Local and Online Algorithms for Facility Location

Dissertation

by

Peter Pietrzyk

Faculty of Computer Science, Electrical Engineering, and Mathematics
Department of Computer Science and Heinz Nixdorf Institute
University of Paderborn, Germany

September 2013

Zusammenfassung

Diese Arbeit beschäftigt sich mit dem Facility Location Problem. Dies ist ein Optimierungsproblem, bei dem festgelegt werden muss an welchen Positionen Ressourcen zur Verfügung gestellt werden, so dass diese von Nutzern gut erreicht werden können. Es sollen dabei Kosten minimiert werden, die zum einen durch Bereitstellung von Ressourcen und zum anderen durch Verbindungskosten zwischen Nutzern und Ressourcen entstehen. Die Schwierigkeit des Problems liegt darin, dass man einerseits möglichst wenige Ressourcen zur Verfügung stellen möchte, andererseits dafür sorgen muss, dass sich Nutzer nicht all zu weit weg von Ressourcen befinden. Dies würde nämlich hohe Verbindungskosten nach sich ziehen.

Das Facility Location Problem wurde bereits sehr intensiv in vielen unterschiedlichen Varianten untersucht. In dieser Arbeit werden drei Varianten des Problems modelliert und neue Algorithmen für sie entwickelt und bezüglich ihres Approximationsfaktors und ihrer Laufzeit analysiert.

Jede dieser drei untersuchten Varianten hat einen besonderen Schwerpunkt. Bei der ersten Varianten handelt es sich um ein Online Problem, da hier die Eingabe nicht von Anfang an bekannt ist, sondern Schritt für Schritt enthüllt wird. Die Schwierigkeit hierbei besteht darin unwiderrufliche Entscheidungen treffen zu müssen ohne dabei die Zukunft zu kennen und trotzdem eine zu jeder Zeit gute Lösung angeben zu können. Der Schwerpunkt der zweiten Variante liegt auf Lokalität, die z.B. in Sensornetzwerken von großer Bedeutung ist. Hier soll eine Lösung verteilt und nur mit Hilfe von lokalen Information berechnet werden. Schließlich beschäftigt sich die dritte Variante mit einer verteilten Berechnung, bei welcher nur eine stark beschränkte Datenmenge verschickt werden darf und dabei trotzdem ein sehr guter Approximationsfaktor erreicht werden muss.

Die bei der Analyse der Approximationsfaktoren bzw. der Kompetitivität verwendeten Techniken basieren zum großen Teil auf Abschätzung der primalen Lösung mit Hilfe einer Lösung des zugehörigen dualen Problems. Für die Modellierung von Lokalität wird das weitverbreitete $\mathcal{LOCAL}$ Modell verwendet. In diesem Modell werden für die Algorithmen subpolynomielle obere Laufzeitschranken gezeigt.

Gutachter:

- **Prof. Dr. Friedhelm Meyer auf der Heide, University of Paderborn, Germany**
- **Prof. Dr. Christian Scheideler, University of Paderborn, Germany**

Contents

CHAPTER 1

Introduction

The topic of this thesis is approximation and online algorithms for an optimization problem known as *Facility Location*. This problem, or one of its many variants, arises as a subproblem in many practical applications, and is thus of significant importance in the field of Operations Research. Furthermore, it is also one of the most studied optimization problems in theoretical computer science with hundreds of research papers published during the last decades. Most of these papers introduce approximation algorithms for different variants of the problem making Facility Location a benchmark problem for various new kinds of approximation techniques.

In this thesis, we focus on the theoretical aspects of Facility Location by designing and, most importantly, analyzing approximation and online algorithms. Our algorithms deal with three distinct scenarios in which Facility Location occurs: (i) networks that are exposed to perpetual changes, (ii) wireless sensor networks with strong locality constraints, and (iii) distributed settings where the focus lies, first and foremost, on the quality of the computed approximation.

