi-commodity online facility location problem* (abbreviated MCOFLP) as a generalization of the *facility location problem*. The model captures a resource allocation problem where the resources cannot be moved. Instead, a new resource instance (called *facility*) can be created at any location for a construction cost. Incoming requests are served by connecting them to a resource. In the spirit of the setting presented in the introduction (see Chapter 1), the model captures situations where migrating a virtual machine is not desired. However, new virtual machines can be instantiated anywhere.

In the multi-commodity case, we consider a set of commodities, for example, representing different services that can be deployed. When creating a new resource, an algorithm must determine the commodities the new instance supplies. Thereby, the set of chosen commodities influences the price of the creation. When multiple services (commodities) are instantiated simultaneously, several costs must be paid only once compared to separate instantiations. For example, only one virtual machine is required, so the overhead in managing machines shrinks (e.g., less memory is required, start-up time is improved, and less communication between machines has to be managed). Consequently, the cost of instantiating a set of services (commodities) is usually smaller than the cost of instantiating them separately, allowing an algorithm to benefit from a combined instantiation.

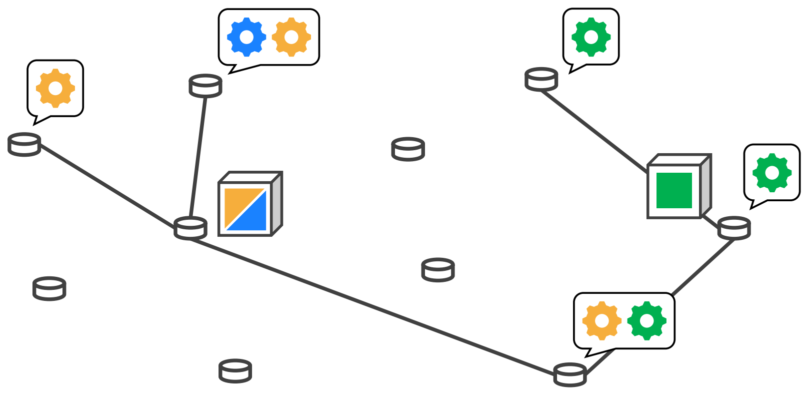

Figure 3.1: In *multi-commodity online facility location*, resources (called facilities) cannot be moved or migrated. Instead, at any location, a new facility (box) can be spawned. An algorithm has to decide which set of commodities (colors) to offer at the newly allocated facility. The chosen configuration (commodity set) influences the cost. Incoming requests (speech bubbles) must be connected (solid lines) to a set of facilities offering all desired commodities jointly.

Next, we frame the model of the MCOFLP formally as a resource allocation problem (see Definition 1.1) in Section 3.1. After that, we present related work to the model in Section 3.2. We proceed with an overview of our results in Section 3.3 followed by three sections laying out the details. In Section 3.4, we show that the introduced commodities significantly influence the performance in the online case. The provided lower bound depends on $\sqrt{|S|}$, where S is the set of commodities. Section 3.5 presents our algorithmic results. We show a deterministic and a randomized algorithm and analyze their competitive ratios. Here, we parameterize the bounds by a property of the construction cost function. In many natural cases, the parameter collapses to $\sqrt{|S|}$ and better. However, for some functions, we show that the parameter cannot be in $\mathcal{O}(\sqrt{|S|})$, i.e., our approach has its limits. Details can be found in Section 3.5.5. Finally, we present how the deterministic algorithm can be combined with a result of the literature to solve an even more general model where facilities must be leased for a certain time in Section 3.6. We additionally consider a more flexible leasing model and show a relation to the classical one.

Chapter Basis. The model, the algorithms, and the analyses build on the following publication:

The Online Multi-Commodity Facility Location Problem

Jannik Castenow, Björn Feldkord, Till Knollmann, Manuel Malatyali, and Friedhelm Meyer auf der Heide
In: Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2020), July 2020, Pages 129–139
Cf. [31]

The results of the publication are generalized as follows. We present a high-level algorithmic approach unifying the deterministic and the randomized algorithm and their analyses. Consequently, we frame the algorithms within it. Further, we loosen the condition required for the algorithms to work and present a definition solely dependent on the construction cost function. Contrasting our publication's algorithms that manage facilities that offer a single or all commodities, our generalized algorithms determine relevant commodity sets based on the construction function. We parameterize our analyses such that analyzing the construction cost function yields adaptive upper bounds by our algorithms. Based thereon, we present functions where the parameter is very low and functions where the parameter is higher than indicated by the lower bound. Furthermore, we combine our model with two leasing approaches and derive deterministic algorithms for them.

3.1 Problem Definition & Model

We define the multi-commodity facility location problem following Definition 1.1 as a resource allocation problem. Recapitulate that the task of an algorithm is to serve every request while minimizing the overall cost.

We consider any metric space (M,d) and a set of commodities S. The resources our algorithm manages are *facilities*. A facility is specified by its location plus the commodity set it offers. A set of commodities is also called a *configuration*. A request $r \in R$ consists of a location of M and the subset $s_r \subseteq S$ of the commodities it demands. We abuse the notation here and refer by r also to the location of a request $r \in R$. Request r is served by connecting it to a set of facilities jointly offering all commodities in s_r. The cost for serving request r is the sum over all distances to connected facilities. More formal, let F_r be the set of facilities that r is connected to, then the *connection cost* for r is $\sum_{f \in F_r} d(f,r)$, where f refers to the location of the respective facility. The actions an algorithm has are the construction of a new facility. When a facility is constructed at location $m \in M$ for a configuration $\sigma \subseteq S$, the *construction cost* is defined by a function f_m^σ. For all $m \in M$, we assume regarding f that for all $\sigma, \tau \subseteq S$ it holds that $f_m^{\sigma \cup \tau} \leq f_m^\sigma + f_m^\tau$ (*sub-additivity*), and for all $\sigma \subseteq S, \tau \subseteq \sigma$ it holds that $f_m^\sigma \geq f_m^\tau$ (*monotonicity*). Sub-additivity and monotonicity can be enforced by defining f_m^σ as the cheapest way to offer all commodities of σ at m. Assume then that sub-additivity is violated for some $m \in M$ and $\sigma, \tau \subseteq S$. Then it holds that $f_m^{\sigma\cup\tau} > f_m^\sigma + f_m^\tau$ and it is cheaper to construct two facilities, one for σ and one for τ at m, yielding a contradiction as $f_m^{\sigma\cup\tau}$ is the cheapest way to offer $\sigma \cup \tau$ at m. Assume that monotonicity is violated, i.e., for some $m \in M$ and $\sigma \subseteq S, \tau \subseteq \sigma$, it holds that $f_m^\sigma < f_m^\tau$. Then, the commodities of τ can be offered cheaper by offering σ yielding a contradiction.

An alternative cost model. One could also formulate the following alternative model for the serving cost. Assume that a single facility serves a request r multiple commodities at $m \in M$. In our model above, the connection of r to m is counted only once for the cost, reflecting that multiple commodities are served by a single communication path (incurring cost). One could argue that the connection should be counted separately per commodity of r that the facility serves. This model can be easily simulated in our model by replacing each request with $s_r \subseteq S$ by $|s_r|$ many requests demanding a single commodity each. Note that this possibly increases the sequence length by a factor of at most $|S|$ in the online case. However, when the number of requests n (before expanding requests) is large in comparison to the number of commodities $|S|$ and $|S|$ is polynomial in n, the competitive ratios of our algorithms increase only by a constant factor.

Comparison to classical facility location. Our model generalizes the classical metric facility location problem that can be instantiated with a single commodity, i.e., $|S| = 1$. In comparison, our model applies to situations where offering a single facility with multiple different commodities is cheaper than providing commodities separately by an adequate cost function f. On the algorithmic side, an algorithm not only has to determine *when* and *where* to open facilities but also *which set* of commodities to offer.

3.2 Related Work

The facility location problem originated from operations research and was already researched back in the 1960s, for example, in the works of [63, 72, 89]. The initial motivation was based on where to locate production plants or warehouses to minimize the construction cost while ensuring locality to clients, hence the name facility location problem. From the computer science perspective, the problem gained much attention in numerous variations. Our work considers the metric variant of the problem in the online version with facilities that can serve unlimited requests (*uncapacitated* case). Next, we outline selected important related work for the facility location problem.

Offline facility location. The metric facility location problem is known to be NP-hard which motivated a large body of literature on approximation algorithms for it. For an extensive overview of the techniques used, we refer to the survey by Shmoys [85]. For the non-metric case, the problem cannot be approximated better than by a factor of $\mathcal{O}(\log n)$ unless NP $\subseteq$ DTIME$(n^{\mathcal{O}(\log\log n)})$, due to a reduction to the set cover problem [40]. Complementary, an algorithm with an approximation factor of $\mathcal{O}(\log n)$ was presented early in [68].

For the metric case, on the one hand, it is known that the problem cannot be approximated better than a factor of 1.463 unless NP $\subseteq$ DTIME$(n^{\mathcal{O}(\log\log n)})$ [51]. On the other hand, a long history of publications led to the currently best approximation factor of 1.488 [66] by building on the previously best-known algorithm that achieves an approximation factor of 1.6774 [30]. As one can see, the gap between the lower and the upper bound is nearly closed.

The competitive ratio in the online case focuses not on the optimization of a constant but on the dependence on the sequence length n. Nevertheless, algorithms for the offline case inspired approaches for the online case. A noteworthy example is the primal-dual approach of Jain and Vazirani [55] that uses the linear program formulation of the problem to achieve a fast 3-approximation. The presented approach inspired online algorithms for facility location [47] and facility leasing [78].

Online facility location. The online version of the facility location problem was introduced by Meyerson [76]. Meyerson introduced the online case, presented a simple randomized online algorithm, and proved that the algorithm achieves a competitive ratio of $\mathcal{O}(\log n)$. By simple adaptations, the analysis can be improved to show that the competitive ratio is indeed in $\mathcal{O}(\frac{\log n}{\log\log n})$ as pointed out by Fotakis [48]. In addition, Meyerson presented a lower bound showing that no online algorithm can achieve a constant competitive ratio under adversarial order of requests. Shortly after that, Fotakis showed that the lower bound for both randomized and deterministic online algorithms is $\Omega(\frac{\log n}{\log\log n})$ [48]. Consequently, the competitive ratio of the randomized algorithm of Meyerson is asymptotically tight. Fotakis also introduced a deterministic online algorithm achieving a competitive ratio of $\mathcal{O}(\frac{\log n}{\log\log n})$. While this closes the gap between the lower and the upper bound, another deterministic algorithm achieving a competitive ratio of $\mathcal{O}(\log n)$ was also introduced [47]. The latter algorithm has the advantage of significantly smaller constants in the competitive ratio. Also, it is much simpler and easier to implement. For an overview of these results, see also [49].

Extended models. The facility location problem has been extensively studied concerning different model extensions. One first step to a generalization is analyzing the adversary's power in the online case. While under adversarial order, the lower bound is $\Omega(\frac{\log n}{\log\log n})$, Meyerson [76] showed that his randomized algorithm achieves a constant expected competitive ratio when the requests arrive randomly. More specifically, the competitive ratio is constant when an oblivious adversary constructs a request sequence which is then randomly permuted before the algorithm solves it. A more fine-grained model for the adversary, called the *t-bounded adversary*, was introduced by Lang [64]. Here, the adversary generates a sequence of requests brought to a random order. Afterward, the adversary can permute the sequence such that for each request, the number of requests that were originally before it while being behind it in the permutation is at most t. For $t = 1$, the input is purely random, while for $t = n$, it is strictly adversarial as the adversary can manipulate the order at will. Lang showed that for online facility location against a t-bounded adversary, the competitive ratio of randomized algorithms is $\Theta(\frac{\log t}{\log\log t})$.

Another approach to overcome the adversarial lower bound is introducing *mobility*. Here, when the online algorithm (but not the offline solution) is allowed to correct the facility positions by a small movement, the lower bound becomes independent of n and dependent on the cost for the movement [41, 42]. The improvement is possible as the online algorithm has more power than the offline solution, commonly known as *resource augmentation*.

The facility location problem was further considered in various extended models. For example, there is the *incremental* facility location problem, where facilities can be merged, and clients are reconnected [46]. In the *connected* facility location problem, connecting all open facilities in a Steiner tree adds to the total cost [83]. Further, there are *multi-level* facility location problems (see [79] for a survey). Notably regarding our results is the research on facility leasing, which is a combination of the facility location problem with the parking permit problem by Meyerson [77]. When opening a facility out of a fixed set of *leases* L with different costs, one has to be selected, dictating how long the facility remains open. The first deterministic algorithm for this problem achieving a competitive ratio of $\mathcal{O}(|L|\log n)$ was presented by Nagarajan and Williamson [78]. The algorithm is an extension of the primal-dual approach of Fotakis [47]. Our deterministic online algorithm for the MCOFLP is based on the same approach and can be combined with the leasing algorithm for a mixed model (see Section 3.6). In [59], Kling et al. presented how the bounds for facility leasing can be improved to be independent of n namely to $\mathcal{O}(\ell\log\ell)$, where ℓ is the length of the longest lease.

Multiple commodities. Ravi and Sinha introduced and considered the multi-commodity facility location problem in its offline metric version in [81]. By a reduction to the weighted set cover problem, they show that the general problem is MAX-SNP-hard. Additionally, one cannot approximate the problem better than by a factor of $\Omega(\log t)$, where t is the maximum number of commodities a facility can offer. Note that in the general case, $t = |S|$ and the problem cannot be approximated better than by a factor of $\Omega(\log |S|)$. The hardness result already applies to a natural set of construction cost functions called *linear functions*. For a linear function, the cost at any location $m \in M$ is $f_m^\sigma = f_m^0 + \sum_{e \in \sigma} f_m^{\{e\}}$ where f_m^0 is a fixed opening cost. Complementary to the lower bound, the authors present an approximation algorithm achieving an asymptotically optimal approximation factor of $\mathcal{O}(\log t)$ for linear functions. In the offline case, the construction cost function plays a major role in designing efficient algorithms. For example, in [81], for $f_m^\sigma = f_m$ for all σ, i.e., when the facility costs are independent of the offered configuration, the problem can be approximated with a constant factor.

A special related problem to the metric multi-commodity facility location problem is the *facility location with service installation cost problem* considered in [86]. Here, if a request for commodity e is connected to a facility, the facility must pay a one-time cost for offering commodity e. In the offline case, the model is equivalent to the multi-commodity facility location problem with linear cost functions mentioned above. The authors of [86] gave a 6-approximation based on a primal-dual approach for the problem under the following assumption: There is an ordering on all facilities such that if a facility at m is before one at m', then for all $s \in S$, it holds that $f_m^{\{s\}} \leq f_{m'}^{\{s\}}$. Compared to [81], the result shows how the construction cost function significantly influences the achievable approximation factor. Note that in the online case, the model is no longer equivalent to the MCOFLP as services can be installed *after* a facility is opened, whereas, in the MCOFLP, the set of offered commodities has to be determined immediately. Adding missing commodities to an existing facility costs as much as if the commodities were included upon construction. So, regarding the construction costs, there is no disadvantage when installing commodities later. For the online non-metric case, Markarian [73] presented an asymptotically optimal online algorithm. Regarding the non-metric offline case, Fleischer showed in [45] an algorithm with an approximation factor logarithmic in the number of requests, facility locations, and commodities. The algorithm is asymptotically optimal and works for the capacitated case as well.

Returning to the MCOFLP, our paper [31] was the first to consider the problem in its online variant. To the best of our knowledge, there were also no further results regarding the MCOFLP yet.

3.3 Our Results

Note that a trivial solution to the MCOFLP is to use any algorithm for the single-commodity case for each commodity separately. Such a solution only constructs facilities offering a single commodity. It results in an additional factor of $|S|$ in the competitive ratio in the worst case when offering all commodities is as cheap as offering a single one, i.e., $f_m^{\{s\}} = f_m^S$ for all $m \in M$ and all $s \in S$. We aim at designing algorithms that achieve a better performance regarding the dependence on $|S|$, i.e., the competitive ratio should be sub-linear in $|S|$.

Lower bounding the competitive ratio. Our first result is a lower bound presented in Section 3.4, indicating a possible sub-linear competitive ratio. However, no algorithm can avoid at least a term of $\sqrt{|S|}$ in the competitive ratio. While the term in the lower bound is *additive*, we aim at a *multiplicative* term on the algorithmic side, coming close to $\sqrt{|S|}$. Additive dependencies in the lower and multiplicative dependencies in the upper bound appeared in the literature in the past. For example, it can be observed in the *online facility leasing problem* [78].

Theorem 3.1 — Lower Bound. No randomized online algorithm for the multi-commodity online facility location problem can achieve a competitive ratio better than $\Omega\left(\sqrt{|S|} + \frac{\log n}{\log\log n}\right)$ against the oblivious adversary, even on a line metric.

The lower bound shows that, as in the single-commodity case, randomization does not significantly improve the competitive ratio. Mainly, the lower bound utilizes that any algorithm cannot correctly guess the exact set of commodities to offer at an initial location. After that, it cannot guess the correct place to which future requests converge. The optimal solution, of course, knows the required set of commodities and can take full advantage of combining all of them immediately at the beginning in one facility. An important implication of the input sequence used in the lower bound is that an algorithm, to take advantage of offering a combination of commodities in one facility, at some point needs to offer commodities that were not yet requested. We call such a behavior *prediction*. If an algorithm does not predict future commodities, it runs behind after the optimal solution and, in the worst case, only achieves a competitive ratio linear in $|S|$. The implication mentioned above motivates the main question for the design of online algorithms: "When and how shall we predict commodities?"

Parameterizing the dependence on the construction cost. On a high level, predicting a set $\sigma \subseteq S$ at a location $m \in M$ becomes reasonable when the algorithm already paid f_m^σ for past requests concerning commodities in σ. In a sense, the predicted facility is financed by past expenses of the algorithm. An algorithm that aims at managing all possible subsets of S as possible configurations is faced with an inherent difficulty: There are $2^{|S|}-1$ many such subsets. Selecting the optimal one for a request sequence is nearly impossible due to the many choices. Investing in all at once lets the cost of the algorithm explode. So, we design algorithms that restrict themselves to a carefully chosen set of possible configurations for facilities. Thus, we need to clarify: "Which of the $2^{|S|}-1$ many subsets of S shall we select?"

The answer lies in the given construction cost function. As a basis, in any case, we consider all single commodity configurations, i.e., configurations $\{s\}$ for all $s \in S$. Such configurations yield basic flexibility for our algorithms, for example, against instances where each request is only interested in a single fixed commodity of S. To determine further configurations, we inspect the construction cost function. For example, the function we use in the proof of Theorem 3.1 is tailored to maximize the resulting competitive ratio by making facilities for $\sqrt{|S|}$ commodities as cheap as facilities for one commodity. Here, an algorithm could select configurations of size $\sqrt{|S|}$. Naturally, the function here is a worst case. If the function is different, the problem gets far easier, and better competitive ratios become possible. An extreme example is a function that is always a constant for any configuration, such as $f_m^\sigma = 1$. Here, an algorithm should select the configuration S because it offers the most commodities (all) for the smallest cost.

For other construction cost functions, choosing an appropriate set of configurations to which the algorithm restricts itself becomes more complex. We develop the following Definition 3.1 that makes it possible to derive a policy on which configurations to consider solely based on the construction cost function. We first introduce the definition formally and afterward give intuitions on what it expresses.

Definition 3.1 — h-dividable. Let h_1 and h_2 be two functions dependent on $|S|$ and $h = h_1 \cdot h_2$. A cost function f is called *h-dividable*, if there is a set of configurations $S_1, \dots, S_x \subseteq S$ such that:

(a) For all $m \in M$ and all $\sigma \subseteq S$ with $|\sigma| > h_1$ there is a set $C_\sigma \subseteq S \cup_{1 \le y \le x} \{S_y\}$ with $|C_\sigma| \le h_1$ such that $\sigma \subseteq \cup_{\tau \in C_\sigma} \tau$ (C_σ *covers* σ) and $\sum_{\tau \in C_\sigma} f_m^\tau \le h_1 f_m^\sigma$.

(b) For all $e \in S$ it holds $|\{S_i \,|\, e \in S_i\}| \le h_2$.

R Technically, h, h_1, and h_2 are functions that depend on $|S|$ or are constant. To improve readability, we always write h, h_1, and h_2 instead of $h(|S|)$, $h_1(|S|)$, and $h_2(|S|)$.

From a technical perspective, Definition 3.1 ensures that the commodities of any fixed configuration σ at any location m can be covered by constructing facilities offering only configurations of $S \cup_{1 \le y \le x} \{S_y\}$. Thereby, the construction cost and the connection cost are each upper bounded by

h_1 times the optimal cost of f_m^σ. Consider Figure 3.2 for a graphical presentation of the construction cost property. Note that regarding the connection cost, the number of involved configurations for σ is at most $|C_\sigma| \le h_1$. Therefore, any request connected to at least one optimal facility can be connected to at most h_1 facilities offering C_σ. Additionally, due to (b) of Definition 3.1, each commodity is involved in at most $h_2 + 1$ configurations of the restricted set $S \cup_{1 \le y \le x} \{S_y\}$. Beyond the technical advantages, Definition 3.1 captures the following core intuition on the handling of most cost functions: Commodities can be grouped based on the cost function, such that any combination of commodities can be covered by groups offering at least the same set of commodities.

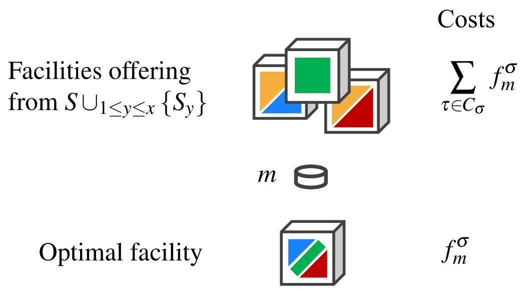
Costs
Facilities offering from $S \cup_{1 \le y \le x} \{S_y\}$
$\sum_{\tau \in C_\sigma} f_m^\sigma$
m
Optimal facility
f_m^σ

Figure 3.2: Condition (a) of Definition 3.1 ensures that for every $m \in M$, the cost for covering any configuration σ with facilities offering configurations from $S \cup_{1 \le y \le x} \{S_y\}$ is bounded by $h_1 \cdot f_m^\sigma$, where f_m^σ is the optimal cost.

R Constraint (a) of Definition 3.1 compares configurations at a single location. In general, for an optimal facility at location m, our algorithms construct facilities of $S \cup_{1 \le y \le x} \{S_y\}$ at locations *different* than m. Therefore, the condition that $|C_\sigma| \le h_1$ is crucial to bound the connection costs. We see in the proofs how we can relate the respective construction and connection costs to the cost of the same set of facilities at m. The latter is then related to the optimal cost.

As an example, in the realm of the introductory scenario (Chapter 1), assume a set of services that is rather cheap and one expensive set of services. The former can be simple services offering computational power or hardware resources without additional support. The latter could be complex services with valuable highly-complex techniques, e.g., machine learning or encryption services. When combining services within the cheap/expensive group, the cost of the combination scales with the size, i.e., the larger the combination, the smaller the cost per commodity. However, when mixing groups, the cost per commodity for all cheap services involved rapidly increases as soon as an expensive service is added. An algorithm that keeps both groups separate can compete well. Intuitively, when only cheap commodities are requested, there is no need to consider offering a configuration containing an expensive commodity. The algorithm should, at most, consider offering the entire cheap group in a facility. Offering a mix of cheap and expensive commodities comes with unnecessarily high costs. Suppose (also) expensive commodities are requested. In that case, the algorithm can consider offering all expensive commodities together at some point, as the optimal solution must also offer the expensive commodities. Again, offering cheap and expensive commodities together is no relevant option as the cost of separately offering cheap commodities is negligible compared to the (required) expensive commodities. Definition 3.1 allows us to determine the groups above and capture the idea of treating the groups separately. In addition, it quantifies the increase in cost due to restricting the algorithm to a few allowed configurations by h.

The main idea for our upcoming algorithmic approaches is to restrict algorithms to manage only facilities offering either a single commodity or a configuration of $S_1, \ldots, S_x$. Constraint (a) ensures that the connection and construction costs of such an algorithm can be at most a factor of h_1 apart from the optimal cost while covering the required commodities. Constraint (b) restricts the algorithm's number of prediction choices for a fixed commodity. Assume an algorithm is willing to spend a cost of c based on one request r and one of its commodities $e \in s_r$. Then the constraint ensures that even spending c for all configurations of $S_1, \ldots, S_x$ containing e results in a cost of at most $c \cdot h_2$.

Note that there is a trade-off between h_1 and h_2. Decreasing h_1 means an algorithm must be able to cover any commodity set with few configurations and a cost close to the optimal one. Consequently, at some point, the set of configurations that the algorithm allows must increase to ensure Constraint (a), implying an increase of h_2 due to Constraint (b). Similarly, a decrease of h_2 implies an increase of h_1. The parameter $h = h_1 \cdot h_2$ balances that trade-off. We aim to make h as small as possible, which is strongly limited by the given construction function.

Algorithmic solutions. Based on Definition 3.1 mentioned above, we design two algorithms for the MCOFLP. Both algorithms follow major design decisions based on well-established results for the single-commodity case. In Section 3.5.2, we elaborate on the main design and analysis ideas.

Our deterministic algorithm PD-MCOFLP is presented in Section 3.5.3 and achieves the competitive ratio stated in Theorem 3.2. It is based on Fotakis' deterministic algorithm for the single-commodity case [47], which achieves a competitive ratio of $\mathcal{O}(\log n)$. The algorithm follows a primal-dual approach; it maintains a set of dual variables that can be scaled to a feasible dual solution. We remark that Fotakis also presented a deterministic algorithm that achieves a slightly better competitive ratio of $\mathcal{O}(\frac{\log n}{\log\log n})$ [49]. Fotakis himself reasons that the primal-dual approach is simpler to understand and implement and achieves better constants in the analysis [47]. Therefore, we chose the algorithm of [47] as a base for our algorithm. In principle, we believe that following the high-level approach of Section 3.5.2 allows designing an adaptation of the other deterministic algorithm for the MCOFLP. Note that for the single-commodity case, PD-MCOFLP behaves exactly as Fotakis' algorithm [47].

From a technical perspective, we use the primal-dual scheme and show that the dual solution maintained by PD-MCOFLP can be scaled to a feasible one. We build upon the analysis of Fotakis' algorithm by Nagarajan and Williamson [78]. In the single-commodity case, there is a dual variable for each request, whereas the multi-commodity case introduces dual variables for each appearing request commodity tuple. Thus, variable assignments from the single-commodity case cannot easily be adapted for the multi-commodity case. Therefore, we carefully determine how much cost a request is willing to take for each of its commodities based on the present facilities and their configurations. As there are so many possible commodity sets for which dual feasibility must be ensured, we require new arguments in the analysis. Notably, we encounter special instances of the *weighted set cover problem* that we name *c-ordered covering*. More specifically, we require a solution to such instances to select a suitable set of inequalities when determining the scaling factor for the dual variables. Thus, we design and analyze a simple algorithm solving c-ordered covering sufficiently well in the analysis.

Theorem 3.2 — Deterministic Competitive Ratio. Given a h-dividable construction cost function, PD-MCOFLP achieves a competitive ratio of $\mathcal{O}(h \log n)$ for the multi-commodity online facility location problem.

Our randomized algorithm, RA-MCOFLP, is based on Meyerson's randomized algorithm for the single-commodity case [76]. We present RA-MCOFLP in Section 3.5.4. It achieves the competitive ratio stated in Theorem 3.3 against the oblivious adversary. Mainly, it determines probabilities for constructing facilities for each managed configuration that are carefully chosen to ensure bounded expected expenses while profiting from the expected prediction. In the single-commodity case, RA-MCOFLP behaves exactly like Meyerson's algorithm [76].

From a technical perspective, we build upon the analysis of Meyerson's algorithm [76] by Fotakis [49]. We extend the analysis to commodity sets and non-uniform metric spaces before we present arguments (similar to the ones of PD-MCOFLP) to relate the cost of the algorithm's configurations to optimal facilities. As for PD-MCOFLP, to avoid a factor of $|S|$ in the competitive ratio, we determine the probabilities for constructing facilities based on a carefully chosen measure of the cost a request is willing to take for each commodity. Seeing the similarities, we frame a high-level approach for both algorithms in Section 3.5.2 and present the algorithms respectively. On an intuitive level, the high-level approach outlines the general strategy to tame the multi-commodity case. We believe that it can be followed to derive other algorithms if desired, e.g., based on the deterministic algorithm for the single-commodity case with a competitive ratio of $\mathcal{O}(\frac{\log n}{\log\log n})$ by Fotakis [49].

Theorem 3.3 — Randomized Competitive Ratio. Given a h-dividable construction cost function, RA-MCOFLP achieves a competitive ratio of $\mathcal{O}\left(h\frac{\log n}{\log\log n}\right)$ against the oblivious adversary for the multi-commodity online facility location problem.

Both our algorithms achieve a competitive ratio dependent on h. While the competitive ratio is a worst-case measure, the beauty of our results and their parameterization in h is that more fine-grained bounds are offered if knowledge about the cost function is provided.

Results tailored to different cost functions. Our analysis allows us to directly derive bounds for specific cost functions by solely analyzing the cost function itself. There is no additional need for adapting the analyses of the algorithms. In Section 3.5.5, we show how various function classes imply different competitive ratios. As a starting point, we show that every construction cost function is $|S|$-dividable, i.e., in the worst case, our algorithms fall back to the trivial approach of handling commodities separately.

Theorem 3.4 frames the positive results we presented in [31] with regard to Definition 3.1. Here, the construction cost functions fulfill a condition (Condition 1 in the paper) that ensures the following. The more commodities are offered in a single facility, the better the construction cost normalized by the number of offered commodities. Slightly more general, the condition demands minimal cost per commodity when offering all commodities. For such a condition, the dependence on $|S|$ in the competitive ratio is $\sqrt{|S|}$, thus matching the dependence posed by the lower bound. Note that the function used to prove Theorem 3.1 fulfills the requirement. Therefore, the performance of our algorithms against the input sequence of the lower bound is asymptotically optimal regarding the dependence on $|S|$. Here, adapting the proof of Theorem 3.4 even allows us to avoid the factor of $c = 2$ in the competitive ratio. Note that for functions of Theorem 3.4, it suffices if the algorithm constructs facilities offering a single commodity or all commodities as in our publication.

Theorem 3.4 For $c \geq 1$, a cost function f where for all $\sigma \subseteq S$ and for all $m \in M$ it holds that $c \cdot \frac{f_m^\sigma}{|\sigma|} \geq \frac{f_m^S}{|S|}$ is $c \cdot \sqrt{|S|}$-dividable. Any sub-additive and monotone function solely dependent on the size of a configuration and the location fulfills the condition with $c = 2$.

The condition models situations where the commodities are somewhat similar, i.e., require the same capabilities. Then, adding multiple ones into a single facility is worthwhile because all commodities benefit from their shared requirements. For example, assume that the commodities are software services using the same libraries in the virtual machine scenario described in the introduction. Here, the more commodities are deployed into the same facility (virtual machine), the lower the cost per commodity. The integration of the shared libraries is only paid for once and amortizes over all deployed commodities.

Note that our definition of h-dividable allows for a far larger class of functions where $h \in \mathcal{O}(\sqrt{|S|})$. Examples are functions for groups of commodities where a group contains commodities that can be combined for a low price. For example, assume the commodity set can be split into at most $\mathcal{O}(\sqrt{|S|})$ many groups such that the commodities within a group can be combined at a low cost, but commodities of different groups cannot. Then the parameter h is in $\mathcal{O}(\sqrt{|S|})$ as well, and our algorithms have an asymptotically optimal dependence on $|S|$. In such cases, our algorithms intuitively offer facilities covering each group without configurations that cut multiple groups.

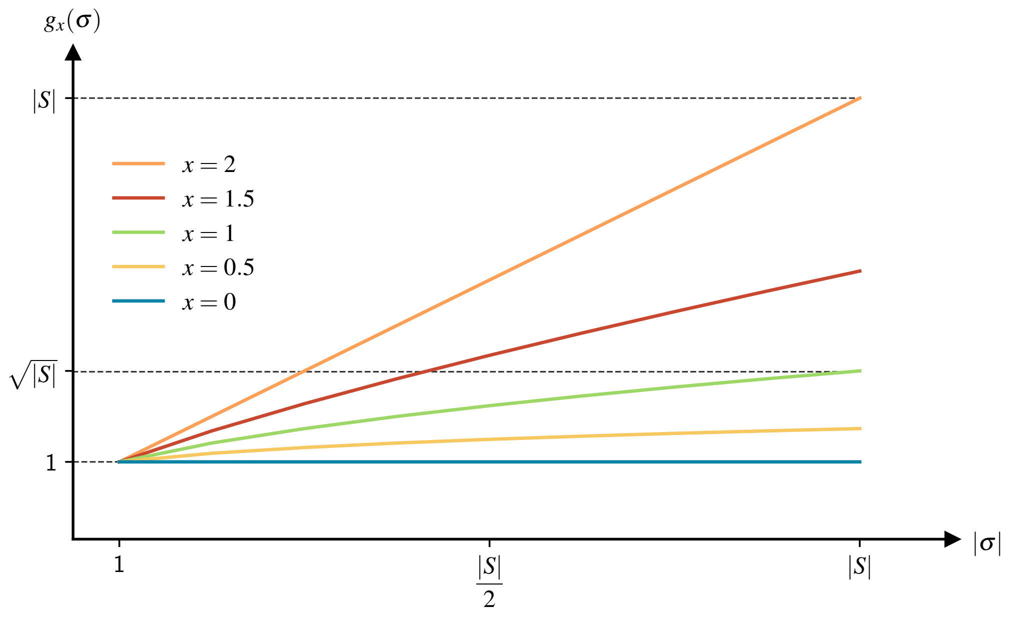

Figure 3.3: The figure sketches several examples for functions of the family $\mathcal{F}$. Note that the family expresses functions that behave like a square-root-function between a constant function ($x = 0$) and a linear function ($x = 2$).

As mentioned, the cost function is much more friendly in many cases, so competitive ratios with $h < \sqrt{|S|}$ are possible. Theorem 3.5 shows an upper bound for cost functions in a class $\mathcal{F}$ that captures monotone functions parameterized by x (not to be confused with the number of configurations of Definition 3.1). The construction cost function is a constant for $x = 0$. For $x = 2$, it is linear in the size of the offered configuration. In between, we have functions that are strictly monotonically increasing with the number of offered commodities while being sub-linear. Figure 3.3 shows example functions for several values of x. In [31], we stated results for functions of $\mathcal{F}$ only for the deterministic algorithm we presented back then. We see here how the arguments we used in the paper are really due to a property of the construction cost function and hold for both algorithms. As a side remark, the upper bound result for functions in $\mathcal{F}$ of Theorem 3.15 of our publication [31] is technically not correct because the argument on the scaling factor ignores the connections costs. The bounds of Theorem 3.5 are the correct ones.

Theorem 3.5 Consider the family of functions $\mathcal{F} = \{g_x(|\sigma|) = |\sigma|^{\frac{x}{2}} \,|\, x \in [0,2]\}$. A cost function f, where $f_m^\sigma = g_x(|\sigma|)$ for $g \in \mathcal{F}$, is $|S|^{\frac{x}{x+2}}$-dividable.

Observe how the competitive ratios follow the intuition that for small x, an algorithm can benefit a lot from combining commodities. In contrast, for increasing x, the problem's difficulty rises as a correct guess of the required commodity set becomes important. We complement the upper bound above with a lower bound for the same class of functions shown in Theorem 3.6.

Theorem 3.6 Consider the family of functions $\mathcal{F} = \{g_x(|\sigma|) = |\sigma|^{\frac{x}{2}} \,|\, x \in [0,2]\}$. For a cost function f, where $f_m^\sigma = g_x(|\sigma|)$ for $g \in \mathcal{F}$, every randomized online algorithm has a competitive ratio of at least $\Omega\left(\min\left\{\sqrt{|S|}^{\frac{2-x}{2}}, \sqrt{|S|}^{\frac{x}{2}}\right\} + \frac{\log n}{\log\log n}\right)$ against the oblivious adversary.

Figure 3.4 depicts a comparison of the dependence on S of the upper bound of Theorem 3.5 and the lower bound of Theorem 3.6. Our bounds diverge for larger values of x, likely due to the

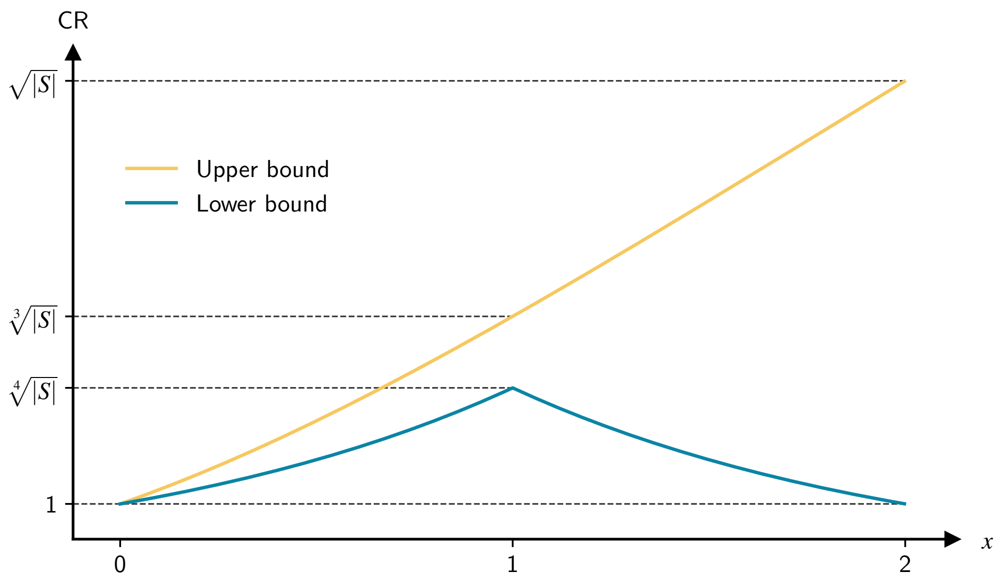

Figure 3.4: The competitive ratios by Theorems 3.5 and 3.6 can be compared regarding their dependence on $|S|$. For $x = 0$, the dependence on $|S|$ is a constant and optimal. For increasing x, the upper bound follows the lower bound up to $x = 1$. Here, the lower bound reaches a dependence of $\sqrt[3]{|S|}$ while the lower bound peaks at $\sqrt[4]{|S|}$. For $x > 1$, the lower bound falls until it is constant again. In the meantime, the upper bound increases to $\sqrt{|S|}$. The divergence is likely due to the lower bound ignoring increased connection costs even when the algorithm can cover a commodity set with a cost close to the optimal one.

following. For larger values of x, from the perspective of the construction cost, it is not worthwhile to consider configurations besides single commodity ones. Since the lower bound works for a single location, it shrinks. However, the lower bound does not consider connection costs, which are still important regarding larger values of x. The more facilities the algorithm constructs to cover an optimal facility, the higher the parameter h_1 and the competitive ratio. Thus, we believe the upper bound comes closer to reality than the lower one. In Section 3.5.5, we elaborate on this.

In the previous examples, it holds that $h \in \mathcal{O}(\sqrt{|S|})$. Unfortunately, we can also design a construction cost function where $h \notin \mathcal{O}(\sqrt{|S|})$. Theorem 3.7 shows thereby the limits of our approach. The function considers all commodity sets of size $|S|/2$. For each such configuration σ, there is exactly one location where σ costs 1 while everywhere else, it costs $|S|$. We show that it is impossible to select sufficiently few configurations such that Constraints (a) and (b) of Definition 3.1 hold simultaneously for $h \in \mathcal{O}(\sqrt{|S|})$. Theorem 3.7 implies that the approach of h-dividable cost functions is not sufficient. We see no straightforward way of improving our approach here and believe new techniques are required to deal with such functions.

Theorem 3.7 There is a construction cost function f and a metric space (M,d) such that there is no $h \in \mathcal{O}(\sqrt{|S|})$ for which f is h-dividable.

The function used for Theorem 3.7 exploits that the costs for a fixed configuration vary for different locations. The proof no longer works when the location does not influence the construction cost. We believe that in such cases, the construction cost function is always $\mathcal{O}(\sqrt{|S|})$-dividable. In Section 3.5.5, we elaborate on Conjecture 3.1.

Conjecture 3.1 Every construction cost function independent of the location is $\mathcal{O}(\sqrt{|S|})$-dividable.

Towards a flexible leasing model. The facility location problem is usually considered incremental, i.e., facilities never close. However, for some settings, a one-time payment to allocate a resource is unsuitable, and, at some point, a facility might be shut down again when not financed further. The facility leasing model introduced in [78] can model such settings. Here, whenever a facility is constructed, a fixed time interval has to be chosen during which it remains open. The available intervals, called *leases*, influence the construction cost. Regarding the MCOFLP, we present in Section 3.6 how our model can be extended regarding leasing. On the one hand, the lower bound Theorem 3.8 below shows the additional difficulty in the online case of choosing the correct lease length out of the set L of available leases. The bound holds for deterministic algorithms. Since the algorithm of Nagarajan and Williamson [78] for facility leasing is also deterministic, we focus on such algorithms. For randomized algorithms, a combination of Meyerson's bound from the parking permit problem [77] and our bound yields a competitive ratio of at least $\Omega(\log|L| + \sqrt{|S|} + {}^{\log n}/_{\log\log n})$ against the oblivious adversary.

Theorem 3.8 No deterministic online algorithm for the multi-commodity online facility leasing problem with fixed leases can achieve a competitive ratio better than $\Omega\left(|L| + \sqrt{|S|} + \frac{\log n}{\log\log n}\right)$, even on a line metric.

On the other hand, PD-MCOFLP can be combined with the deterministic algorithm for facility leasing presented in [78]. On a high level, the combination is possible because both are based on Fotakis' deterministic algorithm [47] and follow the primal-dual approach. The resulting bound is captured in Theorem 3.9.

Theorem 3.9 Given a h-dividable cost function (adapted to leasing), an adaptation of PD-MCOFLP for leasing achieves a competitive ratio of at most $\mathcal{O}(|L|\, h \log n)$ for the multi-commodity online facility leasing problem with fixed leases.

Further, we present in Section 3.6 an alternative leasing model where each facility has construction and maintenance costs. After a facility is constructed, it can be kept open each time step by paying the maintenance cost, or it is closed and can only be re-opened later by paying the construction cost again. This model allows for more flexible management of facilities which is especially convenient when an algorithm desires closing facilities. For the special case where construction costs are independent of the location, and the maintenance costs are equal for all locations and configurations, we show Theorem 3.10. We show the theorem by reducing the problem to a special case of the extension of the MCOFLP by facility leasing presented above. The reduction allows using the adapted algorithm of Theorem 3.9 with a slight change in the analysis.

Theorem 3.10 Consider construction costs independent of the location and maintenance costs equal for all locations and configurations. There is a deterministic algorithm for the multi-commodity online facility leasing problem with a maintenance cost that achieves a competitive ratio of at most $\mathcal{O}(h \log n)$ if the construction cost function is h-dividable.

3.4 The Lower Bound

In the following, we present our lower bound for the MCOFLP. Due to Fotakis [48], we know that the lower bound on the competitive ratio for MCOFLP with one commodity is $\Omega(\frac{\log n}{\log \log n})$ even on line metrics. Our lower bound, presented in Theorem 3.1 below, extends the classical one by a preceding phase in which the online algorithm has to guess the correct subset of commodities required by all requests. Intuitively, even at a single location, the additional difficulty of the multi-commodity case is that the required set of commodities is naturally unknown to the online algorithm. In contrast, the optimal solution can fully benefit from constructing a single facility with the correct set for the cheapest cost possible. The performance loss at a single location of any online algorithm, even a randomized one, significantly depends on S, as we see in the bound.

Theorem 3.1 — Lower Bound. No randomized online algorithm for the multi-commodity online facility location problem can achieve a competitive ratio better than $\Omega\left(\sqrt{|S|}+\frac{\log n}{\log \log n}\right)$ against the oblivious adversary, even on a line metric.

Proof. We construct a probability distribution over demand sequences for which the expected ratio between the costs of the deterministic online algorithm performing best against the distribution and the optimal cost is $\Omega(\sqrt{|S|}+\frac{\log n}{\log \log n})$. The lower bound follows due to Yao's Principle (see Theorem 1.1). Let ALG be a deterministic online algorithm and OPT an optimal offline algorithm. We first present a sequence for a single location $m \in M$ where ALG has a cost of $\Omega(\sqrt{|S|})$ while OPT has a constant cost. A combination with the sequence of the lower bound for the single-commodity case by Fotakis [48] yields a cost of $\Omega(\sqrt{|S|}+\frac{\log n}{\log \log n})$ for ALG and a constant cost for OPT. Repeating the combination at sufficiently many parts of the line yields the theorem. We first define a suitable function for the facility opening costs.

Facility opening costs. Let the cost function for location $m \in M$ be $g(|\sigma|) = f_m^\sigma = \lceil |\sigma| / \sqrt{|S|} \rceil$: i.e., the cost depends only on the size of the configuration. To improve readability, we assume that $\sqrt{|S|} \in \mathbb{N}$. Note that g (depicted in Figure 3.5) is sub-additive and monotone.

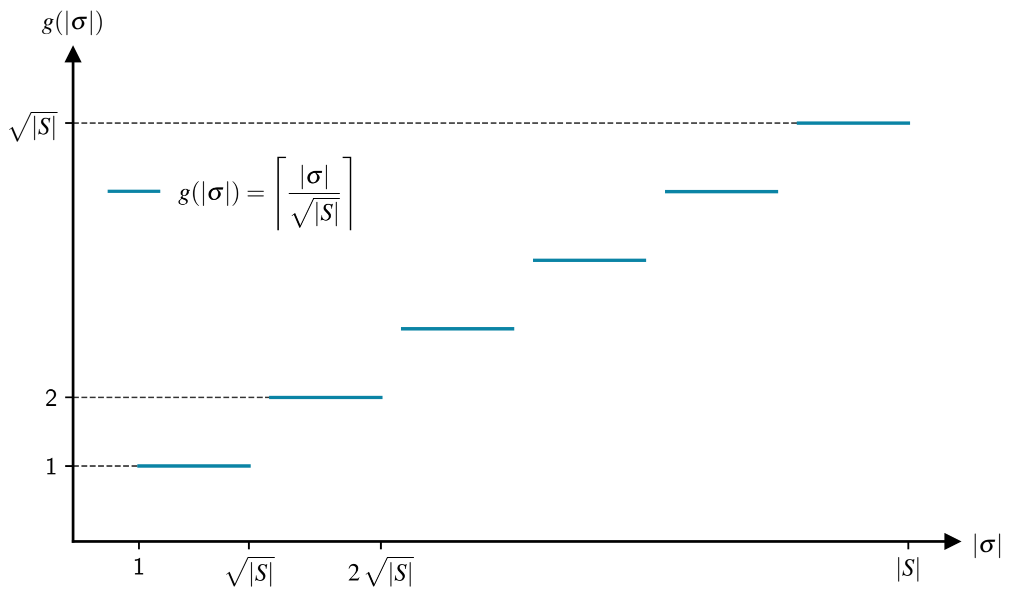

Figure 3.5: In the function used in the general lower bound for the MCOFLP, the cost solely depends on the configuration size and increases in steps of $\sqrt{|S|}$. Consequently, a reasonable algorithm offers only configurations of size $c \cdot \sqrt{|S|}$ for $c \in \mathbb{N}$.

Sequence definition. Consider a set $S' \subset S$ of the size $|S'| = \sqrt{|S|}$ containing randomly selected commodities. The commodities are selected uniformly and independently of each other. One at a time, trigger a request demanding one commodity in S' not yet requested at m.

Competitive ratio. An optimal algorithm builds a single facility serving all the commodities in S' at m. Hence, OPT pays no more than a total of $g(\sqrt{|S|}) = 1$. Contrary to OPT, ALG does not know the set S' until it has been revealed after $|S'| = \sqrt{|S|}$ requests. In each time step, ALG has to serve the commodity being requested and can additionally buy commodities to cover potential future requests. Observe that if ALG does not predict, i.e., it only includes commodities that were already requested when building a facility, it builds $\sqrt{|S|}$ facilities for a total price of $\sqrt{|S|}$.

We observe that ALG constructs facilities in *rounds*, where in the i-th round, a not yet covered commodity $s \in S'$ is requested, and ALG builds a facility serving s and t_i other commodities. t_i is entirely chosen by ALG, and some of the additionally covered commodities might be requested later. Note that we may assume that ALG does not build new facilities when an already covered commodity is requested. We can move whatever ALG then buys to the next round, and the rounds in which nothing happens can be removed from the following calculation. Let X be the number of rounds needed by ALG. Then the cost of ALG is determined by both X and $T := \sum_i t_i$, because ALG builds X facilities and covers T many commodities in total: i.e., the cost of ALG is at least $\max\{X, {}^{T}\!/\!\sqrt{|S|}\}$. Consider Figure 3.6 for a depiction. If $X \geq \sqrt{|S|}/2$, ALG's cost is at least $\sqrt{|S|}/2$. Therefore, assume that $X < \sqrt{|S|}/2$. Next, we show that in this case, T is large on expectation.

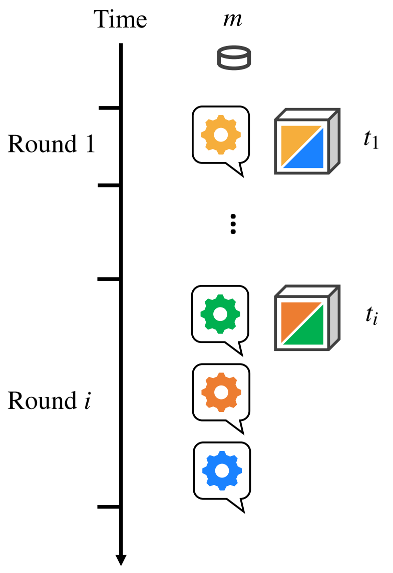

Figure 3.6: ALG's behavior can be separated in rounds $1,\ldots,X$ for a location $m \in M$. In round i, a not yet covered commodity (green) is requested and covered by a facility of ALG. Thereby, ALG covers t_i additional commodities (orange). ALG's cost is determined by X as well as $T := \sum_i t_i$, because it builds X facilities and covers at least T commodities. Note how additionally covered commodities (blue in round 1) might be requested first in future rounds (round i), lengthening them.

Let $S'_a \subset S'$ be the set of commodities that are not covered by ALG before they are requested: i.e., they are requested but not predicted by ALG. Similarly, let $S'_b = S' \setminus S'_a$ be the set of commodities that are requested and predicted by ALG. Observe that $|S'_a| = X < \sqrt{|S|}/2$ and $|S'_b| \geq \sqrt{|S|}/2$. Let $S_b = S \setminus S'_a$ be the total set of commodities out of which ALG predicts in total T many, including the ones in S'_b. Then $|S_b| = |S| - |S'_a| \geq |S| - \sqrt{|S|}/2 \geq |S|/2$. We are interested in bounding T. Since the commodities of S_b are indistinguishable for ALG, and they all have the same probability of being chosen for S' (unknown to ALG), ALG's decision on which commodities are predicted can be viewed as arbitrary and independent of the chosen S'_b. Thus, it is equivalent to model ALG's selection by assuming that ALG draws T times without replacement out of the set S_b and covers all commodities of S'_b. Let $c \geq 4$ be a constant. Observe that $|S'_b| \leq |S'| \leq \sqrt{|S|} \leq \frac{|S|}{c}$ for sufficiently large $|S| \geq 16$. Then the expected number of draws $\mathrm{E}[T]$ until S'_b is covered can be represented by

$$\begin{aligned}
\mathrm{E}[T] = \sum_{i=|S'_b|}^{|S_b|} \Pr[T=i]\cdot i \overset{\left(|S_b|\geq \frac{|S|}{2}\right)}{\geq} \sum_{i=|S'_b|}^{|S_b|} \Pr[T=i]\cdot i \overset{\left(|S'_b|\leq \frac{|S|}{c}\right)}{\geq} \sum_{i=|S|/c}^{|S|/2} \Pr[T=i]\cdot i \\
\geq \sum_{i=|S|/c}^{|S|/2} \Pr[T=i]\cdot \frac{|S|}{c} = \Pr\left[T \geq \frac{|S|}{c}\right]\cdot\frac{|S|}{c} \geq \Pr\left[T > \frac{|S|}{c}\right]\cdot\frac{|S|}{c}. \qquad (3.1)
\end{aligned}$$

Next, we show that with constant probability $|S|/c$ draws are not sufficient to cover S'_b: i.e., $T > |S|/c$. Assume that we draw exactly $|S|/c$ many times out of $|S|/2$ commodities of which $\sqrt{|S|}/2$ ones are requested. Let Y be the number of drawn requested commodities. Then Y is hypergeometrically distributed ($Y \sim \text{Hypergeometric}(|S|/2, \sqrt{|S|}/2, |S|/c)$) with mean $\mathrm{E}[Y] = \sqrt{|S|}/c$. In case $Y < \sqrt{|S|}/2$, not all commodities of S'_b are covered: i.e., the number of draws must be $T > |S|/c$. Since Y is hypergeometrically distributed we can apply the bounds of [36, 54] and get

$$\begin{aligned}
\Pr\left[T > \frac{|S|}{c}\right] &= \Pr\left[Y < \frac{\sqrt{|S|}}{2}\right] = 1 - \Pr\left[Y \geq \frac{\sqrt{|S|}}{2}\right] \\
&= 1 - \Pr\left[Y \geq \frac{\sqrt{|S|}}{c} + \frac{\sqrt{|S|}}{2} - \frac{\sqrt{|S|}}{c}\right] \\
&\overset{\left(\mathrm{E}[Y]=\frac{\sqrt{|S|}}{2}\right)}{=} 1 - \Pr\left[Y \geq \mathrm{E}[Y] + \frac{\sqrt{|S|}\,(c-2)}{2c}\right] \\
&= 1 - \Pr\left[Y \geq \mathrm{E}[Y] + \frac{(c-2)}{2\sqrt{|S|}}\,\frac{|S|}{c}\right] \\
&\geq 1 - \mathrm{e}^{-2\frac{(c-2)^2}{4|S|}\frac{|S|}{c}} = 1 - \mathrm{e}^{-\frac{(c-2)^2}{2c}} \overset{(c\geq 4)}{\geq} 1 - \mathrm{e}^{-\frac{1}{2}} \geq \frac{1}{4}. \qquad (3.2)
\end{aligned}$$

Combining Equation (3.1) and Equation (3.2) yields

$$\mathrm{E}[T] \geq \frac{|S|}{16}. \qquad (3.3)$$

Therefore, the expected cost for ALG is at least

$$\max\{X, g(\mathrm{E}[T])\} = \max\left\{X, \frac{\mathrm{E}[T]}{\sqrt{|S|}}\right\} \geq \frac{\sqrt{|S|}}{16}.$$

Recapitulate that OPT's cost is 1, and the cost of ALG for a single location is at least $\Omega(\sqrt{|S|})$. When we combine our sequence with the bound on MCOFLP with one commodity from Fotakis [48], the optimal solution has a cost of 2 while ALG pays at least $\Omega(\sqrt{|S|} + \frac{\log n}{\log\log n})$. Repeating the combined sequence sufficiently many times, the theorem holds. ■

Our lower bound motivates the usage of *prediction*. Any algorithm that aims at achieving a competitive ratio depending on $|S|$ by less than a linear factor has to offer commodities that were not yet requested at some point. Otherwise, one can easily force it to build $\Omega(|S|)$ facilities. At the same time, the optimal solution needs only a single one combining all necessary commodities for a cost that is a $1/|S|$ fraction of the algorithm's cost (with the choice of a suitable cost function). The main question for the algorithm design is now *how* to predict which commodities will be needed in the future.