We deal with each of these three scenarios within its own dedicated chapter. Chapter 2 covers Scenario (i). It presents an online algorithm designed for a highly dynamic network where additional nodes are perpetually added. The difficulty here is that these new nodes' requests have to be handled efficiently without any knowledge of the network's future development. Scenario (ii) is considered in Chapter 3. Two distributed algorithms for wireless sensor networks are presented here. Due to the nodes' limited communication range, locality is of high importance in this scenario. Additional aspects like inaccurate measurement data, power consumption, and dynamics are also taken into account. Finally, Scenario (iii) is considered in Chapter 4. Our objective here is to distributedly compute a solution with an approximation ratio that is as close as possible to the best achievable ratio. In order to accomplish this, we allow, compared to Scenario (ii), a higher running time, but still require that the algorithm terminates in sub-linear

time. The thesis ends with Chapter 5 where possible directions of future work are presented.

In the remainder of this introductory chapter, we first give an informal explanation of the Facility Location problem in Section 1.1, followed by a formal definition of the problem in Section 1.2. Hereafter, in Section 1.3, an overview of related work is presented. This section also contains a short overview of the main techniques used to analyze Facility Location algorithms. Note that a more detailed discussion of related work dedicated to the three scenarios (and their corresponding Facility Location variants) can be found later in the chapters. Section 1.4 presents a compact description of my contribution in each of the three aforementioned scenarios (i.e., it describes the contents of Chapter 2, 3 and 4). We conclude the introduction with Section 1.5. Here, additionally to the three Facility Location variants considered in this thesis, we present several further very common modifications to Facility Location and explain their relevance to our work.

1.1 The Facility Location Problem

Assume you want to decide on which geographical positions power plants have to be constructed such that cities are provided with electricity. Your solution should be good from an economical point of view: On the one hand, you want to make sure that each city is not too far away from the nearest power plant, because the cost of constructing electricity lines increases with the distance they have to cover. But on the other hand, it is not wise to build too many power plants (due to the high costs of their construction). Thus, from an intuitive point of view, you want to build power plants on locations where the density of cities is high and connect cities to power plants that are closest to them. Especially, building a power plant close to a single remote city where it will be used by that city only, might – in some cases – not be a good idea.

Of course, the power plants and the cities are only an example to illustrate the Facility Location problem. One can also consider a computer network, where the objective is to install servers on the network's computers. Installing a server can be seen as equivalent to building a power plant and users of the network as the equivalent of cities. Here, the distance between a server and a user can be represented by the latency of the connection between them.

In general, the locations where entities can be placed are referred to as *facilities* (this can be, e.g., power plants or servers). Whenever we decide to place an entity on such a location (e.g., construct a power plant, install a server, ...), we say that the facility representing this location is *opened*. The entities that use open facilities are referred to as *clients* (e.g., cities, users of the network, ...).

1.2 Formal Definition

We are given a complete weighted bipartite graph $G = (F, C, E)$. The node sets F and C represent the two partitions of G and the set E contains the edges of G. Nodes in the set F are *facilities*, while the set C contains the *clients*. The weight of the edge between facility i and client j represents the *connection costs* and is denoted by c_{ij}. Each node $i \in F$ has a nonnegative weight f_i referred to as the *opening costs* of facility i. In order to get a feasible solution to our problem, we first have to choose a non-empty subset $F' \subseteq F$ of facilities and declare them to be *open*. Next, we have to assign each client $j \in C$ to an open facility i (i.e., $i \in F'$). This assignment is represented by the function $\phi : C \to F'$. The set of open facilities F' and the assignment ϕ have to be chosen such that sum of the facility opening costs ($\sum_{i \in F'} f_i$) and the connection costs ($\sum_{j \in C} c_{\phi(j)j}$) is minimized. Note that the assignment ϕ is directly determined by the set of open facilities (as we obviously cannot do better than connect each client to its closest open facility).

The Facility Location problem can also be formulated as an integer linear program in the following way:

Facility Location IP

$$\text{minimize} \quad \sum_{i \in F} f_i y_i + \sum_{i \in F, j \in C} c_{ij} x_{ij}$$

$$\begin{array}{lrcll} \text{subject to} & \sum_{i \in F} x_{ij} & \geq & 1 & j \in C \quad (1) \\ & y_i - x_{ij} & \geq & 0 & i \in F, j \in C \quad (2) \\ & x_{ij} & \in & \{0,1\} & i \in F, j \in C \\ & y_i & \in & \{0,1\} & i \in F \end{array}$$

The program contains two kinds of indicator variables (they can be set to either 0 or 1), namely x_{ij} and y_i. A variable y_i indicates whether facility i is open ($y_i = 1$) or closed ($y_i = 0$). The other indicator variable x_{ij} has the value 1 if the client j is connected to facility i, and 0 otherwise. The constraints of type (1) guarantee that each client is connected to at least one facility, while the constraints of type (2) make sure that a client can only be connected to an open facility. Note, that in order to keep the program as simple as possible we allow clients to be connected to more than a single facility. This does not cause any problems, since every solution can be easily improved (while retaining feasibility) by deleting all but one connection of the client. Finally, the cost function consists of two parts, where the first part represents the amount we have to pay for the opened facilities, and the second one represents the costs of connecting each client to a facility.

The Facility Location problem is sometimes defined on a complete graph (in contrast to the definition above, where a complete *bipartite* graph is used). Each node in this graph can take one of two different roles. It can become either a client or a facility. If the node becomes a facility, we have to pay opening costs associated with this node. Otherwise, the node becomes a client and we have to connect it to a node which has become a facility. Here we pay for the connection between these two nodes. This definition will be applied and explained in a more detailed way in Chapter 3.

Throughout this thesis we will consider the problem known as *metric uncapacitated Facility Location*. It is called *metric*, since the values c_{ij} are required to satisfy the triangle inequality (i.e., $\forall i, j, i', j' : c_{ij} \leq c_{ij'} + c_{j'i'} + c_{i'j}$). This means that the direct path between facilities and/or clients is always at most as long as some detour. Since arbitrary many clients can be connected to a single open facility, we refer to the problem as *uncapacitated* (see Section 1.5 for a more detailed description).

This thesis deals, for the most part, with *approximation algorithms* for this metric uncapacitated Facility Location problem. Such algorithms do not necessarily compute an optimal solution, but a solution with a cost that is guaranteed to be quite close to the optimal solution's cost. In order to measure an algorithm's quality, the concept of the *approximation factor* is used. For a minimization problem (e.g., Facility Location), we say that an algorithm has an approximation factor ρ if – for all possible problem instances – the solution it computes has a cost that is at most ρ-times higher than the optimal solution's cost. We also require that the runtime of our algorithms is polynomial in the length of the input. The reason we are interested in approximation algorithms (as opposed to exact algorithms) is due to Facility Location being an *NP*-hard problem (the Set Cover problem can be easily reduced to Facility Location).

1.3 General Related Work

First work with respect to the Facility Location problem was done in the field of Operations Research. This research, as can be found in e.g., [8, 39, 47, 63], dates back as far as 50 years. Heuristics for numerous variants of the problem were presented (i.e., no worst case analysis was performed). For a survey of more than fifty applications for Facility Location see [20]. Some examples of applications are airports, blood banks, brewery depots, computer service centers, bus stops, day-care centers, emergency medical services, electric power generating plants, fire stations, fast-food restaurants, hazardous waste disposal sites, railroad sidings, regional health facilities, satellite homing stations, schools, social service centers, solid waste collection, vehicle inspection stations, warehouses, chip manufacturing, data base management, ... and now we are out of breath.

During the last decades, the uncapacitated metric Facility Location problem was also of great interest in theoretical computer science. A lot of progress has been made concerning the running time and approximation factor of sequential algorithms solving it: Aardal et al. introduced the first polynomial time algorithm yielding a 3.16-approximation [1]. Improving this approximation factor was the topic of a multitude of research papers. For example, in [15] Chudak et al. improved the approximation factor to $(1+2/e) \approx 1.74$ which was later on improved by Byrka to a 1.5-approximation algorithm in [13]. Currently, the best known approximation that can be computed in polynomial time is 1.488 (intorduced by Li in [41]). This approximation is very close to the best known lower bound by Guha and Khuller (see [31]). They showed that under the assumption that $\mathcal{NP} \not\subseteq \mathcal{DTIME}(n^{\log(\log(n))})$, the best possible approximation factor (computed in polynomial time) is 1.463.