3.5 Algorithmic Results

In the following section, we turn to our algorithmic results. We present two algorithms; a deterministic and a randomized one. Each achieves a competitive ratio dependent on the given construction cost function. For many natural construction cost functions, the algorithms' competitive ratios are $\mathcal{O}(\sqrt{|S|}\log n)$ (deterministic) and $\mathcal{O}(\sqrt{|S|}\frac{\log n}{\log\log n})$ (randomized), which is close to the lower bound of $\Omega(\sqrt{|S|}+\frac{\log n}{\log\log n})$ presented in Section 3.4.

Before we begin, we introduce additional notation in Section 3.5.1. Both algorithms are designed to manage only certain commodities based on the construction cost function to simplify the question *how* to predict future commodities implied by the lower bound. Therefore, before we present the concrete algorithms, we first present their common design approach in Section 3.5.2. Afterward, we present and analyze our deterministic algorithm PD-MCOFLP in Section 3.5.3. Our randomized algorithm RA-MCOFLP is presented and analyzed in Section 3.5.4. The competitive ratio of both algorithms is not simply dependent on S but is parameterized in a property of the construction cost function we call h-dividable. Therefore, we can derive varying competitive ratios for both algorithms by solely analyzing the construction cost function. We show multiple classes of cost functions for which we can derive specific competitive ratios in Section 3.5.5. There, a function where it holds that $h \notin \mathcal{O}(\sqrt{|S|})$ reveals the limits of our approach.

3.5.1 Additional Notation

We require some additional notation for the descriptions and analyses of our algorithms. First, in our notation, we usually omit an explicit mention of the time step. The point in time is usually clear from the context and is given implicitly by the current request we consider. Further, at any point in time, we denote by F the set of open facilities of the online algorithm. By $F(\sigma)$ for $\sigma \subseteq S$, we denote the subset of the open facilities of the online algorithm that offer all commodities of σ. In many cases, an algorithm needs to know the distance of a location $m \in M$ to the closest facility with configuration $\sigma \subseteq S$. We denote the smallest such distance by $d(F(\sigma), m)$. For better readability, we further abbreviate $\max\{0, a\}$ for any number a as $(a)_+ := \max\{0, a\}$.

3.5.2 High-Level Algorithmic Idea

On a high level, both our algorithms – the deterministic and the randomized one – operate as follows: When a request r with a set of desired commodities s_r appears, for each commodity $e \in s_r$, we define an *investment* $I_{(r,e)}$ that the algorithm is willing to allocate to serve commodity e of r. Usually, the investment is determined by considering the minimum cost to serve e of r based on the constructed facilities the algorithm has when r arrives. $I_{(r,e)}$ is then invested (multiple times) into (a) the construction of facilities and (b) the connection of r to a facility offering e. In the analyses, we show that the total investment $\sum_{r\in R}\sum_{e\in s_r} I_{(r,e)}$ bounds the cost of the algorithm and that the latter in turn is bounded by the cost of an optimal solution C_{OPT}. By that, a competitive ratio dependent on the previous bounds follows. One major design choice is now in which facilities an algorithm should invest, i.e., which sets of commodities are of interest for construction.

Here, we observe a trade-off; The number of configurations an algorithm considers influences the number of times the investment is spent. In turn, the more times the investment is spent, the higher the competitive ratio. Simultaneously, when allowing more configurations, the cost of covering any required (optimal) commodity set can approach the optimal one. Reducing the set of considered configurations shrinks the times the investment is spent but increases the gap between the cost of an optimal covering and the algorithm's covering of a commodity set. To grasp the above trade-off, we introduce the term of h-dividable cost functions (see Definition 3.1).

Definition 3.1 — h-dividable. Let h_1 and h_2 be two functions dependent on $|S|$ and $h = h_1 \cdot h_2$. A cost function f is called *h-dividable*, if there is a set of configurations $S_1, \dots, S_x \subseteq S$ such that:

(a) For all $m \in M$ and all $\sigma \subseteq S$ with $|\sigma| > h_1$ there is a set $C_\sigma \subseteq S \cup_{1 \leq y \leq x} \{S_y\}$ with $|C_\sigma| \leq h_1$ such that $\sigma \subseteq \cup_{\tau \in C_\sigma} \tau$ (C_σ *covers* σ) and $\sum_{\tau \in C_\sigma} f_m^\tau \leq h_1 f_m^\sigma$.

(b) For all $e \in S$ it holds $|\{S_i \,|\, e \in S_i\}| \leq h_2$.

Definition 3.1 allows exploiting structures in the construction cost function to determine a set of facility configurations an algorithm manages. The parameter h splits into two parameters h_1 and h_2. h_1 expresses the closeness to the optimal costs of the cost of covering any configuration when offering only the configurations dictated by Definition 3.1. These configurations are $S_1, \dots, S_x$ as well as all single commodity configurations, i.e., configurations of the set $S \cup_{1 \leq y \leq x} \{S_y\}$. h_2 expresses the times the investment is spent. Observe how $h = h_1 \cdot h_2$ expresses the aforementioned trade-off. When allowing only configurations of $S \cup_{1 \leq y \leq x} \{S_y\}$, a h-dividable cost functions allows bounding the competitive ratio in h. Consequently, the main remaining difficulty is determining low values for h. The latter is what we consider in Section 3.5.5.

Our deterministic algorithm PD-MCOFLP implements the mentioned ideas. It is based on the linear program formulation of the multi-commodity (offline) facility location problem and defines the $I_{(r,e)}$ respecting the constraints of the dual of the problem. Here, the $I_{(r,e)}$ can be interpreted as dual variables maintained by PD-MCOFLP. For the analysis, we show that rounding down all of them yields a feasible solution to the dual, giving us a lower bound on the optimal solution by weak duality (see Section 1.1).

For our randomized algorithm RA-MCOFLP, we also use the above approach and analyze the investment of RA-MCOFLP for a set of requests associated with the same optimal facility. Here, the algorithm invests the $I_{(r,e)}$ into probabilities for the construction of facilities offering commodity sets induced by Definition 3.1. The probabilities are chosen such that the expected investment spent is linear in the involved $I_{(r,e)}$.

3.5.3 A Deterministic Algorithm

In the following section, we present PD-MCOFLP and prove that it achieves a competitive ratio of $\mathcal{O}(h \log n)$ given that the construction cost function is h-dividable. Our algorithm follows the primal-dual approach (see Section 1.1). Therefore, we begin by showing the primal and the dual linear programs in Section 3.5.3.1. Afterward, we present PD-MCOFLP in Section 3.5.3.2 and prove the bound on the competitive ratio in Section 3.5.3.3.

3.5.3.1 The Linear Programs

Next, we present the primal and the dual linear programs representing the multi-commodity facility location problem and show how the dual can be simplified. The following linear program represents the problem.

Primal for the multi-commodity facility location problem

$$\min \sum_{m \in M} \sum_{\sigma \subseteq S} f_m^\sigma y_m^\sigma + \sum_{m \in M} \sum_{\sigma \subseteq S} \sum_{r \in R} \sum_{s \subseteq s_r} d(m,r) x_{mrs}^\sigma$$

$$\text{s.t.} \sum_{m \in M} \sum_{\sigma \subseteq S} \sum_{s \subseteq \sigma : e \in s} x_{mrs}^\sigma \geq 1 \qquad \forall r \in R, \forall e \in s_r$$

$$x_{mrs}^\sigma \leq y_m^\sigma \qquad \forall m \in M, \forall \sigma \subseteq S, \forall r \in R, \forall s \subseteq s_r$$

$$x_{mrs}^\sigma, y_m^\sigma \in \{0,1\} \qquad \forall m \in M, \forall \sigma \subseteq S, \forall r \in R, \forall s \subseteq s_r$$

Here, y_m^σ represents a variable indicating that at $m \in M$ there is a facility in configuration $\sigma \subseteq S$. x_{mrs}^σ indicates that the subset $s \subseteq s_r$ of commodities requested by request r is served by a facility at $m \in M$ in configuration σ. The first set of constraints ensures that for request r, every commodity is served by a facility connected to r. The second set of constraints ensures that requests are connected to and served only by facilities opened with a respective configuration.

Observe that, given fixed m, σ, r, the serving cost for serving any subset $s \subseteq s_r \cap \sigma$ by configuration σ at m is the same, namely $d(m,r)$. Therefore, it is safe to assume that it is always better to tackle x_{mrs}^σ for maximal $s \subseteq s_r \cap \sigma$, allowing us to eliminate explicitly reflecting s in x_{mrs}^σ. The linear program simplifies to:

Primal for the multi-commodity facility location problem (simplified)

$$\min \sum_{m \in M} \sum_{\sigma \subseteq S} f_m^\sigma y_m^\sigma + \sum_{m \in M} \sum_{\sigma \subseteq S} \sum_{r \in R} d(m,r) x_{mr}^\sigma$$

$$\begin{aligned}
\text{s.t.} \sum_{m \in M} \sum_{\sigma \subseteq S: e \in \sigma} x_{mr}^\sigma &\geq 1 && \forall r \in R, \forall e \in s_r \\
x_{mr}^\sigma &\leq y_m^\sigma && \forall m \in M, \forall \sigma \subseteq S, \forall r \in R \\
x_{mr}^\sigma, y_m^\sigma &\in \{0,1\} && \forall m \in M, \forall \sigma \subseteq S, \forall r \in R
\end{aligned}$$

Let $z_e^\sigma = 1$ if and only if $e \in \sigma$. The corresponding dual is then as follows.

Dual for the multi-commodity facility location problem

$$\max \sum_{r \in R} \sum_{e \in s_r} a_{re}$$

$$\begin{aligned}
\text{s.t.} \sum_{e \in s_r} a_{re} z_e^\sigma - d(r,m) &\leq b_{mr}^\sigma && \forall r \in R, \forall m \in M, \forall \sigma \subseteq S \\
\sum_{r \in R} b_{mr}^\sigma &\leq f_m^\sigma && \forall m \in M, \forall \sigma \subseteq S \\
a_{re}, b_{mr}^\sigma &\geq 0 && \forall e \in s_r, \forall r \in R, \forall m \in M, \forall \sigma \subseteq S
\end{aligned}$$

Recapitulate that we define $(a)_+ := \max\{a, 0\}$ for any number a. For $z_e^\sigma = 0$, the constraint $-d(m,r) \leq b_{mr}^\sigma$ is tautological as $d(m,r), b_{mr}^\sigma \geq 0$. So, the first set of constraints can be reduced to $\left(\sum_{e \in s_r \cap \sigma} a_{re} - d(r,m)\right)_+ \leq b_{mr}^\sigma$. Combined with the second set of constraints, this yields a simplified dual. We use the simplified dual formulation in the following description of our algorithm and the analysis.

Dual for the multi-commodity facility location problem (simplified)

$$\max \sum_{r \in R} \sum_{e \in s_r} a_{re}$$

$$\begin{aligned}
\text{s.t.} \sum_{r \in R} \left(\sum_{e \in s_r \cap \sigma} a_{re} - d(m,r) \right)_+ &\leq f_m^\sigma && \forall m \in M, \forall \sigma \subseteq S \\
a_{re} &\geq 0 && \forall r \in R, \forall e \in s_r
\end{aligned}$$

R We observe the similarity between the simplified dual for the multi-commodity case and the dual for the single-commodity case presented in [49]. The similarity again motivates us to base our algorithm on Fotakis' primal-dual algorithm [47].

3.5.3.2 The Algorithm

Our algorithm PD-MCOFLP is inspired by the primal-dual formulation of Fotakis' deterministic algorithm [47] presented in [78]. For the multi-commodity case, a non-trivial solution is necessary for deciding which configurations to offer at facilities and how to maintain the dual variables. On the technical side, our analysis introduces several new arguments to show the competitive ratio dependent on the cost function. Furthermore, due to the added commodities, special weighted set cover instances appear and must be solved in the analysis.

PD-MCOFLP works with any given function f for the construction cost that is h-dividable with regard to Definition 3.1. Let $S_1,\dots,S_x$ be the configurations of Definition 3.1 for f. Our algorithm only constructs and maintains facilities offering either any single commodity $s\in S$ or offering all commodities of a set S_y for $1\le y\le x$. When a request r with a commodity set s_r appears, the algorithm initializes variables $I_{(r,e)}$ with zero for each $e\in s_r$. $I_{(r,e)}$ corresponds to the investment defined in the high-level approach in Section 3.5.2. To determine how high this investment is, PD-MCOFLP considers the following constraints.

The constraints for PD-MCOFLP

$$(1)\quad I_{(r,e)} \le d(F(e),r) \qquad \forall e\in s_r$$

$$(2)\quad (I_{(r,e)}-d(m,r))_+ + \sum_{j\in R: e\in s_j} (\min\{I_{(j,e)}, d(F(e),j)\} - d(m,j))_+ \le f_m^{\{e\}} \qquad \forall m\in M, \forall e\in s_r$$

$$(3)\quad \sum_{e\in s_r\cap S_y} I_{(r,e)} \le d(F(S_y),r) \qquad \forall 1\le y\le x$$

$$(4)\quad \left(\sum_{e\in s_r\cap S_y} I_{(r,e)} - d(m,r)\right)_+ + \sum_{j\in R}\left(\min\left\{\sum_{e\in s_j\cap S_y} I_{(j,e)}, d(F(S_y),j)\right\} - d(m,j)\right)_+ \le f_m^{S_y} \qquad \forall m\in M, \forall 1\le y\le x$$

Note that all sets and distances are taken considering the time step in which r arrives.

The first two constraints consider facilities offering only e. Constraint (1) ensures that the investment stays below the cost to connect r to the closest facility offering e. Constraint (2) bounds the investment in the price to pay to construct a new facility offering only e at any location m and to connect r to this facility. For Constraint (2), not only r invests in the facility but also earlier requests that demanded e. The last two constraints consider all facilities offering a configuration S_y such that $e\in s_r\cap S_y$. Constraint (3) ensures that the combined investment for all $e\in s_r\cap S_y$ is smaller than the price to pay to connect r to the closest facility offering S_y. Constraint (4) is similar to Constraint (2) and ensures that the combined investment for all $e\in s_r\cap S_y$ is bounded by the price to construct a new facility offering S_y at any location m and to connect r to it. Again, for Constraint (4), not only r invests in the construction but also earlier requests which are affected by S_y.

For a depiction, consider Figure 3.7. For each configuration $\sigma\in S\cup_{1\le y\le x}\{S_y\}$ for which our algorithm manages facilities, we consider all locations $m\in M$ ensuring the following. The cumulated investment of the previous requests and the current one interested in a facility for σ at m becomes no larger than the respective construction cost f_m^σ. The requests construct the

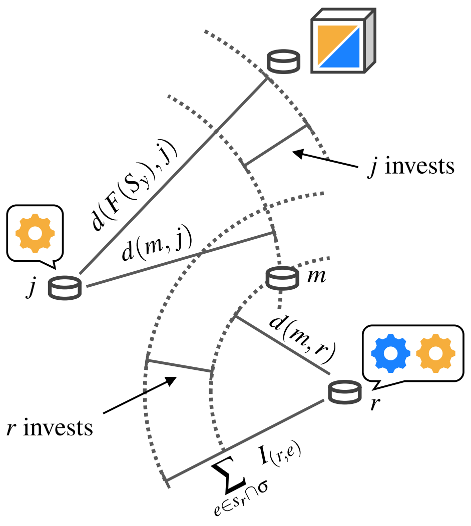

Figure 3.7: Consider the configuration $S_y = \{blue, gold\}$. r is the current request and j a previous one. The amount j invests into any point $m \in M$ is in the example given by the distance of j to the closest facility offering S_y minus the distance of j to m. The current request invests the total investment of all commodities affected by σ minus the distance of r to m.

respective facility if the construction cost is accumulated. The investment for each request r and each commodity $e \in s_r$ is determined as in the pseudo code.

PD-MCOFLP **on arrival of request** r **with commodity set** s_r

```
1:  while Not all $I_{(r,e)}$ for $e \in s_r$ are frozen do
2:      Simultaneously increase all $I_{(r,e)}$ that are not yet frozen
3:      if Constraint (1) is tight for $e \in s_r$ then
4:          Freeze $I_{(r,e)}$ and connect $r$ to the closest facility of $F(e)$
5:      if Constraint (2) is tight for $m \in M$ then
6:          Open a facility for $\{e\}$ at $m$
7:          Freeze $I_{(r,e)}$ and connect $r$ to the facility at $m$
8:      if Constraint (3) is tight for $S_y$ then
9:          Freeze all $I_{(r,e)}$ with $e \in s_r \cap S_y$ and connect $r$ to the closest facility of $F(S_y)$
10:     if Constraint (4) is tight for $m \in M$ and $S_y$ then
11:         Open a facility for $S_y$ at $m$
12:         Freeze all $I_{(r,e)}$ with $e \in s_r \cap S_y$ and connect $r$ to the facility at $m$
```

PD-MCOFLP does the following: All $I_{(r,e)}$ are simultaneously raised until one of the constraints below becomes tight. When a constraint becomes tight, the affected $I_{(r,e)}$ are frozen, and the algorithm keeps on increasing the remaining $I_{(r,e)}$ until eventually all of them are fixed. Intuitively, a tight constraint implies enough investment was accumulated to perform one of the following actions: When a constraint of type (1) or (3) becomes tight, PD-MCOFLP connects r to the closest facility considered in the constraint. When a constraint of type (2) or (4) becomes tight, PD-MCOFLP constructs a facility at the respective location in the configuration implied by the constraint and connects r to it. When multiple constraints become tight simultaneously, we have the following tiebreaks. When constraints for the same configuration σ at different locations become tight simultaneously, execute the action for any of them and ignore the rest. When constraints for different configurations become tight simultaneously, execute the actions for all of them.

As one can see here, $I_{(r,e)}$ is invested in the connection of r to a facility offering e (in all constraints when r is the current request). In addition, $I_{(r,e)}$ is (implicitly) invested in the construction of future facilities (in Constraints (2) and (4)) when r is the current request or a past request. Each $I_{(r,e)}$ can be invested once in the construction of facilities for every configuration in $S_1, \dots, S_x$ that offers e. Viewed differently, when a facility in configuration σ is constructed, its construction cost has been paid with by investments of request commodity tuples where the commodity is in σ. Constraints (2) and (4) bound the investment of earlier requests by the distance to the closest facility of a respective configuration. Therefore, $I_{(r,e)}$ is only invested once per configuration S_y that e is part of and once for facilities offering only e as we will see in the analysis.

By ensuring the constraints, PD-MCOFLP is a primal-dual algorithm. It maintains the dual-variables $I_{(r,e)}$, which are revealed over time (once the individual requests appear). While the variables maintained by PD-MCOFLP do not provide a feasible solution to the dual presented in Section 3.5.3.1, we will see in Section 3.5.3.3 that scaling the $I_{(r,e)}$ provides a feasible one.

3.5.3.3 The Analysis

Next, we analyze the competitive ratio of PD-MCOFLP. We proceed according to the high-level approach presented in Section 3.5.2 and similar to the analysis in [78]. First, following the analysis of [78], we show that the total cost of the algorithm is bounded within the maintained dual variables (the $I_{(r,e)}$), which is essentially enforced by the constraints. In our case, the parameter h_2 also plays a role in the proofs. In other words, we show that the total cost of the algorithm is bounded by $\mathcal{O}\left(h_2 \sum_{r\in R}\sum_{e\in s_r} I_{(r,e)}\right)$ by showing how $I_{(r,e)}$ is invested in the construction of facilities and the connection of r. Afterward, similar to the proof of Lemma 4.3 of [78], we aim at showing that scaling all $I_{(r,e)}$ by $\gamma = 1/(5h_1 H_n)$ yields a feasible solution to the dual problem presented in Section 3.5.3.1. Here, $H_n = \sum_{k=1}^{n} \frac{1}{k} \in \Theta(\log n)$ is the n-th harmonic number. By weak duality, a feasible solution to the dual is a lower bound to an optimal solution, and we can follow a competitive ratio of $\mathcal{O}(h_1 \cdot h_2 \log n) = \mathcal{O}(h \log n)$ as claimed in Theorem 3.2. Compared to [78], showing dual feasibility for the scaled variables is more challenging. In [78], the enforced constraints are rearranged to an inequality that is afterward applied for every request to generate the dual constraints. In our case, the introduced commodities change the constraints so that several inequalities can be derived that cannot simply be applied to arrive at the dual constraints. Instead, we show that selecting the right inequalities can be modeled by a special class of weighted set cover instances we call *c-ordered covering* problems. We present an algorithm for such problems and show that it achieves a sufficiently good solution. Then, we use it to show that an appropriate set of inequalities can be selected to arrive at the dual constraints again. In fact, we first only show that scaling all $I_{(r,e)}$ by the (smaller) factor $1/(5H_n)$ allows for dual feasibility for dual constraints involving configurations of $S \cup_{1\le y\le x} \{S_y\}$. Using these dual constraints and our definition of h-dividable cost functions we show that a scaling of the $I_{(r,e)}$ by $\gamma = 1/(5h_1 H_n)$ achieves the dual feasibility of the remaining configurations. So, the dependence on h_1 in the competitive ratio stems from the last step in the proof.

Bounding the algorithm's cost in the investment. First, we aim to limit the algorithm's cost by its investment. The cost of the algorithm is its cost of serving the requests plus the construction cost of facilities. The investment is determined by the constraints, which, in turn, relate each investment $I_{(r,e)}$ for a commodity e of request r to the distance to the nearest facility offering e. Therefore, we can analyze the serving cost of any request in terms of its investment as in Lemma 3.1.

Lemma 3.1 The serving cost of PD-MCOFLP is at most $(h_2+1)\sum_{r\in R}\sum_{e\in s_r} I_{(r,e)}$.

Proof. Consider a request r and a commodity $e \in s_r$. Consider all the facilities connected to r. For any such facility at $m \in M$ in configuration $\sigma \subseteq S$, the respective constraint must have become tight. If r is connected to a facility offering only $e \in s_r$, either Constraint (1) or (2) is tight. Then, the connection cost to that facility is either $d(F(e),r) = I_{(r,e)}$ (Constraint (1)) or $d(m,r) \le I_{(r,e)}$ (Constraint (2)). If r is connected to a facility offering multiple facilities, either Constraint (3) or (4) is tight for σ. In the case of Constraint (3), the connection cost to that facility is $d(F(\sigma),r) = \sum_{e\in s_r\cap\sigma} I_{(r,e)}$. In the case of Constraint (4), the connection cost is

$d(m,r) \leq \sum_{e \in s_r \cap \sigma} I_{(r,e)}$. In any case, the connection cost of r to a facility in configuration σ is thus at most $\sum_{e \in s_r \cap \sigma} I_{(r,e)}$. In how many constraints does a fixed $I_{(r,e)}$ appear? Due to Definition 3.1, e appears in at most h_2 many configurations of $S_1, \dots, S_x$. Additionally, it appears in the configuration $\{e\}$. Thus, a fixed $I_{(r,e)}$ appears in at most (h_2+1) constraints that became tight. Therefore, the total serving cost of r is at most $(h_2+1)\sum_{e \in s_r} I_{(r,e)}$. Summing up for all $r \in R$ yields the lemma.

■

Next, we show that the investment bounds the construction cost of the algorithm. Again the main argument is that the constraints hold and relate the construction cost to the investment. For a fixed request r and a commodity $e \in s_r$, we show that $I_{(r,e)}$ is invested in total only once into the construction of facilities for a fixed configuration managed by the algorithm. More specifically, $I_{(r,e)}$ is usually invested partly into many such facilities. So, the proof of Lemma 3.2 argues that as soon as a part of $I_{(r,e)}$ is invested into a facility of configuration σ, the investment in other facilities for σ shrinks accordingly.

Lemma 3.2 Fix a configuration $\sigma \in S \cup_{1 \leq y \leq x} \{S_y\}$. The construction cost for facilities in configuration σ of PD-MCOFLP is at most $\sum_{r \in R} \sum_{e \in s_r \cap \sigma} I_{(r,e)}$.

R Our proof follows the one of Lemma 4.1 [78] adapted to our notation and applied for every configuration of $S \cup_{1 \leq y \leq x} \{S_y\}$.

Proof. Throughout the proof, we only consider a request's bid towards facilities in configuration σ at all locations. The *bid* of a request is its contribution to the respective term in the sum of Constraints (2) and (4). We can ignore facilities of other configurations since any investment is handled once for all configurations of $S \cup_{1 \leq y \leq x} \{S_y\}$ separately. Observe that the construction cost for a facility in configuration σ is by Constraint (2) and (4) bounded by the sum of all bids of requests for commodities in σ. Any request r bids at most $\sum_{e \in s_r \cap \sigma} I_{(r,e)}$ towards the construction of a facility at a location $m \in M$ due to the minimum terms in Constraints (2) and (4). We show the following: When the bid a request r towards commodities of $s_r \cap \sigma$ is used to open a facility for σ at $m \in M$, all outstanding bids of r regarding $s_r \cap \sigma$ towards other facilities for σ are reduced by the amount r bids towards m.

Assume that there are two locations m_1, m_2 without a facility for σ. Before all commodities of $s_r \cap \sigma$ of r are served, r bids $\left(\sum_{e \in s_r \cap \sigma} I_{(r,e)} - d(m_1,r)\right)_+$ and $\left(\sum_{e \in s_r \cap \sigma} I_{(r,e)} - d(m_2,r)\right)_+$ towards σ at both locations. Assume a facility for σ opens at m_1 and r is connected to it. Then r had a positive bid on it. Then the bid of r regarding $s_r \cap \sigma$ towards m_2 reduces to $(d(m_1,r) - d(m_2,r))_+$. Thus, it was reduced by the amount spent for the facility at m_1:

$$\left(\sum_{e \in s_r \cap \sigma} I_{(r,e)} - d(m_2,r)\right)_+ - (d(m_1,r) - d(m_2,r))_+ = \left(\sum_{e \in s_r \cap \sigma} I_{(r,e)} - d(m_1,r)\right).$$

For the next part, see Figure 3.8 for an example. Assume next that all commodities of $s_r \cap \sigma$ are served and m_1, m_2 are two locations without a facility for σ. When a facility for σ at m_1 opens and the bid of r regarding $s_r \cap \sigma$ reduces, it reduces by $\left(\min\left\{\sum_{e \in s_r \cap \sigma} I_{(r,e)}, d(F(\sigma),r)\right\} - d(m_1,r)\right)_+$ which is greater than zero, i.e., m_1 is closer to r than any already open facility offering σ and $d(m_1,r) \leq \sum_{e \in s_r \cap \sigma} I_{(r,e)}$. We show that the bid of r regarding $s_r \cap \sigma$ at m_2 reduces by exactly this amount. Once the facility at m_1 for σ is opened, $\min\left\{\sum_{e \in s_r \cap \sigma} I_{(r,e)}, d(F'(\sigma),r)\right\} = d(F'(\sigma),r) = d(m_1,r)$, where F' denotes the new facility set containing the facility for σ at m_1. As a side note,

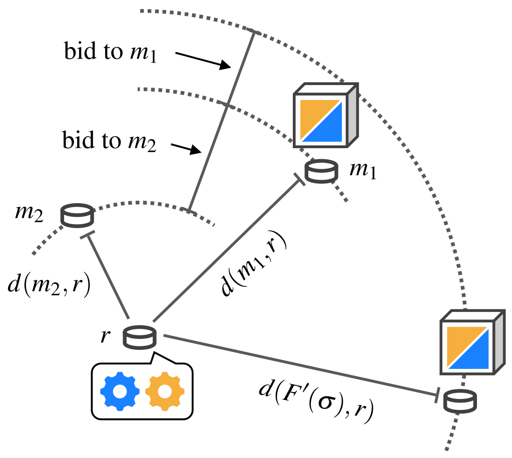

Figure 3.8: Assume that the investment of r for $\sigma = \{blue, gold\}$ is limited by $d(F'(\sigma),r)$ for the ease of explanation (the same arguments hold in the other case). Then due to the construction of a facility in σ at m_1, the amount r bids for σ at m_2 shrinks by the exact amount it bid to m_1.

when $d(m_1,r) \leq \sum_{e\in s_r\cap\sigma} I_{(r,e)}$ holds once, it holds for all future time steps because no facilities are deleted. The bid r spends for $s_r \cap \sigma$ towards m_2 reduces by:

$$\begin{aligned}
&\left(\min\left\{\sum_{e\in s_r\cap\sigma} I_{(r,e)}, d(F(\sigma),r)\right\} - d(m_2,r)\right)_+ \\
&\quad - \left(\min\left\{\sum_{e\in s_r\cap\sigma} I_{(r,e)}, d(F'(\sigma),r)\right\} - d(m_2,r)\right)_+ \\
&= \left(\min\left\{\sum_{e\in s_r\cap\sigma} I_{(r,e)}, d(F(\sigma),r)\right\} - d(m_2,r)\right)_+ - (d(m_1,r) - d(m_2,r))_+ \\
&= \left(\min\left\{\sum_{e\in s_r\cap\sigma} I_{(r,e)}, d(F(\sigma),r)\right\} - d(m_1,r)\right)_+ .
\end{aligned}$$

From the above, this is precisely the amount spent towards m_1. Thus, for any request r:

1. The maximum bid invested in a facility for σ is at most $\sum_{e\in s_r\cap\sigma} I_{(r,e)}$.
2. Whenever a facility for σ is constructed with an investment of r, the bids towards other facilities for σ reduce by the amount r bid towards the constructed facility.

Summing up all requests, the lemma follows. ■

Until now, we bounded the serving cost of a request and the construction cost regarding a fixed configuration that the algorithm manages. Next, we use that each commodity $e \in s_r$ of a request r is in at most h_2 many such configurations. Then we can show Lemma 3.3 below, bounding the total cost of PD-MCOFLP by the investments of requests and h_2.

Lemma 3.3 The total cost of PD-MCOFLP is at most $2\,(h_2+1)\,\sum_{r\in R}\sum_{e\in s_r} I_{(r,e)}$.

Proof. The total cost of the algorithm is given by its serving and construction cost. By Lemma 3.1, the serving cost of PD-MCOFLP is at most $(h_2+1)\sum_{r\in R}\sum_{e\in s_r} I_{(r,e)}$. For the construction cost, Lemma 3.2 gives us a cost of at most $\sum_{r\in R}\sum_{e\in s_r\cap\sigma} I_{(r,e)}$ for all facilities of a configuration $\sigma \in S \cup_{1\leq y\leq x} \{S_y\}$. By Definition 3.1, every $e \in S$ is in at most h_2 configurations in $S_1,\dots,S_x$. Thus, every $e \in S$ is in at most (h_2+1) configurations of $S\cup_{1\leq y\leq x}\{S_y\}$. Therefore, every $I_{(r,e)}$ also appears in at most (h_2+1) configurations of $S\cup_{1\leq y\leq x}\{S_y\}$ and the total construction cost is bounded by $(h_2+1)\sum_{r\in R}\sum_{e\in s_r} I_{(r,e)}$. ■

Solving c-ordered covering. Before we proceed with the analysis, we introduce a special class of the weighted set cover problem. A good solution to instances of this class is needed in the proof of Lemma 3.8. Our instances are defined below, and we aim at finding a minimal weight covering of the set $\{1,\dots,n\}$. Formally, c-ordered covering is defined as in Definition 3.2.

Definition 3.2 — c-ordered Covering. Consider elements $1,\dots,n$ and a given parameter $c \geq 1$. c-ordered covering is defined as finding a covering of all elements $1,\dots,n$ using the following sets (given as input): For every $i = 1,\dots,n$, there is a set $\{i\}$ with weight $\frac{c}{|B_i|+1}$ and a set $\{i\} \cup A_i$ with weight c, where $A_i, B_i \subseteq \{1,\dots,i-1\}$ are defined as follows. For element i, A_i and B_i are arbitrary subsets of $\{1,\dots,i-1\}$ such that $A_i \cap B_i = \emptyset$, $A_i \cup B_i = \{1,\dots,i-1\}$, and for any two elements i and j with $i < j$ it holds $B_i \subseteq B_j$.

An instance of c-ordered covering can be depicted as in Figure 3.9. Note the special structure: When regarding the elements i in increasing order, both sets A_i and B_i are disjoint and supply all elements $1,\dots,i-1$. From i to $i+1$, the B set can only grow, while the A set can lose elements to the B set. For the covering, one can (for each i) either choose to cover solely i with a cost that shrinks as B grows, or one can cover i and all elements of A_i with a fixed weight. Roughly, the more B grows, the less worthwhile it is to choose the latter option, whereas if B stays small for many consecutive elements, one should prefer the latter option.

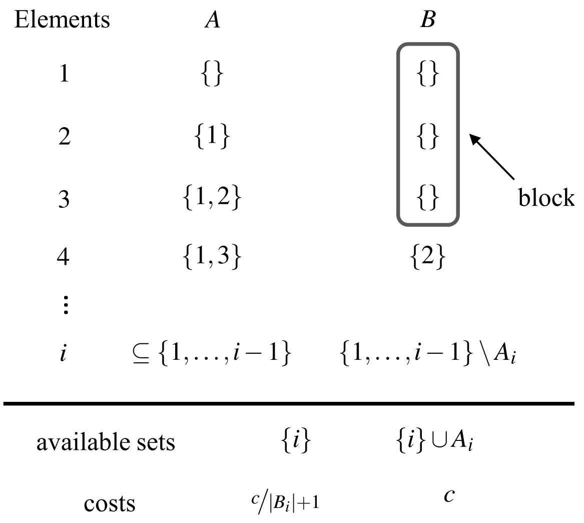

Figure 3.9: Consider the figure for an example of a c-ordered covering instance. For every element $i \in \{1,\dots,n\}$, there are disjoint sets A_i and B_i composing $\{1,\dots i-1\}$. With increasing i, the sets B_i can only grow. A consecutive sequence of sets B of equal contents is called *block*. For element i, there are two sets available; a cheap set containing only i for a price dependent on $|B_i|$ (and, thus $|A_i|$), and an expensive set containing i and A_i for a price of c.

We show that a greedy approach can always achieve a covering with a weight of at most $2cH_n$, where $H_n \in \Theta(\log n)$ is the n-th harmonic number. For this, we introduce additional notation. We call a set of elements $\{i,\dots,j\} \subseteq \{1,\dots,n\}$ with $i \leq j$ of maximum cardinality a *block* if $B_i = B_j$. For convenience, we say an element i covers the elements in A_i. Note that the sets B_i do not change within a block. Thus, each element covers all previous elements in its block and possibly more elements. Our proof consists of the following two steps:

1. Given a c-ordered covering instance of length n, we can cover $x \geq 1$ elements with a total weight of $2c\sum_{i=n-x}^{n}\frac{1}{i}$ (Lemma 3.4).
2. Given a c-ordered covering instance of length n, the x previously covered elements can safely be removed from the instance, and we can create a new ordered covering instance of length $n-x$ (Lemma 3.5).

It inductively follows that the set $\{1,\dots,n\}$ can be covered by a c-ordered covering instance with a weight of $2cH_n$ (Lemma 3.6). We start by Lemma 3.4, where we define a simple process to cover at least one of the last elements in the sequence. Interestingly, the covering itself is achieved by a simple greedy algorithm that goes from back to front through the instance.

Lemma 3.4 Given a c-ordered covering instance of length n, we can cover $x \geq 1$ elements with a total weight of $2c\sum_{i=n-x}^{n}\frac{1}{i}$.

Proof. Consider the following two choices covering at least the last block's elements.

1. Select the set $\{n\} \cup A_n$ with a weight of c.
2. For every element i of the last block, select the set $\{i\}$ with a weight of $c/(|B_n|+1)$ each.

Observe that the set of case 1 covers $n-|B_n|$ elements. Hence, the weight per covered element in case 1 is $c/(n-|B_n|)$. In case 2, the weight per covered element is $c/(|B_n|+1)$. Select one of the two choices depending on which is cheaper per element. Now, the weight per selected element is bounded by

$$\min\left\{\frac{c}{n-|B_n|}, \frac{c}{|B_n|+1}\right\} \overset{\left(\max\{n-|B_n|,|B_n|+1\}\geq\frac{n}{2}\right)}{\leq} \frac{2c}{n}.$$

Assume x elements were covered. Then the total weight for covering them is

$$\sum_{i=n-x}^{n}\frac{2c}{n} \overset{(i\leq n)}{\leq} 2c\sum_{i=n-x}^{n}\frac{1}{i}.$$

■

Next, we show that when at least the last element of a c-ordered covering instance is covered, the instance can be shrunk to a c-ordered covering instance of shorter length.

Lemma 3.5 Given a c-ordered covering instance of length n, the element n and $x \geq 0$ arbitrary elements that are covered by it can be removed from the instance. We can transform the remaining instance into a new c-ordered covering instance of length $n-x-1$.

Proof. Observe that the element n can safely be removed from the instance by simply deleting the sets $\{n\}$ and $\{n\} \cup A_n$. Any other element i that is covered by n is not in any B_j for all $j = 1,\ldots,n$. Removing i (including the sets $\{i\}$ and $\{i\} \cup A_i$) thus does not influence any B_j, such that the following still holds:

- The weights of all remaining sets are untouched.
- For all remaining j: $A_j \cap B_j = \emptyset$.
- For all remaining elements j and k with $j < k$: $B_j \subseteq B_k$.

The condition that for all remaining j it has to hold $A_j \cup B_j = \{1,\ldots,j-1\}$ is violated due to the removal of i. However, it can easily be fixed by consistently renaming every element $j > i$ to $j-1$. The resulting instance is a c-ordered covering instance not containing i. The described procedure can be repeated for $x > 1$ arbitrary elements covered by n, resulting in a c-ordered covering instance of length $n-1-x$. ■

As described above, the implication is that every c-ordered covering instance can be covered from back-to-front such that the total weight of the covering is bounded as in Lemma 3.6.

Lemma 3.6 The set $\{1,\ldots,n\}$ can be covered by a c-ordered covering instance with a weight of $2cH_n$.

Proof. By Lemma 3.4, we can cover x elements of a c-ordered covering instance of length n with a weight of $2c\sum_{i=n-x}^{n}\frac{1}{i}$. Since the last element covers all covered elements, the covered elements can safely be removed by Lemma 3.5, yielding a c-ordered covering instance of length $n-x$. Repeatedly applying Lemma 3.4 and Lemma 3.5 yields a covering of $\{1,\ldots,n\}$ with a weight of $2c\sum_{i=1}^{n}\frac{1}{i} = 2cH_n$. ■

Bounding the total investment in C_{OPT}**.** With the notion of c-ordered covering and a solution to instances of it, we are ready to show that scaling all $I_{(r,e)}$ by $\gamma = 1/(5h_1 H_n)$ results in a feasible solution to the simplified dual of Section 3.5.3.1. Since by weak duality, any solution to the dual is a lower bound on the optimal solution of the primal, $\sum_r \sum_{e\in s_r} \gamma I_{(r,e)}$ is a lower bound to the cost of the optimal solution C_{OPT}. First, we present a generalization of Lemma 4.2 of [78] tailored to our multi-commodity setting. This lemma will help us analyze the maintained variables $I_{(r,e)}$ to show that scaling them by γ achieves dual feasibility.

Lemma 3.7 Consider a configuration $\sigma \in S \cup_{1\leq y\leq x} \{S_y\}$ and two requests j, r such that j arrives before r. At the time when the $I_{(r,e)}$ variables are increased, it holds for all $m \in M$ that $d(F(\sigma), j) - d(m, j) \geq \sum_{e\in s_r\cap\sigma} I_{(r,e)} - d(m,r) - 2d(m, j)$.

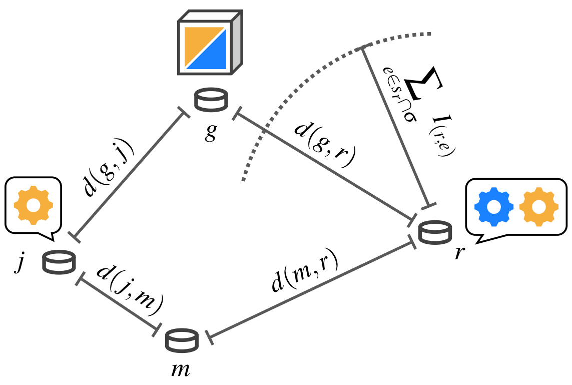

Figure 3.10: When r arrives, the facility for $\sigma = \{blue, gold\}$ is open at g. Both commodities of r can be served when r connects to g. Therefore, the total investment ($\sum_{e\in s_r\cap\sigma} I_{(r,e)}$) of r into σ is at most $d(g,r)$ that can be bounded by using the triangle inequality in $d(g, j) + d(j,m) + d(m,r)$ for any previous request j and location m.

Proof. For a depiction, consider Figure 3.10. Consider the facility at $g \in F(\sigma)$ closest to j when $\sum_{e\in s_r\cap\sigma} I_{(r,e)}$ is increased. Notice that $\sum_{e\in s_r\cap\sigma} I_{(r,e)} \leq d(g,r)$, because g is open and we could have connected r to g to serve all commodities in $s_r \cap \sigma$ (based on Constraint (1) or (3) dependent on σ). By the triangle inequality, it holds that

$$\begin{aligned} \sum_{e\in s_r\cap\sigma} I_{(r,e)} &\leq d(g,r) \leq d(g,j) + d(j,m) + d(m,r) \\ \Rightarrow \qquad d(g,j) - d(m,j) &\geq \sum_{e\in s_r\cap\sigma} I_{(r,e)} - d(m,r) - 2d(m,j). \end{aligned}$$

At the time we increase the $I_{(r,e)}$ it holds that $d(F(\sigma), j) - d(m, j) = d(g, j) - d(m, j)$ by definition of g. Together, this yields

$$d(F(\sigma), j) - d(m, j) = d(g, j) - d(m, j) \geq \sum_{e\in s_r\cap\sigma} I_{(r,e)} - d(m,r) - 2d(m, j).$$

■

Next, we aim to show that scaling the $I_{(r,e)}$ provides a feasible solution to the dual. We have to show that all the constraints in the simplified dual presented in Section 3.5.3.1 hold if we set $a_{re} := \gamma I_{(r,e)}$. We continue by showing for the set of configurations that are constructed by PD-MCOFLP the slightly sharper claim that dual feasibility is achieved for $a_{re} := 1/(5H_n) I_{(r,e)}$. The notion and the analysis of c-ordered covering are used for the following proof.

Lemma 3.8 Fix a configuration $\sigma \in S \cup_{1 \le y \le x} \{S_y\}$. For any $R' \subseteq R$ and any facility serving σ at any $m \in M$ it holds that

$$\sum_{r \in R'} \left(\sum_{e \in s_r \cap \sigma} \frac{I_{(r,e)}}{5H_n} - d(m,r) \right)_+ \le f_m^\sigma .$$

Proof. First, consider any request $r \in R'$ with $s_r \cap \sigma \neq \emptyset$ and only those requests in R' that appeared before r at the time when r arrives, and its investments are increased. All other requests do not influence the variables $I_{(r,e)}$ for $e \in s_r \cap \sigma$. Due to Constraint (2) (if σ is a single commodity) and Constraint (4) (if σ consists of multiple commodities), it holds for any location $m \in M$ that:

$$f_m^\sigma \ge \left(\sum_{e \in s_r \cap \sigma} I_{(r,e)} - d(m,r) \right)_+ + \sum_{j \in R'} \left(\min\left\{ \sum_{e \in s_j \cap \sigma} I_{(j,e)}, d(F(\sigma), j) \right\} - d(m,j) \right)_+ .$$

Let A_r be the set of requests of R' for which $\min\{\sum_{e \in s_j \cap \sigma} I_{(j,e)}, d(F(\sigma), j)\} = \sum_{e \in s_j \cap \sigma} I_{(j,e)}$ and analogously, let B_r be the set of requests of R' for which $\min\{\sum_{e \in s_j \cap \sigma} I_{(j,e)}, d(F(\sigma), j)\} = d(F(\sigma), j)$. For requests in B_r, we can apply Lemma 3.7, because all considered facilities serve σ. Therefore,

$$\begin{aligned}
f_m^\sigma \quad &\ge \quad \left(\sum_{e \in s_r \cap \sigma} I_{(r,e)} - d(m,r) \right)_+ + \sum_{j \in A_r} \left(\sum_{e \in s_j \cap \sigma} I_{(j,e)} - d(m,j) \right)_+ \\
&\quad + \sum_{j \in B_r} \left(d(F(\sigma), j) - d(m,j) \right)_+ \\
&\overset{\text{Lemma 3.7}}{\ge} \left(\sum_{e \in s_r \cap \sigma} I_{(r,e)} - d(m,r) \right)_+ + \sum_{j \in A_r} \left(\sum_{e \in s_j \cap \sigma} I_{(j,e)} - d(m,j) \right)_+ \\
&\quad + \sum_{j \in B_r} \left(\sum_{e \in s_r \cap \sigma} I_{(r,e)} - d(m,r) - 2d(m,j) \right)_+ \\
&= \quad \left(\sum_{e \in s_r \cap \sigma} I_{(r,e)} - d(m,r) \right)_+ + \sum_{j \in A_r} \left(\sum_{e \in s_j \cap \sigma} I_{(j,e)} - d(m,j) \right)_+ \\
&\quad + |B_r| \left(\sum_{e \in s_r \cap \sigma} I_{(r,e)} - d(m,r) \right)_+ - 2 \sum_{j \in B_r} d(m,j) \\
&= \quad (|B_r| + 1) \left(\sum_{e \in s_r \cap \sigma} I_{(r,e)} - d(m,r) \right)_+ + \sum_{j \in A_r} \left(\sum_{e \in s_j \cap \sigma} I_{(j,e)} - d(m,j) \right)_+ \\
&\quad - 2 \sum_{j \in B_r} d(m,j).
\end{aligned}$$

Denote by $2\sum_{j \in B_r} d(m,j)$ by λ for readability. Then, the inequality above implies the following two weaker but simpler inequalities:

$$\left(\sum_{e \in s_r \cap \sigma} I_{(r,e)} - d(m,r) \right)_+ \le \frac{f_m^\sigma + \lambda}{|B_r| + 1} \tag{3.4}$$

$$\left(\sum_{e \in s_r \cap \sigma} I_{(r,e)} - d(m,r) \right)_+ + \sum_{j \in A_r} \left(\sum_{e \in s_j \cap \sigma} I_{(j,e)} - d(m,j) \right)_+ \le f_m^\sigma + \lambda \tag{3.5}$$

R Essentially, we follow the structure of the proof of Lemma 4.3 of [78]. However, in the paper, due to considering only one commodity, the set A_r is not necessary, and one can apply the lemma equivalent to Lemma 3.7 for all past requests. Thereby, one arrives at one inequality for each current request that can be applied for all requests to generate the dual constraints. Due to our approach of investing each $I_{(r,e)}$ into multiple configurations, we have to handle the two inequalities above to avoid a factor of $|S|$ in the scaling factor.

Recapitulate that our goal is to find an upper bound on $X := \sum_{r\in R'} \left(\sum_{e\in s_r\cap\sigma} I_{(r,e)} - d(m,r)\right)_+$. For this, we use Equations (3.4) and (3.5). However, we are not able to statically apply both inequalities. Instead, we show that there is a way of applying them such that X is bounded dependent on f_m^σ. For that, we model the task to bound X by Equations (3.4) and (3.5) as a c-ordered covering problem concerning Definition 3.2.

The idea behind this is the following. We aim at covering all $(\sum_{e\in s_r\cap\sigma} I_{(r,e)} - d(m,r))_+$. Each time we cover an element $(\sum_{e\in s_r\cap\sigma} I_{(r,e)} - d(m,r))_+$ of X, we do so by applying either Equation (3.4) or Equation (3.5). If we apply Equation (3.4), we remove the respective element $(\sum_{e\in s_r\cap\sigma} I_{(r,e)} - d(m,r))_+$ from X and add a weight of $\frac{f_m^\sigma+\lambda}{|B_r|+1}$. If we apply Equation (3.5), we remove multiple elements from X and add a weight of $f_m^\sigma + \lambda$. We then ask ourselves how high the total weight is when removing all elements from X. The resulting weight is immediately an upper bound on X.

Next, we define the c-ordered covering instance based on Equations (3.4) and (3.5). Our instance is as follows: The elements are $1,\dots,|R'|$. Consider the element i. It represents $(\sum_{e\in s_r\cap\sigma} I_{(r,e)} - d(m,r))_+$ of the i-th arriving request r of R'. The sets A_i and B_i are given by A_r and B_r as defined above. The parameter c is set to $f_m^\sigma + \lambda$. For every element i there is a set $\{i\}$ of weight $c/(|B_i|+1)$ and a set $\{i\}\cup A_i$ of weight c. Notice how the weights are set to correspond to Equations (3.4) and (3.5), respectively.

We show that the above instance is a proper instance of c-ordered covering. For any element i, by definition $A_i \cap B_i = \emptyset$ and $A_i \cup B_i = \{1,\dots,i-1\}$, because exactly the requests of R' that arrived before the i-th one have a defined value for $\left(\sum_{e\in s_r\cap\sigma} I_{(r,e)} - d(m,r)\right)_+$. If for some i a request r is in A_i and in B_{i+1}, it contributed towards the construction of a facility for σ of which the distance to the request itself is less than $\sum_{e\in s_r\cap\sigma} I_{(r,e)}$. Thus, for all following requests $j > i$, r stays in B_j. In other words, for any two elements i, j with $i < j$ it holds that $B_i \subseteq B_j$.

By Lemma 3.6, the set $\{\left(\sum_{e\in s_r\cap\sigma} I_{(r,e)} - d(m,r)\right)_+ \mid r \in R'\}$ can be covered with a total weight of $2\,(f_m^\sigma + \lambda)\,H_n$. Each time an element is covered, this corresponds to applying either Equation (3.4) or Equation (3.5) to the respective $\left(\sum_{e\in s_r\cap\sigma} I_{(r,e)} - d(m,r)\right)_+$ term, indicated by an increase in the weight of the covering. Thus, we conclude:

$$\begin{aligned}
&\sum_{r\in R'}\left(\sum_{e\in s_r\cap\sigma} I_{(r,e)} - d(m,r)\right)_+ \le 2\,(f_m^\sigma+\lambda)\,H_n \overset{\text{(Def. of }\lambda)}{=} 2\,f_m^\sigma\,H_n + 4H_n\sum_{r\in R'} d(m,r)\\
\Rightarrow\quad &\sum_{r\in R'}\left(\sum_{e\in s_r\cap\sigma} I_{(r,e)} - 5\,H_n\,d(m,r)\right)_+ \le 2\,f_m^\sigma\,H_n\\
\Rightarrow\quad &\sum_{r\in R'}\left(\sum_{e\in s_r\cap\sigma} \frac{I_{(r,e)}}{5\,H_n} - d(m,r)\right)_+ \le \frac{2}{5}\,f_m^\sigma\\
\Rightarrow\quad &\sum_{r\in R'}\left(\sum_{e\in s_r\cap\sigma} \frac{I_{(r,e)}}{5\,H_n} - d(m,r)\right)_+ \le f_m^\sigma
\end{aligned}$$

■

Next, we use Lemma 3.8 to show that for possible configurations σ the constraints of the dual hold if $a_{re} := \gamma I_{(r,e)} = 1/(5\,h_1\,H_n)\,I_{(r,e)}$. Starting from the configurations considered in Lemma 3.8, we utilize the definition of h-dividable, Definition 3.1 Constraint (a), to ensure that dual feasibility holds for all configurations. Using the definition requires us to increase the scaling factor of the dual variables by h_1.