Although both last mentioned algorithms yield very good approximation factors, they have, due to applying LP-Rounding, high running times. Thus, the design of combinatorial algorithms (possibly with slightly worse approximation factors, but better running times) was also of interest in the past. Jain and Vazirani developed and analyze an algorithm with approximation factor 3 and a running time of $O(n^2 \log(n))$ [34]. A simplified and faster $O(n^2)$ version of this algorithm was introduced by Mettu and Plaxton [48]. Later on, Jain et al. improved the former results by presenting two algorithms in [35]: One with running time $O(m \log(m))$ and an approximation factor of 1.861 and another one with running time $O(n^3)$ and an approximation factor of 1.61 (n denoting the number of nodes and m the number of edges in the complete bipartite graph of facilities and clients). Later on, building upon [35], Mahdian et al. improved the factor to 1.52 [45].

As discussed by Shmoys in [61] there are, roughly speaking, three different categories of approximation algorithms for the Facility Location problem. The *LP-rounding algorithms* first solve a relaxed linear program for the given Facility Location variant optimally, which results in a not necessarily integral (thus possibly infeasible) solution. This solution is then modified by rounding its variables' values to integers. During this rounding procedure, it is made sure that the solution's costs are increased by a factor as small as possible (this will be the approximation factor of the algorithm). The rounding process can be both deterministic and randomized. Examples of such algorithms can be found in [42, 62]. The second category is given by *primal-dual algorithms*. These algorithms compute simultaneously an integral solution to the primal program and a dual solution to the relaxed primal's dual program. In their analysis, it is shown that the cost of the primal solution is within some factor c of the dual's. Due to the Weak Duality theorem, c is an upper bound of the algorithm's approximation factor. Prominent algorithms of this type are presented in [35, 36]. Finally, there is the category of *local search algorithms* (see, e.g., [3, 16, 38]). Here, the algorithm starts with some feasible

solution and then iteratively modifies it with small changes. The current solution is only changed if this change lowers the current costs. Once the solution cannot be improved by such small (i.e., "local") changes, the algorithms stops. A common example of such a small change is to close a single facility while opening some another in a single step. To compute an approximation factor ρ, one now has to show that the cost of each local minimum is at most ρ times larger than the optimal costs.

1.4 Structure and Content of the Thesis

As mentioned before, the thesis is composed of three parts, each dealing with different variants and aspects of the Facility Location problem. Most of the content, namely [2, 18, 37] (as listed below), was published in peer reviewed conferences. An exception are the results in [12] which, at the point of writing this thesis, were only published in the *CoRR* database [1]. The publication listed first is our strongest research result. It deals with the online aspect of Facility Location and provides the content of Chapter 2. In Chapter 3, the results of the second and third publication are described. Here, the focus is mostly on the Facility Location problem's locality aspect. Finally, the publication listed last is presented in Chapter 4 where the quality of the approximation and the message complexity play a major role.

P. Kling, F. Meyer auf der Heide, and P. Pietrzyk. An algorithm for online facility leasing. In Structural Information and Communication Complexity - 19th International Colloquium (SIROCCO), pages 61-72, 2012 (c.f. [37] and Chapter 2)

S. Abshoff, A. Cord-Landwehr, B. Degener, B. Kempkes, and P. Pietrzyk. Local approximation algorithms for the uncapacitated metric facility location problem in power-aware sensor networks. In Proceedings of the 7th International Symposium on Algorithms for Sensor Systems, Wireless Ad Hoc Networks and Autonomous Mobile Entities (ALGOSENSORS), pages 13-27, 2011 (c.f. [2] and Chapter 3)

B. Degener, B. Kempkes, and P. Pietrzyk. A local, distributed constant-factor approximation algorithm for the dynamic facility location problem. In Proceedings of the 24th IEEE International Parallel & Distributed Processing Symposium (IPDPS), pages 1-10, 2010 (c.f. [18] and Chapter 3)

[1] Computing Research Repository (http://arxiv.org/abs/1209.3868)

P. Briest, B. Degener, B. Kempkes, P. Kling, and P. Pietrzyk. A distributed approximation algorithm for the metric uncapacitated facility location problem in the congest model. CoRR, arXiv/1105.1248, 2011 (c.f. [12] and Chapter 4)

The subsections below present an overview of each of the three areas this thesis deals with. They describe the used models and the attained results.