Lemma 3.9 For any configuration $\sigma \subseteq S$, any $R' \subseteq R$, and any facility serving σ at any $m \in M$ it holds that

$$\sum_{r \in R'} \left(\sum_{e \in s_r \cap \sigma} \gamma I_{(r,e)} - d(m,r) \right)_+ \leq f_m^\sigma .$$

Proof. We split the proof into two parts. First, consider configurations σ which are relatively small, i.e., where $|\sigma| \leq h_1$. Here, observe that Lemma 3.8 holds for all configurations considering a single commodity. Let $s := \arg\max_{e \in \sigma} f_m^{\{e\}}$. Then the inequality of the lemma summed up over all commodities of σ yields

$$\begin{aligned}
& \sum_{e \in \sigma} \sum_{r \in R'} \left(\sum_{e \in s_r \cap \{e\}} \frac{I_{(r,e)}}{5H_n} - d(m,r) \right)_+ \leq \sum_{e \in \sigma} f_m^{\{e\}} \leq |\sigma| f_m^{\{s\}} \overset{\substack{(f_m^{\{s\}} \leq f_m^\sigma) \\ (|\sigma| \leq h_1)}}{\leq} h_1 f_m^\sigma \\
\Rightarrow \quad & \sum_{e \in \sigma} \sum_{r \in R'} \left(\sum_{e \in s_r \cap \{e\}} \frac{I_{(r,e)}}{5 h_1 H_n} - \frac{d(m,r)}{h_1} \right)_+ \leq f_m^\sigma \\
\Rightarrow \quad & \sum_{r \in R'} \left(\sum_{e \in s_r \cap \sigma} \frac{I_{(r,e)}}{5 h_1 H_n} - \sum_{e \in s_r \cap \sigma} \frac{d(m,r)}{h_1} \right)_+ \leq f_m^\sigma \\
\Rightarrow \quad & \sum_{r \in R'} \left(\sum_{e \in s_r \cap \sigma} \frac{I_{(r,e)}}{5 h_1 H_n} - |\sigma| \frac{d(m,r)}{h_1} \right)_+ \leq f_m^\sigma \\
\overset{(\sigma \leq h_1)}{\Rightarrow} \quad & \sum_{r \in R'} \left(\frac{I_{(r,e)}}{5 h_1 H_n} - d(m,r) \right)_+ \leq f_m^\sigma \\
\overset{(\text{Def. of } \gamma)}{\Rightarrow} \quad & \sum_{r \in R'} \left(\gamma I_{(r,e)} - d(m,r) \right)_+ \leq f_m^\sigma .
\end{aligned}$$

It holds that $f_m^\sigma \geq f_m^{\{s\}}$ due to the sub-additivity of f and the fact that $s \in \sigma$.

Next, consider a configuration σ which is relatively large, i.e., where $|\sigma| > h_1$. Based on Property (a) of Definition 3.1, it holds for all $m \in M$ that there is a cover C_σ with $|C_\sigma| \leq h_1$ of σ with subsets $\{e\}$ for all $e \in S$ and S_i for all $1 \leq i \leq x$ such that $\sum_{\tau \in C_\sigma} f_m^\tau \leq h_1 f_m^\sigma$. Observe that for every configuration allowed in C_σ Lemma 3.8 holds. Thus, summing up all elements of C_σ yields

$$\begin{aligned}
& \sum_{\tau \in C_\sigma} \sum_{r \in R'} \left(\sum_{e \in s_r \cap \tau} \frac{I_{(r,e)}}{5H_n} - d(m,r) \right)_+ \leq \sum_{\tau \in C_\sigma} f_m^\tau \leq h_1 f_m^\sigma \\
\Rightarrow \quad & \sum_{\tau \in C_\sigma} \sum_{r \in R'} \left(\sum_{e \in s_r \cap \tau} \frac{I_{(r,e)}}{5 h_1 H_n} - \frac{d(m,r)}{h_1} \right)_+ \leq f_m^\sigma \\
\Rightarrow \quad & \sum_{r \in R'} \left(\sum_{e \in s_r \cap \sigma} \frac{I_{(r,e)}}{5 h_1 H_n} - \sum_{\tau \in C_\sigma} \frac{d(m,r)}{h_1} \right)_+ \leq f_m^\sigma \\
\Rightarrow \quad & \sum_{r \in R'} \left(\sum_{e \in s_r \cap \sigma} \frac{I_{(r,e)}}{5 h_1 H_n} - \frac{|C_\sigma|}{h_1} d(m,r) \right)_+ \leq f_m^\sigma \\
\overset{(|C_\sigma| \leq h_1)}{\Rightarrow} \quad & \sum_{r \in R'} \left(\sum_{e \in s_r \cap \sigma} \frac{I_{(r,e)}}{5 h_1 H_n} - d(m,r) \right)_+ \leq f_m^\sigma
\end{aligned}$$

$$\overset{\text{(Def. of } \gamma)}{\Rightarrow} \qquad \sum_{r \in R'} \left(\sum_{e \in s_r \cap \sigma} \gamma I_{(r,e)} - d(m,r) \right)_+ \leq f_m^\sigma$$

■

Theorem 3.2 — Deterministic Competitive Ratio. Given a h-dividable construction cost function, PD-MCOFLP achieves a competitive ratio of $\mathcal{O}(h \log n)$ for the multi-commodity online facility location problem.

Proof. By Lemma 3.9, all $I_{(r,e)}$ scaled by γ provide a feasible solution to the dual. Referring to Section 3.5.3.1, the value of the dual is $\sum_{r \in R} \sum_{e \in s_r} \gamma I_{(r,e)}$. Since the value of a feasible dual solution is a lower bound to the value of any primal solution, the cost of an optimal solution is at least $C_{\text{OPT}} \geq \sum_{r \in R} \sum_{e \in s_r} \gamma I_{(r,e)}$. By Lemma 3.3, the total cost of the algorithm is $C_{\text{PD-MCOFLP}} \leq 2(h_2+1) \sum_{r \in R} \sum_{e \in s_r} I_{(r,e)} \leq 4h_2 \sum_{r \in R} \sum_{e \in s_r} I_{(r,e)}$. Thus, the competitive ratio is

$$\frac{C_{\text{PD-MCOFLP}}}{C_{\text{OPT}}} \leq \frac{2(h_2+1) \sum_{r \in R} \sum_{e \in s_r} I_{(r,e)}}{\sum_{r \in R} \sum_{e \in s_r} \gamma I_{(r,e)}} \leq 4h_2 \cdot 5h_1 H_n \leq 20hH_n \in \mathcal{O}(h \log n).$$

■

3.5.4 A Randomized Algorithm

In the following section, we present RA-MCOFLP, a randomized algorithm for the MCOFLP that achieves a competitive ratio of $\mathcal{O}(h \frac{\log n}{\log \log n})$ against the oblivious adversary for h-dividable cost functions. The algorithm is based on the randomized algorithm for the online facility location problem by Meyerson [76] that achieves a competitive ratio of $\mathcal{O}(\frac{\log n}{\log \log n})$ against the oblivious adversary. We explain our algorithms within the framework of Section 3.5.2 in Section 3.5.4.1 and afterward, show the analysis in Section 3.5.4.2.

3.5.4.1 The Algorithm

RA-MCOFLP works for construction cost functions that are h-dividable respecting Definition 3.1 in a similar manner to PD-MCOFLP. Let $S_1, \ldots, S_x$ be the configurations of Definition 3.1 for f. We only construct facilities offering either any single commodity $s \in S$ or a configuration of a set S_y for $1 \leq y \leq x$. Notice how this is the same design decision we made for PD-MCOFLP.

Our approach follows the high-level idea presented in Section 3.5.2. We first define for each arriving request r and each commodity e it requests an investment $I_{(r,e)}$. This investment intuitively represents the share of e on the cost of connecting r in the cheapest possible way based on the existing facilities of the algorithm. Then, each commodity invests its $I_{(r,e)}$ in the construction of all facilities that would offer e by influencing the probability of such a facility being built ($I_{(r,e)}$ is invested once for each such facility). In other words, for all possible configurations that RA-MCOFLP considers, with a probability dependent on the investments of the affected commodities, we construct a facility for the configuration as close as possible to the request. The probabilities are chosen such that no more than the investment is paid for constructing a facility on expectation.

Handling the non-uniform metric space. Before we show how to define the probabilities, similar to Meyerson [76], we introduce classes for facility costs of a fixed configuration to handle the non-uniformity of the metric space. Fix a configuration $\sigma \subseteq S$. Consider the set of all possible f_m^σ rounded down to the nearest power of 2 in increasing order $K_1^\sigma, K_2^\sigma, \ldots$. We call K_i^σ the *class i of σ*, representing a facility cost for σ occurring at a set of locations of M. Observe that for any i and any σ it holds that $2K_i^\sigma \leq K_{(i+1)}^\sigma$ by definition. Let $d(K_i^\sigma, m)$ denote the minimal distance of a location $m \in M$ to a location of class i for σ. Rounding down the facility costs to the nearest power of 2 increases the competitive ratio of RA-MCOFLP by at most a factor of 2.

Defining the investment. Next, we show how to define the investment for each commodity of a fixed request. For that, consider the situation at the arrival of a request r asking for the commodity set s_r. Consider any configuration $\sigma \in S \cup_{1 \leq y \leq x} \{S_y\}$ of the set of configurations that RA-MCOFLP maintains. If the algorithm would connect r to an existing facility offering σ, it would pay $d(F(\sigma), r)$. Alternatively, the algorithm could construct a facility for σ and connect r to it. Then, for a fixed class i of σ, the cheapest way to do so is $K_i^\sigma + d(K_i^\sigma, r)$. Thus, the cheapest overall way to construct a facility for σ and connect it to r would be $\min_i\{K_i^\sigma + d(K_i^\sigma, r)\}$. In total, the cheapest way to connect r to a facility offering σ is then

$$X(r, \sigma) := \min\{\min_i\{K_i^\sigma + d(K_i^\sigma, r)\}, d(F(\sigma), r)\}.$$

The goal is now to define an investment $I_{(r,e)}$ for each $e \in s_r$ of request r such that the investment is spent into all $X(r, \sigma)$ where $e \in \sigma$ such that each commodity invests as little as possible to be served by a facility. For that, RA-MCOFLP proceeds similarly to PD-MCOFLP. Consider here the first phase in the pseudo code. When a request r arrives, we initialize $I_{(r,e)}$ variables for all $e \in s_r$ by zero and gradually increase them simultaneously. Thereby, we make sure that for all configurations $\sigma \in S \cup_{1 \leq y \leq x} \{S_y\}$ that RA-MCOFLP manages it holds that $\sum_{e \in s_r \cap \sigma} I_{(r,e)} \leq X(r, \sigma)$. As soon as for some σ it holds that $\sum_{e \in s_r \cap \sigma} I_{(r,e)} = X(r, \sigma)$, we freeze all involved $I_{(r,e)}$ and connect r to a facility for σ as follows. If $X(r, \sigma) = K_i^\sigma + d(K_i^\sigma, r)$ for some class i, we construct a facility for σ at the closest location of class i. Then, in any case, we connect r to the closest facility for σ.

RA-MCOFLP **on arrival of request r with commodity set s_r**

```
    # Phase 1
1:  Initialize $I_{(r,e)} = 0$ for all $e \in s_r$
2:  while Not all $I_{(r,e)}$ are frozen do
3:      Simultaneously raise all unfrozen $I_{(r,e)}$ until for some $\sigma \in S \cup_{1 \leq y \leq x} \{S_y\}$
        it holds that $\sum_{e \in s_r \cap \sigma} I_{(r,e)} = X(r, \sigma)$
4:      Freeze $I_{(r,e)}$ for all $e \in s_r \cap \sigma$
5:      if $X(r, \sigma) = K_i^\sigma + d(K_i^\sigma, r)$ for class $i$ of $\sigma$ then
6:          Construct a facility for $\sigma$ at the location of $K_i^\sigma$ closest to $r$
7:      Connect $r$ to the closest facility for $\sigma$
    # Phase 2
8:  for all $\sigma \in S \cup_{1 \leq y \leq x} \{S_y\}$ do
9:      Let $K_0^\sigma := \sum_{e \in s_r \cap \sigma} I_{(r,e)}$
10:     for all Classes $i$ of $\sigma$ do
11:         Build a facility in configuration $\sigma$ of class $i$ at the location closest to $r$ with probability:
            $\Pr[r, \sigma, i] = \frac{d(K_{(i-1)}^\sigma, r) - d(K_i^\sigma, r)}{K_i^\sigma}$
```

Determining probabilities for facility construction. The current request is already connected after phase 1. In phase 2, RA-MCOFLP calculates probabilities for constructing additional facilities based on the investments of commodities determined in the first phase. Assume for ease of explanation that facility costs are uniform. Then the overall probability for constructing a facility in configuration σ is given by the sum of all commodities' investments in σ divided by the cost of constructing the facility. On expectation, investments of commodities that benefit from a facility offering σ add up to the construction cost until the facility is built. However, since the facility costs are not uniform, RA-MCOFLP has to decide on the class of σ. To this end, the probability is split up over all classes of σ in a way that class i gets a share that is proportional to the advantage of considering class i over class $(i-1)$. Since class $(i-1)$ is cheaper for σ, this advantage only exists if a facility of class i can be constructed much closer to r than one of class $(i-1)$. More formally, the probability of constructing a facility of configuration σ is for class i given by

$$\Pr[r,\sigma,i] = \frac{d(K^\sigma_{(i-1)},r) - d(K^\sigma_i,r)}{K^\sigma_i}, \text{ where } K^\sigma_0 = \sum_{e\in s_r\cap\sigma} I_{(r,e)}.$$

Notice how the investment of all commodities is equal to the cost of RA-MCOFLP for connecting the respective request. Also, every commodity invests its share of the cost into all facilities that could have been advantageous (because the associated facility can serve the commodity). The investment is then shared over all classes for a fixed configuration to handle the non-uniformity of the metric. Of course, the construction of additional facilities does not improve the serving cost of the current request, but it improves the situation for future requests that potentially demand similar commodities.

3.5.4.2 The Analysis

Next, we show that RA-MCOFLP achieves a competitive ratio of $\mathcal{O}(h\frac{\log n}{\log\log n})$ (Theorem 3.3) against the oblivious adversary. Our analysis follows the high-level idea of Section 3.5.2 and is inspired by Fotakis' analysis [49] of Meyerson's algorithm [76]. However, we must carefully consider the different configurations managed by RA-MCOFLP.

We show that the total cost of RA-MCOFLP is upper bounded by $\mathcal{O}\left(h_2\sum_{r\in R}\sum_{e\in s_r} I_{(r,e)}\right)$ on expectation. Afterward, following Fotakis [49], we consider each *optimal center* and the requests it serves separately. In contrast to the single-commodity case, the optimal center facility does not always have the same configuration as the algorithm's facilities. Therefore, we cannot directly relate the cost of the algorithm's facilities to the optimal cost as in [49]. Instead, we show the following: For any optimal center and any affected configuration $\sigma \in S\cup_{1\le y\le x}\{S_y\}$ managed by RA-MCOFLP, the cost for σ for the optimal center is bounded roughly by $\frac{\log n}{\log\log n}$ times the cost of a facility for σ at the location of the optimal one. Then, we relate the cost of the optimal facility to the facilities that RA-MCOFLP constructs to cover the same commodities and show that we are a factor of at most h_1 apart. The facilities constructed by the algorithm are funded by the investment of affected requests and their commodities. The investment poses an upper bound on the total cost of the algorithm. Therefore, the competitive ratio follows.

Bounding the algorithm's cost in the investment. The algorithm's cost is bounded by its serving and construction cost. The serving cost can be bounded easily, as they directly define the investments. Therefore, we begin by analyzing the expected construction cost in Lemma 3.10.

Lemma 3.10 Consider a configuration $\sigma \in S\cup_{1\le y\le x}\{S_y\}$. The expected construction cost for facilities in configuration σ due to a request r is

$$\mathrm{E}[r,\sigma] = \sum_{e\in s_r\cap\sigma} I_{(r,e)}.$$

Proof. The expected construction cost of facilities for σ is given by

$$\begin{aligned}\mathrm{E}[r,\sigma] &= \sum_i \Pr[r,\sigma,i]\cdot K^\sigma_i = \sum_i \frac{d(K^\sigma_{(i-1)},r) - d(K^\sigma_i,r)}{K^\sigma_i}\cdot K^\sigma_i \\ &= \sum_i d(K^\sigma_{(i-1)},r) - d(K^\sigma_i,r) = K^\sigma_0 = \sum_{e\in s_r\cap\sigma} I_{(r,e)}\end{aligned}$$

■

The previous lemma states the construction cost for a fixed configuration managed by the algorithm. By definition of h-dividable, every commodity is in at most h_2+1 many such configurations. Therefore, we can analyze the total expected cost a request r has for a commodity $e\in s_r$ as below.

Lemma 3.11 Consider a request r and any commodity $e \in s_r$. The total expected cost of RA-MCOFLP for r regarding e is at most

$$\mathrm{E}_{\text{total}}[r,\{e\}] \leq 2\,(h_2+1)I_{(r,e)}.$$

Proof. Observe that by phase 1 of the RA-MCOFLP, $I_{(r,e)}$ is invested at most once in every connection to facilities for configurations σ where $e \in \sigma$. By Definition 3.1, e is in at most h_2 many of the sets $S_1,\dots,S_x$ and in the set consisting of only $\{e\}$. So, the cost RA-MCOFLP pays with regard to e for connecting r to facilities serving e is at most $(h_2+1)I_{(r,e)}$. Additionally, RA-MCOFLP invests $I_{(r,e)}$ for every configuration $\sigma \in S \cup_{1\leq y\leq x} \{S_y\}$ the expected construction cost which is $\mathrm{E}[r,\sigma] = \sum_{e\in s_r\cap\sigma} I_{(r,e)}$ due to Lemma 3.10. As before the number of configurations in which $I_{(r,e)}$ is invested if by Definition 3.1 bounded by h_2+1. Therefore, the total expected construction cost r pays with the investment of e is at most $(h_2+1)I_{(r,e)}$ and the total expected cost of r for e is then at most $\mathrm{E}_{\text{total}}[r,\{e\}] \leq 2\,(h_2+1)I_{(r,e)}$. ■

As a direct consequence, the expected cost of any request r is bounded as in Corollary 3.1.

Corollary 3.1 Consider a request r and any configuration $\sigma \subseteq S$. The total expected cost of RA-MCOFLP for r regarding σ is at most

$$\mathrm{E}_{\text{total}}[r,\sigma] \leq 2\,(h_2+1)\sum_{e\in s_r\cap\sigma} I_{(r,e)}.$$

Proof. Any commodity in $\sigma \setminus s_r$ does not incur a cost in RA-MCOFLP. For all commodities in $s_r \cap \sigma$, we can apply Lemma 3.11, and the result follows. ■

Relating facilities of the algorithm to the location of an optimal center. In the upcoming part of the analysis, we relate the expected construction cost of the algorithm to an optimal center. We consider first only the facilities managed by the algorithm. For any configuration σ of the algorithm, we consider any optimal center c and the set of requests R_c that is connected to c in the optimal solution. The class of c with respect to σ is j, i.e, constructing a facility for σ at c would cost K_j^σ. See Figure 3.11 (on the next page) for a depiction.

For the requests of R_c, we show in Lemma 3.12: The expected construction cost for facilities offering σ is bounded by the serving cost of the optimal solution, given as $\sum_{r\in R_c} d(r,c)$, and the construction cost for a facility for σ at c, denoted by K_j^σ. Note that we do not yet relate the cost to the construction cost of the optimal solution for c, but only to the location of the optimal facility.

Lemma 3.12 Consider an optimal center c and let R_c be the set of requests connected to c by the optimal solution. Fix a configuration $\sigma \in S \cup_{1\leq y\leq x} \{S_y\}$. Let j be the class of location c with respect to σ. Then the expected construction cost for facilities in configuration σ due to requests of R_c is at most $\mathrm{E}[R_c,\sigma] \leq 20\left(K_j^\sigma + \sum_{r\in R_c} d(r,c)\right)\frac{\log n}{\log\log n}$.

R Our proof follows the analysis of Fotakis [49] and Meyerson [76]. For the sake of precision, we present it respecting non-uniform metrics. Further, we generalize it to configurations of $S \cup_{1\leq y\leq x} \{S_y\}$ and carefully verify that our probabilities are chosen correctly by making all arguments explicit. Due to the added commodities, we need a second step to relate the cost to the optimal facility at c.

Proof. We analyze the requests based on their distance to c. For this, we divide the request set R_c as follows: Let B_α be the set of locations in a distance of $[t^{\alpha-1}\overline{\mathrm{Asg}_c}, t^\alpha\overline{\mathrm{Asg}_c}]$, where $t := \frac{\log n}{\log\log n}$ and $\overline{\mathrm{Asg}_c} := \frac{\sum_{r\in R_c} d(r,c)}{|R_c|}$ is the *average serving cost* of OPT for c. Observe that B_{t+1} is empty because

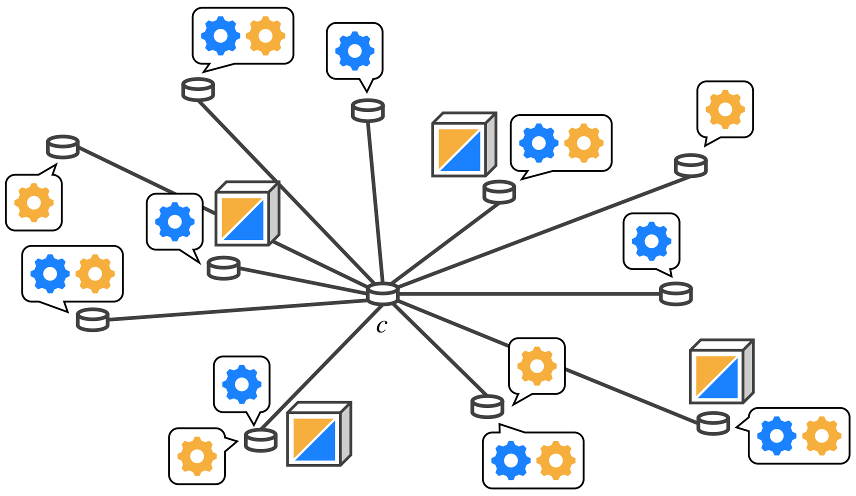

Figure 3.11: First, we consider every configuration σ the algorithm manages. For the figure assume $\sigma = \{blue, gold\}$. c is the location of an optimal center, and we consider the set of requests R_c that are connected to c in the optimal solution (all requests in the figure). We then relate the expected cost for facilities of configuration σ that the algorithm constructs due to requests in R_c (all depicted facilities) to the cost of a (hypothetical) facility for σ at c. Afterward, we relate the cost of a facility for σ to the optimal cost of the facility at c (in an optimal configuration).

$t^{t+1} > n$. A location of B_{t+1} would thus incur a serving cost for c, which is higher than the serving cost of all requests in R_c because $|R_c| \leq n$. The former cannot be by definition.

Defining events. Fix a B_α. The distance of any request r to a location of any class i for σ is at most $d(K_i^\sigma, r) \leq d(K_i^\sigma, c) + d(c,r) \leq d(K_i^\sigma, c) + t^\alpha \overline{\mathrm{Asg}_c}$ by the triangle inequality. We say *event i occurs* if a facility opens in a distance at most $d(K_i^\sigma, c) + 2t^\alpha \overline{\mathrm{Asg}_c}$ to c. Any request at a location of B_α that causes a facility for σ of class i or higher to open triggers event i. To see this, consider the request r that constructs a facility for σ of class i at the location closest to itself. The facility that was opened by r is at a location of K_i^σ and thus a distance of at most $d(K_i^\sigma, c) + t^\alpha \overline{\mathrm{Asg}_c}$ from r. Since $r \in B_\alpha$, it is in a distance of at most $d(c,r) \leq t^\alpha \overline{\mathrm{Asg}_c}$ to c. The distance from the constructed facility to c is by triangle inequality at most the distance between c and r, and r and the facility. This distance is then at most $t^\alpha \overline{\mathrm{Asg}_c} + d(K_i^\sigma, c) + t^\alpha \overline{\mathrm{Asg}_c} \leq d(K_i^\sigma, c) + 2t^\alpha \overline{\mathrm{Asg}_c}$ which implies that event i occurs.

Let $\delta = 4$ be a constant. We say *event δ^* occurs* if a facility for σ in distance $\delta t^\alpha \overline{\mathrm{Asg}_c}$ to c opens. Observe that for $i \geq j$ it holds that $d(K_i^\sigma, c) = 0$, because the closest location to c of class K_i^σ is by definition c which has class j. Therefore, any facility for σ of class $i \geq j$ built triggers both, event i and event δ^*. The events i can be seen as intermediate events that help us to show how long it takes for δ^* to happen. First, we show a limit on the investment of commodities in σ of requests that arrive until δ^* happens. Second, we show how high the investment of commodities in σ is for requests that appear after δ^* happened.

Consider a fixed set B_α and let B_α^σ be the set of requests at locations of B_α that request a commodity in σ. Let $B_\alpha^\sigma(v)$ be the requests of B_α^σ that appear *before* event v has happened.

The cost before event *i* occurs. The expected construction cost any request r pays for a facility for σ of class i is

$$\mathrm{E}[r,\sigma,i] = \Pr[r,\sigma,i] \cdot K_i^\sigma = d(K_{(i-1)}^\sigma, r) - d(K_i^\sigma, r).$$

Further, the expected total investment in facilities for σ of class i until such a facility is constructed and event i happens is

$$\sum_{r\in B^\sigma_\alpha(i)} \mathrm{E}[r,\sigma,i] = K^\sigma_i.$$

Lower bounding $d(K^\sigma_i, r)$ after event i occurred. Assume, i occurred, but δ^* has not, and consider a request $r \in B^\sigma_\alpha(\delta^*)\setminus B^\sigma_\alpha(i)$. Next, we show how to give a lower bound on $d(K^\sigma_i, r)$ by the expected cost of r. Since i already occurred, it holds that $d(F(\sigma),c) \le d(K^\sigma_i,c) + 2t^\alpha\overline{\mathrm{Asg}_c}$. Therefore, it holds for the expected investment of r into facilities for σ:

$$\begin{aligned}\mathrm{E}[r,\sigma\,|\,B^\sigma_\alpha(\delta^*)\setminus B^\sigma_\alpha(i)] &= \sum_{e\in s_r\cap\sigma} I_{(r,e)} \le X(r,\sigma) \le d(F(\sigma),r) \le d(r,c) + d(F(\sigma),c)\\ &\le t^\alpha\overline{\mathrm{Asg}_c} + d(K^\sigma_i,c) + 2t^\alpha\overline{\mathrm{Asg}_c} \le d(K^\sigma_i,c) + 3t^\alpha\overline{\mathrm{Asg}_c} \qquad (3.6)\end{aligned}$$

By definition, there is a location of class i in a distance of $d(K^\sigma_i,r)$ to r. Due to the triangle inequality:

$$\begin{aligned}&d(K^\sigma_i,c) \le d(c,r) + d(K^\sigma_i,r) \le d(K^\sigma_i,r) + t^\alpha\overline{\mathrm{Asg}_c}\\ \Leftrightarrow\ &d(K^\sigma_i,r) \ge d(K^\sigma_i,c) - t^\alpha\overline{\mathrm{Asg}_c} \qquad (3.7)\end{aligned}$$

Since δ^* has not occurred yet, we also have a lower bound on $d(K^\sigma_i,c)$ as follows:

$$\begin{aligned}&\delta t^\alpha\overline{\mathrm{Asg}_c} \le d(F(\sigma),c) \le d(K^\sigma_i,c) + 2t^\alpha\overline{\mathrm{Asg}_c}\\ \Leftrightarrow\ &d(K^\sigma_i,c) \ge (\delta-2)t^\alpha\overline{\mathrm{Asg}_c} \qquad (3.8)\end{aligned}$$

Here, we used that i already occurred. Next, we can derive the following:

$$\begin{aligned}\frac{d(K^\sigma_i,r)}{\mathrm{E}[r,\sigma\,|\,B^\sigma_\alpha(\delta^*)\setminus B^\sigma_\alpha(i)]} &\overset{\text{Equation (3.7)}}{\ge} \frac{d(K^\sigma_i,c) - t^\alpha\overline{\mathrm{Asg}_c}}{\mathrm{E}[r,\sigma\,|\,B^\sigma_\alpha(\delta^*)\setminus B^\sigma_\alpha(i)]}\\ &\overset{\text{Equation (3.6)}}{\ge} \frac{d(K^\sigma_i,c) - t^\alpha\overline{\mathrm{Asg}_c}}{d(K^\sigma_i,c) + 3t^\alpha\overline{\mathrm{Asg}_c}} \qquad (3.9)\end{aligned}$$

Next, we chose $\delta = 4$. Then, based on Equation (3.8) it holds that $d(K^\sigma_i,c) \ge (\delta-2)t^\alpha\overline{\mathrm{Asg}_c} = 2t^\alpha\overline{\mathrm{Asg}_c}$. For $d(K^\sigma_i,c)$ and $t^\alpha\overline{\mathrm{Asg}_c}$ in such a relation, it holds that

$$\frac{d(K^\sigma_i,r)}{\mathrm{E}[r,\sigma\,|\,B^\sigma_\alpha(\delta^*)\setminus B^\sigma_\alpha(i)]} \overset{\text{Equation (3.9)}}{\ge} \frac{d(K^\sigma_i,c) - t^\alpha\overline{\mathrm{Asg}_c}}{d(K^\sigma_i,c) + 3t^\alpha\overline{\mathrm{Asg}_c}} \overset{\delta=4}{\ge} \frac{(2-1)t^\alpha\overline{\mathrm{Asg}_c}}{(2+3)t^\alpha\overline{\mathrm{Asg}_c}} = \frac{1}{5}. \qquad (3.10)$$

Rearranging Equation (3.10) gives us the desired bound:

$$d(K^\sigma_i,r) \ge \frac{1}{5}\mathrm{E}[r,\sigma\,|\,B^\sigma_\alpha(\delta^*)\setminus B^\sigma_\alpha(i)] \qquad (3.11)$$

Bounding the cost until δ^* occurs. Assume event i occurred. How much cost is accumulated on expectation until event $i+1$ happens? By RA-MCOFLP, the expected cost for facilities for σ of classes higher than i is

$$\sum_{r\in B^\sigma_\alpha(i+1)\setminus B^\sigma_\alpha(i)}\ \sum_{k\ge i}\left(d(K^\sigma_k,r) - d(K^\sigma_{k+1},r)\right) = \sum_{r\in B^\sigma_\alpha(i+1)\setminus B^\sigma_\alpha(i)} d(K^\sigma_i,r).$$

Also, the expected cost for facilities for σ until one of class at least $i+1$ is constructed is at most $K^\sigma_{(i+1)}$, giving us:

$$\begin{aligned}
&& \sum_{r\in B^\sigma_\alpha(i+1)\setminus B^\sigma_\alpha(i)} d(K^\sigma_i, r) \le K^\sigma_{(i+1)} \\
\overset{\textit{Equation (3.11)}}{\Rightarrow} && \sum_{r\in B^\sigma_\alpha(i+1)\setminus B^\sigma_\alpha(i)} \frac{1}{5}\mathrm{E}[r,\sigma\,|\,B^\sigma_\alpha(\delta^*)\setminus B^\sigma_\alpha(i)] \le K^\sigma_{(i+1)} \\
\Leftrightarrow && \sum_{r\in B^\sigma_\alpha(i+1)\setminus B^\sigma_\alpha(i)} \mathrm{E}[r,\sigma\,|\,B^\sigma_\alpha(\delta^*)\setminus B^\sigma_\alpha(i)] \le 5K^\sigma_{(i+1)}
\end{aligned} \tag{3.12}$$

Recapitulate that j was the class for σ of the location of the optimal center c. Therefore, $\delta^* \le j$. Additionally, the classes were defined such that for all i it holds that $2K^\sigma_i \le K^\sigma_{(i+1)}$ Summing up over all classes yields that the total expected costs for facilities until δ^* happens is:

$$\begin{aligned}
\sum_{r\in B^\sigma_\alpha(\delta^*)} \mathrm{E}[r,\sigma] &= \mathrm{E}[r,\sigma,1] + \sum_{i=1}^{\delta^*-1}\sum_{r\in B^\sigma_\alpha(i+1)\setminus B^\sigma_\alpha(i)} \mathrm{E}[r,\sigma,i] \overset{\textit{Equation (3.12)}}{\le} K^\sigma_1 + \sum_{i=1}^{\delta^*-1} 5K^\sigma_{(i+1)} \\
&\le 5\sum_{i=1}^{\delta^*} K^\sigma_i \overset{(\delta^*\le j)}{\le} 5\sum_{i=1}^{j} K^\sigma_i = 5\sum_{i=1}^{j}\frac{K^\sigma_j}{2^{j-i}} = 5K^\sigma_j\sum_{i=0}^{j-1}\frac{1}{2^i} \le 10K^\sigma_j
\end{aligned} \tag{3.13}$$

Bounding the cost after δ^* occurred. After δ^* happens, there is a facility close by to serve σ for future requests. We distinguish here between $\alpha > 0$ and $\alpha = 0$. Assume that $\alpha > 0$ and consider a request $r \in B^\sigma_\alpha \setminus B^\sigma_\alpha(\delta^*)$. By definition, the distance between r and c is at least $d(r,c) \ge t^{\alpha-1}\overline{\mathrm{Asg}_c}$. Then, r has an expected cost for facilities for σ of

$$\begin{aligned}
\mathrm{E}[r,\sigma\,|\,r\in B^\sigma_\alpha\setminus B^\sigma_\alpha(\delta^*)] &= \sum_{e\in s_r\cap\sigma} I_{(r,e)} \le X(r,\sigma) \le d(F(\sigma),r) \le d(r,c) + d(F(\sigma),c) \\
&\le d(r,c) + \delta t^\alpha\overline{\mathrm{Asg}_c} = d(r,c) + \delta t\cdot t^{\alpha-1}\overline{\mathrm{Asg}_c} \\
&\le d(r,c) + \delta t\, d(r,c) = (\delta t + 1)\, d(r,c).
\end{aligned} \tag{3.14}$$

Next, assume $\alpha = 0$. Then, there is a facility in distance at most $\delta\overline{\mathrm{Asg}_c}$ of c such that the expected cost for facilities for σ of any request $r \in B^\sigma_0 \setminus B^\sigma_0(\delta^*)$ is

$$\begin{aligned}
\mathrm{E}[r,\sigma\,|\,r\in B^\sigma_0\setminus B^\sigma_0(\delta^*)] &= \sum_{e\in s_r\cap\sigma} I_{(r,e)} \le X(r,\sigma) \le d(F(\sigma),r) \le d(r,c) + d(F(\sigma),c) \\
&\le d(r,c) + \delta\overline{\mathrm{Asg}_c}.
\end{aligned} \tag{3.15}$$

Bounding the expected cost for σ. Finally, we can give an upper bound on the expected cost of requests for facilities for σ:

$$\begin{aligned}
\sum_{r\in R_c}\mathrm{E}[r,\sigma] &= \sum_{\alpha=0}^{t}\sum_{r\in B^\sigma_\alpha}\mathrm{E}[r,\sigma] = \sum_{\alpha=0}^{t}\left(\sum_{r\in B^\sigma_\alpha(\delta^*)}\mathrm{E}[r,\sigma] + \sum_{r\in B^\sigma_\alpha\setminus B^\sigma_\alpha(\delta^*)}\mathrm{E}[r,\sigma]\right) \\
&\overset{\textit{Equation (3.13)}}{\le} \sum_{\alpha=0}^{t}\left(10K^\sigma_j + \sum_{r\in B^\sigma_\alpha\setminus B^\sigma_\alpha(\delta^*)}\mathrm{E}[r,\sigma]\right) \\
&= \sum_{\alpha=0}^{t} 10K^\sigma_j + \sum_{\alpha=0}^{t}\sum_{r\in B^\sigma_\alpha\setminus B^\sigma_\alpha(\delta^*)}\mathrm{E}[r,\sigma] \\
&= 10(t+1)K^\sigma_j + \sum_{r\in R_c}\mathrm{E}[r,\sigma\,|\,r\in B^\sigma_0\setminus B^\sigma_0(\delta^*)] + \sum_{\alpha=1}^{t}\sum_{r\in B^\sigma_\alpha\setminus B^\sigma_\alpha(\delta^*)}\mathrm{E}[r,\sigma]
\end{aligned}$$

$$\begin{aligned}
&\overset{\textit{Equation } (3.15)}{\leq} 10(t+1)K_j^\sigma + \sum_{r\in R_c}(d(r,c)+\delta\overline{\mathrm{Asg}_c}) + \sum_{\alpha=1}^{t}\sum_{r\in B_\alpha^\sigma\setminus B_\alpha^\sigma(\delta^*)} \mathrm{E}[r,\sigma]\\
&\leq 10(t+1)K_j^\sigma + (\delta+1)\sum_{r\in R_c} d(r,c) + \sum_{\alpha=1}^{t}\sum_{r\in B_\alpha^\sigma\setminus B_\alpha^\sigma(\delta^*)} \mathrm{E}[r,\sigma]\\
&\overset{\textit{Equation } (3.14)}{\leq} 10(t+1)K_j^\sigma + (\delta+1)\sum_{r\in R_c} d(r,c) + \sum_{\alpha=1}^{t}\sum_{r\in B_\alpha^\sigma\setminus B_\alpha^\sigma(\delta^*)} (\delta t+1)\,d(r,c)\\
&= 10(t+1)K_j^\sigma + (\delta+1)\sum_{r\in R_c} d(r,c) + (\delta t+1)\sum_{\alpha=1}^{t}\sum_{r\in B_\alpha^\sigma\setminus B_\alpha^\sigma(\delta^*)} d(r,c)\\
&= 10(t+1)K_j^\sigma + (\delta+1)\sum_{r\in R_c} d(r,c) + (\delta t+1)\sum_{r\in R_c} d(r,c)\\
&\overset{(\delta=4)}{=} 10(t+1)K_j^\sigma + 5\sum_{r\in R_c} d(r,c) + (4t+1)\sum_{r\in R_c} d(r,c)\\
&= (10t+10)K_j^\sigma + (4t+6)\sum_{r\in R_c} d(r,c)\\
&\leq (10t+10)\left(K_j^\sigma + \sum_{r\in R_c} d(r,c)\right) \leq 20t\left(K_j^\sigma + \sum_{r\in R_c} d(r,c)\right)\\
&\overset{\left(t\leq\frac{\log n}{\log\log n}\right)}{\leq} 20\left(K_j^\sigma + \sum_{r\in R_c} d(r,c)\right)\frac{\log n}{\log\log n}
\end{aligned}$$

Then, the lemma holds. ■

Comparing the cost to the optimal center. Finally, we are ready to compare the expected total cost of the algorithm to the optimal cost. We again consider an optimal center as before. We use that the investments of the algorithm bound the expected total cost. Further, the investments are bounded by the expected construction cost, which is in turn bounded by the cost the algorithm would have if it placed a facility at the location of the optimal center (Lemma 3.12). Using the definition of h-dividable cost functions, we can relate these costs to the cost the optimal solution pays for c and the requests of R_c.

Lemma 3.13 Consider an optimal center c with configuration σ and let R_c be the set of requests connected to c by the optimal solution. The expected total cost of RA-MCOFLP for requests of R_c regarding σ is at most $\mathcal{O}\left(h\frac{\log n}{\log\log n}\right)$ times the cost of the optimal center.

R The arguments of the proof are similar to the ones used in the proof of Lemma 3.9 for PD-MCOFLP. As there, we exploit our notion of h-dividable and relate to optimal configurations with less than h_1 and more than h_1 commodities separately.

Proof. Assume that the configuration of the optimal solution for c is σ with $|\sigma|\leq h_1$. Applying Corollary 3.1 for all $r\in R_c$ yields that the total expected cost of RA-MCOFLP for requests in R_c regarding σ is at most

$$\mathrm{E}_{\text{total}}[R_c,\sigma] \leq \sum_{r\in R_c}\mathrm{E}_{\text{total}}[r,\sigma] \leq \sum_{r\in R_c} 2(h_2+1)\sum_{e\in s_r\cap\sigma} I_{(r,e)} = 2(h_2+1)\sum_{r\in R_c}\sum_{e\in s_r\cap\sigma} I_{(r,e)}.$$

Observe that Lemma 3.10 implies that

$$\sum_{r\in R_c}\sum_{e\in s_r\cap\sigma} I_{(r,e)} = \sum_{r\in R_c}\sum_{e\in s_r\cap\sigma}\mathrm{E}[r,\{e\}] = \sum_{e\in\sigma}\sum_{r\in R_c}\mathrm{E}[r,\{e\}] = \sum_{e\in\sigma}\mathrm{E}[R_c,\{e\}].$$

Additionally, for every commodity $e \in \sigma$, Lemma 3.12 yields that the expected construction cost for facilities in configuration $\{e\}$ is at most

$$\mathrm{E}[R_c, \{e\}] \le 20 \left(K_j^{\{e\}} + \sum_{r \in R_c} d(r,c) \right) \frac{\log n}{\log\log n}.$$

Observe that for any $K_j^{\{e\}}$ it holds that $K_j^{\{e\}} \le 2 f_c^{\{e\}} \le 2 f_c^{\sigma}$, because of the rounding. Combining the above inequalities, we have that

$$\begin{aligned}
\mathrm{E}_{\text{total}}[R_c, \sigma] \quad &\le \quad 2\,(h_2+1) \sum_{r \in R_c} \sum_{e \in s_r \cap \sigma} I_{(r,e)} \le 2\,(h_2+1) \sum_{e \in \sigma} \mathrm{E}[R_c, \{e\}] \\
&\le \quad 2\,(h_2+1)\, 20 \sum_{e \in \sigma} \left(K_j^{\{e\}} + \sum_{r \in R_c} d(r,c) \right) \frac{\log n}{\log\log n} \\
&\overset{(K_j^{\{e\}} \le 2 f_c^{\sigma})}{\le} \quad 40\,(h_2+1) \sum_{e \in \sigma} \left(2 f_c^{\sigma} + \sum_{r \in R_c} d(r,c) \right) \frac{\log n}{\log\log n} \\
&\le \quad 80\,(h_2+1)\,|\sigma| \left(f_c^{\sigma} + \sum_{r \in R_c} d(r,c) \right) \frac{\log n}{\log\log n} \\
&\overset{(|\sigma| \le h_1)}{\le} \quad 160\, h_2\, h_1 \left(f_c^{\sigma} + \sum_{r \in R_c} d(r,c) \right) \frac{\log n}{\log\log n} \\
&\overset{(h_1 \cdot h_2 = h)}{=} \quad 160\, h \left(f_c^{\sigma} + \sum_{r \in R_c} d(r,c) \right) \frac{\log n}{\log\log n}.
\end{aligned}$$

The optimal solution pays at least $f_c^{\sigma} + \sum_{r \in R_c} d(r,c)$ for the center and thus, the lemma follows.

Next, assume that $|\sigma| > h_1$. By Definition 3.1, we know that there is a cover C_σ with $|C_\sigma| \le h_1$ of σ with subsets $\{e\}$ for all $e \in S$ and S_i for all $1 \le i \le x$ such that $\sum_{\tau \in C_\sigma} f_c^{\tau} \le h_1 f_c^{\sigma}$. Consider the set $\sigma' = \cup_{\tau \in C_\sigma} \tau$ which is the set of all commodities covered by C_σ. Applying Corollary 3.1 for all $r \in R_c$ yields that the total expected cost of RA-MCOFLP for requests in R_c regarding σ' is at most

$$\mathrm{E}_{\text{total}}[R_c, \sigma'] \le \sum_{r \in R_c} \mathrm{E}_{\text{total}}[r, \sigma'] \le \sum_{r \in R_c} 2\,(h_2+1) \sum_{e \in s_r \cap \sigma'} I_{(r,e)} = 2\,(h_2+1) \sum_{r \in R_c} \sum_{e \in s_r \cap \sigma'} I_{(r,e)}.$$

Observe that Lemma 3.10 implies that

$$\sum_{r \in R_c} \sum_{e \in s_r \cap \sigma'} I_{(r,e)} = \sum_{r \in R_c} \sum_{\tau \in C_\sigma} \mathrm{E}[r, \tau] = \sum_{\tau \in C_\sigma} \sum_{r \in R_c} \mathrm{E}[r, \tau] = \sum_{\tau \in C_\sigma} \mathrm{E}[R_c, \tau].$$

As each $\tau \in C_\sigma$ is a configuration that RA-MCOFLP manages, we can apply Lemma 3.12 for each $\tau \in C_\sigma$ such that the expected construction cost for facilities in configuration τ is at most

$$\mathrm{E}[R_c, \tau] \le 20 \left(K_j^{\tau} + \sum_{r \in R_c} d(r,c) \right) \frac{\log n}{\log\log n}.$$

Again, due to our rounding and since the class of c for τ is j, it holds for every τ that $K_j^{\tau} \le 2 f_c^{\tau}$. Combing the inequalities yields

$$\mathrm{E}_{\text{total}}[R_c, \sigma'] \qquad \le \qquad 2\,(h_2+1) \sum_{r \in R_c} \sum_{e \in s_r \cap \sigma'} I_{(r,e)} \le 2\,(h_2+1) \sum_{\tau \in C_\sigma} \mathrm{E}[R_c, \tau]$$

$$\begin{aligned}
&\leq && 2\,(h_2+1)\,20 \sum_{\tau\in C_\sigma}\left(K_j^\tau + \sum_{r\in R_c} d(r,c)\right)\frac{\log n}{\log\log n}\\
&\overset{(K_j^\tau\leq 2f_c^\tau)}{\leq} && 40\,(h_2+1) \sum_{\tau\in C_\sigma}\left(2f_c^\tau + \sum_{r\in R_c} d(r,c)\right)\frac{\log n}{\log\log n}\\
&= && 80\,(h_2+1)\left(\sum_{\tau\in C_\sigma} f_c^\tau + \sum_{\tau\in C_\sigma}\sum_{r\in R_c} d(r,c)\right)\frac{\log n}{\log\log n}\\
&\overset{(\sum_{\tau\in C_\sigma} f_c^\tau\leq h_1 f_c^\sigma)}{\leq} && 80\,(h_2+1)\left(h_1 f_c^\sigma + |C_\sigma|\sum_{r\in R_c} d(r,c)\right)\frac{\log n}{\log\log n}\\
&\overset{(|C_\sigma|\leq h_1)}{\leq} && 80\,(h_2+1)\left(h_1 f_c^\sigma + h_1\sum_{r\in R_c} d(r,c)\right)\frac{\log n}{\log\log n}\\
&\leq && 160\,h_2 h_1\left(f_c^\sigma + \sum_{r\in R_c} d(r,c)\right)\frac{\log n}{\log\log n}\\
&\overset{(h_1\cdot h_2=h)}{=} && 160\,h\left(f_c^\sigma + \sum_{r\in R_c} d(r,c)\right)\frac{\log n}{\log\log n}.
\end{aligned}$$

As before, the optimal solution pays at least $f_c^\sigma + \sum_{r\in R_c} d(r,c)$ for the center and thus, the lemma follows. ■

The previous analysis can be done for every optimal center yielding Theorem 3.3.

Theorem 3.3 — Randomized Competitive Ratio. Given a h-dividable construction cost function, RA-MCOFLP achieves a competitive ratio of $\mathcal{O}\left(h\frac{\log n}{\log\log n}\right)$ against the oblivious adversary for the multi-commodity online facility location problem.

Proof. The theorem follows when applying Lemma 3.13 for every optimal center separately. ■

3.5.5 On h-dividable Cost Functions

Based on the analyses of PD-MCOFLP and RA-MCOFLP, we can immediately derive bounds on the competitive ratios by analyzing the construction cost function regarding Definition 3.1. In the following, we state our findings for different classes of cost functions and show how the parameter h can be chosen suitably. We first state a general upper bound on h and extend the results presented in our publication [31]. After that, we present functions where an improved h can be determined. Further, we design a function that is not $\mathcal{O}(\sqrt{|S|})$-dividable showing the limits of our concept of h-dividable cost functions.

A trivial upper bound on h. Note that by the sub-additivity property of all reasonable cost functions (see Section 3.1), every construction cost function is $|S|$-dividable as stated by Observation 3.1. A suitable instantiation is an algorithm that only constructs facilities containing exactly one commodity, i.e., in the worst case, our algorithms can always fall back to the trivial approach that treats every commodity separately. Of course, we aim at lower values for h.

Observation 3.1 Every cost function f, where f_m^σ reflects the cheapest way of offering commodities of σ at m, is $|S|$-dividable.

Functions dependent on the number of offered commodities. In our publication [31], we framed our algorithms as less general. We show that they achieve competitive ratios dependent on $\sqrt{|S|}$ when for all $m \in M$ and all $\sigma \subseteq S$ it holds that $f_m^\sigma/|\sigma| \geq f_m^S/|S|$. From the perspective of a h-dividable cost function, we can extend the result to Theorem 3.4 below. Observe how the theorem applies to any function depending on the location and the size of the offered configuration. So, our algorithms' dependence on $|S|$ is asymptotically optimal for all those functions. Especially, the function used in the lower bound (see Theorem 3.1) falls into that class. For the latter, a small adaptation of the proof allows avoiding the factor of 2 from Theorem 3.4.

Theorem 3.4 For $c \geq 1$, a cost function f where for all $\sigma \subseteq S$ and for all $m \in M$ it holds that $c \cdot \frac{f_m^\sigma}{|\sigma|} \geq \frac{f_m^S}{|S|}$ is $c \cdot \sqrt{|S|}$-dividable. Any sub-additive and monotone function solely dependent on the size of a configuration and the location fulfills the condition with $c = 2$.

Proof. We show the properties of Definition 3.1 with $h_1 = c \cdot \sqrt{|S|}$, $h_2 = 1$, and a single configuration consisting of all commodities S. Thus, any algorithm for achieving the bounds of the theorem constructs only facilities for a single commodity or all commodities. Constraint (b) of Definition 3.1 is fulfilled as each commodity is inexactly one considered configuration. For constraint (a), we use the cover $C_\sigma = \{S\}$. Clearly, $|C_\sigma| = 1 \leq c\sqrt{|S|} = h_1$ and the cover consists of members of $S \cup \{S\}$. Consider any location $m \in M$. Then it holds for all $\sigma \subseteq S$ with $|\sigma| > h_1$:

$$\sum_{\tau \in C_\sigma} f_m^\tau = f_m^S = \frac{f_m^S}{|S|}|S| \overset{\left(c \cdot \frac{f_m^\sigma}{|\sigma|} \geq \frac{f_m^S}{|S|}\right)}{\leq} c\,\frac{|S|}{|\sigma|} f_m^\sigma \overset{(|\sigma| > h_1 \geq \sqrt{|S|})}{\leq} c\,\frac{|S|}{\sqrt{|S|}} f_m^\sigma = c\sqrt{|S|}\, f_m^\sigma = h_1\, f_m^\sigma$$

Thus, constraint (a) of Definition 3.1 is fulfilled, and the first part of the theorem holds. Consider any function dependent on the size of the configuration and the location. Then, for any $m \in M$ and any $\sigma \subset S$, the lowest cost for covering S at m must be lower than the cost of covering S by subsets of size $|\sigma|$ at m. Therefore, it holds:

$$f_m^S \leq \left\lceil \frac{|S|}{|\sigma|} \right\rceil f_m^\sigma \leq \left(\frac{|S|}{|\sigma|} + 1\right) f_m^\sigma \Leftrightarrow \frac{f_m^S}{|S| + |\sigma|} \leq \frac{f_m^\sigma}{|\sigma|} \overset{(|\sigma| \leq |S|)}{\Rightarrow} \frac{f_m^S}{|S|} \leq 2\,\frac{f_m^\sigma}{|\sigma|}$$

Thus, the cost function fulfills the condition. ■

Functions where $h \leq \sqrt{|S|}$. Next, we show a class of functions for which we can determine a parameter h usually lower than $\sqrt{|S|}$. For such functions, our algorithms achieve competitive ratios that depend on $|S|$ better than in the worst-case lower bound. The class contains functions that depend on the location and the offered configuration's size. We parameterize the functions by x (not to be confused with the number of configurations for Definition 3.1). For a visualization of the functions of $\mathcal{F}$, consider Figure 3.12. Observe that the class captures functions that behave like a square root function dependent on x from a linear function ($x = 2$) to a linear function ($x = 0$).

Theorem 3.5 Consider the family of functions $\mathcal{F} = \{g_x(|\sigma|) = |\sigma|^{\frac{x}{2}} \,|\, x \in [0,2]\}$. A cost function f, where $f_m^\sigma = g_x(|\sigma|)$ for $g \in \mathcal{F}$, is $|S|^{\frac{x}{x+2}}$-dividable.

Proof. Set $h_1 = |S|^{\frac{x}{x+2}}$, $h_2 = 1$ and use a single configuration consisting of S. Thus, any algorithm for this setting constructs only facilities for a single commodity or all commodities. Constraint (b) of Definition 3.1 is fulfilled as each commodity is inexactly one considered configuration. Next, we show that constraint (a) of Definition 3.1 holds. Consider any location $m \in M$ and any configuration

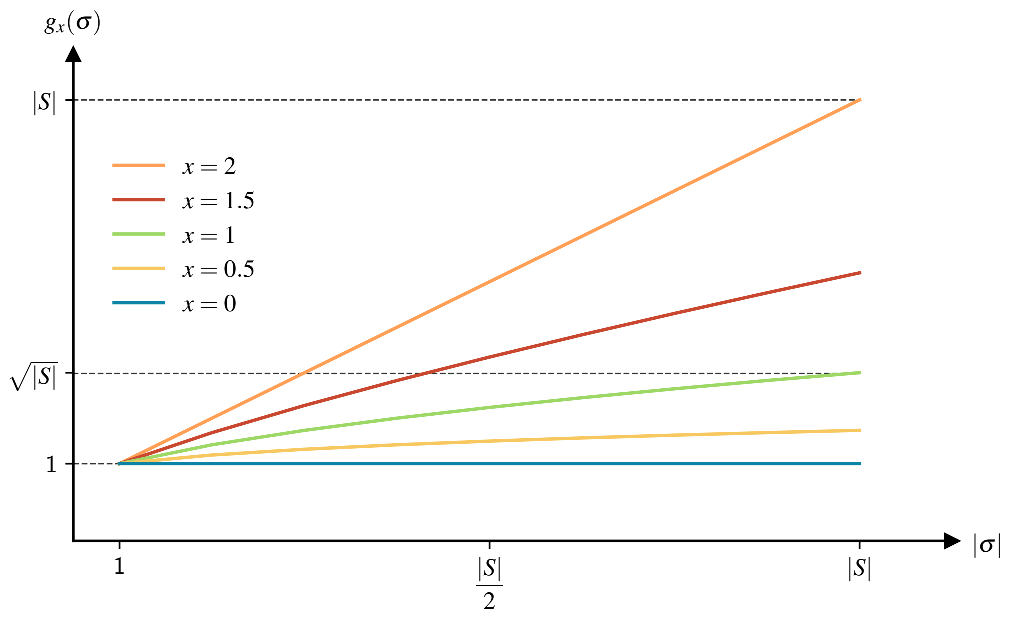

Figure 3.12: (Repetition of Figure 3.3) The figure sketches several examples for functions of the family $\mathcal{F}$. Note that the family expresses functions that behave like a square-root-function between a constant function ($x = 0$) and a linear function ($x = 2$).

$\sigma \subseteq S$ with $|\sigma| > h_1$. Then, we use the cover $C_\sigma = \{S\}$. Clearly, $|C_\sigma| = 1 \leq h_1$ and the cover is of the set $S \cup \{S\}$. For the cost, we have the following:

$$\sum_{\tau \in C_\sigma} f_m^\tau = f_m^S = g_x(|S|) = \frac{g_x(|S|)}{g_x(|\sigma|)} \cdot g_x(|\sigma|) \overset{(g_x(|\sigma|)=f_m^\sigma)}{=} \frac{g_x(|S|)}{g_x(|\sigma|)} \cdot f_m^\sigma \overset{(|\sigma| \geq h_1)}{\leq} \frac{|S|^{\frac{x}{2}}}{\left(|S|^{\frac{x}{x+2}}\right)^{\frac{x}{2}}} \cdot f_m^\sigma$$

$$= \left(|S|^{1-\frac{x}{x+2}}\right)^{\frac{x}{2}} \cdot f_m^\sigma = \left(|S|^{\frac{2}{x+2}}\right)^{\frac{x}{2}} \cdot f_m^\sigma = |S|^{\frac{x}{x+2}} \cdot f_m^\sigma = h_1 \cdot f_m^\sigma$$

Therefore, h_1 and h_2 fulfil the definition of h-dividable and the theorem holds. ■

On the one hand, we prove upper bounds dependent on x as in Theorem 3.5 that show how the competitive ratios behave. On the other hand, for such a parameterized class of functions, we can derive lower bounds such as the one in Theorem 3.6 below.

Theorem 3.6 Consider the family of functions $\mathcal{F} = \{g_x(|\sigma|) = |\sigma|^{\frac{x}{2}} \mid x \in [0,2]\}$. For a cost function f, where $f_m^\sigma = g_x(|\sigma|)$ for $g \in \mathcal{F}$, every randomized online algorithm has a competitive ratio of at least $\Omega\left(\min\left\{\sqrt{|S|}^{\frac{2-x}{2}}, \sqrt{|S|}^{\frac{x}{2}}\right\} + \frac{\log n}{\log\log n}\right)$ against the oblivious adversary.

Proof. Consider the sequence constructed in the proof of the lower bound in Theorem 3.1. Fix $g_x \in \mathcal{F}$. Independent of the cost function, we concluded in Equation (3.3) that if the algorithm does not proceed $\frac{\sqrt{|S|}}{2}$ rounds it has to cover expectedly $\mathrm{E}[T] \geq \frac{|S|}{16}$ commodities. In the former case, ALG pays at least $(\frac{\sqrt{|S|}}{2})\, g(1) = \frac{\sqrt{|S|}}{2}$. In the latter case, ALG pays at least

$$g_x(\mathrm{E}[T]) \geq g_x\left(\frac{|S|}{16}\right) = \left(\frac{|S|}{16}\right)^{\frac{x}{2}} \geq \frac{\sqrt{|S|}^x}{16}. \tag{3.16}$$

Thus, ALG's expected cost is at least

$$\frac{1}{16}\min\{\sqrt{|S|}, \sqrt{|S|}^x\}.$$

OPT pays at most $g_x(\sqrt{|S|}) = \sqrt{|S|}^{\frac{x}{2}}$. Thus, the resulting competitive ratio is at least

$$\frac{\frac{1}{16}\min\{\sqrt{|S|}, \sqrt{|S|}^{x}\}}{\sqrt{|S|}^{\frac{x}{2}}} = \frac{1}{16}\min\{\sqrt{|S|}^{\frac{2-x}{2}}, \sqrt{|S|}^{\frac{x}{2}}\}.$$

As in the proof of Theorem 3.1, we can extend the sequence by using Fotakis' sequence of the lower bound in [48] to arrive at the theorem. ■

For a depiction of how x influences the dependence on $|S|$ in the upper/lower bounds for the competitive ratios, see Figure 3.13. Here, we see that with increasing x, our upper bounds increase while the lower bound increases up to $x = 1$ and decreases after that. Notably are the positions for $x = 0$, $x = 1$, and $x = 2$. For $x = 0$, the cost function is constant, i.e., offering all services is as cheap as offering any single one. Our lower and upper bounds are constant and independent of x and $|S|$. For $x = 1$, the lower bound reaches a peak at $\sqrt[4]{|S|}$. Here, our upper bound slightly diverges from the lower bound with a value of $\sqrt[3]{|S|}$. After that, for $x = 2$, the cost function is linear, i.e., offering any set of commodities is as costly as offering each separately. The lower bound is constant and independent of $|S|$ again. Unfortunately, our upper bound increases for $x \geq 1$ even further but is bounded by $\sqrt{|S|}$. The latter is not surprising as the functions of $\mathcal{F}$ underlie the bound of Theorem 3.4.

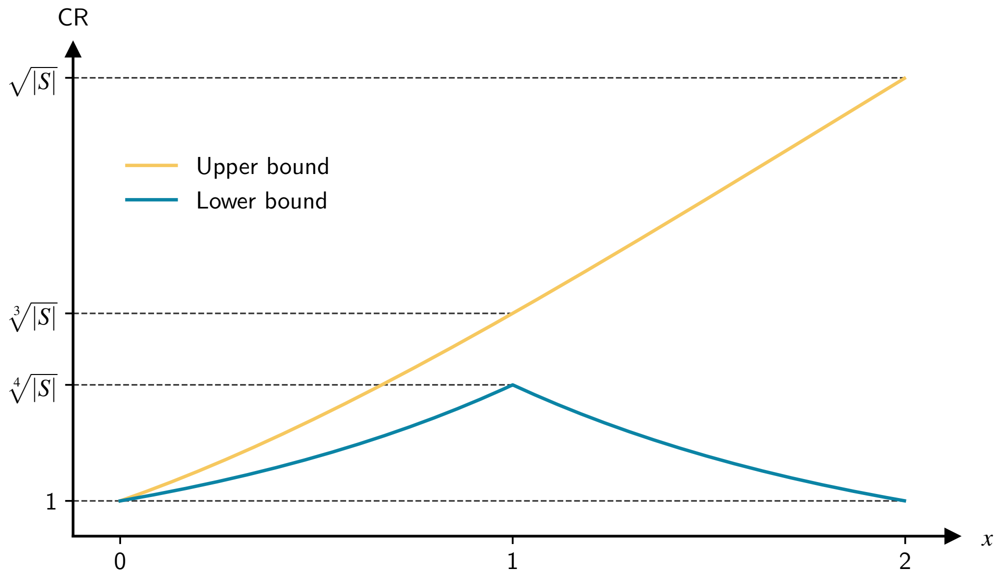

Figure 3.13: (Repetition of Figure 3.4) The competitive ratios by Theorems 3.5 and 3.6 can be compared regarding their dependence on $|S|$. For $x = 0$, the dependence on $|S|$ is a constant and optimal. For increasing x, the upper bound follows the lower bound up to $x = 1$. Here, the lower bound reaches a dependence of $\sqrt[3]{|S|}$ while the lower bound peaks at $\sqrt[4]{|S|}$. For $x > 1$, the lower bound falls until it is constant again. In the meantime, the upper bound increases to $\sqrt{|S|}$. The divergence is likely due to the lower bound ignoring increased connection costs even when the algorithm can cover a commodity set with a cost close to the optimal one.

The gap between the lower and the upper bound for increasing x can be explained as follows. Observe that the lower bound considers the construction cost function, not the connection cost. However, when a request is connected to multiple facilities, the connection cost might be significantly higher than the optimal connection cost to a single facility in an optimal configuration. For example, in the case of $x = 2$, assume we offer only facilities with a single commodity. Then, the construction cost is always optimal, as combining commodities does not yield an advantage. Consider that an algorithm has all its facilities at the position of an associated optimal one in configuration σ. Then, for each request connected to the optimal facility, the optimal solution pays the distance between both. On the other hand, the algorithm pays $|\sigma|$ times the connection costs. The same effect does not appear for small x. For example, when $x = 0$, the algorithm could always construct facilities offering all commodities without extra cost. Thus, in the scenario above, where for $x = 2$, the algorithm pays $|\sigma|$ times the connection cost, it only pays it once for $x = 1$. For a technical perspective, consider the proofs of Lemmas 3.9 and 3.13. Here, we use $|C_\sigma| \leq h_1$ to bound the connection costs of the algorithms in the optimal connection cost. For higher x, an algorithm would tend to construct only facilities offering few commodities. As a result, larger covers are required to satisfy all optimal configurations. Therefore, h_1 increases immediately increasing the dependence on $|S|$ in the competitive ratio. Thus, the upper bound of Figure 3.13 comes closer to reality than the lower bound.

Functions where $h \notin \mathcal{O}(\sqrt{|S|})$. Next, we show that there are functions such that $h \notin \mathcal{O}(\sqrt{|S|})$. More specifically, no parameter $h \in \mathcal{O}(\sqrt{|S|})$ can be determined using a sufficiently large metric space. We construct such a function in the proof of Theorem 3.7 below.

Theorem 3.7 There is a construction cost function f and a metric space (M,d) such that there is no $h \in \mathcal{O}(\sqrt{|S|})$ for which f is h-dividable.

Proof. We prove the theorem by contradiction using a suitable construction cost function. Assuming there are commodity sets such that the function is h-dividable for $h \in \mathcal{O}(\sqrt{|S|})$, we apply a combinatorial argument to show that not all configurations can be covered as demanded by Constraint (a) of Definition 3.1.

Consider all possible commodity sets $S_1, \ldots, S_x$ of size $\frac{|S|}{2}$ (not to be confused with the configurations of Definition 3.1). There are $x = \binom{|S|}{|S|/2}$ many. Assume there are at least x locations. We define the following construction cost function f. Consider the m-th location for $1 \leq m \leq x$. Let f be defined as follows for m:

$$f_m^\sigma = \begin{cases} 1 & \text{if } \sigma \subseteq S_m \\ |S| & \text{else.} \end{cases}$$

Note that f fulfills all properties we demand in Section 3.1. Intuitively, for each location, there is one set of $|S|/2$ commodities that can be combined for a low price, while all other configurations are significantly more expensive.

Assume that f is $\mathcal{O}(\sqrt{|S|})$-dividable where the configurations are given by the set $\mathcal{C}$. As $h = h_1 \cdot h_2$, it follows that $h_1 \leq c_1 \sqrt{|S|}$ and $h_2 \leq c_2 \sqrt{|S|}$ for constants c_1 and c_2. Without loss of generality, we assume that $|S|$ is sufficiently large such that $c_1, c_2 < \sqrt{|S|}$. Next, we analyze how many of the sets $S_1, \ldots, S_x$ can be covered at their respective locations using configurations of $\mathcal{C}$ with a cost of h_1 times the optimal. For the rest of the proof, we mean by A covers B that A covers the commodities of B with a construction cost of at most h_1 times the optimal one.

We first consider a fixed subset $\mathcal{C}' \subseteq \mathcal{C}$ and analyze how many sets of $S_1, \ldots, S_x$ can be covered by using facilities offering configurations of $\mathcal{C}'$ and single commodity facilities. Note that, by Constraint (a) of h-dividable, it holds that

$$|\mathcal{C}'| \leq h_1 \leq c_1 \sqrt{|S|}. \tag{3.17}$$

For any S_m that is covered by $\mathcal{C}'$, assume that $\cup_{\sigma \in \mathcal{C}'} \sigma \nsubseteq S_m$. Then, $\cup_{\sigma \in \mathcal{C}'} \sigma$ contains a commodity outside of S_m and its cost is at least $|S| > h_1 f_m^{S_m}$ for sufficiently large $|S|$. Therefore, it holds that

$$\cup_{\sigma \in \mathcal{C}'} \sigma \subseteq S_m. \tag{3.18}$$

Assume that $|\cup_{\sigma\in\mathcal{C}'}\sigma| < |S|/2 - c_1\sqrt{|S|}$. Then for any location m, more than $c_1\sqrt{|S|} \geq h_1$ additional commodities must be supplied to cover S_m. These have a cost of more than $c_1\sqrt{|S|} \geq h_1 \geq h_1 f_m^{S_m}$. Therefore, it holds that

$$|\cup_{\sigma\in\mathcal{C}'}\sigma| \geq |S|/2 - c_1\sqrt{|S|}. \tag{3.19}$$

Assume that $|\cup_{\sigma\in\mathcal{C}'}\sigma| = |S|/2$. Then, $\mathcal{C}'$ can be used to cover exactly one configuration S_m because for every other configuration S_j, it holds that $\cup_{\sigma\in\mathcal{C}'}\sigma \not\subseteq S_j$ violating Equation (3.18). If $|\cup_{\sigma\in\mathcal{C}'}\sigma| = |S|/2 - 1$, $\mathcal{C}'$ can be used to cover exactly $|S|/2+1$ configurations of $S_1,\dots,S_x$ by supplying one additional commodity. Observe how the number of configurations of $S_1,\dots,S_x$ that can be covered with $\mathcal{C}'$ increases the smaller $|\cup_{\sigma\in\mathcal{C}'}\sigma|$becomes. Therefore, the maximum number of configurations of $S_1,\dots,S_x$ that can be covered with $\mathcal{C}'$ occurs for minimal $|\cup_{\sigma\in\mathcal{C}'}\sigma|$. Formally, the maximum number of configurations of $S_1,\dots,S_x$ that can be covered with $\mathcal{C}'$ is

$$x_{\mathcal{C}'} = \binom{|S| - |\cup_{\sigma\in\mathcal{C}'}\sigma|}{|S|/2 - |\cup_{\sigma\in\mathcal{C}'}\sigma|}.$$

Next, we show that $x_{\mathcal{C}'}$ is monotonically decreasing with increasing $y := |\cup_{\sigma\in\mathcal{C}'}\sigma|$. For that, we show that the following ratio is smaller than 1:

$$\frac{\binom{|S|-(y+1)}{|S|/2-(y+1)}}{\binom{|S|-y}{|S|/2-y}} = \frac{(|S|-y-1)!}{(|S|/2)!\,(|S|/2-y-1)!} \cdot \frac{(|S|/2)!\,(|S|/2-y)!}{(|S|-y)!} = \frac{|S|/2-y}{|S|-y} \overset{(0<y<|S|)}{<} 1$$

Therefore, the maximum number of configurations of $S_1,\dots,S_x$ that can be covered with $\mathcal{C}'$ is

$$x_{\mathcal{C}'} = \binom{|S| - |\cup_{\sigma\in\mathcal{C}'}\sigma|}{|S|/2 - |\cup_{\sigma\in\mathcal{C}'}\sigma|} \overset{\textit{Equation (3.19)}}{\leq} \binom{|S|/2 + c_1\sqrt{|S|}}{c_1\sqrt{|S|}}. \tag{3.20}$$

How many subsets $\mathcal{C}'$ can exist? Since there are $|S|$ many commodities and each can be in at most $h_2 \leq c_2\sqrt{|S|}$ configurations of $\mathcal{C}$, it holds that $|\mathcal{C}| \leq h_2|S| \leq c_2\sqrt{|S|}\cdot|S| \leq c_2|S|^2$. Due to Equation (3.17), the number of subsets $\mathcal{C}'$ is at most

$$|\mathcal{C}|^{c_1\sqrt{|S|}+1} \leq (c_2|S|^2)^{c_1\sqrt{|S|}+1}. \tag{3.21}$$

Combining the results from Equations (3.20) and (3.21), the number of configurations from $S_1,\dots,S_x$ one can cover with $\mathcal{C}$ is at most

$$x_{\mathcal{C}} \leq (c_2|S|^2)^{c_1\sqrt{|S|}+1} \cdot \binom{|S|/2 + c_1\sqrt{|S|}}{c_1\sqrt{|S|}}. \tag{3.22}$$

Next, we show that the number of Equation (3.22) is too small in comparison to the number x of configurations of $S_1,\dots,S_x$. More specifically we prove that $x_{\mathcal{C}} \in \omega(x)$. Consider the ratio of x and $x_{\mathcal{C}}$ as $|S|$ approaches infinity:

$$\begin{aligned}
\lim_{|S|\to\infty} \frac{x}{x_{\mathcal{C}}} &\overset{\textit{Equation (3.22)}}{\geq} \lim_{|S|\to\infty} \frac{\binom{|S|}{|S|/2}}{(c_2|S|^2)^{c_1\sqrt{|S|}+1}\cdot\binom{|S|/2+c_1\sqrt{|S|}}{c_1\sqrt{|S|}}} \\
&= \lim_{|S|\to\infty} \frac{\frac{|S|!}{|S|/2!\,|S|/2!}}{(c_2|S|^2)^{c_1\sqrt{|S|}+1}\cdot\frac{(|S|/2+c_1\sqrt{|S|})!}{|S|/2!\,(c_1\sqrt{|S|})!}} \\
&= \lim_{|S|\to\infty} \frac{|S|!}{|S|/2!}\cdot\frac{(c_1\sqrt{|S|})!}{(|S|/2+c_1\sqrt{|S|})!(c_2|S|^2)^{c_1\sqrt{|S|}+1}}
\end{aligned}$$

$$\begin{aligned}
&= \lim_{|S|\to\infty} \frac{(\frac{|S|}{2}+1)\cdot\ldots\cdot S}{(c_1\sqrt{|S|}+1)\cdot\ldots\cdot(\frac{|S|}{2}+c_1\sqrt{|S|})\cdot(c_2|S|^2)^{c_1\sqrt{|S|}+1}} \\
&= \lim_{|S|\to\infty} \frac{(\frac{|S|}{2}+c_1\sqrt{|S|}+1)\cdot\ldots\cdot S}{(c_1\sqrt{|S|}+1)\cdot\ldots\cdot(\frac{|S|}{2})\cdot(c_2|S|^2)^{c_1\sqrt{|S|}+1}} \\
&= \lim_{|S|\to\infty} \frac{1}{(c_2|S|^2)^{c_1\sqrt{|S|}+1}} \prod_{i=1}^{\frac{|S|}{2}-c_1\sqrt{|S|}} \frac{\frac{|S|}{2}+c_1\sqrt{|S|}+i}{c_1\sqrt{|S|}+i} \\
&\geq \lim_{|S|\to\infty} \frac{1}{(|S|^3)^{(c_1+1)\sqrt{|S|}}} \prod_{i=1}^{\frac{|S|}{2}-c_1\sqrt{|S|}} 2 \\
&\geq \lim_{|S|\to\infty} \frac{2^{\frac{|S|}{2}-c_1\sqrt{|S|}}}{|S|^{6c_1\sqrt{|S|}}} \geq \lim_{|S|\to\infty} \frac{2^{\frac{|S|}{4}}}{|S|^{6c_1\sqrt{|S|}}} = \lim_{|S|\to\infty} 2^{\frac{|S|}{4}-\log(|S|^{6c_1\sqrt{|S|}})} \\
&= \lim_{|S|\to\infty} 2^{\frac{|S|}{4}-6c_1\sqrt{|S|}\log(|S|)} \geq 2^{\frac{|S|}{4}-\frac{|S|}{8}} = 2^{\frac{|S|}{8}} = \infty
\end{aligned}$$

Therefore, the configurations of $\mathcal{C}$ are not sufficient to cover all configurations of $S_1,\ldots,S_x$ with a cost of at most h_1 times the optimal one, yielding a contradiction to the assumption that f is $\mathcal{O}(\sqrt{|S|})$-dividable. Thus, the theorem holds. ■

Consider the construction cost function of Theorem 3.7. It shows the limits of our approach and reveals the flaw of Definition 3.1. Theorem 3.7 holds because Constraint (a) of Definition 3.1 requires the configurations to be fixed *for all* locations. An algorithm could manage suitable commodity sets for different locations separately to perform well against the used construction cost function. Intuitively, one should consider single commodity facilities and, for each location m separately, a facility offering the cheap commodity set S_m. However, the problem is that our high-level approach of bidding an investment per commodity per request requires non-trivial adaptations. Consider a request r asking for all commodities at some location m. Assume at m any facility is very expensive. Further, assume there are $\binom{|S|}{|S|/2}$ locations in equal distance to m with a construction cost as in the proof of Theorem 3.7. If r invests in all facilities equally, the investment is too large. So, r has to select facilities (locations) receiving a larger investment. First, it is unclear how to determine such investment, as all choices are equally valid. Second, the choice of an algorithm can lead to an additional increase in the competitive ratio as it can be far from the optimal one. Therefore, we believe that handling functions as the one of Theorem 3.7 might require approaches very different from our current solutions.

Functions independent of the location. The function of Theorem 3.7 uses a very large metric space and exploits that a h-dividable cost function manages a set of configurations globally, i.e., for all locations. Consider any configuration σ of size $|S|/2$. The construction cost function enforces that σ is only cheap for exactly one location and very expensive for all others. When σ is selected, we can cover one of the configurations of size $|S|/2$ at a single location. At every other location, σ is too expensive and should not be used. Especially, one cannot combine σ with any other configuration to cover a set of commodities of size $|S|/2$ with a low cost. In contrast, assume a construction cost function independent of the location. Here, selecting σ can be good. As it is cheap for one location, it is cheap for all locations and can be used with other configurations for a low cost. Selecting configurations for Definition 3.1 becomes much simpler when the construction cost is independent of the location. Following that intuition, we claim Conjecture 3.1.

Conjecture 3.1 Every construction cost function independent of the location is $\mathcal{O}(\sqrt{|S|})$-dividable.

3.6 Leasing Facilities

In the following section, we consider the MCOFLP under the assumption that facilities are not open forever after construction. More specifically, we consider a model extension where a constructed facility has to be *leased* for future time steps.

One established model of the literature is the facility leasing model [1, 78], where there is a fixed set of available leases L. A lease determines the interval of time a facility remains open and can serve arriving requests. An algorithm must determine which lease it selects upon a facility's construction. The selected lease directly influences the construction cost, making it a tough decision for an online algorithm to select the optimal lease. This model can be seen as a combination of the classical facility location problem and the leasing problem introduced in Meyerson's parking permit problem [77]. We call it *facility leasing with fixed leases*. For example, it models situations where a third party sells the facilities and only offers certain fixed lease periods. In Section 3.6.1, we present a combined model for the MCOFLP with fixed leases and show how our algorithm PD-MCOFLP can be adapted to the combined model. The adapted algorithm achieves a competitive ratio of $\mathcal{O}(|L|\, h \log n)$ for h-dividable cost functions.

In addition, we consider a model where the time a facility remains open must not be determined during the construction. Instead, an open facility can be kept open each consecutive time step by paying a *maintenance cost*. If this cost is not paid, the facility closes and can only be re-opened if the construction cost is paid again. We call the model *facility leasing with maintenance cost*. It allows for more flexible management of facilities and models, for example, situations in which an open facility poses reoccurring costs, e.g., for power consumption or computation time on a server. Interestingly, the model is related to the fixed leases model. When the construction costs are independent of the location, and the maintenance costs are equal for all locations and configurations, we can use that relationship to design a deterministic algorithm for the maintenance cost model. In Section 3.6.2, we examine the relationship and present an analysis that shows that the extension mentioned above of PD-MCOFLP achieves a competitive ratio of $\mathcal{O}(h \log n)$. For a graphical comparison of the two considered leasing models, see Figure 3.14.

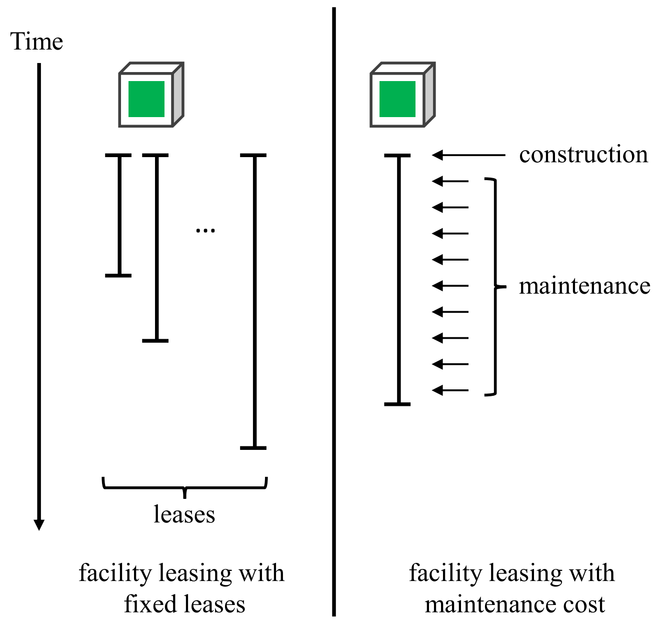
Time
...
leases
facility leasing with fixed leases
construction
maintenance
facility leasing with maintenance cost

Figure 3.14: In the facility leasing with fixed leases model, when a facility is constructed, the algorithm has to decide on a lease determining the construction cost and the time the facility remains open. The facility leasing with maintenance cost model allows a more flexible period a facility remains open. When the facility is constructed, a construction cost is paid. Maintenance costs must be paid to keep it open in every consecutive step.

3.6.1 Facility Leasing with Fixed Leases

We extend our results by combining our model of MCOFLP with the leasing model introduced in [78]. Below, we show the combined model, sketch a lower bound, explain how our deterministic algorithm PD-MCOFLP can be adapted to the leasing model, and sketch its analysis. The algorithmic idea combines PD-MCOFLP and the algorithm for facility leasing presented in [78]. Similarly, the analysis combines our analysis with the one of Nagarajan and Williamson. Thus, the techniques are not novel, and we concentrate on a high-level overview and a sketch of the results rather than completely repeating the proofs.

The model. Each arriving request is considered served as soon as it is connected to a set of facilities jointly offering the requested commodities. In addition to the model of MCOFLP presented in Section 3.1, there is a set of leases L. Each lease has an associated length, and the construction cost function $f_m^{\sigma\,\ell}$ depends on the location $m \in M$, the configuration $\sigma \subseteq S$ and the selected lease $\ell \in L$. A facility constructed at m for σ using a lease of length ℓ at time t is open up to time $t+\ell$. Afterward, it is closed and can no longer serve future requests. For convenience, we denote by $\mathcal{I}_t^\ell = [t, t+\ell]$ the time interval from t to $t+\ell$.

Linear programs. Our algorithm is based on PD-MCOFLP, so we first introduce the primal and dual linear program for the multi-commodity facility leasing problem below.

Primal for the multi-commodity facility leasing problem (simplified)

$$\min \sum_{m\in M}\sum_{\sigma\subseteq S}\sum_{t\in T}\sum_{\ell\in L} f_m^{\sigma\,\ell} y_m^{\sigma\,\ell t} + \sum_{m\in M}\sum_{\sigma\subseteq S}\sum_{r\in R}\sum_{(t,\ell):r_r\in\mathcal{I}_t^\ell} d(m,r)x_{mr}^{\sigma\,\ell t}$$

$$\text{s.t.} \sum_{m\in M}\sum_{\sigma\subseteq S:e\in\sigma}\sum_{(t,\ell):r_r\in\mathcal{I}_t^\ell} x_{mr}^{\sigma\,\ell t} \geq 1 \qquad \forall r\in R, \forall e\in s_r$$

$$x_{mr}^{\sigma\,\ell t} \leq y_m^{\sigma\,\ell t} \qquad \forall m\in M, \forall\sigma\subseteq S, \forall r\in R, \forall t\in T, \forall \ell\in L$$

$$x_{mr}^{\sigma\,\ell t}, y_m^{\sigma\,\ell t} \in \{0,1\} \qquad \forall m\in M, \forall\sigma\subseteq S, \forall r\in R, \forall t\in T, \forall \ell\in L$$

Here, $y_m^{\sigma\,\ell t}$ is a variable indicating that a facility in configuration $\sigma \subseteq S$ has been leased at $m \in M$ with the lease $\ell \in L$ starting at time $t \in T$. $x_{mr}^{\sigma\,\ell t}$ indicates that request r is connected to a facility at $m \in M$ for configuration $\sigma \subseteq S$ with lease $\ell \in L$ opened at $t \in T$. The first set of constraints ensures that connections to respective facilities serve all commodities of a request. The second set of constraints ensures that a request is only connected to facilities that are also open upon its arrival.

In comparison to the primal of the multi-commodity facility location problem as presented in Section 3.5.3.1, we can see that essentially, the selected lease ℓ and its starting time t come into play for the variables y and x. Intuitively, the lease and the starting time are needed to specify a facility. Below, we present the simplified dual:

Dual for the multi-commodity facility leasing problem (simplified)

$$\max \sum_{r\in R}\sum_{e\in s_r} a_{re}$$

$$\text{s.t.} \sum_{r\in R:\, r_t\in\mathcal{I}_t^\ell}\left(\sum_{e\in s_r\cap\sigma} a_{re} - d(m,r)\right)_+ \leq f_m^{\sigma\,\ell} \qquad \forall m\in M, \forall\sigma\subseteq S, \forall t\in T, \forall\ell\in L$$

$$a_{re} \geq 0 \qquad \forall r\in R, \forall e\in s_r$$

Again, the only major difference to the dual for the multi-commodity facility location problem is that the constraints now depend on the selected lease. As one can see, the dual variables are the same as in the non-leasing variant. Therefore, we can extend the deterministic algorithm

PD-MCOFLP presented in Section 3.5.3 by leasing. Intuitively, the extension is similar to the extension of Fotakis' algorithm [48] by leasing presented in [78].

A lower bound. Before considering an algorithmic solution, we consider the lower bound on the parking permit problem presented in [77]. Meyerson showed that every deterministic online algorithm for the parking permit problem has a competitive ratio of at least $\Omega(|L|)$, where L is the set of permits (leases). Since the parking permit problem is a special case of the facility leasing problem at a single location, we can extend our lower bound for MCOFLP to the following for deterministic algorithms. The lower bound sequence starts with Meyerson's sequence for the parking permit problem [77] at a starting location for a fixed commodity. Afterward, we continue with the sequence of Theorem 3.1. For randomized algorithms against the oblivious adversary, one can show a lower bound of $\Omega(\log|L| + \sqrt{|S|} + {}^{\log n}/_{\log\log n})$ in the same way.

Theorem 3.8 No deterministic online algorithm for the multi-commodity online facility leasing problem with fixed leases can achieve a competitive ratio better than $\Omega\left(|L| + \sqrt{|S|} + \frac{\log n}{\log\log n}\right)$, even on a line metric.

A deterministic algorithm. We use PD-MCOFLP with the following adaptation. The constraints are changed such that they respect that requests only consider facilities open when they arrive. Additionally, the sets of constraints (2) and (4) are expanded to hold for all $\ell \in L$. Of course, the actions the algorithm executes when a constraint becomes tight will reflect the respective lease ℓ for which the constraint became tight. Intuitively, in PD-MCOFLP, for each request r and the desired commodity $e \in s_r$, the investment $I_{(r,e)}$ was invested in the connection of r to a facility serving e and the construction of facilities serving e (once for each such facility). Now, we have for each potential facility that serves e different lease types. Therefore, the investment is once invested for each lease type of L. The additional investment is reflected in the competitive ratio's additional factor of $|L|$. The algorithm must invest in all lease types because it does not know the optimal one and cannot circumvent buying the latter to be competitive. All other lease types may be very expensive when applied for the time interval a facility has to stay open.

The analysis. Note that due to the extended model, Definition 3.1 must be adjusted such that the first condition reads:

(a) For all $\ell \in L$, for all $m \in M$, and for all $\sigma \subseteq S$ with $|\sigma| > h_1$ there is a set C_σ with $|C_\sigma| \leq h_1$ where each $\tau \in C_\sigma$ is an element of $S \cup_{1 \leq y \leq x} \{S_y\}$ such that $\sigma \subseteq \cup_{\tau \in C_\sigma} \tau$ (C_σ *covers* σ) and $\sum_{\tau \in C} f_m^{\tau\ell} \leq h_1 f_m^{\sigma\ell}$.

The adaptation of the definition is more of a technical one. It ensures that all requests possibly served by any facility of some lease ℓ can be served by a set of facilities managed by our algorithm for the entire lease ℓ. Next, we sketch how the analysis of PD-MCOFLP presented in Section 3.5.3.3 has to be adapted to show the theorem below.

Theorem 3.9 Given a h-dividable cost function (adapted to leasing), an adaptation of PD-MCOFLP for leasing achieves a competitive ratio of at most $\mathcal{O}(|L|\, h \log n)$ for the multi-commodity online facility leasing problem with fixed leases.

We consider all lemmas of the proof of Section 3.5.3.3 step by step and explain how they must be adapted. In general, we assume more structure to the leases as done in [3, 77, 78], losing only a constant factor of at most 2.

Lemma 3.14 — Lemma 5.1 of [78] with adapted notation. Given an instance P of the facility leasing problem, it can be converted into another instance P' of the facility leasing problem such that: Leases of type ℓ are only available at times t divisible by ℓ such that any solution to P can be converted into a solution to P' that costs no more than twice as much.

We call the time interval from the point in time t where a lease of type ℓ can be purchased until time $t+\ell$ an *interval of* ℓ. First, consider the cost of the algorithm regarding its investments $I_{(r,e)}$. Note that the adapted algorithm's investment per request/commodity is the same as the one of PD-MCOFLP except that it respects the additional constraints due to the lease types. Since the serving cost does not change and the second condition of Definition 3.1 remains the same, Lemma 3.1 remains true as is.

Lemma 3.1 The serving cost of PD-MCOFLP is at most $(h_2+1)\sum_{r\in R}\sum_{e\in s_r} I_{(r,e)}$.

The next lemma, Lemma 3.2, needs to be adapted to incorporate the number of leases $|L|$ in the upper bound. The upper bound becomes $|L|\cdot\sum_{r\in R}\sum_{e\in s_r\cap\sigma} I_{(r,e)}$. This is, as the algorithm invests for each fixed $\sigma\in S\cup_{1\le y\le x}\{S_y\}$ in each $\ell\in L$.

Lemma 3.2 Fix a configuration $\sigma\in S\cup_{1\le y\le x}\{S_y\}$. The construction cost for facilities in configuration σ of PD-MCOFLP is at most $\sum_{r\in R}\sum_{e\in s_r\cap\sigma} I_{(r,e)}$.

As a consequence of the higher bound in Lemma 3.2, the bound of Lemma 3.3 increases by a factor of $|L|$ to $2|L|(h_2+1)\sum_{r\in R}\sum_{e\in s_r} I_{(r,e)}$ as well.

Lemma 3.3 The total cost of PD-MCOFLP is at most $2(h_2+1)\sum_{r\in R}\sum_{e\in s_r} I_{(r,e)}$.

In the previous lemmas, we can see how the factor of $|L|$ enters the competitive ratio. The remaining lemmas must be adjusted to the leasing setting but do not influence the competitive ratio other than before. Lemma 3.7 needs to be restricted to a fixed lease length $\ell\in L$. Additionally, both requests j, r must appear during the same interval of lease type ℓ.

Lemma 3.7 Consider a configuration $\sigma\in S\cup_{1\le y\le x}\{S_y\}$ and two requests j,r such that j arrives before r. At the time when the $I_{(r,e)}$ variables are increased, it holds for all $m\in M$ that $d(F(\sigma),j)-d(m,j)\ge\sum_{e\in s_r\cap\sigma} I_{(r,e)}-d(m,r)-2d(m,j)$.

As the previous lemma, Lemmas 3.8 and 3.9 must be restricted to a fixed lease length $\ell\in L$. Similar to the proof of Lemma 5.2 in [78], we only consider requests for a commodity of σ in the current interval of ℓ, as other requests do not contribute to the construction of a facility in the interval. The rest of the lemmas can be shown as in the proof in Section 3.5.3.3. Especially the statements regarding c-ordered covering remain the same as they are independent of the rest of the analysis. Technically, the adapted constraint (a) of Definition 3.1 ensures the following for a fixed lease ℓ: The construction cost of the algorithm to cover any configuration is bounded by the construction cost for the configuration itself.

Lemma 3.8 Fix a configuration $\sigma\in S\cup_{1\le y\le x}\{S_y\}$. For any $R'\subseteq R$ and any facility serving σ at any $m\in M$ it holds that

$$\sum_{r\in R'}\left(\sum_{e\in s_r\cap\sigma}\frac{I_{(r,e)}}{5H_n}-d(m,r)\right)_+\le f_m^{\sigma}.$$

Lemma 3.9 For any configuration $\sigma \subseteq S$, any $R' \subseteq R$, and any facility serving σ at any $m \in M$ it holds that

$$\sum_{r \in R'} \left(\sum_{e \in s_r \cap \sigma} \gamma I_{(r,e)} - d(m,r) \right)_+ \leq f_m^\sigma.$$

Finally, Theorem 3.9 can be shown similar to Theorem 3.2 by applying that scaling all $I_{(r,e)}$ by γ implies a feasible solution to the dual and thus, a lower bound to the optimal solution. Additionally, the adapted Lemma 3.3 upper bounds the cost of the algorithm by all $I_{(r,e)}$. By both facts, the competitive ratio of the adapted algorithm is bounded by $\mathcal{O}(|L|\, h \log n)$.

3.6.2 Facility Leasing with Maintenance Cost

In contrast to the model presented in Section 3.6.1 above, we can also consider the following model, which allows for more flexibility regarding the lease lengths. The alternative model is especially useful in extended scenarios where closing a facility might be necessary. For example, imagine a situation where certain commodities conflict and cannot be stored close to each other. Here, it might be necessary to close facilities at a location again to make room for commodities that conflict with previously offered ones. Next, we present the alternative model and show that our algorithm PD-MCOFLP can be applied to this model while losing at most a constant factor in the competitive ratio if the construction costs are independent of the location and the maintenance costs are equal for all locations and configurations.

The model. Each request is considered served as soon as it is connected to a set of facilities jointly offering the requested commodities. When a facility in configuration $\sigma \subseteq S$ is constructed at location $m \in M$, the cost is f_m^σ. After that, in each time step the facility should remain open, the algorithm has to pay a *maintenance cost* of c_m^σ. As soon as the maintenance cost for a facility is not paid, the respective facility closes and must be opened for a cost of f_m^σ again if wanted.

Our results. Interestingly, the new model is closely related to the leasing model introduced above in Section 3.6.1. We can show the following theorem:

Theorem 3.11 Any instance P of the maintenance cost model can be transformed into an instance P' of the fixed leases model such that:
(a) A solution to P' translates to a solution to P with the same cost.
(b) A solution to P translates to a solution to P' which costs at most twice as much.

Proof. Consider the instance P for the maintenance cost model. To have an instance P' for the fixed leases model, define for every configuration at every location a lease of length $\frac{f_m^\sigma}{c_m^\sigma}$ with a cost of $f_m^{\sigma\ell} = 2 f_m^\sigma$. Consider a solution to P'. For every facility for σ that is leased at m at time t, construct a facility for σ at m at time t and pay the maintenance cost for $\frac{f_m^\sigma}{c_m^\sigma}$ time steps. The cost of the facility in the solution to P' is $2 f_m^\sigma$. The cost of the facility in the solution to P is $f_m^\sigma + \frac{f_m^\sigma}{c_m^\sigma} \cdot c_m^\sigma = 2 f_m^\sigma$. Thus, (a) holds.

Figure 3.15 depicts the following construction. Consider a solution for P. A solution for P' can be constructed as follows. For each facility constructed at m in σ, consider the number of time steps t until the maintenance cost is no longer paid. Subdivide t in intervals of length $\frac{f_m^\sigma}{c_m^\sigma}$. The last such interval may be shorter than $\frac{f_m^\sigma}{c_m^\sigma}$. For each such interval, construct and lease the facility in P' for a cost of $f_m^{\sigma\ell} = 2 f_m^\sigma$ with the lease of length $\frac{f_m^\sigma}{c_m^\sigma}$. If the solution in P results in only one interval, it has a cost of at least f_m^σ, and the cost at most doubled in P'. If the solution in P yields more than one interval, it has a cost of at least $f_m^\sigma + \frac{f_m^\sigma}{c_m^\sigma} \cdot c_m^\sigma = 2 f_m^\sigma$ for the first and the last interval. The first

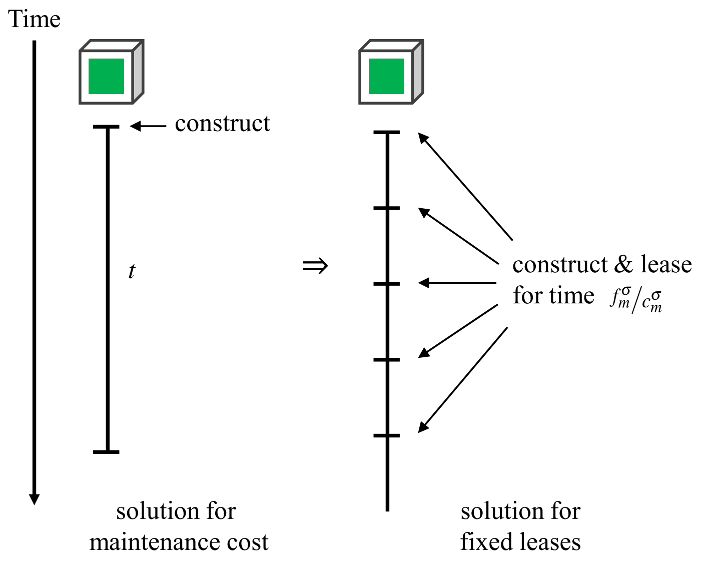

Figure 3.15: A solution for the maintenance cost model can be transformed into a solution for the fixed lease model with one lease per commodity per location. Here, each time interval t during which a facility for σ stays open is split into multiple time spans of length f_m^σ/c_m^σ. In each, the facility is constructed and leased in the fixed lease model for a cost of $2\,f_m^\sigma$.

and the last interval cost $4\,f_m^\sigma$ in P', and the cost doubled at most. For any interval between the first and the last, the cost in P is at least $\frac{f_m^\sigma}{c_m^\sigma} \cdot c_m^\sigma = f_m^\sigma$ while it costs at most double in P'. Applying the transformation for every facility in P yields a solution to P' with at most twice the cost. ■

The implication of Theorem 3.11 is the following: Suppose we have an algorithm ALG for the fixed leases model that achieves a competitive ratio of c. In that case, we can construct an algorithm $\overline{\text{ALG}}$ for the maintenance cost model with a competitive ratio of $2c$. Given an instance of the maintenance cost model, construct an instance for the fixed leases model following Theorem 3.11. Execute ALG on the latter and translate its solution to a solution $\overline{\text{ALG}}$ for the maintenance cost model. By Theorem 3.11, the costs of ALG and $\overline{\text{ALG}}$ are equal. Let $\overline{\text{OPT}}$ be an optimal solution to the problem in the maintenance cost model. Then, we can transform $\overline{\text{OPT}}$ to a solution for the fixed leases model OPT with a cost of at most twice the cost of $\overline{\text{OPT}}$. As ALG has a competitive ratio of c in the fixed leases model, it holds for the cost that $C_{\overline{\text{ALG}}} = C_{\text{ALG}} \leq c\,C_{\text{OPT}} \leq 2\,c\,C_{\overline{\text{OPT}}}$.

Next, we consider the case of uniform construction costs: for all $\sigma \subseteq S$ and all $m_1, m_2 \in M$ the construction costs $f_{m_1}^\sigma = f_{m_2}^\sigma$ are equal. Further, assume the maintenance costs are equal for all locations and configurations. These assumptions limit the possible leases in the fixed leases instance created by Theorem 3.11. Only one lease is available for every $\sigma \subseteq S$. However, applying the algorithm of Section 3.6.1 as described above only yields a competitive ratio of $\mathcal{O}(h_2\,|S|\,h\log n)$ because all leases for every configuration the algorithm builds could be distinct and $|L|$ as large as the number of considered configurations. The algorithm considers at most $(h_2+1)\,|S|$ and at least $|S|$ different configurations, and thus, the competitive ratio is relatively high. Fortunately, we can strengthen the analysis of the algorithm for the fixed leases model by using that for each commodity σ, only one lease is possible. This results in a much stronger bound on the competitive ratio:

Theorem 3.10 Consider construction costs independent of the location and maintenance costs equal for all locations and configurations. There is a deterministic algorithm for the multi-commodity online facility leasing problem with a maintenance cost that achieves a competitive ratio of at most $\mathcal{O}(h\log n)$ if the construction cost function is h-dividable.

Proof. Consider an instance P for the maintenance cost model with cost functions f^σ and c^σ and transform it to an instance P' for the fixed leases model as in the proof of Theorem 3.11. Since the maintenance cost is equal for all locations and all configurations, we assume it is 1 for simplicity. Then, for $\sigma \subseteq S$, let $\ell(\sigma)$ be the lease of length f^σ and cost $f^{\sigma\,\ell(\sigma)} = 2\,f^\sigma$ in P'.

Next, we define an algorithm for P'. The algorithm is similar to the adaptation of PD-MCOFLP to leasing as presented in Section 3.6.1. However, each investment $I_{(r,e)}$ of a request r regarding commodity $e \in s_r$ is not invested for all possible leases but only the one associated with σ introduced in the transformation in Theorem 3.11 for $\sigma \in S \cup_{1 \leq y \leq x} \{S_y\}$. More specifically, it is invested for any such fixed σ in the lease $\ell(\sigma)$. As we have seen in the sketch of the analysis in Section 3.6.1, this implies that the overall cost of the algorithm is at most $(h_2+1)\sum_{r\in R}\sum_{e\in s_r} I_{(r,e)}$. Note that it is especially not blown up by $|L|$ as in the general case.

Next, observe that the instance P' is a special case of the facility leasing with fixed leases model. For any configuration σ, there is only one available lease $\ell(\sigma)$. The optimal solution can only select one lease for σ, too. So, our adaption of Definition 3.1 can be loosened. It suffices if the first condition reads:

(a) For all $\sigma \subseteq S$ with $|\sigma| > h_1$ there is a set C_σ with $|C_\sigma| \leq h_1$ where each $\tau \in C_\sigma$ is an element of $S \cup_{1 \leq y \leq x} \{S_y\}$ such that $\sigma \subseteq \cup_{\tau \in C_\sigma} \tau$ (C_σ *covers* σ) and $\sum_{\tau \in C_\sigma} f^{\tau \ell(\sigma)} \leq h_1 f^{\sigma \ell(\sigma)}$.

Note that it is still required that the configurations of C_σ are offered the entire duration of $\ell(\sigma)$. Next, we show that the above constraint is fulfilled. Since the construction cost function is h'-dividable (in the classical sense), for any $\sigma \subseteq S$ with $|\sigma| > h'_1$, there is a set C_σ with $|C_\sigma| \leq h'_1$ that covers σ by subsets $\{e\}$ for all $e \in S$ and S_i for all $1 \leq i \leq x$ such that $\sum_{\tau \in C_\sigma} f^\tau \leq h'_1 f^\sigma$. Note that any configuration τ can be leased for a length of f^τ. Condition (a) requires that τ is leased for a length of f^σ. The cost for that is:

$$f^{\tau \ell(\sigma)} := f^{\tau \ell(\tau)} \cdot \left\lceil \frac{f^\sigma}{f^\tau} \right\rceil \leq 2 f^{\tau \ell(\tau)} \cdot \frac{f^\sigma}{f^\tau} \tag{3.23}$$

So, the total cost for all $\tau \in C_\sigma$ to be open the entire lease of length f^σ is:

$$\begin{aligned}\sum_{\tau \in C_\sigma} f^{\tau \ell(\sigma)} &\overset{\textit{Equation } (3.23)}{\leq} \sum_{\tau \in C_\sigma} 2 f^{\tau \ell(\tau)} \cdot \frac{f^\sigma}{f^\tau} \overset{(f^{\tau\ell(\tau)}=2f^\tau)}{=} \sum_{\tau \in C_\sigma} 4 f^\tau \cdot \frac{f^\sigma}{f^\tau} = \sum_{\tau \in C_\sigma} 4 f^\sigma = 4|C_\sigma| f^\sigma \\ &\overset{(|C_\sigma| \leq h'_1)}{\leq} 2 h'_1 f^{\sigma \ell(\sigma)}\end{aligned}$$

Setting $h_1 = 2h'_1$ ensures that Condition (a) from above holds, while we only lose a constant factor of 2 in the competitive ratio. Using the same techniques as before for the fixed leases model, one can show the adaptations of Lemmas 3.8 and 3.9 again. Finally, one can show Theorem 3.10 in the same way as Theorems 3.2 and 3.9. ■

4. Multi-Commodity Online k-Server

In the following chapter, we present the *multi-commodity online k-server problem* (MCOKSP) as a generalization of the k-server problem. The k-server problem incorporates various fundamental online problems such as paging. Here, k identical resources (called *servers*) are moved to serve incoming requests. Any request can be served by moving one of the servers to its location. In the spirit of the introductory scenario (see Chapter 1), virtual machines have to be always migrated. For example, situations can be modeled where each request has so much data that must be communicated to the desired service that a route through the network is no option. Another motivation is security concerns requiring virtual machines to run at the client's location when in use.

Each resource offers a fixed set of commodities in the problem's multi-commodity variant. Any arriving request also declares a set of possible commodities, of which one is required to serve it. Still, an algorithm can only migrate resources and has to place a resource providing one of the declared commodities at each request's location. The multi-commodity case naturally extends the classical case by allowing for heterogeneous virtual machines offering different bundled services. Such situations are common, for example, in the introductory example. Compared to the multi-commodity online page migration problem (MCOPMP) and the multi-commodity online facility location problem (MCOFLP), the extension does not offer an algorithm any *benefit* from managing the commodities together. Instead, managing heterogeneous resources at once becomes a *necessity* as requests leave a choice of which resource to supply.

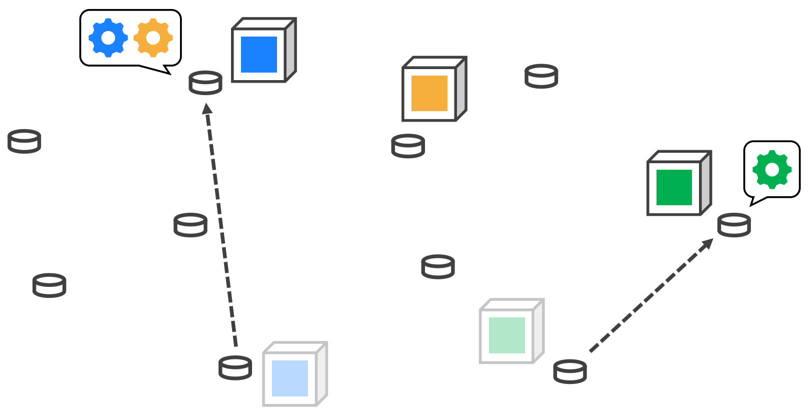

Figure 4.1: In the *multi-commodity k-server problem*, each resource (called a server) offers a subset of the commodities (colors). Here, we depicted a special case where each server (box) has one unique commodity. Each request (speech bubble) presents a set of commodities that can all serve it. An algorithm must place a server offering one of the desired commodities at a request's location and pays for the movement of servers (a movement is a dashed arrow pointing at the destination).

We proceed to formalize the model in Section 4.1 below. After that, we present related work in Section 4.2 and our results in Section 4.3. We show that already in simple cases, the problem gets significantly more complex than the k-server problem. In Section 4.4, we show that the any-or-one case, where each request either can be served by any or one specific server, on uniform metrics requires non-trivial algorithms. More specifically, simple adaptations of known algorithms have an unbounded competitive ratio, and the general lower bound for deterministic algorithms raises from k to $2k-1$. Motivated thereby, we show that the lower bound can be framed dependent on how many requests for specific servers appear. As a result, we have a complete picture from instances of the classical k-server problem (zero specific requests) to trivial instances where all requests leave no choice on which server must move. Perhaps surprisingly, there is no algorithm performing best for all instances. We classify an important behavioral rule and show its influence on the competitive ratio dependent on the input sequence. Consequently, in Section 4.5, we present several deterministic algorithms complementing each other for the uniform any-or-one case. We analyze their competitive ratio in the same parameterization as the lower bound and relate their actions to the previously identified behavior. Finally, we present a first algorithm with an asymptotically optimal competitive ratio for $k = 2$ servers on line metrics in Section 4.6.