1.4.1 Facility Location in Dynamic Networks

The major part of the results presented in Chapter 2 was published in [37] in 2012. We consider an online Facility Location problem where clients arrive over time and their demands have to be served by assigning the clients to currently open facilities. When opening a facility, we must choose which one of K different lease types is going to be used. Each lease type k has a certain lease length l_k. Opening a facility i using lease type k causes a cost of f_i^k and ensures that i is open (i.e., leased) for the next l_k time steps. Only during this time interval it is possible to connect clients to facility i. In order to be able to use i after the time interval ends, one has to lease it again, which also means that one has to pay the corresponding leasing costs again. In addition to costs for opening (i.e., leasing) facilities, we have to pay connection costs c_{ij} for assigning client j to facility i. We develop and analyze the first online algorithm for this problem that has a time-independent competitive factor.

This Facility Location variant was introduced by Nagarajan and Williamson [52] and is strongly related to both the Online Facility Location problem by Meyerson [49] and the Parking Permit problem also by Meyerson [50]. Nagarajan and Williamson gave a 3-approximation algorithm for the offline version of this problem (here it is known when and where clients will arrive beforehand) and an $O(K \log n)$-competitive algorithm for the online variant. Here, n denotes the total number of clients arriving over time. We extend their result for the online variant by removing the dependency on n (and thereby on the time). In general, our algorithm is $O(l_{\max} \log(l_{\max}))$-competitive. Here $l_{\max}$ denotes the maximum lease length. Moreover, we prove that it is $O(\log^2(l_{\max}))$-competitive for many "natural" client arrival patterns. Such patterns include, for example, situations where the number of clients arriving in each time step does not vary too much, or is non-increasing, or is polynomially bounded in $l_{\max}$.

1.4.2 Facility Location in Sensor Networks

In Chapter 3, we present two distributed, constant factor approximation algorithms for the metric Facility Location problem. They were published in [2] and build upon the work published in [18]. Both algorithms have been designed with a strong emphasis

on applicability in the area of wireless sensor networks: in order to execute them, each sensor node only requires limited local knowledge and simple computations. Also, the algorithms can cope with measurement errors and take into account that communication costs between sensor nodes do not necessarily increase linearly with the distance, but can be represented by a polynomial of higher degree. Since it cannot always be expected that sensor nodes execute algorithms in a synchronized way, our algorithms are executed in an asynchronous model (but they are still able to break symmetry that might occur when two neighboring nodes act at exactly the same time). Furthermore, they can deal with dynamic scenarios: if a node moves, the solution is updated and the update affects only nodes in the local neighborhood. Finally, the algorithms are robust in the sense that incorrect behavior of some nodes during some round will, in the end, still result in a good approximation. The first algorithm runs in expected $O(\log_{1+\varepsilon} n)$ communication rounds and yields a $\mu^4(1+4\mu^2(1+\varepsilon)^{1/p})^p$ approximation, while the second has a running time of expected $O(\log^2_{1+\varepsilon} n)$ communication rounds and an approximation factor of $\mu^4(1+2(1+\varepsilon)^{1/p})^p$. Here, $\varepsilon > 0$ is an arbitrarily small constant, p the exponent of the polynomial representing the communication costs, and μ the relative measurement error. Thus, as a simplification, if we assume that no measurement errors occur ($\mu = 1$), the connection costs are linear ($p = 1$) and ε is set to 1, we get an approximation factor of 4 respectively 6 with a runtime of $O(\log^2 n)$ respectively $O(\log n)$.

1.4.3 Facility Location in Distributed Settings

Chapter 4, the final part of this thesis, we present a randomized distributed approximation algorithm for the metric uncapacitated Facility Location problem. The algorithm is executed on a bipartite graph in the $\mathcal{CONGEST}$ model (with a restriction of $O(\log n)$ bits on the message size) yielding a $(1.861+\varepsilon)$ approximation factor, where ε is an arbitrary small positive constant. It requires $O(n^{3/4}\log^2_{1+\varepsilon}(n))$ communication rounds with high probability (n denoting the number of facilities and clients). At the time of publishing, our algorithm had, to the best of our knowledge, the best approximation factor for the Facility Location problem in a distributed setting. It is based on a greedy sequential approximation algorithm by Jain et al. presented in [35]. Note that, while the presented runtime bound of our algorithm is "with high probability", the approximation factor is not "in expectation" but always guaranteed to be $(1.861+\varepsilon)$.