Chapter Basis. The model, the algorithms, and the analyses build on the following publication:

The k-Server with Preferences Problem
Jannik Castenow, Björn Feldkord, Till Knollmann, Manuel Malatyali, and Friedhelm Meyer auf der Heide
In: Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2022), July 2022, Pages 345–356
Cf. [32]

Compared to the publication, we sharpen our upper and lower bounds. The lower bound is now parameterized in the ratio between specific requests and the total number of requests, smoothly going through all previously known lower bound instances. Further, we present a tight analysis for all our algorithms when specific requests dominate the input and generalize the lower bounds for defensive algorithms. We formulate an algorithm that generalizes both algorithms presented in the paper and allows a fine-tuned competitive ratio. Additionally, we present a result for $k = 2$ servers on line metrics.

4.1 Problem Definition & Model

Next, we define the multi-commodity k-server problem as a resource allocation problem following Definition 1.1. Recapitulate that an algorithm must serve every request while it aims to minimize the total cost.

We consider any metric space (M, d) with a set of commodities S. The set of resources consists of k servers. Each offers a subset of the commodities S. Any request $r \in R$ consists of a location and a set of commodities $s_r \subseteq S$ which can satisfy it. We abuse the notation here and refer by r also to the location of a request $r \in R$. A request is served when a server offering one commodity of s_r is located at the request's position. The serving itself has a cost of zero. The actions of an algorithm are the movement of each server. Moving any server from location $i \in M$ to location $j \in M$ incurs a movement cost of $d(i, j)$. At any point in time, we call the placements of servers at locations the *configuration* and all locations where a server resides *covered*.

Comparison to classical k-server. Our model generalizes the classical k-server problem, which can be instantiated by allowing only one commodity, i.e., $|S| = 1$. Compared to the classical case, requests in our model have more flexibility in their needed resources. We can model situations where not all servers are identical but offer different capabilities. The additional difficulty on the algorithmic side is not only in determining the correct locations that should be occupied by servers but also in determining the *exact* position of each server based on its commodities.

The any-or-one case. We are especially interested in the any-or-one case in which a request can either be served by any server or a specifically requested one. More formal, for a request r either $s_r = S$ or $s_r = \{e\}$ for $e \in S$ and each server has exactly one unique commodity, i.e., there are k commodities. We call a request that any server can serve a *general request*. In contrast, a request explicitly asking for server $s \in S$ is a *specific request for s*. Note that the any-or-one case also reduces to the classical k-server problem if only general requests appear. We call such input sequences *pure general inputs*. The other extremum is when only specific requests appear. Such input sequences are called *pure specific inputs*, and they can be solved optimally in a trivial way because the input sequence leaves no freedom of choice for any algorithm. Input sequences where general and specific requests arrive are called *mixed inputs*.

Lazy and non-lazy algorithms. For completeness, we introduce a fundamental design aspect for algorithms for the MCOKSP called *lazy* in Definition 4.1. The definition is equivalent to the original definition for the k-server problem, for example, used in [60]. Note that without loss of generality, any non-lazy algorithm can be turned lazy by postponing movements until needed due to an arriving request. For this, it is necessary to remember the postponed movements to reflect the behavior of the non-lazy algorithm. Laziness is also discussed further in [60].

Definition 4.1 — Laziness. An algorithm for the multi-commodity online k-server problem is called *lazy* if, in each time step, it only moves one server to the location of the current request if necessary.

4.2 Related Work

The *k-server problem* was introduced in 1988 by Manasse et al. [70]. On the one hand, it is a special case of so-called *metrical task systems* [22]. On the other hand, it naturally generalizes fundamental online problems such as the paging problem or the k-headed disk problem. For example, the paging problem is equivalent to the k-server problem on uniform metrics (where all distances are the same). As a result, the problem has gained much interest since its appearance. Next, we outline relevant related work to overview the problem. For a detailed survey on the work on the k-server problem, we refer to the work of Koutsoupias [60].

Offline k-server. In its offline version, the k server problem can be solved optimally using a dynamic programming approach. Another approach is reducing the problem to finding a minimum cost flow of maximum value. In [34], Chrobak et al. present such a reduction and show that the runtime is $\mathcal{O}(kn^2)$ (see Theorem 5.1 in [34]).

The k-server problem as a metrical task system. A metrical task system [22] is given by a metric space (S,d) with a set of states S (not to be confused with our commodity set) and a distance function $d : S \times S \to \mathbb{R}$. An algorithm for a metrical task system is always in a state of S and can switch to another state in S with a cost dictated by d. The task of an algorithm is to serve a sequence T of tasks that define a serving cost for each state of S. The k-server problem (on finite metrics) can be viewed as a metrical task system: S consists of all possible configurations the servers can express, and $d(x,y)$ is the cheapest way of moving servers in configuration x to achieve configuration y. A configuration here is the set of locations that all servers cover. The request sequence generates the sequence of arriving tasks as follows. Any request corresponds to a task with zero cost for all states that express a configuration that covers the request's locations and a cost of infinity for all other states. This way, when a request arrives, the algorithm must move to a configuration where the request is served. For the special case of $k+1$ locations, the k-server problem is even equivalent to a metrical task system of $k+1$ states [70]. For metrical task systems, the deterministic competitive ratio is bounded by exactly $2n-1$, where n is the number of states [22]. The algorithm that achieves the bound of $2n-1$ is the *work function algorithm* which is currently also the most general algorithm for the k-server problem. Regarding randomized algorithms, the competitive ratio can be improved. The lower bound for randomized algorithms against metrical task systems is due to [10] $\Omega(\log n/\log^2 \log n)$. Bartal et al. [9] presented a randomized online algorithm with a polylogarithmic competitive ratio of $\mathcal{O}(\log^6 n)$. Recent improvements used randomized embeddings of metric spaces with hierarchically separated trees [8] to achieve a competitive ratio of $\mathcal{O}(\log^2 n)$ [25]. Compared to metrical task systems, the k-server problem is of special interest since its competitive ratio does not depend on the number of states but on the number of servers.

Online k-server. The first results on the k-server problem were by Manasse et al. [70, 71]. After introducing the problem, they presented a lower bound of at least k for every deterministic online algorithm if the metric space has at least $k+1$ locations. For the special case of $k=2$, the competitive ratio is exactly $2=k$, and for metrics with $k+1$ many locations, there is an algorithm with an optimal competitive ratio. In [70], the authors stated the famous k-server conjecture captured by Conjecture 4.1. Until today the conjecture is still open. Currently, algorithms with a competitive ratio of k are known for several special cases. For the special case of paging (k-server on uniform metrics), the conjecture was shown to hold already before the work of Manasse et al. by Sleator and Tarjan [87].

Conjecture 4.1 — k-server Conjecture [70]. For any $k \geq 1$, there is a deterministic algorithm for any symmetric (the distances between locations are symmetric) k-server problem with a competitive ratio of k.

Notable results for other metrics are the *double coverage* algorithm and the *work function* algorithm. The double coverage algorithm achieves an optimal competitive ratio on the real line and tree metrics [34, 35]. It exploits that on such metrics, either any request is in between two (closest) servers or all servers are to one side of it. In the former case, the algorithm moves the two closest servers at equal speed toward the request, while in the latter case, the closest server is moved. Note that the algorithm is an example of a non-lazy algorithm with respect to Definition 4.1 since it may move two servers upon the arrival of one request.

The work function algorithm was already established for metrical task systems [22] and was shown by Koutsoupias and Papadimitriou [61] to achieve a competitive ratio of $2k-1$ on general metrics for the k-server problem. The algorithm aims at choosing a configuration for each request that minimizes the work needed to answer the request sequence seen so far and the cost to transition from the current configuration of the algorithm. Besides being the best-known general algorithm for the k-server problem so far, the work function algorithm achieves an optimal competitive ratio on line and star metrics as well [12]. While the algorithm seems to need much computational power in each step, Rudec et al. [82] demonstrated that it indeed could be implemented efficiently to be used in real-world scenarios.

Much literature considered randomized solutions for the k-server problem. Fiat et al. [43] were among the first to present an algorithm for uniform metrics with a competitive ratio of $\mathcal{O}(\log k)$, which is called the marking algorithm. The current best-performing randomized algorithm for general metrics achieves a competitive ratio of $\mathcal{O}(\log^2 k \log n)$ [26] using that any metric can be embedded into a hierarchically separated tree with a loss of $\mathcal{O}(\log n)$ in the competitive ratio [8]. Here, n is the number of locations. The same result can be achieved with a different approach presented in [28]. Building upon these techniques, [65] achieves a competitive ratio independent of n and polylogarithmic in k. Very recent work [24] indicates improvements to the known lower bounds. For example, the paper presents a lower bound of $\Omega(\log^2 k)$ for a metric with $k+1$ locations negating the hope for a $\mathcal{O}(\log k)$ competitive algorithm for metrics with at least $k+1$ locations.

Extended models. The difficulties in proving the k-server conjecture in general metrics lead to model extensions that consider the influences on the competitive ratio. One way to approach the conjecture is using *resource augmentation* as in the (h,k)-server problem, where the online algorithm has k and the offline algorithm $h \leq k$ servers. Also considered the *weak adversary model*, the problem was first studied for uniform metrics in [87]. The exact competitive ratio for deterministic algorithms is fixed to $\frac{k}{k-h+1}$, i.e., the more servers the online algorithm has, the better the competitive ratio. The same bounds hold for star metrics [92]. While the bound approaches 1 as $k/h \to \infty$, recently, Bansal et al. showed that, on tree metrics, the competitive ratio for $k/h \to \infty$ is larger than 2.4 [7]. Interestingly, for some algorithms, such as the double coverage, using more than h servers even increases the competitive ratio [6, 7]. Bienkowski et al. [17] showed that for general metrics, even if $k \to \infty$, the competitive ratio depends on h.

Further extensions of the k-server problem are, for example, allowing to reject requests [19], allowing to delay the service of requests [4], and introducing more complex request types as in the k-taxi problem [37, 44]. In the latter, a request consists of two locations and must be served by a server visiting the two locations in order.

***k*-server on uniform metrics.** On uniform metrics, the k-server problem is equivalent to the *paging problem* where a set of n pages is managed in a cache of size k. Here, when a page outside the cache is requested, an algorithm must load the respective page to the cache (for a uniform cost) and potentially evict some other page. The k-server problem models paging by interpreting any location as a page and all locations where a server resides as being in the cache. Loading a page into the cache is equivalent to moving a server onto the location.

Regarding deterministic algorithms, the lower bound of k mentioned earlier directly applies. An optimal offline algorithm for the paging problem is the *farthest in the future* algorithm presented by Belady [14]. The algorithm always evicts the page in the cache that will be requested latest. For online algorithms, a class of algorithms called *marking algorithms* achieve an optimal competitive ratio. The general idea is to divide the request sequence into *phases*. In one phase, the online algorithm moves each server at most once. When a server is used to answer a request, it gets *marked* and will not move for the rest of the phase. When all servers are marked upon the arrival of a request at an uncovered location, the algorithm starts a new phase by declaring all servers unmarked. Intuitively, right at the first request after each phase, the optimal solution requires at least one movement, while a marking algorithm moves at most k times during each phase. The main design choice of an algorithm is in which order the unmarked servers are selected. Two well-known selection strategies are *least recently used* (LRU) and *first in, first out* (FIFO). Sleator and Tarjan [87] showed an optimal competitive ratio for LRU and FIFO. A different proof technique for the competitive ratio of LRU, FIFO, and *flush, when full* is given by Karlin et al. [57]. For an extended explanation of the marking approach, we refer to the book by Kleinberg and Tardos [58].

For randomized paging, the research of Fiat et al. [43] showed that the competitive ratio is between H_k and $2H_k$, where $H_k \in \Theta(\log k)$ is the k-th harmonic number. The competitive ratio became tight with the introduction of a H_k-competitive algorithm by McGeoch and Sleator [75]. Since then, several extended models have been researched. For an overview, we refer again to the survey by Koutsoupias [60]. Going into the direction of heterogeneous paging, as pointed out by Chrobak et al. [33], the *restricted caching* model [23] considers a special case of the any-or-one case on uniform metrics where each page has a fixed subset of the servers that can serve it. Here, the least recently used policy is no longer competitive [23], already indicating additional hardness introduced by heterogeneity. Using randomization, a competitive ratio of $\mathcal{O}(\log^2 k)$ can be achieved [27].

Multi-commodity k-server. No heterogeneity is considered in the classical formulations of the k-server problem. To the best of our knowledge, the first work explicitly considering servers with different capabilities is the master's thesis by Patel [80]. Here, a variant of the MCOKSP with two kinds of requests/servers is studied. General requests can be answered by any server, and special requests only by any server of a fixed subset of the servers. For this special case, Patel bounded the competitive ratio for line and uniform metrics in $\mathcal{O}(k)$.

Previous results that come closest to our model are in the work of Haney [52]. Haney considered the any-or-one case which he called *all-or-one k-server* on uniform metrics. In [52], he presented a lower bound of $2k-1$, a deterministic algorithm with a competitive ratio of $3k$, and a randomized algorithm with a competitive ratio of $\mathcal{O}(\log k)$. The lower bound and the deterministic algorithm are similar in spirit to our worst-case lower bound and our first algorithm CONF-MCOKSP. Our lower bound uses the same requests in a different order, and our algorithm can be seen as a lazy version of the algorithm presented by Haney. Regarding these results, we present improvements by parameterized bounds dependent on the frequency of specific requests for both the lower and the upper bound, a better upper bound of $3k-2$ in the worst case, and an optimal competitive ratio when specific requests dominate the input.

In a very recent dissertation, Liaee [67] considered the MCOKSP and showed that, in general, the problem has a lower bound of at least $\Omega(2^k/\sqrt{k})$ for deterministic algorithms. Therefore, more structured instances are researched. It is shown that if the allowed sets, each request can pose, form a laminar family, the achievable competitive ratio for deterministic algorithms is $\mathcal{O}(h^2 k)$, where $h \leq k$ is the height of the laminar family. Further, the author presents an algorithm for the any-or-one case on star metrics with a competitive ratio of $\mathcal{O}(k)$.

Shortly after our publication [32], Chrobak et al. [33] researched further in the same direction on uniform and star metrics. The authors present the results of [67]. In addition, they present a lower bound of $\Omega(k)$ for randomized algorithms in the general case. They again consider the laminar case and show that the achievable competitive ratio for randomized algorithms is $\mathcal{O}(h^2 \log k)$. For the any-or-one case, the authors present the worst-case lower bound of $2k-1$ and an algorithm with a competitive ratio of $3k-2$. The latter is similar to the idea of Haney [52] and our algorithm CONF-MCOKSP. Further, the paper offers proof that the any-or-one case of the MCOKSP is NP-complete in the offline case, underlining how this very limited case already fundamentally differs from the classic k-server problem.

As pointed out by Chrobak et al. (see Section 3 of [33]), the MCOKSP is a special case of the *generalized k-server* problem introduced in [62], where each server lies in its own metric space. An arriving request presents a location in each metric space and is served as soon as, in any metric space, the corresponding server moves to the requested location. While there are results for uniform metrics for the generalized k-server problem [5], the competitive ratios are $k2^k$ (deterministic) and $\mathcal{O}(k^3 \log k)$ (randomized), and thus, relatively large.

4.3 Our Results

Next, we sketch our results for the MCOKSP. All our results consider the any-or-one case where each request can be served by either any server or a specific one (see Section 4.1). Additionally, we focus on deterministic online algorithms.

Trivial solutions may not be competitive. Recapitulate that in the MCOPMP and the MCOFLP, a trivial solution was to treat each commodity separately as an instance of the single-commodity case. For both problems, the competitive ratio of such an algorithm is increased by a factor of at most $|S|$. In the MCOKSP, however, the situation is different. Even for the simple uniform metric and only a few servers, trivial approaches yield an unbounded competitive ratio in the any-or-one case. We give an example of a naive approach for which we show an unbounded competitive ratio for $k=3$ in Section 4.4.1. Intuitively, in the MCOPMP and the MCOFLP, an algorithm can improve its cost by at most a factor of $1/|S|$ when moving commodities together or instantiating them together, because any input sequence can be separated into sequences for each commodity. In MCOKSP, the input cannot be separated so clearly as the serving criteria of a request is rather soft, i.e., the algorithm can move any demanded commodity to the current request. A wrong movement can yield further wrong decisions increasing the competitive ratio drastically. Adapting the model of the MCOPMP to allow requests that any commodity of a given subset can serve yields similar effects in that model.

Lower bounding the competitive ratio. Our first result is a lower bound. The lower bound for the k-server problem, and thus, for the MCOKSP on pure general inputs, is k [71]. What happens if we induce specific requests into the lower bound sequence? While every specific request is trivial to serve – the request itself tells the algorithm which server to move – the lower bound for mixed inputs turns out to be significantly higher than k. As we can see in Theorem 4.1, the worst-case lower bound is $2k-1$, i.e., by an additive $k-1$ higher than the lower bound for pure general inputs. The bound holds already for uniform metrics on $k+1$ locations. Essentially, the sequence is designed such that the online algorithm accumulates a cost of k until it covers the correct locations with its servers that are also covered by the optimal solution. Up to this point, the sequence is the same as the classic lower bound for the k-server problem. After that, specific requests force the online algorithm to sort out precisely which server has to be located where incurring an additional cost of $k-1$.

Theorem 4.1 — Lower Bound. Consider the any-or-one case of the multi-commodity online k-server problem. Every deterministic online algorithm has a competitive ratio of at least $2k-1$, even in a uniform metric space with $k+1$ locations.

We remark that a similar lower bound was presented by Haney in [52]. The lower bound sequence uses the same requests in a different order than ours. Our sequence starts with general requests, followed by specific ones. The sequence of Haney alternates general and specific requests. Both structures have their strengths allowing us to derive more involved bounds.

Parameterizing the lower bounds. We adapt and extend the structure of Haney's lower bound sequence to derive a more fine-grained bound. We observe that the lower bound is still k for pure general inputs and not $2k-1$. Also, for pure specific inputs, the lower bound is only 1 because such inputs can easily be solved optimally. The bound of Theorem 4.1 uses a mixed input. To better understand the situation, we define a measure for the input sequence parameterizing to what extent specific requests happen. For this, consider the ratio $s = \frac{f}{f+g}$, where f is the number of specific requests requiring a movement of the online algorithm and g is the number of general requests requiring a movement of the online algorithm. s expresses the ratio of the specific requests versus the total number of requests. If $s = 0$, the input is a pure general one; if $s = 1$, the input is a pure specific one; and if $0 < s < 1$, the input is a mixed one. We define s respecting the *required movements* as otherwise, it could be arbitrarily manipulated by inserting general or specific requests that do not require a movement and do not influence the competitive ratio.

R By definition, s depends on the considered online algorithm. For upper bounds, the definition is clear. For lower bounds, we assume lazy algorithms. The lazy property allows determining requests that require a movement. The assumption does not weaken the bounds as every algorithm can be turned lazy without increasing the competitive ratio [60].

The beauty of s as a measure is that it allows us to show how the performance of online algorithms gradually depends on the number of induced specific requests. Speaking of the lower bound, we can give an adaptive lower bound stated in Theorem 4.2 that shows how the problem gets more and more difficult in the online case until a peak is reached at the lower bound given by Theorem 4.1.

Theorem 4.2 — Adaptive Lower Bound. Consider any deterministic online algorithm for the any-or-one case of the multi-commodity online k-server problem. Let s be the ratio between the number of specific requests and the total number of requests that require a movement by the algorithm. Already in a uniform metric space with $k+1$ locations it holds: The algorithm has a competitive ratio of at least $(1+\frac{s}{1-s})k$ if $s \leq \frac{k-1}{2k-1}$, a competitive ratio of at least $2k-1$ if $\frac{k-1}{2k-1} < s < \frac{k}{2k-1}$, and a competitive ratio of at least $\frac{1}{2s-1}$ if $s \geq \frac{k}{2k-1}$.

For a graphical representation of the lower bound of Theorem 4.2, see Figure 4.2. One can see here that the adaptive lower bound starts at $s = 0$ with the classical lower bound of k and gradually increases up to the worst-case lower bound for $\frac{k-1}{2k-1} \leq s \leq \frac{k}{2k-1} \approx \frac{1}{2}$. Thereafter, it decreases *independently of* k until it is constant for $s = 1$. The lower bound already suggests that the adversary's power is severely limited as soon as specific requests dominate the input sequence. Intuitively, roughly up to $s = \frac{1}{2}$, the adversary can induce specific requests to make an online algorithm pay for an earlier movement due to a general request. However, it can only do this once for every general request. Both our lower bounds are proven in Section 4.4.2.

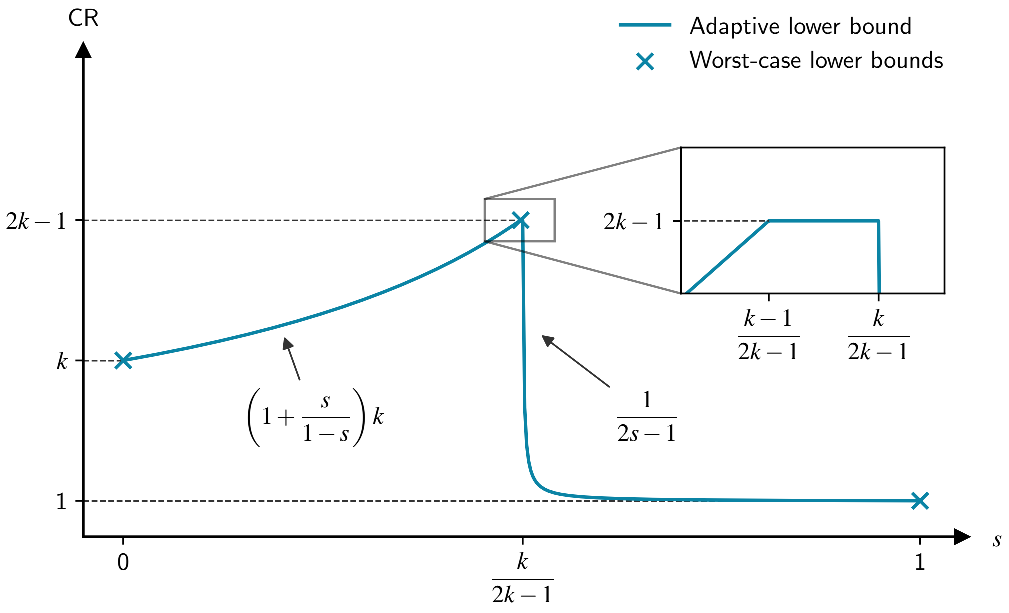

Figure 4.2: The figure plots the adaptive lower bound of Theorem 4.2 and the worst-case lower bounds against the ratio s of the number specific requests and the total number of requests. The worst-case bounds are for the k-server problem (k at $s = 0$), the MCOKSP ($2k-1$ at $s = \frac{k}{2k-1}$), and the special case where all requests are specific (1 at $s = 1$). The adaptive bound is defined for every possible $s \in [0,1]$ and goes through the worst-case points.

A deterministic algorithm for uniform metrics. On the algorithmic side, we present in Section 4.5.2 our algorithm CONF-MCOKSP. The algorithm is based on marking approaches explained in Section 4.2, i.e., it operates in phases during which the optimal solution has a cost of at least one. A phase ends, and another begins as soon as the algorithm detects that the optimal solution must have made a move. During a phase, the algorithm tracks the locations at which only general requests appear and the specifically requested servers. As soon as too many servers are required to cover the locations of general requests with servers that were not specifically requested, the optimal solution must have moved, and the phase ends. The main insight for specifically requested servers is that they should stick to the position they were specifically requested for as long as possible. Whenever a specific request appears, the online algorithm learns the current position of the requested server in the optimal solution, as the latter has to serve the request, too. Of course, at some point, the algorithm has to move specifically requested servers again. These servers are released when the current phase ends because the optimal solution could have moved any of them at that point. Considering the competitive ratio of CONF-MCOKSP, we can show an adaptive upper bound dependent on s again. The bound of Theorem 4.3 is depicted in Figure 4.3.

Theorem 4.3 Consider the any-or-one case of the multi-commodity online k-server problem on uniform metrics. Let s be the ratio between the number of specific requests and the total number of requests that require a movement by the algorithm. The competitive ratio of CONF-MCOKSP is at most $\min\{k+\frac{2s}{1-2s}k,\, 3k-2,\, 1+2\frac{1-s}{s}k\}$ for $s<\frac{k}{2k-1}$ and at most $\frac{1}{2s-1}$ for $s\geq\frac{k}{2k-1}$.

Unfortunately, the competitive ratio in the worst case is $3k-2$, and thus, by $k-1$ higher than the worst-case lower bound. Theorem 4.4 shows that this is not an artifact of the analysis but a result of the algorithm's decisions.

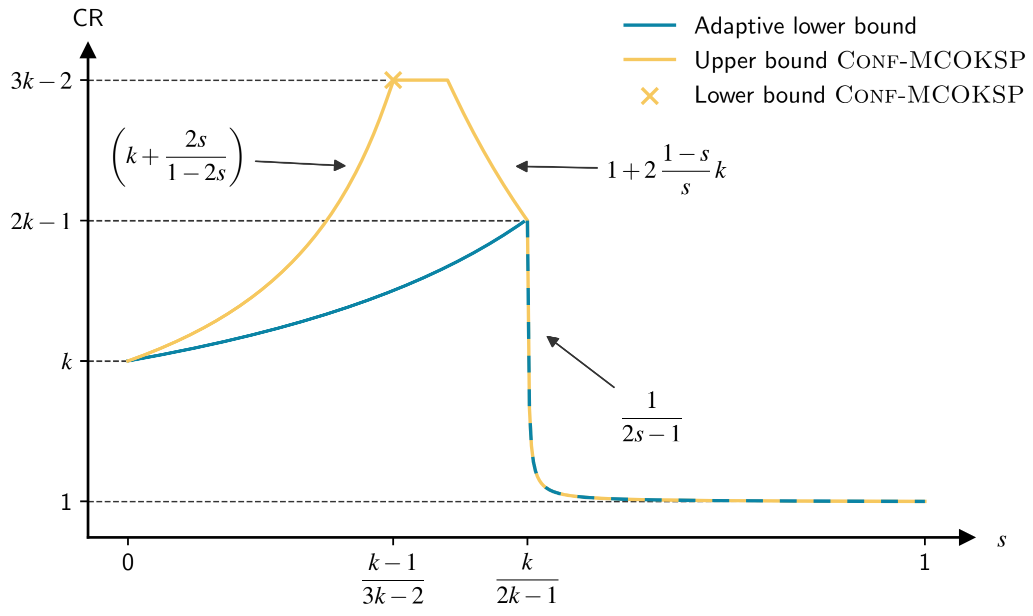

Figure 4.3: The plot compares the upper and lower bounds for CONF-MCOKSP (Theorems 4.3 and 4.4) and the adaptive lower bound (Theorem 4.2) plotted against s, the ratio between the number of specific requests and the total number of requests. The competitive ratio of the algorithm roughly follows the lower bound and is optimal for $s=0$ and $s\geq\frac{k}{2k-1}$. However, for $s=\frac{k-1}{3k-2}$, the algorithm has a competitive ratio of at least $3k-2$.

Theorem 4.4 Consider the any-or-one case of the multi-commodity online k-server problem on uniform metrics. The worst-case competitive ratio of CONF-MCOKSP is at least $3k-2$.

Figure 4.3 shows the competitive ratio of CONF-MCOKSP against the lower bound introduced before. One can see that the bound is tight both for $s=0$ and $s\geq\frac{k}{2k-1}$. Thus, the algorithm achieves an optimal competitive ratio in the worst-case lower bound of Theorem 4.1. For $s\geq\frac{k}{2k-1}$, the competitive ratio is tight, confirming that the power of the adversary is most relevant for $s<\frac{k}{2k-1}$. For $s=\frac{k-1}{3k-2}\approx{}^1/_3$, unfortunately, the algorithm's competitive ratio overshoots the worst-case lower bound. Theorem 4.4 shows that the reason for this is an issue with the algorithm itself.

An important behavioral rule. Why does CONF-MCOKSP miss the worst-case lower bound by $k-1$? It turns out that there is one crucial behavioral rule that enables the bound of Theorem 4.4.

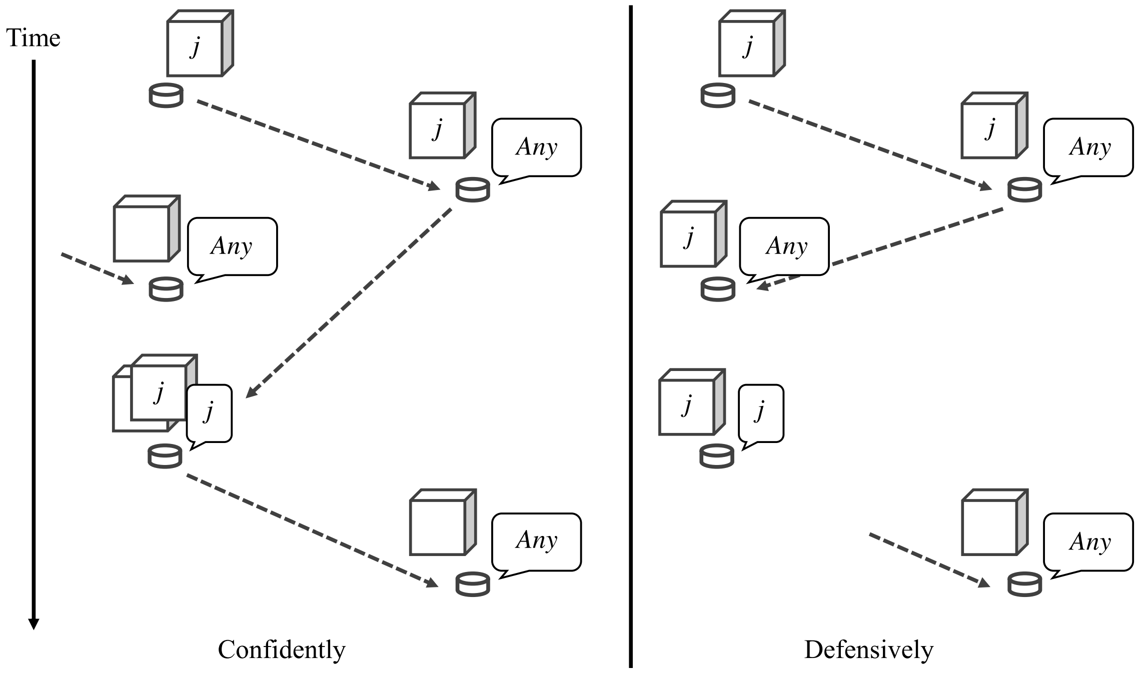

Figure 4.4: Assume that initially, j is on $p^*(j)$ (the last known optimal position of j). Then j can either act confidently or defensively. *Left:* When the algorithm acts confidently for j, it fills the gap at $p^*(j)$ with another server on the second request, as it assumes that moving j away earlier was correct. *Right:* When the algorithm acts defensively for j, it moves j back to $p^*(j)$ on the second request, as it assumes, moving away earlier was wrong.

Consider Figure 4.4 for the following. For any server j of the online algorithm, let $p^*(j)$ be the last known optimal location of j. $p^*(j)$ is known initially (by assumption, the algorithm's servers are initially at the same locations as in the optimal solution) and updated by specific requests for j. Assume j is at $p^*(j)$. If j is moved away by the algorithm, and another general request on $p^*(j)$ appears, either the algorithm could move another server to $p^*(j)$, or it could move j back to $p^*(j)$. If the algorithm decides to move another server, it is *confident* that j is not required at $p^*(j)$ anymore. In the case that the algorithm moves j back to $p^*(j)$, it *defensively* assumes that moving j earlier was the wrong decision. Note that the formerly introduced decision is a behavior that can be determined for each server independently. We formally grasp the behavioral rule behind by Definition 4.2 also presented in Section 4.4.3.

Definition 4.2 — Defensive/Confident. An algorithm acts defensively for j if, when a general request appears on $p^*(j)$ that requires a movement, it moves j to the request. A deterministic algorithm is *ℓ-defensive*, if for a fixed subset of ℓ servers, it *always* acts defensively. An algorithm is *ℓ-confident*, if, for ℓ servers, it does not always act defensively. An algorithm is *strictly-ℓ-confident*, if for a fixed subset of ℓ servers, it *never* acts defensively if it can be avoided.

A trade-off in the competitive ratio. Consider the scenario from above (pictured in Figure 4.4) again. If the algorithm acted confidently for j and a specific request at $p^*(j)$ appears, the algorithm has to move j back to $p^*(j)$ while the optimal solution could have kept j there the whole time. Further, the algorithm can be forced to move one additional server to the location of the first general request. The total cost of the algorithm, in this case, is 4, while the optimal solution has only a cost of 1 to move any server other than j to the first request. If the optimal solution already covers the first location, it does not even have a cost. Intuitively, the lower bound on CONF-MCOKSP of Theorem 4.4 uses the setting above for $k-1$ servers and can map a cost of 3 to j each time. With an additional movement for the remaining server, we arrive at a total cost of $3k-2$. The input sequence can enforce such a high cost because CONF-MCOKSP does not act defensively. In fact, for the sequence of Theorem 4.4, CONF-MCOKSP acts strictly confident for all servers. Based on this observation and our sequence for Theorem 4.1, we can derive a lower bound for online algorithms that explicitly do not act defensively for at least ℓ servers in Theorem 4.5.

Theorem 4.5 Consider the any-or-one case of the multi-commodity online k-server problem. No strictly-ℓ-confident algorithm (for $\ell \geq 1$) can achieve a competitive ratio better than $2k+\ell-2$.

As one can see, each server for which an algorithm acts confidently inherently moves the competitive ratio one step away from the worst-case lower bound. Considering the scenario above again, if an online algorithm acts defensively for j, its cost is 3. So, it seems like a dominating strategy to act defensively. Unfortunately, we show in Theorem 4.6 that each server for which an algorithm acts defensively increases the lowest achievable competitive ratio for $s \leq \frac{k}{2k-1}$ by one. Especially, the competitive ratio for pure general inputs is increased.

Theorem 4.6 Consider any ℓ-defensive online algorithm for the any-or-one case of the multi-commodity online k-server problem (for $\ell \geq 1$). Let s be the ratio between the number of specific requests and the total number of requests that require a movement by the algorithm. For $s \in (0, \frac{1}{k+\ell-1})$, the algorithm has a competitive ratio of at least $k+\ell-2$. For $s = 0$ and $\frac{1}{k+\ell-1} \leq s \leq \frac{k}{2k-1}$, the algorithm has a competitive ratio of at least $k+\ell-1$.

R The lower bound is by 1 smaller for the very few cases where $s \in (0, \frac{1}{k+\ell-1})$ due to a technical subtlety in the analysis. Noteworthy, for $\ell = k$, one can show a bound of $k+\ell-1$ for any $s \leq \frac{k}{2k-1}$.

So, there is a trade-off between acting defensively or confidently. Acting defensively allows approaching the worst-case lower bound, while when specific requests are rare, it improves the performance to act confidently. Our bounds leave space for algorithms that not always or never act defensively, but *sometimes* do. Naturally, bounds on the competitive ratios of all such algorithms are hard to grasp as they can act in uncountable ways. We note here that CONF-MCOKSP, while not being *strictly* k-confident, still suffers from the same lower bound of Theorem 4.5. We believe that even for algorithms that act confidently or defensively flexibly, a mixture of Theorem 4.5 and Theorem 4.6 implies the same trade-off. To support this claim, we show Theorem 4.7.

Theorem 4.7 Consider the any-or-one case of the multi-commodity online k-server problem. No deterministic algorithm can achieve a competitive ratio of k on pure general inputs and $2k-1$ on mixed inputs.

No algorithm can achieve a competitive ratio of k on pure general inputs while simultaneously achieving a competitive ratio of $2k-1$ in the worst case. An algorithm has to act confidently to have a competitive ratio of k on pure general inputs. If it does so, it has already missed the opportunity to act defensively and has increased its competitive ratio in the worst case. The bound of Theorem 4.7 is possible due to the structure of our lower bound sequence for Theorem 4.1 in which the specific requests are revealed *after* the online algorithm already built up a competitive ratio of at least k by general requests. In Section 4.4.3, we present the proofs for the results on defensive and confident algorithms and discuss them in more detail.

Algorithms for the trade-off on uniform metrics. The previously discussed trade-off motivates the design of (partly) defensive algorithms. Note that we already have a k-confident algorithm CONF-MCOKSP. In Section 4.5.3, we present a k-defensive algorithm called DEF-MCOKSP. It can be seen as an extension of CONF-MCOKSP that acts defensively for every server whenever possible. We frame the algorithm as non-lazy first to improve the readability of the analysis for the worst-case competitive ratio. As a result, the performance for large values of $s \to 1$ cannot undercut the $2k-1$ bound. When considering the lazy version of DEF-MCOKSP, the bound for $s \geq \frac{k}{2k-1}$ equals the lower bound. Our results for this algorithm are combined in Theorem 4.8.

Theorem 4.8 Consider the any-or-one case of the multi-commodity online k-server problem on uniform metrics. Let s be the ratio between the number of specific requests and the total number of requests that require a movement by the algorithm. The competitive ratio of DEF-MCOKSP is at most $2k+14$. For $s \geq \frac{k}{2k-1}$, the *lazy* version of DEF-MCOKSP has a competitive ratio of at most $\frac{1}{2s-1}$.

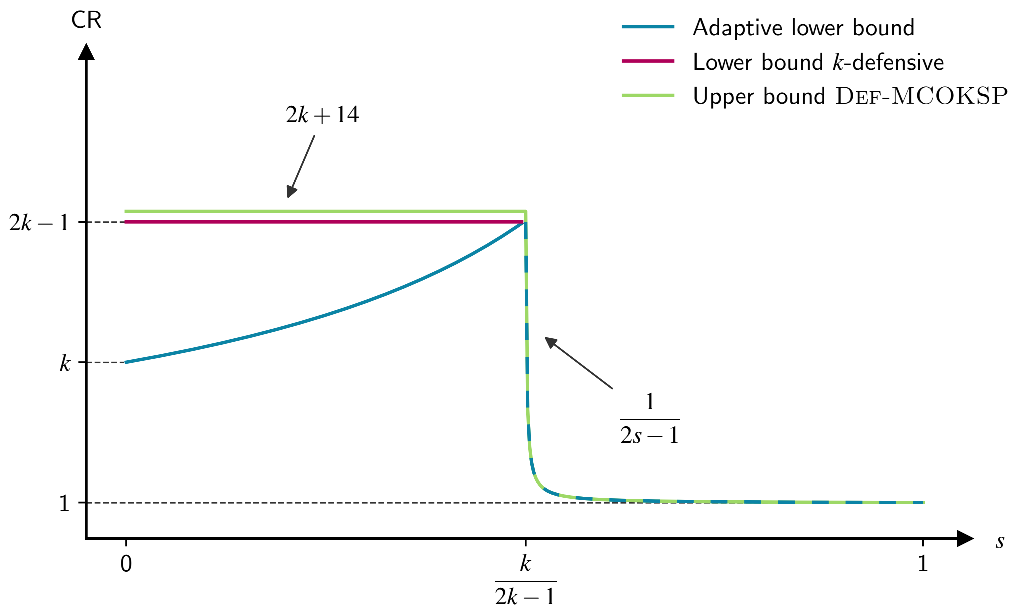

Figure 4.5: The plot compares the upper bound of the lazy version of DEF-MCOKSP (Theorem 4.8), the lower bound for k-defensive algorithms (Theorem 4.12) (that applies to DEF-MCOKSP), and the general adaptive lower bound (Theorem 4.2). All bounds are plotted against s, the ratio between the number of specific requests and the total number of requests. Note how the algorithm achieves a close-to-optimal competitive ratio regarding the lower bound for k-defensive algorithms. In other words, DEF-MCOKSP nearly hits the limits of algorithms of its kind. For $s \geq \frac{k}{2k-1}$, all bounds are the same.

Consider Figure 4.5 for a depiction of the competitive ratio of DEF-MCOKSP. We can see that the competitive ratio is tight for $s \geq \frac{k}{2k-1}$, confirming once more that the adversarial power lies in the area of $s < \frac{k}{2k-1}$. For such s, the competitive ratio is $2k+14$ which is nearly tight respecting the lower bound for k-defensive algorithms of $2k-1$. Technically, analyzing DEF-MCOKSP requires far more complex arguments than CONF-MCOKSP, as it no longer suffices to analyze only the cost of the algorithm for one phase. One has to amortize the cost over multiple phases to get the competitive ratio towards $2k$. We strongly believe the additive constant of 15 compared to the lower bound is due to the analysis.

As mentioned, deciding whether to act defensively or confidently is a per-server decision. Thus, we can mix CONF-MCOKSP and DEF-MCOKSP, yielding MIXED-MCOKSP. It is parameterized by ℓ, the number of servers acting according to CONF-MCOKSP. Consequently, $k-\ell$ servers act as for DEF-MCOKSP. Applying a mix of both analyses, we arrive at Theorem 4.9, giving us a flexible algorithm that can be tailored at will. The algorithm is presented in Section 4.5.4. Also, note how MIXED-MCOKSP contains CONF-MCOKSP (set $\ell = k$) and DEF-MCOKSP (set $\ell = 0$).

Theorem 4.9 Consider the any-or-one case of the multi-commodity online k-server problem on uniform metrics. Let s be the ratio between the number of specific requests and the total number of requests that require a movement by the algorithm. MIXED-MCOKSP achieves a competitive ratio of at most $\left(\min\{\ell + \frac{2s}{1-2s}\ell, 3\ell - 2, 1 + 2\frac{1-s}{s}\ell\} + 2(k-\ell) + 14\right)$ for $s < \frac{k}{2k-1}$ and at most $\frac{1}{2s-1}$ for $s \geq \frac{k}{2k-1}$.

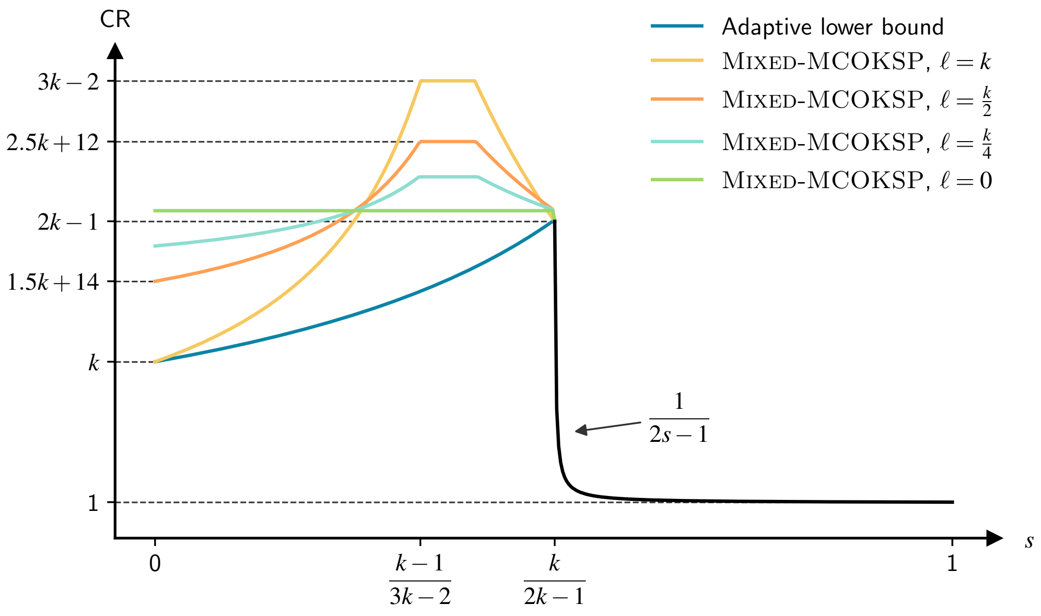

Figure 4.6: The figure plots the upper bounds of MIXED-MCOKSP for $\ell = 0$, $\ell = k/4$, $\ell = k/2$, $\ell = k$, and the adaptive lower bound against s, the ratio of the number of specific requests and the total number of requests. Note how the parameter ℓ of MIXED-MCOKSP demonstrates the trade-off in the competitive ratio. For $\ell = k$, the algorithm is CONF-MCOKSP. The more ℓ decreases, the more the competitive ratio in the worst case at $s = \frac{k-1}{3k-2}$ improves, and the competitive ratio for smaller s worsens. For $\ell = 0$, the algorithm is DEF-MCOKSP. For any choice of ℓ, the competitive ratio is tight for $s \geq \frac{k}{2k-1}$.

For a depiction of the competitive ratio of MIXED-MCOKSP, consider Figure 4.6. Here, we see some example values for the parameter ℓ. Overall, the parameter alternates the competitive ratio between the bound for CONF-MCOKSP and DEF-MCOKSP following the trade-off discussed before. Note how the lower bounds of Theorems 4.5 and 4.6 complement the upper bounds. For example, for $\ell = k/2$ (orange in Figure 4.6) the lower bound for $s = 0$ is $1.5k - 1$ while the upper bound is $1.5k + 14$, and in the worst case the lower bound is $2.5k - 2$ while the upper bound is $2.5k + 12$.

R Technically, Theorem 4.5 only applies to strictly-ℓ-confident algorithms and not to MIXED-MCOKSP (as it might by chance act defensively for the ℓ confidently acting servers). However, one can show the same bound for the algorithm similar as we proved the lower bound of $3k - 2$ for CONF-MCOKSP in Theorem 4.4.

Algorithms for other metrics. Our previous algorithmic results relied on a uniform metric. The lower bounds were also designed on a uniform metric, but naturally, they carry over against algorithms for the any-or-one case in all metrics. The main advantage of a uniform metric for the algorithm design is that correcting the location of one server always costs at most 1. In general metrics, this is no longer true, increasing the potential costs for correcting the position of a server. Consequently, designing algorithms with a close-to-optimal competitive ratio for non-uniform metrics is more difficult.

In Section 4.6, we elaborate on the additional difficulty when considering non-uniform metrics. After that, we show that an adaptation of the famous double coverage algorithm [34] achieves a competitive ratio of $6 = 3k$ for $k = 2$ servers on the real line as shown in Theorem 4.10. The main challenge for more than 2 servers is the following situation. Imagine a server is specifically requested. Then, the online algorithm knows potentially k locations where its servers should be before the specific request. After moving the specifically requested server away, the algorithm does not know which of the k locations can be left without a server. So, the main task is to cover approximately the k previous locations, using $k-1$ servers to avoid uncovering the wrong location. For $k = 2$ servers, the situation can be solved by moving the server not specifically requested slightly toward the other server's former location. For $k > 2$, it is unclear which server should step in, and the design of a competitive algorithm remains an open problem.

Theorem 4.10 Consider the any-or-one case of the multi-commodity online k-server problem on real line metrics. DC-MCOKSP achieves a competitive ratio of 6 for $k = 2$ servers.

4.4 Lower Bounds

The following section presents our lower bounds for the any-or-one case of the MCOKSP. All our lower bounds already hold in uniform metric spaces with $k+1$ locations. We start in Section 4.4.1 where we show an example for a simple algorithm that is not competitive. The example shows that non-trivial approaches are necessary even for a bounded competitive ratio. After that, we present a general worst-case lower bound in Section 4.4.2. The worst-case lower bound is further extended to a lower bound parameterized in s, the ratio of specific requests against all requests in the input sequence (of requests requiring a movement by the online algorithm). Regarding s, we determine a critical behavior – acting defensively or confidently – to design competitive online algorithms in Section 4.4.3. We show further lower bounds for classes of algorithms following the identified behavior. The results of the last section indicate a significant trade-off that influences the competitive ratio dependent on whether an algorithm acts confidently or defensively. Finally, we show in Section 4.4.3 that no algorithm can meet the previously introduced adaptive lower bound for all values of s simultaneously.

4.4.1 Simple Adaptations Are Not Competitive

Next, we show how a simple adaptation of the least recently used (LRU) policy fails in our model. More specifically, the presented algorithm already yields an unbounded competitive ratio on a metric with k locations. Therefore, we require more involved algorithms.

Example algorithm. Use the least recently used server for a general request. For a specific request, use the requested server. This server is then also counted as the most recently used one.

For pure general inputs, i.e., instances of the k-server problem, the LRU approach is optimal. Unfortunately, we can show the following for mixed inputs:

Theorem 4.11 Consider the any-or-one case of the multi-commodity online k-server problem. The example algorithm has an unbounded competitive ratio even for $k = 3$ servers on a uniform metric with k locations.

Proof. Consider the locations v_1, v_2, v_3, the algorithm's servers a_1, a_2, a_3 and the servers o_1, o_2, o_3 of the optimal solution. Let $p(s)$ be the location of server s. Let initially $p(a_i) = p(o_i) = v_i$ for all servers. We assume that the algorithm starts to move its a_i in the order of the indices. The optimal solution moves each server once such that $p(o_1) = v_2$, $p(o_2) = v_3$ and $p(o_3) = v_1$ for a cost of 3. Afterward, the optimal solution never moves its servers.

Start the sequence by requesting server 1 specifically at v_2. Now, the algorithm has its servers a_1, a_2 at v_2 and a_3 at v_3, while the order of movements for the servers is a_2, a_3, a_1. Next, we show how to construct a sequence of requests such that the algorithm has a cost and ends up in the same situation again. First, apply a general request on v_1 causing a_2 to move there. Then, request server 3 specifically at v_1, such that a_3 moves. Do a general request on v_3 to move a_1 there and afterward, a general request on v_2 such that a_2 moves there. Make a specific request for server 1 on v_2 such that the algorithm moves a_1 there. Do a general request on v_3 to move a_3. Finally, make a specific request for server 1 on v_2. Repeating this sequence infinitely long yields the theorem. ■

Theorem 4.11 already shows us multiple pitfalls in our extended model. It seems disadvantageous to cover a location with two servers if not needed due to specific requests. Also, the algorithm might make a mistake when moving a server if it is immediately specifically requested at its previous location. It seems to be a good idea to keep servers at the location where they were lastly specifically requested. Doing so would solve the situation of the proof above, as the algorithm's servers stick to the optimal positions. However, in larger metric spaces, we eventually have to move servers away from their last known optimal location.

4.4.2 General Lower Bounds

Regarding all deterministic online algorithms for the any-or-one case of the MCOKSP, we can show the lower bound of Theorem 4.1 on uniform metrics.

Theorem 4.1 — Lower Bound. Consider the any-or-one case of the multi-commodity online k-server problem. Every deterministic online algorithm has a competitive ratio of at least $2k-1$, even in a uniform metric space with $k+1$ locations.

Proof. Consider the uniform metric with locations $v_1, \dots, v_{k+1}$. Let $a_1, \dots, a_k$ be the servers of an online algorithm and $o_1, \dots, o_k$ be the servers of an optimal solution. Let $p(s)$ be the location of server s. Assume that initially, $p(a_i) = p(o_i) = v_i$ for all $1 \leq i \leq k$.

The request sequence starts with a general request on v_{k+1} and then repeatedly poses general requests at the currently unoccupied location. Since the online algorithm is deterministic, there is a permutation π of $1, \dots, k$ such that the algorithm moves its servers for the first time in the order given by π, i.e., the first time $a_{\pi(i)+1}$ moves, $a_{\pi(i)}$ has already moved once. Rename each i to $\pi(i)$. Now, the algorithm moves its servers in the order given by $1, \dots, k$, i.e., the first time a_{i+1} moves, a_i has already moved once. Next, we build the sequence as above but do not pose any request for v_k. The online algorithm will move a_k the last, while the optimal solution will only move o_k. Note how we can assume that the algorithm moves each server eventually because otherwise, a_k is never moved. Then, the competitive ratio is unbounded because the algorithm never converges towards the optimal locations without using a_k. Separate the sequence into phases: We say phase i starts when a_i is moved for the first time. During each phase i only the servers $a_1, \dots, a_i$ move and the only locations that can become unoccupied are $v_1, \dots, v_i$ and v_{k+1}. The last phase ends when the online algorithm occupies $v_1, \dots, v_{k-1}$ and v_{k+1}, i.e., the same locations as the optimal solution. Until then, by the phases' definition, the online algorithm must have moved each server at least once. For any server a_i, for $i \leq k-1$, at the end of the last phase, it holds that a_i was moved at least twice or a_i is not located at v_i.

Next, we issue a specific request for each of $a_1, \dots, a_{k-1}$ at their initial locations. With the exception of a_k, each server a_i that is not at v_i must be moved there. Hence, every server except a_k moved twice, and a_k moved once yielding a total cost of $2k-1$. The optimal solution answers the total sequence by moving o_k to v_{k+1} for a cost of 1. Now, we are in the same configuration as in the beginning by renaming the locations. ■

For a depiction of the lower bound sequence of Theorem 4.1, consider Figure 4.7. The bound demonstrates that an algorithm has to determine the optimal locations of the servers *and* the precise mapping of specific servers to locations. Therefore, the lower bound for mixed inputs is by $k-1$ higher than for pure general inputs.

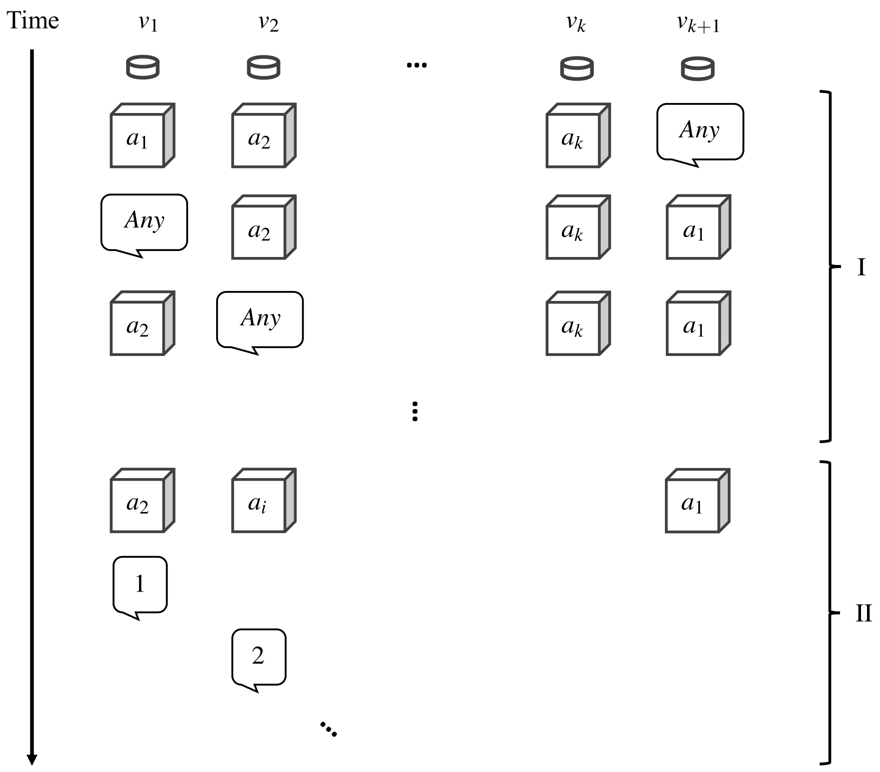

Figure 4.7: The lower bound sequence of Theorem 4.1 is divided into two phases. In phase I, always the currently unoccupied location (except for v_k) is requested. Eventually, the online algorithm covers all locations except v_k. Then phase II forces every server except for a_k back to its initial location using specific requests. The optimal solution solves the entire sequence by moving o_k to v_{k+1}.

The same lower bound was proven by Haney in [52] using the same set of requests but a different structure (see Section 2.4 in [52]). It works by forcing every server s_i back to its initial position with a specific request *immediately* after the respective server was moved to v_{k+1}. For the bounds regarding defensive/confident algorithms (see Section 4.4.3), our structure has the advantage of separating general and specific requests. However, the structure of Haney's bound is beneficial to derive our adaptive bound below. For completeness, we state it in compliance with our notation.

Haney's proof of Theorem 4.1. As before, we consider a uniform metric with locations $v_1, \dots, v_{k+1}$, the algorithm's servers $a_1, \dots, a_k$, and the optimal servers $o_1, \dots, o_k$. Let $p(s)$ be the location of server s. Initially, $p(a_i) = p(o_i) = v_i$ for all $1 \leq i \leq k$. As in the proof of Theorem 4.1, the servers are renamed such that the algorithm moves its servers in the order of the indices.

The sequence works as follows. Pose a general request on v_{k+1}. After that, repeatedly pose a specific and a general request until a_k is at v_{k+1}. Let a_i be the server currently on v_{k+1}. Then, the specific request is for a_i at v_i. The general request is again on v_{k+1}. Observe that since the algorithm moves its servers in order of the indices, it has $2k-1$ movements. As in our lower bound, it must move every server eventually to have a bounded competitive ratio. The optimal solution has a single movement of o_k to v_{k+1}. Additionally, after the sequence, the configuration is equivalent to the initial one by renaming the locations and servers. ■

Based on the above sequence, we can still construct a lower bound when the number of general requests stays the same, but less than $k-1$ specific requests are desired, implying that the number of specific requests is less than the number of general requests. Also, we can extend the sequence above by adding specific requests if more specific than general requests are desired. Using these ideas, we can extend the lower bound above to be parameterized in s, the ratio of specific requests requiring a movement and the total number of requests requiring a movement as done in Theorem 4.2. The sequence above uses precisely $k-1$ specific requests requiring a movement and k general requests requiring a movement. Thus, it applies to $s = \frac{k-1}{2k-1}$. The parameterization allows us to express how the bound changes gradually between pure general, mixed, and pure specific inputs.

When extending the bound, we are faced with an additional technical difficulty. The bound of Theorem 4.1 limits the number of requests, as we want to compare to an optimal solution having as few movements as possible. As a result, the extensions to the sequence of Theorem 4.1 that we use do not give a bound for all rational values of s. Note that $s \in [0,1]$ is, by definition, a rational number, as any number of requests is always an integer. To still be able to give a lower bound for *any valid* s, we need Lemma 4.1 below. Intuitively, it allows us to repeat modified versions of the sequence of Theorem 4.1 sufficiently many times to achieve that the final ratio of the number of specific requests and the number of total requests in the sequence is any desired rational number between 0 and 1. The sequence of Theorem 4.1 can be repeated because the specific requests ensure that the final placement of the algorithm's servers and the optimal servers is the same and equivalent to the initial placement.

Lemma 4.1 Let $x \in \mathbb{Q}$ be a rational number greater than one that is not a natural number. Then there are natural numbers $a, b \in \mathbb{N}$ with $a, b > 0$ such that $x = \frac{a \cdot \lfloor x \rfloor + b \cdot \lceil x \rceil}{a+b}$.

Proof. Since x is not natural but rational, it holds that $\lceil x \rceil = \lfloor x \rfloor + 1$ and $x = \lfloor x \rfloor + \varepsilon$ for $\varepsilon \in \mathbb{Q}$ and $0 < \varepsilon < 1$. Set for any two numbers a, b:

$$\lfloor x \rfloor + \varepsilon = x = \frac{a \cdot \lfloor x \rfloor + b \cdot \lceil x \rceil}{a+b} = \frac{a \cdot \lfloor x \rfloor + b \cdot (1 + \lfloor x \rfloor)}{a+b}.$$

Simplifying and solving for a yields $a = b\left(\frac{1}{\varepsilon} - 1\right)$. Since $\varepsilon \in \mathbb{Q}$ and $\varepsilon < 1$, $\frac{1}{\varepsilon}$ is rational and greater than one such that $\left(\frac{1}{\varepsilon} - 1\right)$ is rational and greater than zero. Thus, it can be expressed as a fraction $\frac{c}{d}$ of two natural numbers c and d greater than zero. Set $b = d$. Then $a = c$ and both a and b are natural numbers greater than zero, which shows the lemma. ■

Equipped with the lemma above, we are ready to show Theorem 4.2. The bound is also depicted in Figure 4.8 below. Note how the bound is defined for all rational numbers $s \in [0,1]$, i.e., the analysis of the theorem is very fine-grained.

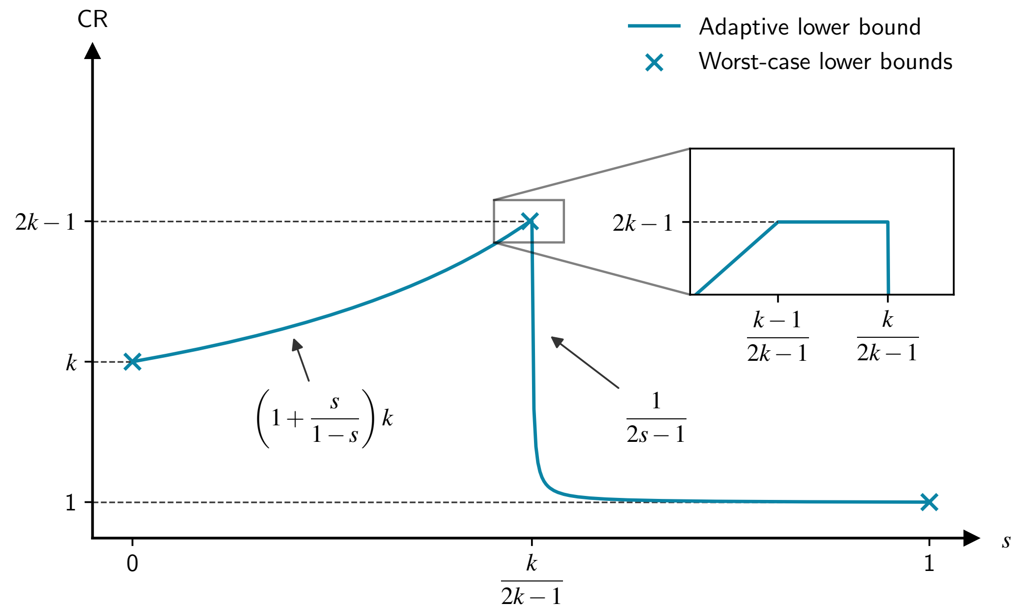

Figure 4.8: (Repetition of Figure 4.2) The figure plots the adaptive lower bound of Theorem 4.2 and the worst-case lower bounds against the ratio s of the number specific requests and the total number of requests. The worst-case bounds are for the k-server problem (k at $s=0$), the MCOKSP ($2k-1$ at $s=\frac{k}{2k-1}$), and the special case where all requests are specific (1 at $s=1$). The adaptive bound is defined for every possible $s \in [0,1]$ and goes through the worst-case points.

Theorem 4.2 — Adaptive Lower Bound. Consider any deterministic online algorithm for the any-or-one case of the multi-commodity online k-server problem. Let s be the ratio between the number of specific requests and the total number of requests that require a movement by the algorithm. Already in a uniform metric space with $k+1$ locations it holds: The algorithm has a competitive ratio of at least $(1+\frac{s}{1-s})k$ if $s \leq \frac{k-1}{2k-1}$, a competitive ratio of at least $2k-1$ if $\frac{k-1}{2k-1} < s < \frac{k}{2k-1}$, and a competitive ratio of at least $\frac{1}{2s-1}$ if $s \geq \frac{k}{2k-1}$.

Proof. Consider the uniform metric with locations $v_1,\dots,v_{k+1}$. Let $a_1,\dots,a_k$ be the servers of an online algorithm and $o_1,\dots,o_k$ be the servers of an optimal solution. Let $p(s)$ be the location of server s. Assume that initially, $p(a_i)=p(o_i)=v_i$ for all $1\leq i\leq k$. We further assume that the algorithm moves *lazy*, i.e., it only moves a server if there is an unserved request at the destination. As pointed out in [60], every algorithm can be turned *lazy* without disadvantage. This assumption allows us to reasonably capture s since it depends on the requests requiring algorithm movement.

Assume $s \leq \frac{k-1}{2k-1} < \frac{1}{2}$. Similar to the proof of Theorem 4.1, rename the indices such that the algorithm moves its a_i in order. The sequence is a modification of the proof of Theorem 4.1 by Haney: First, issue a general request on v_{k+1}. For $x<k$ steps do the following (x is determined later): After the algorithm moved a server a_i to v_{k+1}, issue a specific request for a_i at v_i (which is unoccupied). Issue another general request on v_{k+1} (which is unoccupied again). After x many steps, always pose a general request on the currently unoccupied location until the algorithm covers all locations except v_k. It then holds that x many servers have been moved twice (to v_{k+1} and back to their initial location). At the same time, the remaining $k-x$ servers have been moved at least once (similar to the proof of Theorem 4.1, otherwise, the algorithm has unbounded competitive ratio). Observe that all requests are to unoccupied locations and require a movement while the number of specific requests is x and the total number of requests is $k+x$. Overall, the cost of the algorithm is at least $k+x$. The optimal solution solves the entire sequence by moving o_k to v_{k+1} for a cost of 1,

and the competitive ratio is bounded from below by $k+x$. Observe that x must be a natural number. If $\frac{s}{1-s}k$ is a natural number, we can simply set $x = \frac{s}{1-s}k$ such that $\frac{x}{k+x} = s$ and the competitive ratio is at least $(1+\frac{s}{1-s})k$. Otherwise, observe that after the sequence, the algorithm covers the same locations as the optimal solution. Not all servers of the algorithm might be at the same location as in the optimal solution (if they were not forced back by specific requests). Still, we can repeat the sequence such that the algorithm moves as frequently as before by appropriately renaming the servers and locations. Rename the servers such that the algorithm moves them in the order of their indices again. Rename the locations such that $p(a_i) = v_i$ for every $1 \le i \le k$ afterward. Then, v_{k+1} is unoccupied by the algorithm initially. Also, every server a_i for $1 \le i \le k-1$ that moves to v_{k+1} is not on the location of o_i as o_k is on v_{k+1}. Therefore, it has to move on a specific request to the location of o_i, even though that location might not be v_i. In addition, after the repetition, the algorithm again covers the same locations as the optimal solution. Thus, the sequence can be repeated arbitrarily often.

By Lemma 4.1 we know that there are natural numbers a, b greater than zero such that $\frac{s}{1-s}k = \frac{a\lfloor\frac{s}{1-s}k\rfloor + b\lceil\frac{s}{1-s}k\rceil}{a+b}$. For a phases, set $x = \lfloor\frac{s}{1-s}k\rfloor$ and for b phases, use $x = \lceil\frac{s}{1-s}k\rceil$. Then the ratio between the number of specific requests and the total number of requests is:

$$\frac{a\lfloor\frac{s}{1-s}k\rfloor + b\lceil\frac{s}{1-s}k\rceil}{a\lfloor\frac{s}{1-s}k\rfloor + b\lceil\frac{s}{1-s}k\rceil + (a+b)k} = \frac{(a+b)\frac{s}{1-s}k}{(a+b)\frac{s}{1-s}k + (a+b)k} = s.$$

The competitive ratio is at least:

$$\frac{a\left(k+\lfloor\frac{s}{1-s}k\rfloor\right) + b\left(k+\lceil\frac{s}{1-s}k\rceil\right)}{a+b} = \frac{ak+bk+a\lfloor\frac{s}{1-s}k\rfloor + b\lceil\frac{s}{1-s}k\rceil}{a+b} = \left(1+\frac{s}{1-s}\right)k.$$

Observe for any case that $s \le \frac{k-1}{2k-1}$ ensures that:

$$\frac{s}{1-s}k \le \left\lceil\frac{s}{1-s}k\right\rceil \le \left\lceil\frac{\frac{k-1}{2k-1}\cdot k}{1-\frac{k-1}{2k-1}}\right\rceil \le \lceil k-1\rceil \le k-1.$$

Therefore, truly in each sequence $x < k$. Additionally, observe that for $s = \frac{k-1}{2k-1}$ the competitive ratio is $2k-1$.

Next, consider $s \ge \frac{k}{2k-1} > \frac{1}{2}$. As in the proof of Theorem 4.1, rename the indices such that the algorithm moves its a_i in order. Like before, the sequence modifies the sequence of Haney's proof of Theorem 4.1. In comparison to the case where it holds that $s < \frac{1}{2}$, not all servers can be forced to move due to an insufficient number of general requests. Specific requests compensate for this. In the first phase, do for $x < k$ steps the following (x is determined later): Issue a general request on v_{k+1}. Assume the algorithm moves server a_i. Issue a specific request for a_i on v_i. Since the algorithm moves the servers in order and lazy, it incurs a cost of $2x$ for the first phase and moves only servers a_i with $i \le x$. In phase two, consider $k-x-1$ servers with an index between x and k (so, server k is not considered). Find a permutation π for the initial locations of the above servers such that $v_i \neq \pi(v_i)$ for all of them. Pose a specific request for each of these servers i at $\pi(v_i)$. Finally, phase three is a single specific request for k at v_{k+1}. Observe that all requests require a movement by the algorithm, and the number of specific requests is k while the total number of requests is $k+x$. The total cost of the algorithm is at least $k+x$. The optimal solution only moves the servers of phases two and three and has a cost of at most $k-x$. Simplifying the ratio gives a lower bound of $\frac{k+x}{k-x}$ on the competitive ratio. Again, x must be a natural number. If $\frac{1-s}{s}k$ is a natural number, simply set $x = \frac{1-s}{s}k$ such that $\frac{k}{k+x} = s$ and the lower bound is $\frac{1}{2s-1}$. Otherwise, observe that the final configuration is equivalent to the initial one in the above input sequence. Due to specific requests, every server is forced to the same location as the respective optimal one. Hence, the sequence can be repeated arbitrarily often. By Lemma 4.1, we know that there are natural numbers a, b greater than zero such that $\frac{1-s}{s}k = \frac{a\left\lfloor\frac{1-s}{s}k\right\rfloor + b\left\lceil\frac{1-s}{s}k\right\rceil}{a+b}$. For a phases, set $x = \left\lfloor\frac{1-s}{s}k\right\rfloor$ and for b phases, use $x = \left\lceil\frac{1-s}{s}k\right\rceil$. Then the ratio between the number of specific requests and the total number of requests is:

$$\frac{ak+bk}{ak+bk+a\left\lfloor\frac{1-s}{s}k\right\rfloor + b\left\lceil\frac{1-s}{s}k\right\rceil} = \frac{(a+b)k}{(a+b)k+(a+b)\frac{1-s}{s}k} = s.$$

The competitive ratio is at least:

$$\frac{ak+a\left\lfloor\frac{1-s}{s}k\right\rfloor + bk + b\left\lceil\frac{1-s}{s}k\right\rceil}{ak-a\left\lfloor\frac{1-s}{s}k\right\rfloor + bk - b\left\lceil\frac{1-s}{s}k\right\rceil} = \frac{(a+b)k+(a+b)\frac{1-s}{s}k}{(a+b)k-(a+b)\frac{1-s}{s}k} = \frac{1}{2s-1}.$$

Observe that in any case $s \geq \frac{k}{2k-1}$ ensures that:

$$\frac{1-s}{s}k \leq \left\lceil\frac{1-s}{s}k\right\rceil \leq \left\lceil\frac{\left\lceil\left(1-\frac{k}{2k-1}\right)k\right\rceil}{\frac{k}{2k-1}}\right\rceil \leq \lceil k-1\rceil \leq k-1.$$

Therefore, truly in each sequence $x < k$. Additionally, observe that for $s = \frac{k}{2k-1}$ the competitive ratio is $2k-1$.

Consider the case of $\frac{k-1}{2k-1} < s < \frac{k}{2k-1}$ where $s \approx \frac{1}{2}$. First observe that the above bounds on s imply that (1) $k-1 < s(2k-1) < k$. As observed in the first part of the proof, for $s = \frac{k-1}{2k-1}$, the first input sequence gives a competitive ratio of at least $2k-1$. Also, for $s = \frac{k}{2k-1}$, the second input sequence yields a competitive ratio of at least $2k-1$. In both cases, the configuration after the sequence is the same as the initial one, and we can repeat each sequence arbitrarily often. By (1) $s(2k-1)$ is rational but not natural and, therefore, by Lemma 4.1, there are natural numbers a, b greater than zero such that $s(2k-1) = \frac{a\lfloor s(2k-1)\rfloor + b\lceil s(2k-1)\rceil}{a+b}$. Observe that $\lfloor s(2k-1)\rfloor = k-1$ and $\lceil s(2k-1)\rceil = k$. Repeat a times the sequence for $s = \frac{k-1}{2k-1}$ and b times the sequence for $s = \frac{k}{2k-1}$. Then, the ratio between the number of specific requests and the total number of requests is

$$\begin{aligned}\frac{a\frac{k-1}{2k-1} + b\frac{k}{2k-1}}{a+b} &= \frac{a\frac{\lfloor s(2k-1)\rfloor}{2k-1} + b\frac{\lceil s(2k-1)\rceil}{2k-1}}{a+b} = \frac{a\lfloor s(2k-1)\rfloor + b\lceil s(2k-1)\rceil}{a+b} \cdot \frac{1}{2k-1} \\ &= \frac{s(2k-1)}{2k-1} = s.\end{aligned}$$

Finally, the competitive ratio is $2k-1$. ■

4.4.3 Confident and Defensive Algorithms

In Section 4.3, we already introduced the key idea behind the term of *acting confidently* and *acting defensively*. Recapitulate, Definition 4.2 below, where $p^*(j)$ for server j is the last-known optimal location of j.

Definition 4.2 — Defensive/Confident. An algorithm acts defensively for j if, when a general request appears on $p^*(j)$ that requires a movement, it moves j to the request. A deterministic algorithm is *ℓ-defensive*, if for a fixed subset of ℓ servers, it *always* acts defensively. An algorithm is *ℓ-confident*, if, for ℓ servers, it does not always act defensively. An algorithm is *strictly-ℓ-confident*, if for a fixed subset of ℓ servers, it *never* acts defensively if it can be avoided.

$p^*(j)$ is by assumption known for each sever j initially and can be updated whenever a specific request for j appears because the optimal solution must serve such a request with j as well. Regarding the definition above, if multiple servers share the same p^*, we only require one of them to move in case of a general request. Still, all act defensively, as only one must move. Observe that strictly-ℓ-confident algorithms are also ℓ-confident. Most algorithms that do not explicitly act defensively are ℓ-confident, e.g., the example algorithm in Section 4.4.1. By chance, they could act defensively for a server, but this is not ensured, for example, simply because all general requests are treated the same. We introduce the notion of strictly-ℓ-confident because it allows us to show a significant drawback when not acting defensively at all for some servers:

Theorem 4.5 Consider the any-or-one case of the multi-commodity online k-server problem. No strictly-ℓ-confident algorithm (for $\ell \geq 1$) can achieve a competitive ratio better than $2k+\ell-2$.

Proof. Let ALG be the online algorithm and $p(s)$ be the location of server s. Consider a uniform metric with locations $v_1,\dots,v_{k+1}$. Initially, ALG's servers $a_1,\dots,a_k$ and the servers $o_1,\dots,o_k$ of the optimal solution share the same locations $p(a_i) = p(o_i) = v_i$, for all $i \leq k$. As in the proof of Theorem 4.1, rename the i's such that during the first phase below, ALG moves its a_i in order.

Next, we construct the adversarial sequence. First, issue a general request on v_{k+1}. Whenever a server a_i is moved by the algorithm, issue a request on $v_i = p^*(a_i)$ afterwards. Either the algorithm acts defensively on a_i and moves it back to v_i, or it does not. Since ALG only sees general requests and is deterministic, by our renaming of the i's above, it moves its a_i in order. The first phase ends, when ALG covers all the locations $v_1,\dots,v_{k-1},v_{k+1}$. From now on, we force the algorithm to cover these locations by using enough general requests. Let $\overline{D}$ be the set of servers for which ALG did not act defensively, and D be the other servers. Let $L \subseteq \overline{D}$ be the ℓ servers for which the algorithm never acts defensively.

Observe that for the servers of $\overline{D}$, ALG incurred a cost of 1 and for any of these servers a_i it holds $p(a_i) \neq p^*(a_i)$ and $p(a_j) = p^*(a_i)$ for some $a_j \in \overline{D}$. For all other servers, the algorithm had a cost of at least 2.

Next, we show that it takes ALG additional cost to sort the servers of L back to their initial location. All these servers are in $\overline{D}$. While there is a server in $a_i \in \overline{D} \setminus \{a_k\}$ with $p(a_i) \neq p^*(a_i)$ do the following: First, specifically request a_i at $p^*(a_i)$. Afterward, using general requests, enforce that ALG covers $v_1,\dots,v_{k-1},v_{k+1}$. ALG has a cost of 1 for moving a_i. Additionally, it has a cost of 1 to move a server to a_i's previous location. Observe that a_i's previous location p was either v_{k+1} or on an initial location of one server of $\overline{D}$. Thus, by moving a server to p, if the moved server is one of the servers for which the algorithm always acts confidently, it cannot return to its initial location because only general requests on p appear. After any iteration, all servers of L that were not yet specifically requested are at a location different from their initial one because all those servers never act defensively, and, at their locations, only general requests appeared. Note that a_k cannot be forced back to its initial location. Thus, if (a) $a_k \in L$, the process can be repeated $\ell - 1$ many times. Otherwise, if (b) $a_k \notin L$, the process can be repeated ℓ many times. In total, there is a cost of at least $2(\ell-1)$ in case (a), or 2ℓ in case (b) for moving servers of L. Next, force every other server except v_k back to its initial location using specific requests. Then every server except a_k has at least a cost of two.

Summing up, in the case of (a), there are $k-\ell$ servers having a cost of 2 while the servers of L imply a cost of $\ell + 2(\ell-1)$. In the case of (b), there are $k-\ell-1$ servers (because $a_k \notin L$) with a cost of 2 while the servers of L have a cost of $\ell + 2\ell$. In any case, the total cost is at least $2k+\ell-2$. The optimal solution solves the entire sequence by moving o_k to v_{k+1} for a cost of 1. ■

For Theorem 4.5, the input sequence is depicted in Figure 4.7 (page 98), where additionally, general requests in phase II enforce that all locations except v_k remain covered. By that, the servers of L have further costs. To see this, consider Figure 4.9 (on the next page). The algorithm acted confidently for the servers of L, which the adversary transformed into a mistake. Every server of L should have stuck to its initial position, but the algorithm moved it and has to move it back. By the movement back, the algorithm leaves a location unoccupied and, since it acts confidently, potentially fills it with the wrong server again.

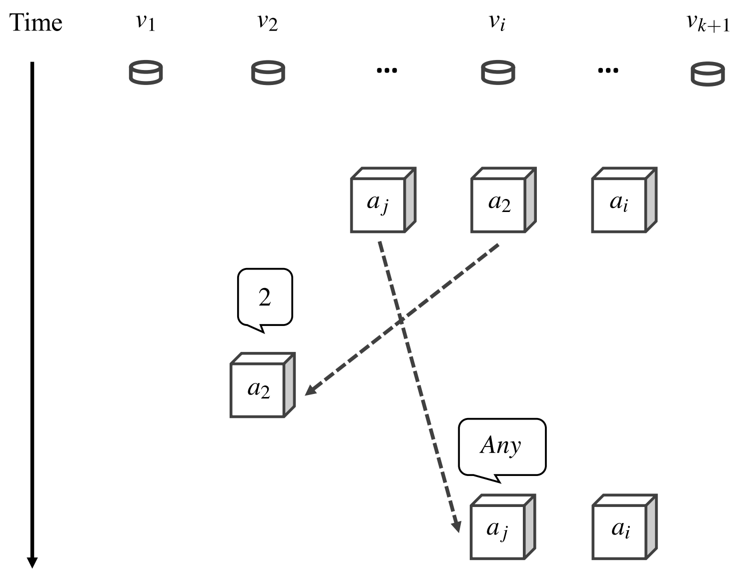

Figure 4.9: Assume, a_2 and a_i are in L. Then forcing a_2 back to v_2 with a specific request incurs cost one and leaves v_i empty. A general request on v_i incurs cost one, but since $a_i \in L$, the algorithm cannot move a_i there and moves some a_j. Therefore, after both requests, a_2 is the only server of L that returned to its initial location. Excluding a_k, each server in L can be forced to its initial location with a cost of at least 2.

The servers of L that induce an unnecessary cost in the sequence of Theorem 4.5 can be reduced by acting defensively for more servers. Hence, one might think acting defensively is always a better option. While acting defensively truly reduces the competitive ratio in the worst case (as we see in Section 4.5.3), there is a major drawback on inputs where there are more general than specific requests. To come to this conclusion, we first show Theorem 4.12 below, showing that the performance for pure general inputs degrades the more servers act defensively.

Theorem 4.12 Consider the any-or-one case of the multi-commodity online k-server problem. No ℓ-defensive algorithm can achieve a competitive ratio better than $k+\ell-1$ on pure general inputs.

Proof. Let ALG be the online algorithm and $p(s)$ be the location of server s. As in the proof of Theorem 4.5, consider a uniform metric with locations $v_1,\dots,v_{k+1}$ such that initially, $p(a_i) = p(o_i) = v_i$ for all servers $a_1,\dots,a_k$ of the algorithm and $o_1,\dots,o_k$ of the optimal solution. As before, we assume the i's are renamed such that the algorithm moves its a_i in order. Let D be the set of servers on which ALG always acts defensively and $\overline{D}$ be the other servers.

The sequence is the first phase of the proof of Theorem 4.5: First, issue a general request on v_{k+1}. Whenever a server a_i is moved by the algorithm, issue a request on $v_i = p^*(a_i)$ afterwards. Either the algorithm acts defensively on a_i and moves it back to v_i, or it does not. Since ALG only sees general requests, is deterministic, and we renamed the i's, it moves its a_i in order. When ALG covers all locations of $v_1,\dots,v_{k-1},v_{k+1}$, the sequence stops. Each server of D except possibly a_k was moved by ALG twice and each other server at least once. Thus, ALG's cost is at least $2\,(|D|-1)+|\overline{D}|+1 \geq k+\ell-1$. The optimal solution solves the entire sequence by moving o_k to v_{k+1} with a cost of 1. ■

The proof of Theorem 4.12 consists of the first phase of the sequence for Theorem 4.5. For the proof, we use that even without specific requests, the online algorithm moves $\ell-1$ servers back to their initial location and thus twice. Next, we can use Theorem 4.1 and Theorem 4.12 to formulate a bound for ℓ-defensive algorithms for $s \leq \frac{k}{2k-1}$, i.e., mixed inputs where there are more general than specific requests.

Theorem 4.6 Consider any ℓ-defensive online algorithm for the any-or-one case of the multi-commodity online k-server problem (for $\ell \geq 1$). Let s be the ratio between the number of specific requests and the total number of requests that require a movement by the algorithm. For $s \in (0, \frac{1}{k+\ell-1})$, the algorithm has a competitive ratio of at least $k+\ell-2$. For $s=0$ and $\frac{1}{k+\ell-1} \leq s \leq \frac{k}{2k-1}$, the algorithm has a competitive ratio of at least $k+\ell-1$.

Proof. We consider four different sequences and show how to apply them multiple times to achieve the desired ratio s. *Type one* is the sequence presented in the proof of Theorem 4.12. It consists of $k+\ell-1$ general requests requiring a movement with $s=0$.

Type two is the same sequence applied afterward. Here, the number of movements may be different. As in the proof, let D be the set of servers for which the algorithm always acts defensively. Consider the configuration at the end of the sequence. Note that the servers of the algorithm cover the same locations as the optimal servers. Additionally, each server of D except possibly a_k is at the same location as in the optimal solution. If $a_k \notin D$ or if a_k is at the same location as in the optimal solution, renaming the servers and locations allows repeating the type one sequence with the same analysis as in the proof of Theorem 4.12. Otherwise, one can still repeat the sequence, but the algorithm has fewer movements. Observe that $a_k \in D$, and, after the type one sequence, the empty location is $p^*(a_k) = v_k$. Therefore, a_k moves *once* to v_k at the first request of the repetition. The last server the algorithm moves (once) might again be in D. Therefore, in the repetition, the algorithm has a cost of 2 for the $|D|-2$ remaining servers and one for $|\overline{D}|+2$ servers. Thus, the competitive ratio in the repetition is only $2(|D|-2)+|\overline{D}|+2 = k+\ell-2$, i.e., by one smaller than for type one. After the repetition, again $|D|-1$ servers of D are at the same location as in the optimal solution. Thus, the type two sequence can be repeated arbitrarily often, yielding a competitive ratio of at least $k+\ell-2$. The sequence respectively consists of $k+\ell-2$ general requests requiring a movement with $s=0$.

Type three consists of type two with a single specific request at v_{k+1} for server k afterward. If server k was not on v_{k+1}, the specific request requires a movement and $s = \frac{1}{k+\ell-1}$. After the sequence, all servers of D are at the same location as in the optimal solution. Further, the algorithm covers the same locations as the optimal solution. Therefore, by renaming the servers/locations appropriately, the sequence can be repeated arbitrarily often. Note that the competitive ratio for a type three sequence is $k+\ell-1$.

R Technically, when repeating type two, type one might reappear. Similarly, for repetitions of type three, the specific request might not require a movement influencing s and the number of relevant movements. Since the algorithm is deterministic, we know how often which case appears when constructing the sequence and could adapt our bound accordingly. The sequence can enforce the desired ratio of s by introducing enough repetitions. To improve the readability, we continue the proof under the assumption that repetitions of type two are always of type two and that the specific request of type three always requires a movement.

Type four is the sequence of Haney's proof of Theorem 4.1, where we swap the last general request with a specific request for server k at v_{k+1}. Every server was specifically requested in the sequence and is at the same location as in the optimal solution afterward. The sequence consists of k specific and $k-1$ general requests requiring a movement with $s = \frac{k}{2k-1}$. Note that it can also be repeated arbitrarily often.

Next, we combine and repeat the sequence types mentioned above to show bounds for all values of $s \leq \frac{k}{2k-1}$. Note that the theorem already holds for $s=0$ by Theorem 4.12. First, we combine types one, two, and three to show a lower bound of $k+\ell-2$ for $s \in (0, \frac{1}{k+\ell-1})$. After that, we combine types three and four for a lower bound of $k+\ell-1$ for $s \in [\frac{1}{k+\ell-1}, \frac{k}{2k-1}]$.

R When after a type three sequence, the respective server k is already at v_{k+1}, the specific request requires no movement of the algorithm. Then, it holds that $s=0$ and one can show a bound of $k+\ell-1$ for any $s \leq \frac{k}{2k-1}$ using types three and four. Especially, the case happens if $\ell = k$. Therefore, we assume for the following that the case does not occur.

Assume $s \in (0, \frac{1}{k+\ell-1})$. Note that after type one, type two can be applied. After that, type three can be applied. Note that since $s \in [0,1]$ is rational, s can be expressed by two integers $s = \frac{a}{b}$. The entire sequence consists of once type one, then $b - a(k+\ell-1) - 1$ times type two, and $a(k+\ell-2)$ times type three. To arrive at s, we leave out the last general request of the last type three repetition. We can do so, as the respective request would move a_k by definition to some location other than v_{k+1}. Since a, b, k, and ℓ are integers greater than zero, the numbers of repetitions are also integers. As $\frac{a}{b} = s < \frac{1}{k+\ell-1}$ it holds that $b > a(k+\ell-1)$ such that the numbers of repetitions are at least zero. Therefore, the sequence is valid. Next, the total number of specific requests requiring a movement equals the number of type three repetitions $a(k+\ell-2)$. The total number of requests requiring a movement is $(k+\ell-1) + (b - a(k+\ell-1) - 1) \cdot (k+\ell-2) + a(k+\ell-2) \cdot (k+\ell-1) - 1$. Therefore, the ratio becomes

$$\frac{a(k+\ell-2)}{(k+\ell-1) + (b-a(k+\ell-1)-1)\cdot(k+\ell-2) + a(k+\ell-2)\cdot(k+\ell-1) - 1}$$
$$= \frac{a}{b} = s.$$

Since the competitive ratio in any sequence is at least $k+\ell-2$, the bound follows.

Assume that $s \in [\frac{1}{k+\ell-1}, \frac{k}{2k-1}]$. We can append a type three sequence after a type four sequence. As before, $s = \frac{a}{b}$ can be expressed by two integers a, b. The entire sequence consists of $x = k^2(k+\ell-1) \cdot a - k^2 \cdot b$ repetitions of type four and $y = k^3 \cdot b - k^2(2k-1)a$ repetitions of type three. As a, b, k, and ℓ are integers greater than zero, x and y are integers. Due to $\frac{a}{b} = s \geq \frac{1}{k+\ell-1}$ we know that $(k+\ell-1)a \geq b$ and, therefore, $x \geq 0$. Due to $\frac{a}{b} = s \leq \frac{k}{2k-1}$, we know that $kb \geq (2k-1)a$ and, therefore, $y \geq 0$. The number of specific requests requiring a movement is $kx + y$. The total number of requests requiring a movement is $(2k-1)x + (k+\ell-1)y$. Then, the ratio becomes

$$\frac{kx+y}{(2k-1)x + (k+\ell-1)y} = \frac{(k^3(k+\ell-1) - k^2(2k-1))a}{(k^3(k+\ell-1) - k^2(2k-1))b} = \frac{a}{b} = s.$$

Since the competitive ratio for types three and four is at least $k+\ell-1$, the bound follows. ∎

Intuitively, an algorithm that acts defensively on pure general inputs already moves more than required to avoid higher costs on mixed inputs. Here, the structure of Theorem 4.1 is crucial as it only reveals the specific requests after the general requests have already appeared. In other words, an algorithm has to decide to act defensively and improve its competitive ratio for the mixed case without even knowing if specific requests will occur. One step further, we can show Theorem 4.7.

Theorem 4.7 Consider the any-or-one case of the multi-commodity online k-server problem. No deterministic algorithm can achieve a competitive ratio of k on pure general inputs and $2k-1$ on mixed inputs.

Proof. Assume we have an algorithm that achieves a competitive ratio of k on pure general inputs and one of $2k-1$ on mixed inputs. As in the proofs of Theorem 4.5 and Theorem 4.12, consider a uniform metric with locations $v_1, \ldots, v_{k+1}$ such that initially, $p(a_i) = p(o_i) = v_i$ for all servers $a_1, \ldots, a_k$ of the algorithm and $o_1, \ldots, o_k$ of the optimal solution. As before, we assume the i's are renamed such that the algorithm moves its a_i in order.

The sequence is as our sequence for the proof of Theorem 4.5: First, issue a general request on v_{k+1}. Whenever a server a_i is moved by the algorithm, issue a general request on $v_i = p^*(a_i)$ afterward. The optimal solution solves the entire sequence by moving o_k to v_{k+1} at a cost of 1.

Since the algorithm has a competitive ratio of k on pure general inputs, it acted confidently for all servers. Otherwise, it would have moved every server at least once and at least one server twice, which implies a cost of at least $k+1$. Since the algorithm moved each server in the first phase, it has cost k for that phase. After the first phase, no server is on its initial location, and some server a_j with $j \neq k$ is on v_{k+1}. Next, issue a specific request for every server on its location in the optimal solution. Then, the algorithm has a cost of at least k to move every server to its final location. The total cost, in this case, is $2k > 2k-1$, and we have a contradiction. ■

The proof of Theorem 4.7 is based on the input sequences of Theorem 4.5 and Theorem 4.12. It uses the fact that any algorithm with a competitive ratio of k on pure general inputs has to be k-confident in the first phase. But then, a_k is not on v_{k+1}. Thus, even if it reverts all changes in the second phase and moves a_k, the algorithm has a cost higher than $2k-1$.

Theorem 4.7 still leaves space for nearly optimal algorithms on all input types. Due to Theorems 4.5 and 4.12, such algorithms must act confidently sometimes but not always for a significant number of servers. Interestingly, most algorithms that treat every general request the same fall into this category, as they might – by chance – act defensively. While our bounds do not yield increased lower bounds for algorithms of the above category, we strongly believe that they suffer similar drawbacks dependent on the number of times they act defensively/confidently. To see this, observe that the lower bounds are very similar: The adversarial sequence of Theorem 4.5 extends the one from Theorem 4.12. To build the lower bound of Theorem 4.5, it even suffices if the given algorithm acts confidently for ℓ servers not always but *always in the sequence given by the lower bound.* As an intuition, to perform well on general inputs, the algorithm should act mostly confidently in the first phase of the lower bound of Theorem 4.5. Then, in the second phase, the algorithm should avoid acting confidently. Here, we can see how algorithms that treat every general request the same probably perform as badly on mixed inputs as strictly-k-confident algorithms. The same effect can be seen by Theorem 4.4, stating that for our k-confident algorithm CONF-MCOKSP, even though it is not strictly-k-confident, the same lower bound as Theorem 4.5 applies. To conclude, our lower bounds indicate a trade-off between performing well on general/mixed inputs controlled by whether or not and to which degree an online algorithm acts defensively/confidently.

4.5 Algorithms for Uniform Metrics

Next, we present algorithms that incorporate acting confidently/defensively and show bounds on their competitive ratio. Our algorithms consider a uniform metric space and the any-or-one case. All our bounds are analyzed respecting the formerly introduced ratio s of the number of specific requests against the total number of requests.

We introduce further notation required for our algorithms in Section 4.5.1. Then we present a k-confident algorithm called CONF-MCOKSP in Section 4.5.2. Thereafter, we introduce a k-defensive algorithm called DEF-MCOKSP in Section 4.5.3. As acting confidently/defensively is a behavior implemented for each server separately, one can also design an algorithm that acts confidently for a specified set of servers and acts defensively for all others. We present such an algorithm, MIXED-MCOKSP in Section 4.5.4. MIXED-MCOKSP thereby generalizes CONF-MCOKSP and DEF-MCOKSP.

4.5.1 Additional Notation

For any server j, denote by $p(j)$ its current location. Denote by $p^*(j)$ the *last known optimal location* of j. We assume that an online algorithm's servers are initially at the same locations as the servers of the optimal solution. Therefore, $p^*(j)$ is well-defined. All our algorithms, CONF-MCOKSP, DEF-MCOKSP, and MIXED-MCOKSP work in *phases* and utilize sets. For a set of servers U, let $p(U)$ be the set of locations occupied by servers of U. In our analyses, we need a precise understanding of time. For this, we define t to be the time step at which the t-th request appears. We denote with *right before* t the time at the beginning of t after the request is revealed and before the algorithm and the optimal solution moved their servers and answered the request. We denote by *right after* t, the point in time at the end of t after the algorithm and the optimal solution moved their servers. Observe that considering time t, right after t is before right before $t+1$. Concerning the managed sets, we denote by $\widehat{U^i}$ the content of a set U^i right after the end of phase i. For a phase, let t_{start} be the time step of the first request and t_{end} be the time step of the last request.

R The assumption that the algorithm's servers are initially at the same locations as the optimal ones can be dropped. Then, the competitive ratios of our algorithms increase by an additive term in $\mathcal{O}(k)$ independent of the optimal solution.

4.5.2 A k-confident Algorithm

In the following section, we present our k-confident algorithm (see Definition 4.2) called Conf-Mcoksp. We analyze Conf-Mcoksp parameterized in the share of specific requests on the total number of requests. Note that we only consider requests that require a movement of the algorithm. All other requests have no relevance to the competitive ratio. This way, our results (captured in Theorem 4.3) not only show how the competitive ratio is bounded for our general model but also for instances of the k-server problem. For a graphical representation, consider Figure 4.10.

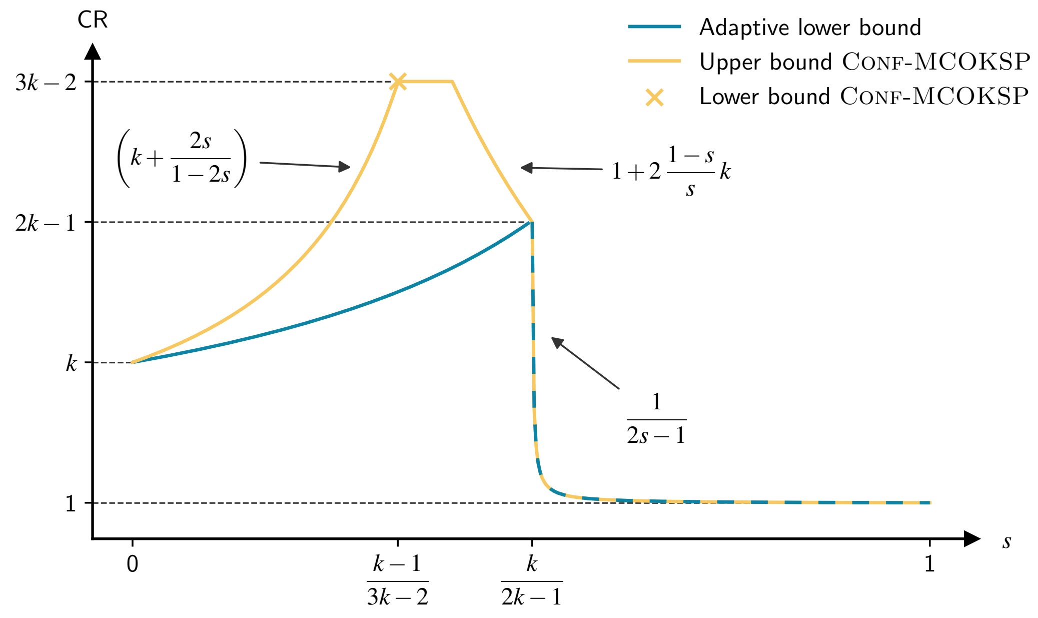

Figure 4.10: (Repetition of Figure 4.3) The plot compares the upper and lower bounds for Conf-Mcoksp (Theorems 4.3 and 4.4) and the adaptive lower bound (Theorem 4.2) plotted against s, the ratio between the number of specific requests and the total number of requests. The competitive ratio of the algorithm roughly follows the lower bound and is optimal for $s=0$ and $s \geq \frac{k}{2k-1}$. However, for $s=\frac{k-1}{3k-2}$, the algorithm has a competitive ratio of at least $3k-2$.

As we can see, Conf-Mcoksp achieves an optimal competitive ratio for $s=0$ and when specific requests dominate the input ($s \geq \frac{k}{2k-1}$). In any case, the competitive ratio is upper bounded by $3k-2$.

Theorem 4.3 Consider the any-or-one case of the multi-commodity online k-server problem on uniform metrics. Let s be the ratio between the number of specific requests and the total number of requests that require a movement by the algorithm. The competitive ratio of CONF-MCOKSP is at most $\min\{k + \frac{2s}{1-2s}k, 3k-2, 1+2\frac{1-s}{s}k\}$ for $s < \frac{k}{2k-1}$ and at most $\frac{1}{2s-1}$ for $s \geq \frac{k}{2k-1}$.

CONF-MCOKSP employs ideas of the marking approach [58, pp. 752-758] and operates in phases. For an explanation of marking approaches, see Section 4.2. A phase tries to capture the longest sequence of requests for which the optimal solution does not need to move. Right after a phase ends, the optimal solution must have a cost of at least one. During a phase, all algorithm servers get marked one by one. If the cost of an algorithm in each phase is at most c, it has a competitive ratio of c. However, the main difference in our approach is that a server can get unmarked again during a phase. Also, we do not only differentiate between marked and unmarked servers but distinguish more carefully using set memberships as described in Section 4.5.2.1. In comparison, our set C (defined later) denotes the unmarked servers, whereas all other sets contain marked ones. After the algorithm, we present a detailed analysis in Section 4.5.2.2.

4.5.2.1 The Algorithm

Next, we describe how CONF-MCOKSP (split into CONF-MCOKSP-GEN and CONF-MCOKSP-SPEC) works. Consider the pseudo code on the next page for the following. As described above, the algorithm operates in phases. CONF-MCOKSP aims at tracking a part of the input sequence in which the optimal solution must have a movement. Such a part is then a phase.

Sets managed by CONF-MCOKSP. During the execution, CONF-MCOKSP handles for each phase different sets. We denote that a set belongs to phase i by an exponent of i that we omit when the phase is clear from the context. At the beginning of any phase, every server is assigned to a candidate set C (CONF-MCOKSP-GEN Lines 5 – 6; CONF-MCOKSP-SPEC Lines 3 – 4). During the phase, CONF-MCOKSP handles four sets C, G, L, and F. We ensure that each server is in exactly one of C, G, or F. L stores locations. More precisely, L stores the locations where only general requests appeared, and G stores the servers at such locations. F contains all specifically requested servers. Servers of F can be at the same location while the locations of L (and thus $p(G)$) are distinct and do not overlap with the locations of $p(F)$. This distinction is necessary for a parameterized bound on the competitive ratio when many specific requests appear. While by definition $p(G) \subseteq L$, locations in L can become unoccupied when a server of G gets specifically requested.

Actions on the arrival of a request. Whenever a general request r appears, if its location is not in $p(F) \cup L$, CONF-MCOKSP stores it in L (CONF-MCOKSP-GEN Line 9). When there is no server of $G \cup F$ on the requested location already, select a server $j \in C$ to be the answering server and assign j to G (CONF-MCOKSP-GEN Line 3, Line 10). When a specific request r appears, we observe the following: The server that is specifically requested must be at the location of r in the optimal solution. Assuming that the optimal solution has no cost within the phase, any such server can no longer move after it is specifically requested. Therefore, CONF-MCOKSP declares server j as *frozen* and assigns it to F (CONF-MCOKSP-SPEC Line 7). The phase ends in two cases: (1) it can no longer be guaranteed that $|L| + |F| \leq k$ when serving r (CONF-MCOKSP-GEN Line 4; CONF-MCOKSP-SPEC Line 2) or (2) a server $j \in F$ is specifically requested at a different location (CONF-MCOKSP-SPEC Line 2). In case (1), the optimal solution cannot cover all locations where requests appeared in the phase with servers, and thus a server must have been moved. In case (2), the optimal solution must have moved j. We remark that the statement to move any server of C to serve a general request is ambiguous, and any order on the servers of C will do. For precision, assume that the servers are selected using the FIFO (first in, first out) rule.

CONF-MCOKSP-GEN: **General request r arrives in phase i**

```
1:  if $r \notin p(G \cup F)$ then
2:      if $r \in L$ then
3:          Move some $j \in C$ to $r$ and assign it to $G$
4:      else if $|L| + |F| = k$ then
5:          Start the next phase $i+1$
6:          Set $C^{i+1} \leftarrow S$ and $G^{i+1}, L^{i+1}, F^{i+1} \leftarrow \emptyset$
7:          Process $r$ again for phase $i+1$
8:      else if $|L| + |F| < k$ then
9:          $L \leftarrow L \cup \{p(r)\}$
10:         Move some $j \in C$ to $r$ and assign it to $G$
```

CONF-MCOKSP-SPEC: **Specific request r for server j arrives in phase i**

```
1:  if $r \neq p(j)$ then
2:      if $j \in F$ or $(|L| + |F| = k$ and $r \notin p(G))$ then
3:          Start the next phase $i+1$
4:          Set $C^{i+1} \leftarrow S$ and $G^{i+1}, L^{i+1}, F^{i+1} \leftarrow \emptyset$
5:          Process $r$ again for phase $i+1$
6:      else if $j \notin F$ and $(|L| + |F| < k$ or $r \in p(G))$ then
7:          Move $j$ to $r$ and assign it to $F$
8:          if There is a $s \neq j$, $s \notin F$ on $r$ then
9:              Remove $p(s)$ from $L$
10:             Assign $s$ to $C$
11: else
12:     Assign $j$ to $F$
```

The initialization. The very first phase is different from all others. Since we assume that the servers of the online algorithm are at the same locations as the servers of the optimal solution, no movements happen in the first phase. To reflect this, we set for the first phase $C^1, G^1, L^1 = \emptyset$ and $F^1 = S$ (the set of all servers).

Differences to classical k-server. Due to specific requests, CONF-MCOKSP incorporates behaviors that are fundamentally different from the classical k-server problem: Observe that a specific request removes a location $x \in L$ when a server becomes frozen there. When this happens, a server $j \in G$ can even be assigned to C again (CONF-MCOKSP-SPEC Lines 8 – 10). From a perspective of a marking algorithm, this means j becomes unmarked again. Intuitively, by the specific request, CONF-MCOKSP detects that j was the wrong server to answer the previous general request on x. Moreover, a specific request for j can yield that j's previous location of G becomes *unoccupied*. To still keep track of it, CONF-MCOKSP stores it in L. For a location of L where no server of G is, it may be necessary to move another server on it due to a later general request. One could ensure that all locations of L are covered the entire time by servers in G, but that behavior has no advantage.

R Indeed, as we see in Section 4.5.3.3, ensuring that all locations of L are covered by servers of G any time increases the competitive ratio for large values of $s \to 1$.

4.5.2.2 An Upper Bound on the Competitive Ratio

Next, we present an analysis for the upper bound on the competitive ratio of CONF-MCOKSP captured by Theorem 4.3. First, we lower bound the cost of the optimal solution OPT per phase of the algorithm. After that, we analyze the cost of CONF-MCOKSP per phase and finally show the competitive ratio.

On the cost of the optimal solution. Before analyzing the cost of OPT, we need a technical lemma on the sets the algorithm manages. The algorithm ensures Lemma 4.2.

Lemma 4.2 At any point in time, L and $p(F)$ are disjoint.

Proof. Assume there is a location $\ell \in L$ where some server $s \in F$ is. If s was first on ℓ, no server of C would be moved to ℓ and ℓ would not be in L, because s is able to answer general requests. If ℓ became part of L first, s moved to ℓ due to some specific request. Then the algorithm ensures that ℓ is no longer part of L. ■

Now we can bound the cost of an optimal solution. First, we make clear that each phase of the algorithm indeed tracks one inevitable movement of OPT.

Lemma 4.3 Consider any phase but the last and the first request r right after the phase ends. OPT has a cost of at least 1 during the time right after t_{start} until right after $t_{\text{end}}+1$.

Proof. Consider any phase that ends. At the end of the phase, it holds either that (i) $|\widehat{L}|+|\widehat{F}|=k$ or (ii) a server $j \in F$ is specifically requested at a location different to $p^*(j)$. Assume OPT has had no movement. Then, during the time interval, OPT has its servers at least at the locations $L \cup p(\widehat{F}) \cup \{r\}$ since at each of these locations, a request appeared (right after t_{start}, OPT has a server on the location of the first request). Also, OPT must have the servers of F at identical locations as CONF-MCOKSP.

In the case of (i), OPT covers due to Lemma 4.2 $|\widehat{L}|+1 > k-|\widehat{F}|$ distinct locations using $k-|\widehat{F}|$ servers which cannot be. In the case of (ii), j is specifically requested at two different locations meaning that OPT must have placed j at two different locations. In any case, there is a contradiction. Thus, OPT has cost at least 1. ■

In addition, the cost of the optimal solution can be lower bounded as follows. Consider the set $\widehat{F^i}$ of the algorithm. It can be decomposed into two disjoint sets of servers. Servers in $\widehat{F_1}$ are those which were specifically requested at the exact location as they were the last time before the phase, while servers in $\widehat{F_2}$ were specifically requested at a location different from the previous one. Every time a server gets specifically requested at a location different than the one where it was specifically requested the last time, OPT has to move the server. Therefore, Observation 4.1 holds.

Observation 4.1 OPT has a cost of at least $\sum_i |\widehat{F_2^i}|$.

On the cost of the algorithm. Next, we show how the cost of the algorithm for a phase is bounded. We show four bounds in total. First, we present a worst-case upper bound of $3k-2$ for any phase in Lemma 4.4. After that, we show three bounds parameterized in s in Lemma 4.5. To relate to s, we restate the cost of the algorithm in a phase as follows.

Observation 4.2 Consider any phase $i>1$. Let g^i be the number of general requests during the phase requiring algorithm movement. Let f^i be the number of specific requests during the phase that require a movement of the algorithm. The cost of CONF-MCOKSP in the phase is at most g^i+f^i.

In the following lemma, we show the worst-case upper bound for our algorithm by analyzing the maximum cost that can occur in a phase.

Lemma 4.4 The competitive ratio of CONF-MCOKSP is at most $3k-2$.

Proof. Assume there are p phases. Due to Lemma 4.3, OPT's cost is at least $p-1$. Based on Observation 4.2, we know that for any phase but the first, CONF-MCOKSP's cost is at most $g^i + f^i$. Observe that it also holds that

$$g^i \leq |\widehat{G}| + |\widehat{F}| + f^i, \tag{4.1}$$

$$f^i \leq |\widehat{F}|, \tag{4.2}$$

$$k \geq |\widehat{G}| + |\widehat{F}|. \tag{4.3}$$

Equation (4.1) holds because general requests requiring a movement could have appeared only at locations covered by a server in the end and at unoccupied locations. The number of former locations is upper bounded by $|\widehat{G}| + |\widehat{F}|$. A location can only become unoccupied when a server moves away from it and joins F, which implies that there are at most f^i many. Equation (4.2) holds by definition, and Equation (4.3) is ensured by the algorithm.

We consider two cases: (a) $|\widehat{F}| < k$ and (b) $|\widehat{F}| = k$. Consider the case of (a). Then in the phase, the cost of the algorithm is at most

$$g^i + f^i \overset{\textit{Equation } (4.1)}{\leq} |\widehat{G}| + |\widehat{F}| + 2f^i \overset{\textit{Equation } (4.2)}{\leq} |\widehat{G}| + 3|\widehat{F}| \overset{\textit{Equation } (4.3)}{\leq} k + 2|\widehat{F}| \overset{(|\widehat{F}|<k)}{\leq} 3k-2.$$

Consider the case of (b). In this case, $|\widehat{G}| = 0$. Consider the last specific request for server j. Either j is already at the requests' location or was not used before, i.e., $j \in C$. If j is already at the request's location, $f^i < |\widehat{F}|$ and thus, it holds

$$g^i + f^i \overset{\textit{Equation } (4.1)}{\leq} |\widehat{G}| + |\widehat{F}| + 2f^i \overset{(|\widehat{G}|=0)}{=} |\widehat{F}| + 2f^i \overset{(f^i<|\widehat{F}|)}{\leq} 3|\widehat{F}| - 2 \overset{(|\widehat{F}|\leq k)}{\leq} 3k-2.$$

Otherwise, j was also in C when the penultimate specific request appeared, implying that due to that request, no location of G became unoccupied. Therefore, $g^i \leq 2|\widehat{F}| - 2$ and therefore it holds that

$$g^i + f^i \overset{(g^i\leq 2|\widehat{F}|-2)}{\leq} 2|\widehat{F}| - 2 + f^i \overset{\textit{Equation } (4.2)}{\leq} 3|\widehat{F}| - 2 \overset{(|\widehat{F}|\leq k)}{\leq} 3k-2.$$

In total, the cost of the algorithm over all phases is at most $(p-1)(3k-2)$. Since OPT has a cost of at least $p-1$, the lemma follows. ■

Next, we show three bounds that parameterize the competitive ratio of CONF-MCOKSP in the structure of the input sequence.

Lemma 4.5 Let s be the ratio between the number of specific requests and the total number of requests that require a movement by the algorithm. The competitive ratio of CONF-MCOKSP is at most $\min\{k + \frac{2s}{1-2s}k,\ 1 + 2\frac{1-s}{s}k\}$ for $s < \frac{k}{2k-1}$ and at most $\frac{1}{2s-1}$ for $s \geq \frac{k}{2k-1}$.

Proof. Assume there are p phases. Denote the cost of the optimal offline solution by C_{OPT}. First, due to Lemma 4.3, OPT's cost is at least $p-1$. Secondly, due to Observation 4.1, OPT has a cost of at least $\sum_i |\widehat{F_2^i}|$.

Next, consider the cost of CONF-MCOKSP denoted by $C_{\text{CONF-MCOKSP}}$. We start with some basics. As before in the proof of Lemma 4.4, for any phase but the first, CONF-MCOKSP's cost is at most $g^i + f^i$ and Equations (4.1) to (4.3) hold. Observe that the first phase costs the algorithm nothing, i.e., $f^1 = g^1 = 0$. Further, we can derive

$$s = \frac{\sum_i f^i}{\sum_i (g^i + f^i)}$$

$$\begin{aligned}
\Leftrightarrow \qquad & s\sum_i (g^i + f^i) = \sum_i f^i \\
\Rightarrow \qquad & s\sum_i (k + 2f^i) \geq \sum_i f^i \\
\Leftrightarrow \qquad & \sum_{i=2}^{p} f^i \leq \frac{s}{1-2s}\sum_{i=2}^{p} k. \qquad (4.4)
\end{aligned}$$

We begin by using Equations (4.1) to (4.4). Summed up over all phases, we end up at:

$$\begin{aligned}
C_{\text{CONF-MCOKSP}} \quad & \leq \quad \sum_{i=2}^{p}(g^i + f^i) \overset{\textit{Equation } (4.1)}{\leq} \sum_{i=2}^{p}(|\widehat{G^i}| + |\widehat{F^i}| + 2f^i) \\
& \overset{\textit{Equation } (4.3)}{\leq} \sum_{i=2}^{p}\left(k + 2f^i\right) \overset{\textit{Equation } (4.4)}{\leq} \sum_{i=2}^{p}\left(k + \frac{2s}{1-2s}k\right) \\
& = \quad (p-1)\left(k + \frac{2s}{1-2s}k\right) \overset{(C_{\text{OPT}} \geq p-1)}{\leq} \left(k + \frac{2s}{1-2s}k\right) C_{\text{OPT}}.
\end{aligned}$$

Next, we turn to the second bound. We can derive that

$$\begin{aligned}
& s = \frac{\sum_i f^i}{\sum_i (g^i + f^i)} \\
\Leftrightarrow \qquad & s\sum_i g^i = (1-s)\sum_i f^i \\
\Leftrightarrow \qquad & \sum_{i=2}^{p} g^i = \frac{1-s}{s}\sum_{i=2}^{p} f^i \leq \frac{1-s}{s}\sum_{i=2}^{p} k. \qquad (4.5)
\end{aligned}$$

Let f_1^i be the number of movements due to servers in $\widehat{F_1^i}$ and f_2^i be the number of movements due to servers in $\widehat{F_2^i}$. Consider the servers of $\bigcup_i \widehat{F_1^i}$. For any such server j for phase i, it holds: If our algorithm has a cost for j when j joins $\widehat{F_1^i}$, then j was moved by a general request since the time it was lastly specifically requested. The implication is that

$$\sum_i f_1^i \leq \sum_i g^i. \qquad (4.6)$$

Then, the cost of the algorithm is, at most:

$$\begin{aligned}
C_{\text{CONF-MCOKSP}} \leq \quad & \sum_{i=2}^{p}(g^i + f^i) \overset{(f^i = f_1^i + f_2^i)}{=} \sum_{i=2}^{p}(g^i + f_1^i + f_2^i) \overset{\textit{Equation } (4.6)}{\leq} \sum_{i=2}^{p}(2g^i + f_2^i) \\
\overset{\textit{Equation } (4.5)}{\leq} \quad & \sum_{i=2}^{p}\left(2\frac{1-s}{s}k + |\widehat{F_2^i}|\right) = (p-1)2\frac{1-s}{s}k + \sum_i |\widehat{F_2^i}| \\
\overset{\left(C_{\text{OPT}} \geq \max\left\{(p-1), \sum_i |\widehat{F_2^i}|\right\}\right)}{\leq} \quad & \left(1 + 2\frac{1-s}{s}k\right) C_{\text{OPT}}.
\end{aligned}$$

Consider now the third bound. By the derivation of Equation (4.5) we know

$$\sum_{i=2}^{p} g^i \leq \frac{1-s}{s}\sum_{i=2}^{p} f^i. \qquad (4.7)$$

Then, since $s \geq \frac{k}{2k-1} > \frac{1}{2}$, we can derive

$$\sum_{i=2}^{p} f_1^i \overset{\textit{Equation } (4.6)}{\leq} \sum_{i=2}^{p} g^i \overset{\textit{Equation } (4.7)}{\leq} \frac{1-s}{s}\sum_{i=2}^{p} f^i \overset{(f^i = f_1^i + f_2^i)}{=} \frac{1-s}{s}\sum_{i=2}^{p}(f_1^i + f_2^i)$$

$$\begin{aligned}
\Leftrightarrow\quad & \left(\frac{s}{s}-\frac{1-s}{s}\right)\sum_{i=2}^{p} f_1^i \le \frac{1-s}{s}\sum_{i=2}^{p} f_2^i \\
\Leftrightarrow\quad & \frac{2s-1}{s}\sum_{i=2}^{p} f_1^i \le \frac{1-s}{s}\sum_{i=2}^{p} f_2^i \\
\Leftrightarrow\quad & \sum_{i=2}^{p} f_1^i \le \frac{s}{2s-1}\,\frac{1-s}{s}\sum_{i=2}^{p} f_2^i \\
\Leftrightarrow\quad & \sum_{i=2}^{p} f_1^i \le \frac{1-s}{2s-1}\sum_{i=2}^{p} f_2^i. \qquad (4.8)
\end{aligned}$$

Finally, we can show

$$\begin{aligned}
C_{\text{CONF-MCOKSP}} &\le \sum_{i=2}^{p}(g^i+f^i) \overset{\textit{Equation } (4.7)}{\le} \left(1+\frac{1-s}{s}\right)\sum_{i=2}^{p} f^i \\
&\overset{(f^i=f_1^i+f_2^i)}{=} \left(1+\frac{1-s}{s}\right)\sum_{i=2}^{p} f_1^i + \left(1+\frac{1-s}{s}\right)\sum_{i=2}^{p} f_2^i \\
&\overset{\textit{Equation } (4.8)}{\le} \left(1+\frac{1-s}{s}\right)\cdot\frac{1-s}{2s-1}\sum_{i=2}^{p} f_2^i + \left(1+\frac{1-s}{s}\right)\sum_{i=2}^{p} f_2^i \\
&= \left(1+\frac{1-s}{s}\right)\cdot\left(\frac{1-s}{2s-1}+1\right)\sum_{i=2}^{p} f_2^i = \frac{1}{s}\cdot\frac{s}{2s-1}\sum_{i=2}^{p} f_2^i \\
&= \frac{1}{2s-1}\sum_{i=2}^{p} f_2^i \overset{\left(f_2^i\le|\widehat{F}_2^i|\right)}{\le} \frac{1}{2s-1}\sum_{i=2}^{p}|\widehat{F}_2^i| \\
&\overset{\left(C_{\text{OPT}}\ge\sum_{i=2}^{p}|\widehat{F}_2^i|\right)}{\le} \frac{1}{2s-1}C_{\text{OPT}}.
\end{aligned}$$

■

Theorem 4.3 now follows from Lemma 4.4 and Lemma 4.5.

4.5.2.3 CONF-MCOKSP **Has a Worst-Case Competitive Ratio of at Least** $3k-2$

This section shows a mixed input for CONF-MCOKSP such that the algorithm's competitive ratio is at least $3k-2$. Thus, even though CONF-MCOKSP is not strictly-k-confident but only k-confident, the same lower bound as the one of Theorem 4.5 applies. The bound below consists of $2k-1$ general request and $k-1$ specific requests such that $s=\frac{k-1}{3k-2}$.

Theorem 4.4 Consider the any-or-one case of the multi-commodity online k-server problem on uniform metrics. The worst-case competitive ratio of CONF-MCOKSP is at least $3k-2$.

Proof. We assume CONF-MCOKSP selects servers of C by the FIFO rule. As a remark, the bound below can be adapted for other orders for the selection.

Our lower bound is constructed as the lower bound of Theorem 4.5: We consider a uniform metric with locations $v_1,\dots,v_{k+1}$. Initially, the algorithm's servers $a_1,\dots,a_k$ as well as OPT's servers $o_1,\dots,o_k$ share the same location $p(a_i)=p(o_i)=v_i$, for all $i\le k$. Rename the i's such that during the first phase (defined below), the algorithm moves its a_i in order (see the proof of Theorem 4.1). First, issue a general request on v_{k+1}. OPT solves the entire sequence by moving o_k to v_{k+1} at a cost of 1. Whenever a server a_i is moved by the algorithm, issue a general request on $v_i=p^*(a_i)$ afterward, except for a_k. After the first phase, CONF-MCOKSP covers the locations $1,\dots,k-1$ and $k+1$ in the following way: a_1 is on v_{k+1} and each a_i for $i>1$ is on v_{i-1}. CONF-MCOKSP has moved every server once, i.e., all servers are in G. Now, for each $i<k$, issue a specific

request on v_i for server i and afterward, a general request on a_i's previous location. For each such two requests, CONF-MCOKSP moves server a_i to v_i, the server a_{i+1} joins C and immediately joins G on v_{k+1}. In total, we can do $k-1$ such pairs of requests until all servers except for a_k are on their initial location and a_k is on v_{k+1}. At this point, CONF-MCOKSP's configuration matches the optimal one. The cost of CONF-MCOKSP is then $k+2(k-1)=3k-2$ while OPT has had a cost of 1. ■

4.5.3 A k-defensive Algorithm

In the following, we present DEF-MCOKSP, a k-defensive algorithm (see Section 4.4.3) achieving a worst-case competitive ratio of $2k+14$ (Theorem 4.8). DEF-MCOKSP comes close to the general lower bound of $2k-1$ (see Section 4.4.2). Similar to CONF-MCOKSP (see Section 4.5.2), the algorithm is loosely inspired by the marking approach. DEF-MCOKSP can be seen as an extended version of CONF-MCOKSP, where for each server, the algorithm acts defensively.

Recapitulate that an algorithm is *lazy* if it only moves at most one server towards a request if necessary. In Section 4.5.3.1, we present the algorithm in its non-lazy version. We introduce the non-lazy version to improve the readability and simplify the analysis. In Section 4.5.3.2, we present an analysis for the worst-case competitive ratio of $2k+14$ for the non-lazy version introduced before. After that, we discuss in Section 4.5.3.3 the limitations of the non-lazy version for $s \geq \frac{k}{2k-1}$ and present an additional analysis for the lazy version of the algorithm. As every non-lazy algorithm can be turned lazy without increasing the competitive ratio, the worst-case analysis from before still holds, and we arrive at the bound stated in Theorem 4.8.

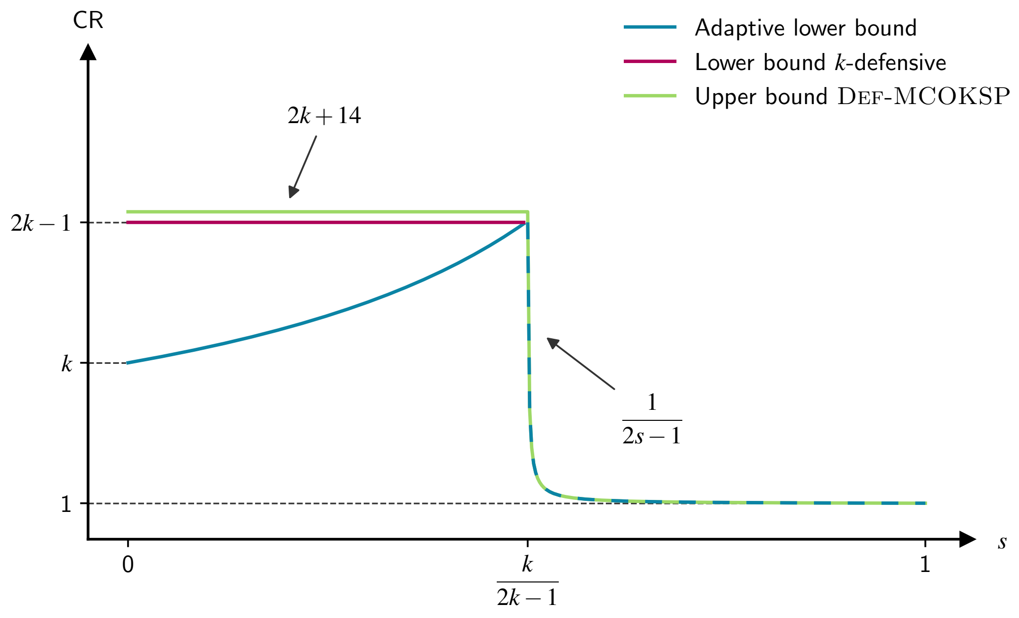

Figure 4.11: (Repetition of Figure 4.5) The plot compares the upper bound of the lazy version of DEF-MCOKSP (Theorem 4.8), the lower bound for k-defensive algorithms (Theorem 4.12) (that applies to DEF-MCOKSP), and the general adaptive lower bound (Theorem 4.2). All bounds are plotted against s, the ratio between the number of specific requests and the total number of requests. Note how the algorithm achieves a close-to-optimal competitive ratio regarding the lower bound for k-defensive algorithms. In other words, DEF-MCOKSP nearly hits the limits of algorithms of its kind. For $s \geq \frac{k}{2k-1}$, all bounds are the same.

The competitive ratio of the stated bound can be seen in Figure 4.11. As we see, the algorithm performs optimally for $s \geq \frac{k}{2k-1}$ and close to optimal concerning the lower bound for k-defensive algorithms (see Theorem 4.12) in all other cases. More specifically, DEF-MCOKSP's competitive ratio is at most an additive term of 15 apart from the optimum for $s < \frac{k}{2k-1}$.

Theorem 4.8 Consider the any-or-one case of the multi-commodity online k-server problem on uniform metrics. Let s be the ratio between the number of specific requests and the total number of requests that require a movement by the algorithm. The competitive ratio of DEF-MCOKSP is at most $2k+14$. For $s \geq \frac{k}{2k-1}$, the *lazy* version of DEF-MCOKSP has a competitive ratio of at most $\frac{1}{2s-1}$.

4.5.3.1 The Algorithm

As CONF-MCOKSP (Section 4.5.2) does, DEF-MCOKSP works in phases and is split into DEF-MCOKSP-GEN for general requests and DEF-MCOKSP-SPEC for specific ones. In addition, it utilizes a routine, DEF-MCOKSP-SELECT, for selecting servers to move during the phase. As before, DEF-MCOKSP manages several sets for each phase i. We denote this by an exponent of i that is omitted if the phase is clear from the context.

DEF-MCOKSP-SELECT: **Server for request r**

```
1: if $C_1$ is not empty then
2:     if There is $j \in C_1$ such that $j$ would not be selected to act defensively for $p^*(j)$ then
3:         return $j$
4:     else
5:         return Any server of $C_1$
6: else if $C_1$ is empty then
7:     return Any server of $C_2$
```

Sets managed by DEF-MCOKSP. At the beginning of a phase, all servers are in a candidate set C (DEF-MCOKSP-GEN, DEF-MCOKSP-SPEC Lines 3 – 4). Similar to CONF-MCOKSP we have the sets G and F. As before, F is the set of servers for which specific requests appeared so far during the phase (as a result, these servers become *frozen* in place, i.e., they do not leave their current position for the rest of the phase). G is defined as the set of servers at locations where only general requests appeared, of which no server acted defensively. In contrast to the definition for CONF-MCOKSP, we do not allow locations where only general requests appeared so far unoccupied. Thus, we do not need L. This change does not influence the worst-case bound and improves the readability. The worst-case bound is unaffected because, at every unoccupied location, a general request increases only the cost of the online algorithm and hence, happens in the worst case. As a side note, this action is exactly why the presented algorithm is non-lazy. To ensure that no unoccupied location appears, DEF-MCOKSP simulates a general request whenever a server not in C moves away from a location (DEF-MCOKSP-GEN, DEF-MCOKSP-SPEC Lines 13 – 14). In addition to the above sets, we have a set D of servers that acted defensively during the current phase. D and G are disjoint, containing all servers at locations where only general requests have appeared. Intuitively, CONF-MCOKSP treats all servers moving due to general requests the same, while DEF-MCOKSP acts defensively whenever possible.

Actions on the arrival of a request. With the same reasoning as in the description of CONF-MCOKSP, whenever a specific request for server j appears, j is never moved for the rest of the phase and thus joins F (DEF-MCOKSP-SPEC Line 7). When a general request r appears, DEF-MCOKSP first determines if there is a server $j \in C \cup G$ such that $p^*(j) = r$ (DEF-MCOKSP-GEN Lines 10 – 12). Note how, by definition, no server of $D \cup F$ can act defensively for r on a location different from its current one. If so, the algorithm acts defensively by moving j to r, and assigns j to D. As

DEF-MCOKSP-GEN: **General request r arrives in a phase i**

```
1:  if $r \notin p(G \cup D \cup F)$ then
2:      if $|G| + |D| + |F| = k$ then
3:          Start the next phase $i+1$
4:          Set $C^{i+1} \leftarrow S$ and $G^{i+1}, D^{i+1}, F^{i+1} \leftarrow \emptyset$
5:          Process $r$ again for phase $i+1$
6:      else if $|G| + |D| + |F| < k$ then
7:          if $r \notin p^*(C) \cup p^*(G)$ then
8:              Pick server $j \in C$ given by SELECT
9:              Move $j$ to $r$ and assign it to $G$ ($G_1$ if it was in $C_1$, $G_2$ else)
10:         else if $r \in p^*(C) \cup p^*(G)$ then
11:             Let $j \notin F$ be the server with $p^*(j) = r$. If there are multiple,
                select the one that was specifically requested the last.
12:             Move $j$ to $r$ ($p^*(j)$) and assign it to $D$
13:             if $j$ was in $G$ then
14:                 Simulate a general request on $j$'s previous location
```

a tiebreak, when there are multiple such servers, DEF-MCOKSP picks the one that was specifically requested the latest. If no such server exists, DEF-MCOKSP moves a server of the candidate set C to r and assign it to G (DEF-MCOKSP-GEN Lines 7 – 9). The respective server is chosen by a scheme prioritizing servers as follows: Prefer servers that did not yet move in the current phase (DEF-MCOKSP-SELECT Lines 1, 6) and those that would not act defensively as there is some other server acting defensively for the same location (DEF-MCOKSP-SELECT Lines 2 – 4). For details, see the algorithm DEF-MCOKSP-SELECT. Note here, how C and G are split into C_1 and C_2, and G_1 and G_2 to keep track of servers that acted defensively. C_1 and G_1 contain servers that never joined D during the current phase, while C_2 and G_2 contain those servers that were in D earlier. Note how DEF-MCOKSP-SELECT is ambiguous for the real choice of a server of C_1 or C_2. As in CONF-MCOKSP, any ordering on the servers will do, and we assume that the FIFO rule is used.

DEF-MCOKSP-SPEC: **Specific request r for server j arrives in phase i**

```
1:  if $r \neq p(j)$ then
2:      if $j \in F$ or ($|G| + |D| + |F| = k$ and $r \notin p(G \cup D)$) then
3:          Start the next phase $i+1$
4:          Set $C^{i+1} \leftarrow S$ and $G^{i+1}, D^{i+1}, F^{i+1} \leftarrow \emptyset$
5:          Process $r$ again for phase $i+1$
6:      else if $j \notin F$ and ($|G| + |D| + |F| < k$ or $r \in p(G \cup D)$) then
7:          Move $j$ to $r$ and assign it to $F$
8:          if There is a $s \neq j$, $s \notin F$ on $r$ then
9:              if $s$ was in $C_1 \cup G_1$ then
10:                 Assign $s$ to $C_1$
11:             else
12:                 Assign $s$ to $C_2$
13:         if $j$ was in $G \cup D$ then
14:             Simulate a general request on $j$'s previous location
15: else
16:     Assign $j$ to $F$
```

End of a phase and first phase. A phase ends when either a server of F is specifically requested at a different location (DEF-MCOKSP-SPEC Line 2) or when $|G|+|D|+|F| \leq k$ would not hold anymore when serving r (DEF-MCOKSP-GEN, DEF-MCOKSP-SPEC Line 2). In the former case, the optimal solution must move the respective server for a cost of at least 1. In the latter case, more than $k-|F|$ locations would need to be covered by $k-|F|$ servers which implies that the optimal solution has cost at least 1. As before in CONF-MCOKSP, we assume that initially, the servers are at the exact locations as in the optimal solution. Hence, $C^1, G^1, D^1 = \emptyset$ and $F^1 = S$ (the set of all servers).

4.5.3.2 The Worst-Case Analysis

Next, we show that DEF-MCOKSP has a competitive ratio of $2k+14$. The starting approach is the same as in the analysis of CONF-MCOKSP (see Section 4.5.2.2), i.e., we bound the cost of DEF-MCOKSP in each phase and use that the optimal solution OPT has cost 1 in each phase. However, the cost of DEF-MCOKSP might be higher than $2k+14$ in each phase. To reason about these higher costs, we first analyze in detail which costs DEF-MCOKSP produces. Afterward, we simplify the bound step-by-step using insights into the behavior of DEF-MCOKSP. Then, we show how to charge the simplified cost of DEF-MCOKSP in a phase to movements of OPT. For this step, we identify further movements of OPT that must happen to answer specific requests.

Before we start, observe that DEF-MCOKSP is k-defensive respecting Definition 4.2 as ensured by Lines 10-14 for serving a general request. If there is a server that can act defensively for r, then $r \in p^*(C) \cup p^*(G)$ holds. In the respective lines, DEF-MCOKSP selects a server that acts defensively for r, and we assign it to D.

On the cost of OPT in a phase. We start with a technical adaptation of Lemma 4.2 shown in Lemma 4.6. The lemma ensures that the locations of the sets managed by DEF-MCOKSP are disjoint at any time.

Lemma 4.6 At any point in time $p(G)$, $p(D)$, and $p(F)$ are disjoint.

Proof. Assume there is one location ℓ with a $j \in G$ and some $s \in D$. Both servers joined their sets due to a general request on ℓ. No matter which server joined its set first, the later one would not join its set on ℓ because there was already a server on that location.

Assume there is a location ℓ with some server $j \in G \cup D$ and some server $s \in F$. If s were first on ℓ, j would not have joined its set there because s can answer general requests. If j was first on ℓ, s moved to ℓ due to some specific request. Then the algorithm ensures that j leaves $G \cup D$ and immediately joins C. ■

Next, we show that the optimal solution has at least a cost of one for each phase except the first. Lemma 4.7 is an adaptation of Lemma 4.3 taking D into account.

Lemma 4.7 Consider any phase but the last and the first request r right after the phase ends. OPT has a cost of at least 1 during the time interval right after t_{start} until right after $t_{\text{end}}+1$.

Proof. Consider any phase that ends. At the end of the phase, it holds either that (i) $|\widehat{G}|+|\widehat{D}|+|\widehat{F}| = k$ or (ii) a server $j \in F$ is specifically requested at a location different to $p^*(j)$. Assume OPT has had no movement. Then, during the time interval, OPT has its servers at least at the locations $p(\widehat{G} \cup \widehat{D} \cup \widehat{F}) \cup \{r\}$ since at each of these locations a request appeared (at $t_{\text{start}}+1$, OPT has a server on the location of the first request). Also, OPT must have each server of F at the exact location as DEF-MCOKSP.

In the case of (i), OPT covers due to Lemma 4.6 $|\widehat{G}|+|\widehat{D}|+1 > k-|\widehat{F}|$ distinct locations using $k-|\widehat{F}|$ servers which cannot be. In the case of (ii), j is specifically requested at two different locations meaning that OPT must have placed j at two different locations. In any case, there is a contradiction. Thus, OPT has a cost of at least 1. ■

Next, we show that the optimal cost can even be higher during the phase based on the configuration of OPT after a phase. The lemma below is partly similar to Observation 4.1 but more involved to enable us a precise determination of costs of OPT during a certain time interval dependent on a phase.

Lemma 4.8 For a phase, let p_1 be the number of locations of $p(\widehat{G} \cup \widehat{D})$ where no server of OPT is located. Let p_2 be the number of servers in $\widehat{F}$ that OPT has not located where they were specifically requested. Let $p := p_1 + p_2$, then OPT has cost at least p during the time interval right after t_{start} until right after t_{end}.

Proof. Right after t_{end}, it holds that there are p_1 locations where requests appeared during the phase and where OPT has no server. Consider any such location. Since a request appeared there, OPT must have had a server on it during the phase. Since there is no server on it after the phase, OPT moved its server for a cost of 1. Consider any server contributing to p_2. During the phase, OPT must have had the respective server on the location where it is specifically requested. Since this is no longer the case after the phase, OPT moved it for a cost of 1. ■

Intuitively, OPT must have a server at all locations that appear during the phase. Hence, not covering all implies an equal movement cost. Observe that for any phase but the last, the cost is $\max\{1, p\}$ while in the last phase, the cost is p, as it ends by definition at t_{end}.

On the cost of DEF-MCOKSP in a phase. Next, we analyze the cost of DEF-MCOKSP for any but the first phase. In the first phase, all servers are frozen; therefore, DEF-MCOKSP has cost zero. Before we start, we state Lemma 4.9. It holds because our algorithm acts defensively for all servers.

Lemma 4.9 If at any time on a location ℓ, a server j joins G, there is no server s with $p^*(s) = \ell$ until the next time a server is specifically requested on ℓ.

Proof. Consider such a location and assume that there is a s with $p^*(s) = \ell$. Then, since s was not specifically requested on ℓ since j joined G on ℓ, $p^*(s) = \ell$ at the time when j joined G on ℓ. Also, s was not in F. Then, it contradicts our algorithm that j joined G on ℓ, as s would act defensively and join D on ℓ. ■

Next, we show by Lemma 4.10 how the cost of DEF-MCOKSP in any but the first phase is bounded by the sizes of the sets G, D and F. As we will see, we need to distinguish the servers more carefully than with the sets C, G, D, and F. During a phase, servers that already are in G or D can transition back to C when some other server gets frozen at their current location. Regarding a server in D, such a transition can happen only once. After it happened, the specifically requested server would always act defensively for the location, not the server that transitioned from D. To reflect this, we split C into C_1 and C_2, and we split G into G_1 and G_2. The sets C_2 and G_2 contain servers that were previously in D. When a server transitions back to C, the transition itself increases the cost of the respective server to reach its final set by 1 or 2. To capture this, we define e_x^s to be the event that server s transitions back to C and incurs an additional cost of x in the current phase. E_x is the respective set of events of the current phase. Also, among others, we introduce the following sets: F_1 is the set of servers frozen at the same location as they were specifically requested before. $F_{1a} \subseteq F_1$ is the subset of these servers for which it holds that for each $j \in F_{1a}$, there was a server $s \in D$ at the location at which j gets frozen. $F_{1b} = F_1 \setminus F_{1a}$ is the respective remaining set of servers of F_1. $F_2 = F \setminus F_1$ is the set of servers that get frozen at a location different from their last specific location.

Lemma 4.10 In any phase $i > 1$, the cost of DEF-MCOKSP is

$$c^i \leq |\widehat{G}| + 2\left(|\widehat{D}| + |\widehat{F_1}| + |\widehat{F_2}|\right) + |\widehat{F_2}| + |E_1| + 2|E_2|.$$

Proof. To prove this, we consider each time step in a phase of the algorithm. First, consider any time step in which either a general request on a location covered by DEF-MCOKSP by $G \cup D \cup F$ happens or a specific request for a frozen server at its location appears. We exclude all such time steps from the phase in the following because the algorithm takes no action in them.

An overview of all actions. From now on, for any time step, we analyze the cost and how servers move between G, D, F_1, and F_2. We denote by an arrow that a server leaves a set and joins another set as here: $j : G \to D$ (j leaves G and joins D). We call the movement of one server between sets a *transition*. Note that C is always given as all servers not in $G \cup D \cup F$.

Observe that there is no cost if a request requires no movement of a server $j \in C$. Thus, there are transitions $j : C \to G$, $j : C \to D$, $j : C \to F_1$ and $j : C \to F_2$ of cost zero. Next, we consider only the remaining cases summarized in Table 4.1.

Case	Transition / Cost	
1.a.1	$j : D \to F_1$ / 0	
1.a.2.a	$j : G \to F_1$ / 1	
1.a.2.b	$j : G \to F_1$ / 1,	$s : D \to C$ / 0
1.b.1	$j : G \to F_2$ / 0	
1.b.2.a.1	$j : C \to F_2$ / 1	
1.b.2.a.2	$j : C \to F_2$ / 1,	$s : G \to C$ / 0
1.b.2.a.3	$j : C \to F_2$ / 1,	$s : D \to C$ / 0
1.b.2.b.1	$j : G \to F_2$ / 1	
1.b.2.b.2	$j : G \to F_2$ / 1,	$s : G \to C$ / 0
1.b.2.b.3	$j : G \to F_2$ / 1,	$s : D \to C$ / 0
1.b.2.c.1	$j : D \to F_2$ / 1	
1.b.2.c.2	$j : D \to F_2$ / 1,	$s : G \to C$ / 0
1.b.2.c.3	$j : D \to F_2$ / 1,	$s : D \to C$ / 0
2.a	$j : C \to G$ / 1	
2.b	$j : G \to D$ / 1,	$s : C \to G$ / 1

Table 4.1: The table lists all possible transitions between the sets C, G, D, F_1, and F_2.

There are two kinds of time steps depending on the current request. Case (1) are time steps with a specific request, and case (2) are time steps without one.

Consider case (1). There are two kinds of such a time step. Either (1.a), a server ends up in F_1 after the time step, or (1.b) it ends up in F_2. When a general request is simulated, during the cases of (1), there can be additional transitions and cost exactly as in time steps of kind (2) below.

In the case of type (1.a), server j is specifically requested at $p^*(j)$. If (1.a.1) $j \in D$, this incurs no cost and $j : D \to F_1$. If (1.a.2) $j \in G$, we have a cost of 1 for $j : G \to F_1$. In case (1.a.2.a), there is no server of D on r, otherwise (1.a.2.b) there is $s : D \to C$ for a cost of zero.

In the case of type (1.b), server j is specifically requested at some location $r \neq p^*(j)$. If (1.b.1) $r = p(j)$, we have cost 0 and $j : G \to F_2$. Otherwise, (1.b.2) $r \neq p(j)$. If (1.b.2.a) $j \in C$, either (1.b.2.a.1) there is no server $s \in G \cup D$ on r, (1.b.2.a.2) there is a server $s \in G$ at r, or (1.b.2.a.3) there is a server $s \in D$ on r. In any case, we have cost 1 for $j : C \to F_2$. Also, in the case of (1.b.2.a.2) $s : G \to C$, or in the case of (1.b.2.a.3) $s : D \to C$. The missing cases are (1.b.2.b) $j \in G$ and (1.b.2.c) $j \in D$ combined with (1.b.2.b.1 and 1.b.2.c.1) there is no server $s \in G \cup C$ on r, (1.b.2.b.2 and 1.b.2.c.2) there is $s \in G$ on r, or (1.b.2.b.3 and 1.b.2.c.3) there is $s \in D$ on r. In any case of (1.b.2.b), we have a cost of 1 for $j : G \to F_2$. In any case of (1.b.2.c), we have a cost of 1 for $j : D \to F_2$. In the cases (1.b.2.b.2) and (1.b.2.c.2), we have $s : G \to C$, and in the cases (1.b.2.b.3) and (1.b.2.c.3), we have $s : D \to C$.

Consider a time step of kind (2). Here, two types of requests can occur: Either (2.a) a general request on some new location appears, and we have a cost of 1 with $j : C \to G$ for some j, or (2.b) a general request on a location of $p^*(j)$ for some $j \in G \cup C$ appears. In the case of (2.b), we have a cost of 1 for $j : G \to D$ and an additional cost of 1 for the server $s : C \to G$ taking j's place. Note that the server s cannot join D, as else, j would not have been in G at the same location.

Table 4.1 lists all cases mentioned above with their respective transitions and costs.

Restrictions to the actions. When looking at the cases above, we notice that any server j has only limited possibilities to be moved between the sets C, G, D, F_1, and F_2. The most obvious limitation is that *no server can ever leave* F_1 or F_2. Any server in one of these sets cannot incur further costs within the phase.

Now, observe that there are only seven cases in which a server s in G or D can end up in C again; cases (1.a.2.b), (1.b.2.a.2), (1.b.2.a.3), (1.b.2.b.2), (1.b.2.b.3), (1.b.2.c.2) and (1.b.2.c.3). In any case, the transition of s does not incur a cost. In all these cases, s was at a location where some other server j was specifically requested. Note, if $s \in D$, then after this time step, the location $p^*(s)$ will always be covered by j for the current phase. That means that while s joins C again, it can no longer join D. To reflect this, split C and G into two sets: $C = C_1 \cup C_2$ and $G = G_1 \cup G_2$. At the beginning of the phase, all servers are in C_1. In the example above, we say s joins the separate set C_2 from which it can transition to G_2, but any server in $C_2 \cup G_2$ cannot transition to D any more. Also, in all four cases, some server j must join $F_{1a} \cup F_2$. Thus, the total number of times a server can transition to C_2 is bounded by the total number of servers in $F_{1a} \cup F_2$ at the end of the phase. Besides this, note that any server transition between two sets has a cost at most 1.

Next, we consider the following graph in Figure 4.12 that depicts all possible transitions between the sets with an over-approximation of the cost of a transition as the weight of the respective edge.

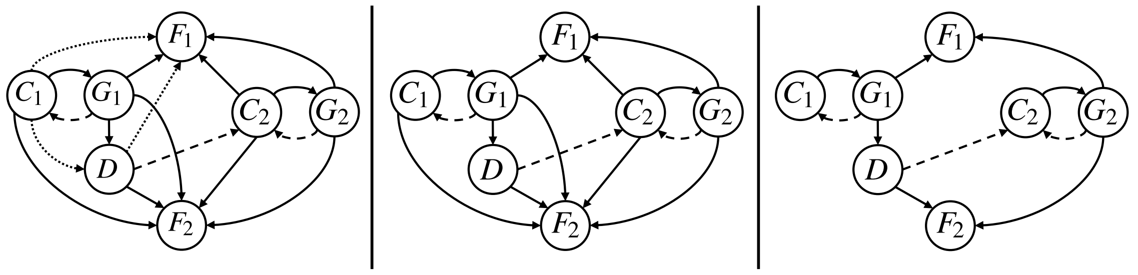

Figure 4.12: *Left:* The graph represents all possible server transitions within a phase. At the beginning of each phase, all servers are in C_1. Every node reflects one of the sets. An edge (U,V) means that a server in set U can transition to set V with a cost of, at most, the weight of the edge. Every dashed or dotted edge weighs 0, and every other edge 1. A dashed edge can only be traversed if another server joins $F_{1a} \cup F_2$ in the same time step. If server s traverses (D,C_2) and ends up in $\widehat{C_2} \cup \widehat{G_2} \cup \widehat{F_1}$, event e_2^s happens. Else, the crossing of a dashed edge implies that e_1^s happens. *Middle:* We can simplify the graph by removing the zero-cost forward edges (dotted). *Right:* The transitive reduction of the reduced graph (ignoring dashed edges) allows analyzing an upper bound on the cost of the algorithm in a phase.

Note that the cost for a sequence of transitions of a server from set U to set V can be upper bounded by finding the longest path from U to V in the graph G of Figure 4.12. Thus, to bound DEF-MCOKSP's cost, we can consider G without all zero cost forward edges. Additionally, it suffices to consider the transitive reduction of G (ignoring dashed edges). This simplification of G is G' depicted on the right of Figure 4.12.

On the cost of a phase. Next, we bound the total cost of a phase. We argue based on G'. First, consider all servers that end up in their final set without traversing a dashed edge (no event of E_1 or E_2 happens for them). For any such server j, we have the following cost: If $j \in \widehat{G_1}$, the cost is 1. If $j \in \widehat{D}$, or $j \in \widehat{F_1}$, the cost is 2, and if $j \in \widehat{F_2}$, the cost is 3. Second, consider the servers ending up in their final set traversing a dashed edge. For any such server j, the cost increases by 2 only once if $j \in \widehat{C_2} \cup \widehat{G_2} \cup \widehat{F_1}$ due to the crossing of the edge (D, C_2), i.e., an event e_2^i happens. Every other increase in the cost due to a dashed edge is at most 1 when an event in E_1 happens. Then, the total cost of the phase i can be bounded by:

$$c^i \leq |\widehat{G}| + 2|\widehat{D}| + 2|\widehat{F_1}| + 3|\widehat{F_2}| + |E_1| + 2|E_2|$$

■

Next, we get rid of the set of events in the bound. Intuitively, simplifying the bound can be achieved by a very fine-grained analysis of how events happen. We try to bound the number of events in $|\widehat{F_2}|$ as far as possible because for each server of $\widehat{F_2}$, the optimal solution must have a movement since it was lastly specifically requested. Costs bounded by $|\widehat{F_2}|$ can later be charged to these movements.

Lemma 4.11 In any phase $i > 1$, the cost of DEF-MCOKSP is

$$c^i \leq 2(|\widehat{C_2}| + |\widehat{G}| + |\widehat{D}| + |\widehat{F}|) + 5|\widehat{F_2}| + 3|\widehat{G_2}|.$$

Proof. For the analysis of the events, we need some more notation. Let $E_2(\widehat{F_2}) \subseteq E_2$ be the set of events of E_2 that are triggered by a server in $\widehat{F_2}$. Similarly, let $E_2(\widehat{F_{1a}}) \subseteq E_2$ be the set of events of E_2 triggered by a server in $\widehat{F_{1a}}$. Then, $E_2 = E_2(\widehat{F_2}) \cup E_2(\widehat{F_{1a}})$. Now, we split up $E_2(\widehat{F_{1a}})$ further: Consider a set $S \in \{\widehat{C_2}, \widehat{G_2}, \widehat{F_{1b}}\}$ (not to be confused with the set of commodities). We denote by $E_2(\widehat{F_{1a}}, S)$ the set of events of E_2 triggered by a server in $\widehat{F_{1a}}$ such that the respective server for which the event happens ends up in S. Then, $E_2(\widehat{F_{1a}}) = E_2(\widehat{F_{1a}}, \widehat{C_2}) \cup E_2(\widehat{F_{1a}}, \widehat{G_2}) \cup E_2(\widehat{F_{1a}}, \widehat{F_{1b}})$.

For each server for which an event in $E_2(\widehat{F_{1a}}, \widehat{F_{1b}})$ happens, we can find a matching server in $\widehat{G_2} \cup \widehat{F_2}$ as follows: If a server $j \in \widehat{F_{1b}}$ incurs cost of two (and an event in $E_2(\widehat{F_{1a}}, \widehat{F_{1b}})$ happens), it is in G_2 at some location ℓ just before it joins F_{1b}. Thereafter, because $\ell \neq p^*(s)$ for all s (Lemma 4.9), only a server of $\widehat{G_1}$, $\widehat{G_2}$, or $\widehat{F_2}$ can be on ℓ at the end. If there is a server $s \in \widehat{G_1}$ on ℓ, s was already in G_1 when j joined G_2, because servers of C_1 are preferred over servers of C_2. Thus, the only way that s moves on ℓ can be that it was moved back to C_1 before due to a server of F_2 (due to Lemma 4.9 the server cannot be in F_1). In total, for server j, there is a unique server $s \in \widehat{G_2} \cup \widehat{F_2}$. Therefore, $|\widehat{G_2}| + |\widehat{F_2}| \geq |E_2(\widehat{F_{1a}}, \widehat{F_{1b}})|$.

Additionally, we have the following: For any server s the event e_2^s can happen at most once, thus for $S \in \{\widehat{C_2}, \widehat{G_2}, \widehat{F_{1b}}\}$ it holds $E_2(\widehat{F_{1a}}, S) \leq |S|$. Additionally, each server of $\widehat{F_2}$ triggers at most one event, and thus $|E_1| + |E_2(\widehat{F_2})| \leq |\widehat{F_2}|$. Using both inequalities and the bound on $|E_2(\widehat{F_{1a}}, \widehat{F_{1b}})|$, we reframe the bound of Lemma 4.10:

$$\begin{aligned}
c^i &\overset{\textit{Lemma } 4.10}{\leq} |\widehat{G}| + 2|\widehat{D}| + 2|\widehat{F_1}| + 3|\widehat{F_2}| + |E_1| + 2|E_2| \\
&\overset{(\text{Def. of } E_2)}{\leq} |\widehat{G}| + 2(|\widehat{D}| + |\widehat{F}|) + |\widehat{F_2}| + |E_1| + 2(|E_2(\widehat{F_2})| + |E_2(\widehat{F_{1a}})|) \\
&\overset{\left(|E_1|+|E_2(\widehat{F_2})| \leq |\widehat{F_1}|\right)}{\leq} |\widehat{G}| + 2(|\widehat{D}| + |\widehat{F}|) + 3|\widehat{F_2}| + 2|E_2(\widehat{F_{1a}})| \\
&\overset{\left(\text{Def. of } E_2(\widehat{F_{1a}})\right)}{\leq} |\widehat{G}| + 2(|\widehat{D}| + |\widehat{F}|) + 3|\widehat{F_2}| \\
&\qquad + 2|E_2(\widehat{F_{1a}}, \widehat{C_2})| + 2|E_2(\widehat{F_{1a}}, \widehat{G_2})| + 2|E_2(\widehat{F_{1a}}, \widehat{F_{1b}})| \\
&\overset{\left(|E_2(\widehat{F_{1a}}, \widehat{C_2})| \leq |\widehat{C_2}|\right)}{\leq} |\widehat{G}| + 2(|\widehat{D}| + |\widehat{F}|) + 3|\widehat{F_2}|
\end{aligned}$$

$$\begin{aligned}
& +2\,|\widehat{C_2}|+2\,|E_2(\widehat{F_{1a}},\widehat{G_2})|+2\,|E_2(\widehat{F_{1a}},\widehat{F_{1b}})| \\
\overset{(|\widehat{G_2}|+|\widehat{F_2}|\geq|E_2(\widehat{F_{1a}},\widehat{F_{1b}})|)}{\leq}\quad & |\widehat{G}|+2\,(|\widehat{D}|+|\widehat{F}|)+3\,|\widehat{F_2}| \\
& +2\,|\widehat{C_2}|+2\,|E_2(\widehat{F_{1a}},\widehat{G_2})|+2\,|\widehat{G_2}|+2\,|\widehat{F_2}| \\
\overset{(|E_2(\widehat{F_{1a}},\widehat{G_2})|\leq|\widehat{G_2}|)}{\leq}\quad & |\widehat{G}|+2\,(|\widehat{D}|+|\widehat{F}|)+3\,|\widehat{F_2}| \\
& +2\,|\widehat{C_2}|+2\,|\widehat{G_2}|+2\,|\widehat{G_2}|+2\,|\widehat{F_2}| \\
=\quad & 2\,(|\widehat{C_2}|+|\widehat{G}|+|\widehat{D}|+|\widehat{F}|)+5\,|\widehat{F_2}|+3\,|\widehat{G_2}|.
\end{aligned}$$

■

On the charging scheme. Next, we charge the cost of DEF-MCOKSP of a phase to movement costs of OPT. From now on, we denote by the exponent i the respective object of the i-th phase. First, let us split the cost of DEF-MCOKSP of the i-th phase into the following:

$$c_1^i = 2\,(|\widehat{C_2^i}|+|\widehat{G^i}|+|\widehat{D^i}|+|\widehat{F_1^i}|+|\widehat{F_2^i}|) \qquad c_2^i = 5\,|\widehat{F_2^i}|$$
$$c_3^i = 3\,|\widehat{G_2}|$$

By Lemma 4.7, we know that OPT has at least one movement o^i for each phase i except the last one. We charge c_1^i to o^i for any phase but the last. We charge c_1^{last} of the last phase to a movement contributing to o^1, because DEF-MCOKSP has cost zero during phase 1.

Next, for any server j regarding the current phase, let t_1^i be the last time step before phase i in which j was specifically requested and let $p(j,t_1^i)$ be the location at which it was requested back then. Regarding any server $j \in \widehat{F_2^i}$, we know that j's location at the end of the current phase is different from $p(j,t_1^i)$ and thus, OPT must have moved j. We charge $c_2^i = 5\,|\widehat{F_2^i}|$ by charging a cost of 5 to OPT's last movement of j for each $j \in \widehat{F_2^i}$.

The charging of c_3^i is a bit more complicated. First, we charge additional costs to the movements of servers in $\widehat{F_2^i}$ by matching $|\widehat{F_2^i}|$ servers of $\widehat{G_2^i}$ to the respective movement of OPT. For the remaining $|\widehat{G_2^i}|-|\widehat{F_2^i}|$ servers, we show Lemma 4.12. Intuitively, for the remaining servers, we observe that our algorithm needed to move them (as they are not in C_2, and the servers of C_1 were already used). Consequently, OPT also needs some movement as it must serve the same requests. Using the servers of $\widehat{F_2}$ and Lemma 4.12, for each server of G_2^i, there is exactly one movement of OPT of a server s since s was lastly specifically requested before the phase until the end of the current phase. We charge the cost of 3 to that movement, and none of these movements receives more than one charge of the current phase. However, if a server ends up multiple times in $\widehat{C_2}\cup\widehat{G_2}$ without being specifically requested in between, the respective movement of OPT could potentially receive multiple charges. We show that in between any two times in which a server ends up in $\widehat{C_2}\cup\widehat{G_2}$, either the server is specifically requested, or the server triggering the event is in $\widehat{F_2}$ (see Lemma 4.13). In the former case, we charge the cost of 3 as explained above. In the latter case, we charge the respective cost of 3 due to the later event for server s to j.

Lemma 4.12 Assume $0 \leq x < |\widehat{G_2}|-|\widehat{F_2}|$ servers of $\widehat{C_2}\cup\widehat{G_2}$ were moved by OPT since they were lastly specifically requested until the end of a phase. Then, OPT has $|\widehat{G_2}|-|\widehat{F_2}|-x$ movements during the phase.

Proof. Assume OPT moved x servers of $\widehat{C_2}\cup\widehat{G_2}$. Then we know that $|\widehat{C_2}|+|\widehat{G_2}|-x$ servers of $\widehat{C_2}\cup\widehat{G_2}$ are the entire phase at the location at which they were lastly specifically requested. For all these servers, another server was specifically requested at their location. By the algorithm, we know that there are $|\widehat{G}|+|\widehat{D}|$ locations where only general requests appeared during the phase. Assume that at the end of the phase, OPT has $p_1 \geq 0$ many servers of $\widehat{F}$ not at the location at which they were specifically requested. Then the optimal solution can cover at the end $|\widehat{C}|+|\widehat{G}|+|\widehat{D}|-|\widehat{C_2}|-|\widehat{G_2}|+x+p_1$ locations of $\widehat{G}\cup\widehat{D}$. Therefore, the number of locations of $\widehat{G}\cup\widehat{D}$ that are not covered by OPT is $|\widehat{G_2}|-|\widehat{C_1}|-x-p_1$.

Next, we show that $|\widehat{C_1}| \leq |\widehat{F_2}|$. Observe that at the point in time where the first server j joins G_2, $C_1=\emptyset$. Else, j would not have moved because it is in C_2 and servers of C_1 are always preferred over servers of C_2. Thus, any server $j \in \widehat{C_1}$ must have been in G_1 at location ℓ before. Such a server can only join C_1 again if another server s joins F_2 on ℓ (due to Lemma 4.9, the other server cannot join F_1). Therefore, $|\widehat{C_1}| \leq |\widehat{F_2}|$. Using this in the above yields that there are at least $|\widehat{G_2}|-|\widehat{F_2}|-x-p_1$ locations of $\widehat{G}\cup\widehat{D}$ which are not covered at the end of the phase. Then, Lemma 4.8 tells us that OPT had $|\widehat{G_2}|-|\widehat{F_2}|-x-p_1+p_1$ movements during the phase and the lemma holds. ■

For Lemma 4.13, consider Figure 4.13 for an intuitive depiction.

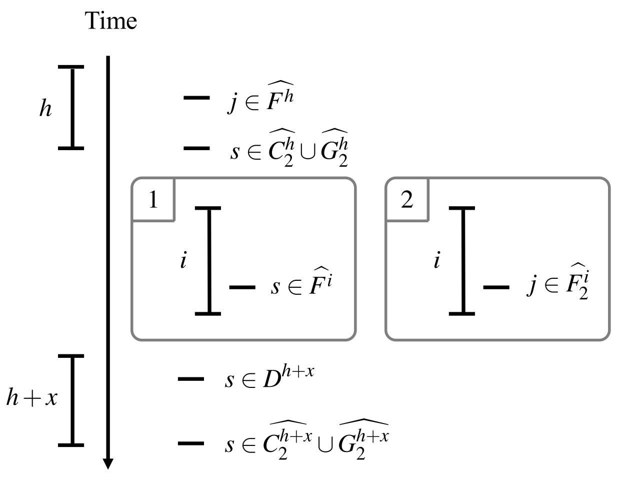

Figure 4.13: The figure depicts the statement of Lemma 4.13. Time goes from the top to the bottom. If the server s is in $\widehat{C_2^h}\cup\widehat{G_2^h}$ and $\widehat{C_2^{h+x}}\cup\widehat{G_2^{h+x}}$, then either (1) in a phase $h<i<h+x$, $s \in \widehat{F^i}$, or (2) the server j triggering the event e_2^s in h is in a phase $h<i\leq h+x$ in $\widehat{F_2^i}$. Intuitively, this must be because s must have been in D^{h+x}. This implies that s was the server which was lastly specifically requested on $p^*(s)$, but after h, j was lastly specifically requested on $p^*(s)$.

Lemma 4.13 Consider a server $s \in \widehat{C_2^h}\cup\widehat{G_2^h}$ with $s \in \widehat{C_2^{h+x}}\cup\widehat{G_2^{h+x}}$ for minimal $x>0$. Either there is a phase $h<i<h+x$ such that $s \in \widehat{F^i}$, or there there is a phase $h<i\leq h+x$ such that $j \in \widehat{F_2^i}$ for the server j that triggered the event e_2^s in phase h.

Proof. It holds that $p^*(j)=p^*(s)$ at the end of phase h and j was lastly specifically requested *after* s. Since $s \in \widehat{C_2^{h+x}}\cup\widehat{G_2^{h+x}}$, s must have been in D in phase $h+x$. For this, s must be the last server for $p^*(s)$ (with respect to phase $h+x$) that was specifically requested. If $s \in \widehat{F^i}$ for $h<i<h+x$ the lemma holds. Else, $p^*(s)$ is the same for h and $h+x$ and j must have changed its $p^*(j)$ in between. This implies $j \in \widehat{F^i}$ for $h<i\leq h+x$. ■

As briefly sketched above, we can now show that the maximum charges to a movement of OPT are limited. Consider Figure 4.14 for a depiction.

Lemma 4.14 Any movement of OPT gets charged at most $2(|\widehat{C}|+|\widehat{G}|+|\widehat{D}|+|\widehat{F}|)+14$ for some phase.

Proof. Consider a movement of OPT for server j in phase h. Let $i>h$ be the next phase in which j is specifically requested.

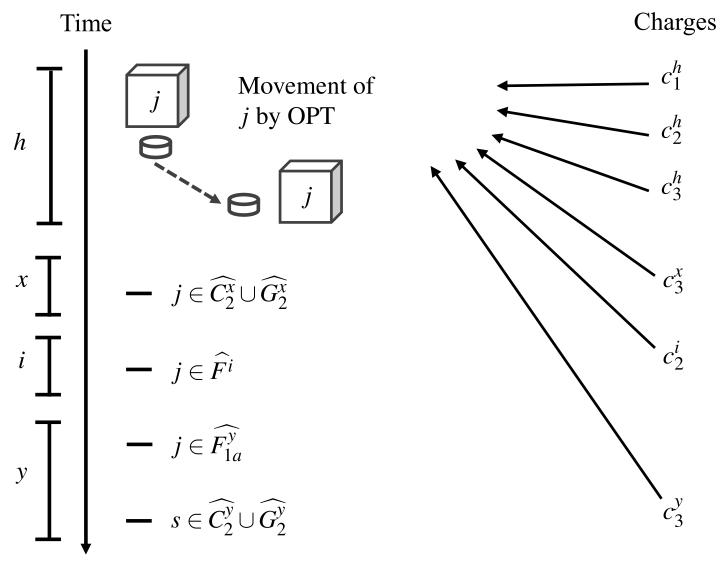
Time
Charges
Movement of j by OPT
h
x
i
y
$j \in \widehat{C_2^x} \cup \widehat{G_2^x}$
$j \in \widehat{F^i}$
$j \in \widehat{F_{1a}^y}$
$s \in \widehat{C_2^y} \cup \widehat{G_2^y}$
c_1^h
c_2^h
c_3^h
c_3^x
c_2^i
c_3^y

Figure 4.14: The figure depicts the situation analyzed in the proof of Lemma 4.14. Time goes from the top to the bottom. A bar represents a point in time for which the adjacent statement holds. For details, consider the proof of Lemma 4.14.

Due to c_1^h, the movement gets charged at most $2\,(|\widehat{C^h}|+|\widehat{G^h}|+|\widehat{D^h}|+|\widehat{F^h}|)$. If $j \in \widehat{F_2^h}$ and the movement happens before j is specifically requested, it receives a charge of at most 5 due to c_2^h. Otherwise, it receives charges of at most 5 due to c_2^i, if j is in $\widehat{F_2^i}$. From one server of $\widehat{C_2^h} \cup \widehat{G_2^h}$ (it could also be j itself), the movement can get an additional charge of 3 (see Lemma 4.12). If j ends up in $\widehat{C_2^x} \cup \widehat{G_2^x}$ for an $h \le x < i$, there could be an additional charge of 3. The latter charge can only be applied once because by Lemma 4.13 any second time j is in $\widehat{C_2} \cup \widehat{G_2}$, the respective phase must be after phase i. However, if j triggers an event e_2^s for some server s by joining F_{1a}^y (in phase $y \ge i$), and if s is in $\widehat{C_2^y} \cup \widehat{G_2^y}$, the movement receives an additional charge of 3 due to c_3^y. After that, j was specifically requested in y, and any more charges to j affect a later movement of j by OPT. ■

On the competitive ratio. Finally, we use that each server is in precisely one of the sets C, G, D, or F at any point in time, i.e., $|C|+|G|+|D|+|F| \le k$ always holds.

Theorem 4.13 — Worst-Case Competitive Ratio of DEF-MCOKSP. The competitive ratio of DEF-MCOKSP is at most $2k+14$.

Proof. Due to Lemma 4.14, the algorithms cost can be charged to OPT's cost such that each movement of OPT receives a maximum charge of $2\,(|\widehat{C}|+|\widehat{G}|+|\widehat{D}|+|\widehat{F}|)+14 \le 2k+14$. ■

4.5.3.3 Lazy vs. Non-lazy Behavior

In the following, we compare the performance of DEF-MCOKSP in its non-lazy and lazy versions. As introduced in Section 4.1, an online algorithm for the k-server problem is called lazy if it always only moves a server when necessary.

Definition 4.1 — Laziness. An algorithm for the multi-commodity online k-server problem is called *lazy* if, in each time step, it only moves one server to the location of the current request if necessary.

While CONF-MCOKSP is a lazy algorithm, DEF-MCOKSP as defined in Section 4.5.3.1 is a non-lazy algorithm. Every time a server j is moved away from a location of $G \cup D$, DEF-MCOKSP simulates a general request on j's previous location. Thereby, we ensure that all locations associated with $G \cup D$, i.e., where a general request occurred during the current phase, are covered by servers at any time. In turn, it allowed us to simplify the analysis presented in Section 4.5.3.2. In the lazy version of DEF-MCOKSP, the simulated general request does not occur. The affected lines of the pseudo code are Lines 13-14 in DEF-MCOKSP-GEN and DEF-MCOKSP-SPEC (see Section 4.5.3.1). For the lazy version, it no longer suffices to remember which servers are members of which set, but we need to track a set of *locations* where general requests appeared during the respective phase. Here, the behavior resembles CONF-MCOKSP, where we also tracked a set of locations L in addition to the sets containing servers. As in CONF-MCOKSP, the criteria for the end of a phase must depend on the location set and not on G and D.

For the non-lazy version of DEF-MCOKSP, we showed a competitive ratio of at most $2k+14$. For $s \leq \frac{k}{2k-1}$, we already know that DEF-MCOKSP cannot achieve a competitive ratio better than $2k-1$, simply because it is k-defensive (see Theorem 4.6). Therefore, we ask ourselves: "Does CONF-MCOKSP outperform CONF-MCOKSP for $s \geq \frac{k}{2k-1}$?" As it turns out, the answer is yes for the non-lazy version of DEF-MCOKSP.

Theorem 4.14 Let s be the ratio between the number of specific requests and the total number of requests that require a movement by the algorithm. For $s = \frac{k-1}{k} \approx 1$, the non-lazy version of DEF-MCOKSP has a competitive ratio of at least $2k-1$.

Proof. The set-up is the same as in the proof of Theorem 4.1, i.e., let the locations be $v_1, \dots, v_{k+1}$ with the optimal servers $o_1, \dots, o_k$ and the algorithm's servers $a_1, \dots, a_k$. Initially, $p(a_i) = p(o_i) = v_i$ for all $1 \leq i \leq k$.

The request sequence is a subset of the sequence of the proof of Theorem 4.1. Intuitively, because DEF-MCOKSP acts defensively and non-lazy, we need fewer general requests to enforce the same movement as in the above-mentioned proof. Start with a general request on v_{k+1}. As we showed when proving Theorem 4.1, there is an initial configuration of the servers such that DEF-MCOKSP moves its servers in the order of the indices. That means, DEF-MCOKSP moves a_1 to answer the request. Next, do a specific request for each server a_i with $1 \leq i \leq k-1$ at the initial location of a_i. By the specific request, DEF-MCOKSP moves a_i to v_i and places a_{i+1} on v_{k+1} to cover v_{k+1}. In total, DEF-MCOKSP has a cost of at least $2(k-1)+1 = 2k-1$. The optimal solution is to move only o_k to v_{k+1} for a cost of 1. Observe that the sequence has $k-1$ specific requests and 1 general request such that $s = \frac{k-1}{k}$. ■

Intuitively, one can see in the sequence of the proof above that the non-lazy version of DEF-MCOKSP always tries to cover the location v_{k+1} with a server even though there was only a single general request on it. The algorithm pessimistically assumes that if it does not cover that location, the adversary poses another general request on it. DEF-MCOKSP does not always have to cover the locations of general requests, but it suffices to remember them as discussed above. For the lazy version of DEF-MCOKSP, the sequence of the former proof can no longer be applied, and we can show Theorem 4.15, which is technically the same as for the third bound of Theorem 4.3. For completeness, we restate the proof here to clarify that it applies.

Theorem 4.15 Let s be the ratio between the number of specific requests and the total number of requests that require a movement by the algorithm. For $s \geq \frac{k}{2k-1}$, the lazy version of DEF-MCOKSP achieves a competitive ratio of $\frac{1}{2s-1}$.

Proof. For the proof, we consider two disjoint sets of servers composing $\widehat{F}$ for any phase. Servers in $\widehat{F}_1$ are those which were specifically requested at the exact location as they were the last time before the phase, while servers in $\widehat{F}_2$ were specifically requested at a location different from the previous one. Assume there are p phases. Denote the cost of the optimal offline solution by C_{OPT}. First, due to Lemma 4.7, OPT's cost is at least $p-1$. Every time a server gets specifically requested at a location different than the one where it was specifically requested the last time, OPT has to move the server. Therefore, secondly, OPT has a cost of at least $\sum_i |\widehat{F_2^i}|$. Observe that, by definition, the cost of DEF-MCOKSP in any phase $i > 1$ is at most $g^i + f^i$, where g^i is the number of general requests and f^i is the number of specific requests during the phase that require a movement. By definition of s, we can derive that

$$\begin{aligned} & s = \frac{\sum_i f^i}{\sum_i (g^i + f^i)} \\ \Leftrightarrow \quad & s \sum_i g^i = (1-s) \sum_i f^i \\ \Leftrightarrow \quad & \sum_{i=2}^{p} g^i = \frac{1-s}{s} \sum_{i=2}^{p} f^i. \end{aligned} \tag{4.9}$$

Let f_1^i be the number of movements due to servers in $\widehat{F_1^i}$ and f_2^i be the number of movements due to servers in $\widehat{F_2^i}$. Consider the servers of $\bigcup_i \widehat{F_1^i}$. For any such server j concerning phase i, it holds: If our algorithm has a cost for j when j joins $\widehat{F_1^i}$, then j was moved by a general request since the time it was lastly specifically requested. Therefore:

$$\sum_i f_1^i \leq \sum_i g^i. \tag{4.10}$$

Therefore, using that $s \geq \frac{k}{2k-1} > \frac{1}{2}$, we can follow that

$$\begin{aligned} & \sum_{i=2}^{p} f_1^i \overset{Equation\ (4.10)}{\leq} \sum_{i=2}^{p} g^i \overset{Equation\ (4.9)}{\leq} \frac{1-s}{s} \sum_{i=2}^{p} f^i \overset{(f^i = f_1^i + f_2^i)}{=} \frac{1-s}{s} \sum_{i=2}^{p} (f_1^i + f_2^i) \\ \Leftrightarrow \quad & \left(\frac{s}{s} - \frac{1-s}{s}\right) \sum_{i=2}^{p} f_1^i \leq \frac{1-s}{s} \sum_{i=2}^{p} f_2^i \\ \Leftrightarrow \quad & \frac{2s-1}{s} \sum_{i=2}^{p} f_1^i \leq \frac{1-s}{s} \sum_{i=2}^{p} f_2^i \\ \Leftrightarrow \quad & \sum_{i=2}^{p} f_1^i \leq \frac{s}{2s-1} \frac{1-s}{s} \sum_{i=2}^{p} f_2^i \\ \Leftrightarrow \quad & \sum_{i=2}^{p} f_1^i \leq \frac{1-s}{2s-1} \sum_{i=2}^{p} f_2^i. \end{aligned} \tag{4.11}$$

Next, the cost of DEF-MCOKSP in its lazy version is at most

$$\begin{aligned} C_{\text{DEF-MCOKSP}} \quad & \leq \quad \sum_{i=2}^{p} (g^i + f^i) \overset{Equation\ (4.9)}{\leq} \left(1 + \frac{1-s}{s}\right) \sum_{i=2}^{p} f^i \\ & \overset{(f^i = f_1^i + f_2^i)}{=} \left(1 + \frac{1-s}{s}\right) \sum_{i=2}^{p} f_1^i + \left(1 + \frac{1-s}{s}\right) \sum_{i=2}^{p} f_2^i \\ & \overset{Equation\ (4.11)}{\leq} \left(1 + \frac{1-s}{s}\right) \cdot \frac{1-s}{2s-1} \sum_{i=2}^{p} f_2^i + \left(1 + \frac{1-s}{s}\right) \sum_{i=2}^{p} f_2^i \\ & = \quad \left(1 + \frac{1-s}{s}\right) \cdot \left(\frac{1-s}{2s-1} + 1\right) \sum_{i=2}^{p} f_2^i = \frac{1}{s} \cdot \frac{s}{2s-1} \sum_{i=2}^{p} f_2^i \\ & = \quad \frac{1}{2s-1} \sum_{i=2}^{p} f_2^i \overset{\left(f_2^i \leq |\widehat{F_2^i}|\right)}{\leq} \frac{1}{2s-1} \sum_{i=2}^{p} |\widehat{F_2^i}| \\ & \overset{\left(C_{\text{OPT}} \geq \sum_{i=2}^{p} |\widehat{F_2^i}|\right)}{\leq} \frac{1}{2s-1} C_{\text{OPT}}. \end{aligned}$$

■

The previous analysis of the non-lazy version of DEF-MCOKSP presented in Section 4.5.3.2 also holds for the lazy version. So, combining Theorem 4.13 and Theorem 4.15 yields the following theorem:

Theorem 4.8 Consider the any-or-one case of the multi-commodity online k-server problem on uniform metrics. Let s be the ratio between the number of specific requests and the total number of requests that require a movement by the algorithm. The competitive ratio of DEF-MCOKSP is at most $2k+14$. For $s \geq \frac{k}{2k-1}$, the *lazy* version of DEF-MCOKSP has a competitive ratio of at most $\frac{1}{2s-1}$.

4.5.4 A Mixed Algorithm

In Section 4.5.2, we presented a k-confident algorithm, while Section 4.5.3 introduced a k-defensive one. By their nature of acting confident or defensive, both algorithms have advantages and disadvantages when compared to each other, as discussed in Section 4.4.3. Simply put, the confident algorithm outperforms the defensive one on pure general inputs, while the defensive one outperforms the confident one on mixed inputs. However, both are rather strict, acting confidently/defensively for all servers. One might ask if we can formulate an algorithm that acts for *some* servers confidently and for the rest defensively. We answer the question positively by showing how we can combine CONF-MCOKSP and DEF-MCOKSP.

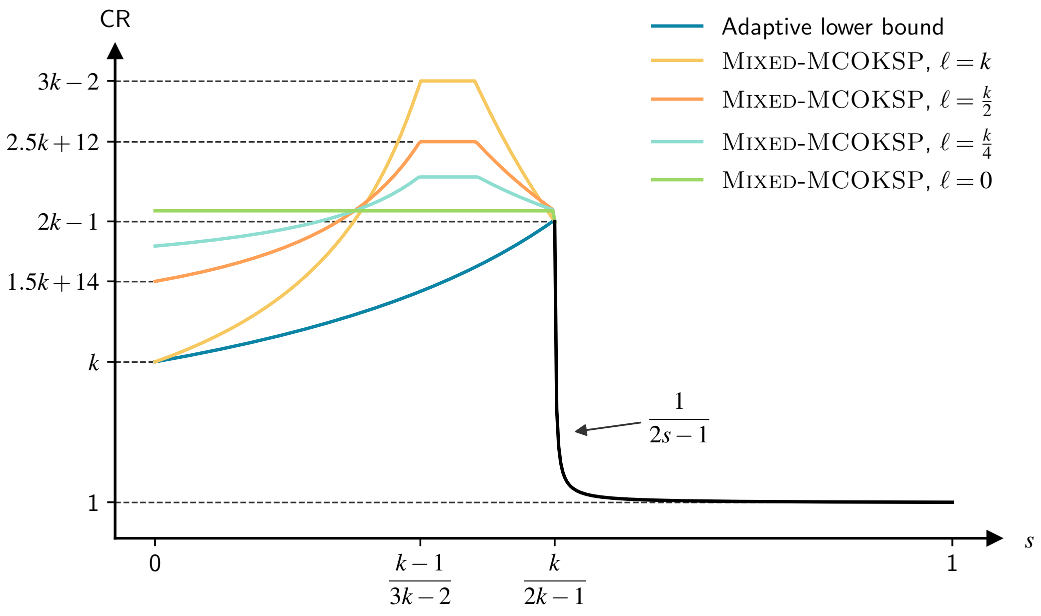

Figure 4.15: (Repetition of Figure 4.6) The figure plots the upper bounds of MIXED-MCOKSP for $\ell = 0$, $\ell = k/4$, $\ell = k/2$, $\ell = k$, and the adaptive lower bound against s, the ratio of the number of specific requests and the total number of requests. Note how the parameter ℓ of MIXED-MCOKSP demonstrates the trade-off in the competitive ratio. For $\ell = k$, the algorithm is CONF-MCOKSP. The more ℓ decreases, the more the competitive ratio in the worst case at $s = \frac{k-1}{3k-2}$ improves, and the competitive ratio for smaller s worsens. For $\ell = 0$, the algorithm is DEF-MCOKSP. For any choice of ℓ, the competitive ratio is tight for $s \geq \frac{k}{2k-1}$.

The algorithm MIXED-MCOKSP**.** Observe that DEF-MCOKSP in its lazy version (introduced in Section 4.5.3.3) can be seen as an extension of CONF-MCOKSP with the additional property that we act defensively for each server. More specifically, both algorithms only differ in the *per server* decision of whether to act confidently or defensively. Thus, a hybrid algorithm can be formulated by letting ℓ of the server act according to CONF-MCOKSP and the remaining $k-\ell$ servers according to DEF-MCOKSP. The resulting algorithm – MIXED-MCOKSP – is then ℓ-confident and $(k-\ell)$-defensive. We can mix the analysis of CONF-MCOKSP and DEF-MCOKSP as both operate in phases and end up with the following result.

Theorem 4.9 Consider the any-or-one case of the multi-commodity online k-server problem on uniform metrics. Let s be the ratio between the number of specific requests and the total number of requests that require a movement by the algorithm. MIXED-MCOKSP achieves a competitive ratio of at most $\left(\min\{\ell + \frac{2s}{1-2s}\ell, 3\ell-2, 1+2\frac{1-s}{s}\ell\} + 2(k-\ell)+14\right)$ for $s < \frac{k}{2k-1}$ and at most $\frac{1}{2s-1}$ for $s \geq \frac{k}{2k-1}$.

Proof. We apply the analysis of CONF-MCOKSP for the ℓ servers that act confidently (Theorem 4.3). For the remaining $(k-\ell)$ servers, we apply the analysis of the lazy version of DEF-MCOKSP (Theorem 4.8). ■

One can see the influences of CONF-MCOKSP and DEF-MCOKSP on the competitive ratio of MIXED-MCOKSP in Theorem 4.9. For a better depiction, consider the plot for different values of ℓ in Figure 4.15. Intuitively, MIXED-MCOKSP allows a fine-grained tuning of the performance trade-off between pure general and mixed inputs parameterized by ℓ.

4.6 Beyond Uniform Metrics

The following section presents our first results regarding the any-or-one case for the MCOKSP on non-uniform metrics. We first present the general difficulty that arises for non-uniform metrics. Afterward, we adapt the double coverage algorithm for the real line.

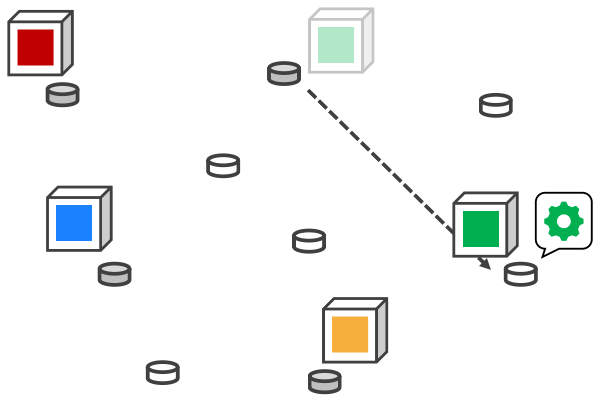

Figure 4.16: The main difficulty for the design of algorithms for non-uniform metrics is the following. When one server is forced away from its location (green), the algorithm has the uncertainty of not knowing which of the shaded locations observed in the past is not occupied by the optimal solution.

In general, one main difficulty for online algorithms for the MCOKSP is the following. Consider Figure 4.16 for a depiction. Assume the k servers cover k different locations. If now one server, say server *green*, is forced to another location by a specific request, the algorithm learns the optimal location of server *green*. However, it cannot be sure which of the previously known k locations should not be occupied by a server because the location from which *green* moved away could optimally be covered by another server or not. Therefore, an algorithm should not simply forget about *green*'s past location because it could be that one of the other servers should be there. However, it does not know which other servers could be incorrectly placed. Intuitively, a promising approach to deal with the situation is balancing all remaining servers between the formerly covered locations.

The Real Line. The following approach is based on the double coverage algorithm presented originally for the k-server problem in [34]. Recapitulate that in the double coverage algorithm, there are two cases. If the request is in between two servers, we move both closest servers with equal speed towards the request until one of them serves it. If the request is not in between two servers, we move the closest server to the request.

The algorithm DC-MCOKSP**.** Every request is treated as a general request, and the algorithm executes the double coverage algorithm. If a request is a specific request for server j, and server i was moved on the request, move i halfway towards j and then j on r.

The algorithm applies the idea mentioned at the beginning of this section. Mainly, for two servers, the situation is rather simple. The remaining server tries to balance its position between the formerly occupied ones. We can show Theorem 4.10 below, i.e., the competitive ratio of the algorithm is $3k = 6$.

Theorem 4.10 Consider the any-or-one case of the multi-commodity online k-server problem on real line metrics. DC-MCOKSP achieves a competitive ratio of 6 for $k = 2$ servers.

Proof. Let a_1 and a_2 be the two servers of DC-MCOKSP and let o_1 and o_2 be the servers of the optimal solution. The potential for $k = 2$ servers in the analysis of the double coverage algorithm [34] is given by

$$\phi = d(a_1,a_2) + 2\,(d(a_1,o_1) + d(a_2,o_2)).$$

The potential contains the distances of the algorithm's servers to each other and the distances of the algorithm's servers to the optimal servers. In the original analysis, the server types do not matter, and the servers are always numbered in increasing order of the line metric. Therefore, the second summand of ϕ is better interpreted as the weight of a minimal matching MATCH between the servers of DC-MCOKSP and the optimal servers and thus,

$$\phi = d(a_1,a_2) + 2 \cdot \text{MATCH}.$$

One can see here that the original potential does not depend on the servers' types simply because it does not need to. We extend the potential by relating the servers of DC-MCOKSP with their counterpart of the optimal solution again. Then, we end up with the following extension and generalization of ϕ:

$$\psi = \alpha \cdot d(a_1,a_2) + \beta \cdot \text{MATCH} + \gamma \cdot (d(a_1,o_1) + d(a_2,o_2))$$

In the following, we determine the values for α, β, and γ by going through all possible cases. First, consider the DC moves. Assume the algorithm moves a_1 outwards by 1. Of course, the algorithm could move a server farther. However, we can argue on the value normalized to 1 because only the relations between α, β, and γ in the potential matters. In this case, the term $d(a_1,a_2)$ increases by 1, the matching decreases by 1, and the term $d(a_1,o_1)$ may also increase up to 1. Hence, $\Delta\psi \leq \alpha - \beta + \gamma$. The cost of the algorithm (1) is canceled if

$$1 + \Delta\psi \leq 0 \Leftarrow 1 + \alpha - \beta + \gamma \leq 0 \Leftrightarrow \alpha - \beta + \gamma \leq -1. \tag{4.12}$$

Now consider the case where both servers move inwards by a distance of 1 each. The matching MATCH remains neutral as at least one optimal server lies between the algorithm's servers. With the same argument, at least one of the terms $d(a_1,o_1)$ and $d(a_2,o_2)$ decreases by 1 as well, making the change of their sum at most 0. Meanwhile, the distance $d(a_1,a_2)$ decreases by 2, giving $\Delta\psi \leq -2\alpha$. The cost of the algorithm (2) is again canceled if

$$2 + \Delta\psi \leq 0 \Leftarrow 2 - 2\alpha \leq 0 \Leftrightarrow -\alpha \leq -1. \tag{4.13}$$

Finally, we have to consider the swap move, performed if the wrong server is on the request after the double coverage move. Consider the following setup depicted in Figure 4.17: The server a_1 is at the location of the request, but server a_2 is needed. The optimal solution's server o_2 is on the request.

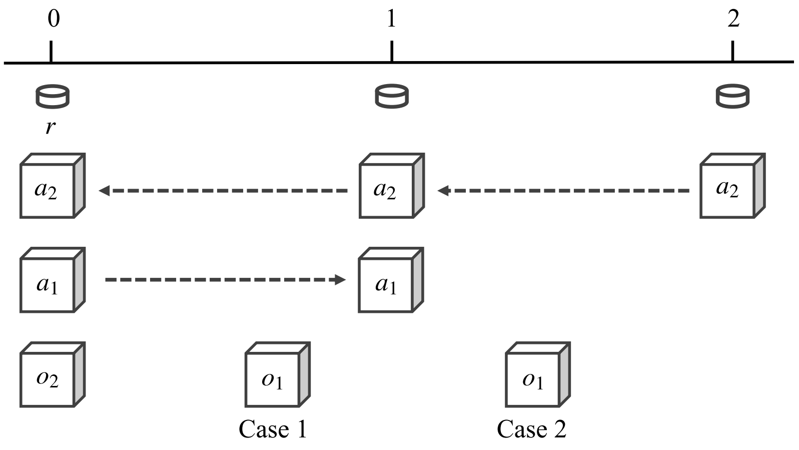

Figure 4.17: The figure depicts the setup for a swap move. r, a_1, and o_2 are at location 0, while a_2 is at location 2. The targets of a_1 and a_2 are locations 1 and 0, respectively. In Case 1, o_1 is between 0 and 1. In Case 2, o_1 is between 1 and 2.

Now, a_2 moves distance 2 onto the request while a_1 moves distance 1 in the opposite direction in which a_2 moves. We can map the locations onto a number line as follows: The request is at 0, and a_2 is at 2. During the move, a_2 moves towards 0 and a_1 towards 1. Since both servers move at equal speed, they arrive at 1 at the same time, where a_1 stops and a_2 continues to go to 0.

In any case, we can see that $d(a_1,a_2)$ decreases by 1. The rest of the change in the potential now depends on the location of o_1. First case: o_1 is on or right of location 1. This means a_1 moves towards o_1 the entire time. Since a_2 moves onto the location of o_2, we get a total decrease of 3 in the term $d(a_1,o_1)+d(a_2,o_2)$. The change in MATCH can be best observed from the perspective of o_1: First, a server moves away from it by at most 2. Then a server moves towards it by distance 1. The potential involving o_2 does not change because there is a server at o_2's location at the beginning and the end. Therefore MATCH increases by at most 1. In total, $\Delta\psi \leq -\alpha+\beta-3\gamma$. The cost of the algorithm (3) is canceled by the potential if

$$3+\Delta\psi \leq 0 \Leftarrow 3-\alpha+\beta-3\gamma \leq 0 \Leftrightarrow -\alpha+\beta-3\gamma \leq -3. \tag{4.14}$$

Second case: o_1 is to the left of 1. Now $d(a_1,o_1)$ increases by up to 1 while $d(a_2,o_2)$ again decreases by 2. The change in the matching is as follows: If o_1 is to the left of 0, then a_1 increases the distance towards both servers by 1 while a_2 decreases it by 2, making an overall decrease by 1. If o_1 is between 0 and 1, observe that when the servers a_1 and a_2 meet at 1, they switch the partners in MATCH. Hence, a_2 moves towards its matching partner the entire time. Overall we have $\Delta\psi \leq -\alpha-\beta-\gamma$ and the cost of the algorithm (3) is canceled by the potential if

$$3+\Delta\psi \leq 0 \Leftarrow 3-\alpha-\beta-\gamma \leq 0 \Leftrightarrow -\alpha-\beta-\gamma \leq -3. \tag{4.15}$$

Whenever OPT moves its servers, the potential increases by at most $(\beta+\gamma)$ times the moved distance because the first term is independent of OPT. Choosing $\alpha=1$, $\beta=4$ and $\gamma=2$ ensures that Equations (4.12) to (4.15) hold while $(\beta+\gamma)$ is minimized. Since the increase in the potential is upper bounded by $(\beta+\gamma)=6$, the competitive ratio is at most 6.

■

For larger k, the situation becomes way more difficult as one has to decide on the correct server to step in. Applying the same algorithm with the potential above does not even work for three servers anymore. It seems like a different potential is needed that encapsulates the distance of the algorithm's configuration to the optimal one.

5. Conclusion and Outlook

In the previous chapters, we have extended three fundamental problems of online resource allocation by heterogeneity. Thereby, we observed different effects of heterogeneity on online computation. First, we considered the *multi-commodity online page migration problem* (MCOPMP) in Chapter 2, where the adversary gained too much flexibility such that no online algorithm can improve upon treating each commodity separately. Second, we studied the *multi-commodity online facility location problem* (MCOFLP) in Chapter 3, where non-trivial algorithms were needed to achieve a competitive ratio close to the worst-case bound. Lastly, we introduced the *multi-commodity online k-server problem* (MCOKSP), where we have seen a trade-off between a good competitive ratio for heterogeneous instances and performing well in classical instances. Next, we conclude our results and present open problems for future research.

Multi-commodity online page migration. In Chapter 2, we extended the online page migration problem by a model with one page per commodity. Moving several commodities from a common source to a destination incurs a cost smaller than moving the commodities separately. We have seen by a lower bound that no online algorithm benefits from joint movements. In the worst case, the adversary can force a separate movement while knowing the commodity set that can be moved together. We complemented the lower bound with an asymptotically tight bound on the competitive ratio for algorithms that treat each commodity separately. One key observation we have seen is that any algorithm that wants to benefit from joint movements has to *predict* which set of commodities will be required at a future location.

The main problem of the MCOPMP is that the adversary is too strong. For future research, one could consider resource augmentation, i.e., settings where the online algorithm has more possibilities than the offline algorithm. One approach could be to allow the online algorithm a lower cost for moving pages together than the offline algorithm. However, other directions seem more interesting. One example could be to carry over ideas from the MCOKSP (see Chapter 4). Here, we think of allowing each request to present a set of commodities of which only one commodity is needed for serving. Seeing how, already in simple metrics, the extension for the k-server problem is non-trivial to solve, probably more insight for the MCOKSP is advantageous for such extension of the MCOPMP. Of course, all future research directions of the MCOKSP now carry over to such an extended model of the MCOPMP.

Multi-commodity online facility location. In Chapter 3, we presented our generalization of the facility location problem to the MCOFLP. Here, an algorithm must determine the commodity set to offer whenever a facility is constructed. The commodity set thereby influences the construction cost. Arriving requests present a commodity set and must be connected to a set of facilities jointly offering the required ones. On the one hand, we provided a lower bound showing that the commodity set S influences the competitive ratio by $\sqrt{|S|}$. The lower bound holds against randomized and deterministic algorithms. Interestingly, an algorithm aiming at getting close to the lower bound requires – similar to the MCOPMP – a *prediction* on which commodity set will be required at a location in the future. In contrast to the MCOPMP, such guessing is worthwhile in the MCOFLP. We showed the latter by presenting two algorithms – a randomized and a deterministic one. Both our algorithms are framed in a unifying way and achieve competitive ratios that depend directly on a parameter of the construction cost function (and only indirectly on $|S|$). The key idea here was to introduce the notion of a h-dividable cost function (see Definition 3.1). Based on the definition, the algorithms consider only a fixed set of possible commodity sets to offer. Thus, the decision of which commodity set to offer is facilitated while the cost an algorithm accumulates for considering multiple ones is limited. h-dividable cost functions allow analyzing the competitive ratio by considering the concrete cost function directly without having to prove further bounds on the algorithms. Consequently, for several classes of cost functions, we derived values for h and showed that, in many cases, the dependence on $|S|$ is asymptotically optimal. In some cases, the dependence even is smaller than $\sqrt{|S|}$, underlining the significance of the construction cost function for the achievable competitive ratio. Further, we designed a construction cost function that is not $\mathcal{O}(\sqrt{|S|})$-dividable. The function exploits the influence of the location on the construction cost. If such an exploit is no longer possible, i.e., the construction cost function is independent of the location, we believe that $h \in \mathcal{O}(\sqrt{|S|})$ always holds. Finally, we presented how our deterministic algorithm can be extended to solve models where facilities are not open forever but must be leased over time.

Regarding future work, dealing with functions that are not $\mathcal{O}(\sqrt{|S|})$-dividable remains challenging. We believe that techniques vastly different from our approaches are necessary. Besides, the model of the MCOFLP offers many potential extensions. Some particular examples are *commodities with conflicts*, *requests with a choice*, and *minimal connection costs*.

By *commodities with conflicts*, we consider a model where some commodities may conflict with others. The existing conflicts are then part of the input, such as a graph where the nodes are commodities, and an edge represents a conflict. For an application, assume two services, one storing sensible client data and another offering complex computations. Offering both services in a virtual machine might not be allowed when the computation service offers too much functionality so that the sensible data might leak. One can model conflicts, for example, by disallowing conflicting commodities to be instantiated together. Another approach could be to enforce a minimum distance in the metric between conflicting commodities. Especially the latter requires a more flexible way of handling where and when facilities are open. Otherwise, an adversary can always enforce that the algorithm places some commodity somewhere and cannot place a conflicting (required) one at the same location for a long time. Here, the second leasing model we presented in Section 3.6 can be applied as it allows an algorithm to shut down facilities anytime.

By *requests with a choice*, we mean a model where each request presents a set of commodities for which only one (or a subset) is required to serve it. Such a model is closely related to the MCOKSP where requests are given the same flexibility. It captures situations where a request might not need all specified services but is satisfied with any of the presented ones as they all offer similar functionality. In the case of the MCOFLP, the resulting flexibility makes the problem significantly more difficult. Assume a request only requires one of the presented commodities. Then an adversary can easily fool an online algorithm by repeatedly requesting all commodities except those offered at a location. As a result, the lower bound linearly depends on $|S|$ in the general case. Future work might consider restricted cases where not all request patterns are allowed.

In the case of *minimal connection costs*, we imagine a model where the serving cost is not the sum over all distances between a request and each connected facility. Consider a request r that is connected to x facilities at $m \in M$. Then, the serving cost for r is x times the distance between r and m. In practice, the cost is probably lower when a request communicates with the same location multiple times. Further, when a request communicates with two facilities, instead of contacting them directly, the communication might be routed over one path from r over the closer facility to the farther one. Implementing such alternative connection cost models leaves interesting future work. Especially in the latter case, the serving cost could be interpreted as the weight of a minimal Steiner tree (see [38] for an overview of Steiner trees) connecting a request and all required facilities.

Multi-commodity online k-server. In Chapter 4, we extended the k-server problem by commodities to the MCOKSP. Here, each server offers one commodity. A request presents a commodity set of which one is required to serve it. We have seen that the lower bound increases in the uniform metric for a special case. The case we considered is the any-or-one case, where a request can choose to be answered by any server or by a specific one. Introducing specific requests, in general, raises the lower bound from k (classical k-server problem instances) to $2k-1$. We established a parameterization in the number of specific requests. Based thereon, we presented a parameterized lower bound showing how increasing the number of specific requests increases the competitive ratio until it rapidly shrinks to a constant after hitting $2k-1$. One behavioral rule – *acting defensively* – significantly influences the achievable competitive ratio. We generalized our lower bounds regarding the behavior. Furthermore, we showed that following it allows approaching the worst-case lower bound while increasing the competitive ratio on instances with few specific requests and vice versa. Further, we presented two algorithms (that can be combined) incorporating the behavior (and avoiding it) and presented respective upper bounds for the problem. The key result we observed is a trade-off between performing well when only a few specific requests arrive and when many arrive. We found that no algorithm can approach the general lower bound in any case. Further, we presented an algorithm for line metrics achieving a competitive ratio of $3k$ for $k = 2$.

For future work, we believe that a similar trade-off regarding the any-or-one case can also be observed for randomized algorithms for the problem. Randomization reduces the competitive ratio, so the trade-off is probably smaller, but it should still be there. Further, researching a similar effect for non-uniform metrics seems interesting. Our algorithm for the real line achieves a competitive ratio of 6 for $k = 2$. For $k > 2$, designing an algorithm with a provably good competitive ratio seems challenging. We believe a solution for $k = 3$ and general k for the real line is an important step forward. A solution for k servers on the real line could probably be extended to tree metrics allowing an algorithm with a bounded competitive ratio for general metrics using hierarchically separated trees [8]. Regarding general metrics, an adaption of the work function algorithm is tempting. However, several definitions and techniques used in the original analysis are undefined and broken in the MCOKSP. Fixing them at least seems not trivial, and an algorithm for the MCOKSP for general metrics remains an open problem.

Besides the any-or-one case, the problem gets much more difficult. The results by Chrobak et al. [33] show that much work remains open, even when restricting request patterns, e.g., to the laminar case or the hierarchical case. For many cases, such as the laminar case, not even better lower bounds than $2k-1$ are known, indicating that the any-or-one case may be among the most significant ones. Also, the case of non-uniform and general metrics poses a major open work.

Further open problems. Various current trends in online algorithms can also be applied to our models. For example, one could introduce a delay allowing an online algorithm to postpone the serving of a request for a cost or even reject requests entirely as in [4, 19]. An interesting direction that holds much potential is *online algorithms with predictions*. Motivated by the growing power of machine learning techniques, the main idea is to supply an online algorithm with a prediction on the unknown input. If the prediction is good, the competitive ratio should improve, while in the worst case, it should be bounded as before. We see potential gains when having a prediction including commodities. In the case of the MCOFLP, a prediction on which commodities will be requested may help an online algorithm to reduce its dependence on $|S|$ in the competitive ratio. Since the major difficulty for the algorithm is guessing the correct commodity set to offer, a good prediction probably helps a lot. Besides, we see a high potential for applying predictions in the MCOKSP. We have seen that the achievable competitive ratio is limited by a decision on how the algorithm should behave, which might be good when the number of specific requests is small but bad when it is high and vice versa. A prediction on the ratio s we defined in Chapter 4 could help to decide which behavior to apply and improve the competitive ratio when considering the general parameterized lower bound.

We believe that, ultimately, the variety of models in the area of online resource allocation allows for a solid foundation for a deep understanding of the general scenario we introduced in the introduction in Chapter 1. Such understanding would greatly benefit the design of efficient and practical algorithms for the age of cloud computing. Hopefully, the results we presented here provide a step forward.

Bibliography

[1] Sebastian Abshoff, Peter Kling, Christine Markarian, Friedhelm Meyer auf der Heide, and Peter Pietrzyk. Towards the price of leasing online. *Journal of Combinatorial Optimization (JOCO)*, 32(4):1197–1216, November 2016.

[2] Susanne Albers. Online algorithms: A survey. *Mathematical Programming*, 97(1):3–26, July 2003.

[3] Barbara M. Anthony and Anupam Gupta. Infrastructure Leasing Problems. In *Integer Programming and Combinatorial Optimization (IPCO)*, Lecture Notes in Computer Science, pages 424–438. Springer, 2007.

[4] Yossi Azar, Arun Ganesh, Rong Ge, and Debmalya Panigrahi. Online Service with Delay. *ACM Transactions on Algorithms*, 17(3):23:1–23:31, July 2021.

[5] Nikhil Bansa, Marek Eliáš, Grigorios Koumoutsos, and Jesper Nederlof. Competitive Algorithms for Generalized k-Server in Uniform Metrics. In *Proceedings of the 2018 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 992–1001, January 2018.

[6] Nikhil Bansal, Marek Eliáš, Łukasz Jeż, Grigorios Koumoutsos, and Kirk Pruhs. Tight Bounds for Double Coverage Against Weak Adversaries. *Theory of Computing Systems (TOCS)*, 62(2):349–365, February 2018.

[7] Nikhil Bansal, Marek Eliéš, Łukasz Jeż, and Grigorios Koumoutsos. The (h,k)-Server Problem on Bounded Depth Trees. *ACM Transactions on Algorithms*, 15(2):28:1–28:26, February 2019.

[8] Yair Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *Proceedings of the 37th Conference on Foundations of Computer Science (FOCS)*, pages 184–193, October 1996.

[9] Yair Bartal, Avrim Blum, Carl Burch, and Andrew Tomkins. A polylog(n)-competitive algorithm for metrical task systems. In *Proceedings of the 29th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 711–719, May 1997.

[10] Yair Bartal, Béla Bollobas, and Manor Mendel. A Ramsey-type theorem for metric spaces and its applications for metrical task systems and related problems. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 396–405, October 2001.

[11] Yair Bartal, Moses Charikar, and Piotr Indyk. On page migration and other relaxed task systems. *Theoretical Computer Science (TCS)*, 268(1):43–66, October 2001.

[12] Yair Bartal and Elias Koutsoupias. On the competitive ratio of the work function algorithm for the k-server problem. *Theoretical Computer Science (TCS)*, 324(2):337–345, September 2004.

[13] Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, Guido Schäfer, and Tjark Vredeveld. Average case and smoothed competitive analysis of the multi-level feedback algorithm. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 462–471, October 2003.

[14] Laszlo A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Systems Journal*, 5(2):78–101, 1966.

[15] Shai Ben-David, Allan Borodin, Richard M. Karp, Gábor Tardos, and Avi Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11(1):2–14, January 1994.

[16] Marcin Bienkowski. Migrating and replicating data in networks. *Computer Science - Research and Development*, 27(3):169–179, August 2012.

[17] Marcin Bienkowski, Jarosław Byrka, Christian Coester, and Łukasz Jeż. Unbounded lower bound for k-server against weak adversaries. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1165–1169, June 2020.

[18] Marcin Bienkowski, Jarosław Byrka, and Marcin Mucha. Dynamic Beats Fixed: On Phase-based Algorithms for File Migration. *ACM Transactions on Algorithms*, 15(4):46:1–46:21, July 2019.

[19] E. Bittner, Csanád Imreh, and Judit Nagy-György. The online k -server problem with rejection. *Discrete Optimization*, 13:1–15, August 2014.

[20] David Black and Daniel Sleator. Competitive Algorithms for Replication and Migration Problems. Technical Report CMU-CS-89-201, Department of Computer Science, Carnegie-Mellon University, January 1989.

[21] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, February 2005.

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

[23] Mark Brehob, Richard Enbody, Eric Torng, and Stephen Wagner. On-line Restricted Caching. *Journal of Scheduling*, 6(2):149–166, March 2003.

[24] Sébastien Bubeck, Christian Coester, and Yuval Rabani. The Randomized k-Server Conjecture is False!, November 2022.

[25] Sébastien Bubeck, Michael B. Cohen, James R. Lee, and Yin Tat Lee. Metrical Task Systems on Trees via Mirror Descent and Unfair Gluing. *SIAM Journal on Computing*, 50(3):909–923, January 2021.

[26] Sébastien Bubeck, Michael B. Cohen, James R. Lee, Yin Tat Lee, and Aleksander Mądry. K-server via multiscale entropic regularization. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 3–16, June 2018.

[27] Niv Buchbinder, Shahar Chen, and Joseph (Seffi) Naor. Competitive Algorithms for Restricted Caching and Matroid Caching. In *Proceedings of the 22nd Annual European Symposium on Algorithms (ESA)*, pages 209–221, 2014.

[28] Niv Buchbinder, Anupam Gupta, Marco Molinaro, and Joseph (Seffi) Naor. K-servers with a smile: Online algorithms via projections. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 98–116, January 2019.

[29] Niv Buchbinder and Joseph (Seffi) Naor. The Design of Competitive Online Algorithms via a Primal–Dual Approach. *Foundations and Trends® in Theoretical Computer Science*, 3(2–3):93–263, May 2009.

[30] Jaroslaw Byrka. An Optimal Bifactor Approximation Algorithm for the Metric Uncapacitated Facility Location Problem. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, Lecture Notes in Computer Science, pages 29–43. Springer, 2007.

[31] Jannik Castenow, Björn Feldkord, Till Knollmann, Manuel Malatyali, and Friedhelm Meyer auf der Heide. The Online Multi-Commodity Facility Location Problem. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 129–139, July 2020.

[32] Jannik Castenow, Björn Feldkord, Till Knollmann, Manuel Malatyali, and Friedhelm Meyer auf der Heide. The k-Server with Preferences Problem. In *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 345–356, July 2022.

[33] Marek Chrobak, Samuel Haney, Mehraneh Liaee, Debmalya Panigrahi, Rajmohan Rajaraman, Ravi Sundaram, and Neal E. Young. Online Paging with Heterogeneous Cache Slots, June 2022.

[34] Marek Chrobak, Howard J. Karloff, T. H. Payne, and Sundar Vishwanathan. New results on server problems. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 291–300, January 1990.

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

[36] Václav Chvátal. The tail of the hypergeometric distribution. *Discrete Mathematics*, 25(3):285–287, January 1979.

[37] Christian Coester and Elias Koutsoupias. The online k-taxi problem. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1136–1147, June 2019.

[38] Ding-Zhu Du, Bing Lu, Huang Ngo, and Panos M. Pardalos. Steiner tree problems. In *Encyclopedia of Optimization*, pages 2451–2464. Springer US, 2001.

[39] IBM Cloud Education. IaaS versus PaaS versus SaaS. https://www.ibm.com/cloud/learn/iaas-paas-saas, September 2, 2021, retrieved December 2022.

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

[41] Björn Feldkord. *Mobile Resource Allocation*. Doctoral Dissertation, Paderborn University, 2020.

[42] Björn Feldkord and Friedhelm Meyer auf der Heide. Online Facility Location with Mobile Facilities. In *Proceedings of the 30th Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 373–381, July 2018.

[43] 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, December 1991.

[44] Amos Fiat, Yuval Rabani, and Yiftach Ravid. Competitive k-server algorithms. *Journal of Computer and System Sciences*, 48(3):410–428, June 1994.

[45] Rudolf Fleischer, Jian Li, Shijun Tian, and Hong Zhu. Non-metric Multicommodity and Multilevel Facility Location. In *Algorithmic Aspects in Information and Management*, pages 138–148, 2006.

[46] Dimitris Fotakis. Incremental algorithms for Facility Location and k-Median. *Theoretical Computer Science (TCS)*, 361(2):275–313, September 2006.

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

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

[49] Dimitris Fotakis. Online and incremental algorithms for facility location. *ACM SIGACT News*, 42(1):97, March 2011.

[50] Gartner and Statista. Market growth forecast for public cloud services worldwide from 2011 to 2023** [Graph]. https://www.statista.com/statistics/203578/global-forecast-of-cloud-computing-services-growth/, October 31, 2022, retrieved December 2022.

[51] Sudipto Guha and Samir Khuller. Greedy strikes back: Improved facility location algorithms. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 649–657, January 1998.

[52] Samuel Mitchell Haney. *Algorithms for Networks With Uncertainty*. Doctoral Dissertation, Duke University, 2019.

[53] Godfrey H. Hardy. *A Mathematician's Apology*. Canto Classics. Cambridge University Press, 2012.

[54] Wassily Hoeffding. Probability Inequalities for sums of Bounded Random Variables. In *The Collected Works of Wassily Hoeffding*, Springer Series in Statistics, pages 409–426. Springer, 1994.

[55] Kamal Jain and Vijay V. Vazirani. Approximation algorithms for metric facility location and k-Median problems using the primal-dual schema and Lagrangian relaxation. *Journal of the ACM*, 48(2):274–296, March 2001.

[56] Holger Karl, Dennis Kundisch, Friedhelm Meyer auf der Heide, and Heike Wehrheim. A Case for a New IT Ecosystem: On-The-Fly Computing. *Business & Information Systems Engineering*, 62(6):467–481, December 2020.

[57] Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel D. Sleator. Competitive snoopy caching. *Algorithmica*, 3(1):79–119, November 1988.

[58] Jon Kleinberg and Éva Tardos. *Algorithm Design*. Pearson, pearson new internat. ed. [der] 1.ed edition, 2014.

[59] Peter Kling, Friedhelm Meyer auf der Heide, and Peter Pietrzyk. An Algorithm for Online Facility Leasing. In *Proceedings of the 19th International Colloquium on Structural Information & Communication Complexity (SIROCCO)*, pages 61–72, 2012.

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

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

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

[63] Alfred A. Kuehn and Michael J. Hamburger. A Heuristic Program for Locating Warehouses. *Management Science*, 9(4):643–666, July 1963.

[64] Harry Lang. Online Facility Location against a t-Bounded Adversary. In *Proceedings of the 2018 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1002–1014, January 2018.

[65] James R. Lee. Fusible HSTs and the Randomized k-Server Conjecture. In *Proceedings of the 59th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 438–449, October 2018.

[66] Shi Li. A 1.488 approximation algorithm for the uncapacitated facility location problem. *Information and Computation*, 222:45–58, January 2013.

[67] Mehraneh Liaee. *Algorithms for Network Resource Allocation under Adversarial Dynamics and Assignment Constraints.* Doctoral Dissertation, Northeastern University, 2022.

[68] Jyh-Han Lin and Jeffrey Scott Vitter. E-approximations with minimum packing constraint violation. In *Proceedings of the 24th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 771–782, July 1992.

[69] Alejandro López-Ortiz. Alternative Performance Measures in Online Algorithms. In *Encyclopedia of Algorithms*, pages 1–7. Springer US, 2008.

[70] Mark S. Manasse, Lyle A. McGeoch, and Daniel D. Sleator. Competitive algorithms for on-line problems. In *Proceedings of the 20th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 322–333, January 1988.

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

[72] Alan S. Manne. Plant Location Under Economies-of-Scale — Decentralization and Computation. *Management Science*, 11(2):213–235, November 1964.

[73] Christine Markarian. Online Non-metric Facility Location with Service Installation Costs. In *Proceedings of the 23rd International Conference on Enterprise Information Systems (ICEIS)*, pages 737–743, November 2022.

[74] Akira Matsubayashi. A 3 + Omega(1) Lower Bound for Page Migration. In *Proceedings of the 3rd International Symposium on Computing and Networking (CANDAR)*, pages 314–320, December 2015.

[75] Lyle A. McGeoch and Daniel D. Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(1):816–825, June 1991.

[76] Adam Meyerson. Online facility location. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 426–431, 2001.

[77] Adam Meyerson. The parking permit problem. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 274–282, October 2005.

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

[79] Camilo Ortiz-Astorquiza, Ivan Contreras, and Gilbert Laporte. Multi-level facility location problems. *European Journal of Operational Research*, 267(3):791–805, June 2018.

[80] Jignesh Patel. Restricted k-server problem. Master's thesis, Michigan State University, 2004.

[81] R. Ravi and Amitabh Sinha. Multicommodity facility location. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 342–349, January 2004.

[82] Tomislav Rudec, Alfonzo Baumgartner, and Robert Manger. A fast work function algorithm for solving the k-server problem. *Central European Journal of Operations Research*, 21(1):187–205, January 2013.

[83] Mário César San Felice, David P. Williamson, and Orlando Lee. The Online Connected Facility Location Problem. In *Proceedings of the 11th Latin American Symposium on Theoretical Informatics (LATIN)*, pages 574–585, 2014.

[84] Guido Schäfer and Naveen Sivadasan. Topology matters: Smoothed competitiveness of metrical task systems. *Theoretical Computer Science (TCS)*, 341(1):216–246, September 2005.

[85] David B. Shmoys. Approximation Algorithms for Facility Location Problems. In *Proceedings of the 3rd International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX)*, pages 27–32, 2000.

[86] David B. Shmoys, Chaitanya Swamy, and Retsef Levi. Facility Location with Service Installation Costs. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, volume 15, pages 1081–1090, 2004.

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

[88] Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM*, 51(3):385–463, May 2004.

[89] John F. Stollsteimer. A Working Model for Plant Numbers and Locations. *Journal of Farm Economics*, 45(3):631–645, 1963.

[90] Jeffery Westbrook. Randomized Algorithms for Multiprocessor Page Migration. *SIAM Journal on Computing*, 23(5):951–965, October 1994.

[91] Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (SFCS)*, pages 222–227, October 1977.

[92] N. Young. The k-server dual and loose competitiveness for paging. *Algorithmica*, 11(6):525–541, June 1994.