The main difficulty in executing this sequential algorithm lies in dealing with situations where multiple facilities are eligible for opening, but (in order to preserve the approximation factor of the sequential algorithm) only a small subset of them can actually be opened. Thus, a significant part of our contribution is a sublinear time selection mechanism that, while increasing the approximation factor by an arbitrary small additive

term, allows us to decide which of the eligible facilities to open.

1.5 Variants of Facility Location

Various aspects of the original Facility Location problem can be modified such that multiple new and interesting problems arise. Even though each of these problems is closely linked to the original Facility Location, there are very distinct difficulties associated to each one of them. Thus, the techniques required to solve them are also very different from each other. Some of them can even be considered to be separate research areas by themselves, each one comprising many publications. This section presents several such modifications and describes their relationship to the work done in this thesis.

1.5.1 Objectives and Constraints

There are multiple variations of Facility Location where the objective is more complicated than the original problem defined in Section 1.2. They all have in common that one has to choose locations where facilities are opened (which incurs some costs) and then connect clients to these open facilities (which incurs a cost proportional to the distance between a client and the facility it is connected to). Other than that, these variants can differ significantly from each other. Here, we present some of the more prominent variants.

The *Fault Tolerant Facility Location* problem is almost the same as the original problem with the exception that each client j is not only required to be connected to one open facility but to at least $r_j \in \mathbb{N}$ such facilities. Also, the r_j facilities j is connected to are required to be distinct (otherwise the problem could be easily solved by duplicating a solution for the standard Facility Location problem). A detailed problem definition and a $O(\log(k))$ primal-dual approximation algorithm, with $k = \max_j\{r_j\}$, can be found in [33]. It is due to Jain and Vazirani, who were the first to consider this problem. Guha et al. give the first constant upper bound of 2.47 in [32] using a local-search technique. This bound is later improved by Swamy and Shmoys in [64] to 2.076. Note that the distributed algorithm presented in Chapter 4 can be extended to this fault tolerant variant in a straight forward way (increasing its approximation factor from constant to $O(\log(k))$). In the special case that for all clients the r_j values are equal, better bounds are known. Here, Jain et al. give a 1.61 approximation in [35], and Swamy and Shmoys a 1.52 approximation in [64].

If there is more than one way to open a facility (i.e., a facility can provide different kinds of services), we have a *Multi Commodity Facility Location* problem. Assume there are $m \in \mathbb{N}$ different kinds of services a facility can provide. Now, for each client

we require that the set of facilities the client is connected to provides all m different services. In the case that we allow a facility to provide all m services (paying a fixed price for each service), solving this problem is the same as solving the original Facility Location problem m times. To make the problem interesting we can, e.g., disallow a facility to provide more than one service. Another possibility (see [60] by Ravi and Sinha) is to decrease the cost of providing a service by a facility if this facility is already providing different services.

Until now we only considered Facility Location Problems where the clients have unit demand and the amount of demand a facility can cover is unbounded. In the *Capacitated Facility Location* we introduce a positive demand value d_j to each client j and a positive capacity κ_i to each facility. When client j is connected to facility i the incurred connection cost is now $d_j * c_{ij}$. We also require that the sum of the demand of clients connected to facility i is not higher than i's capacity (i.e., $\sum_{j \in \phi^{-1}(i)} d_j \leq \kappa_i$). Constant upper bounds concerning this variant can be found in [15, 38, 40, 46].

Yet another branch of research is *Mobile Facility Location*. Even though the name suggest some kind of dynamics (e.g., an online problem), this problem can be considered as static (i.e., the entire input is known right at the beginning). Here, we are given a complete graph with k nodes marked as open facilities. It is now possible to move the facilities, for a certain cost that depends on the distance, before connecting each client to its closest facility. The cost of the solution is the sum of the movement costs (incurred by displacing the facilities) and the connection costs of clients (which are proportional to the distance between a client and the facility it is connected to, and are weighted with the client's demands). Mobile Facility Location belongs to the research area of *movement problems* that was initiated by Demaine et al. in [19]. For the unweighted case, Friggstad and Salavatipour present the first approximation algorithm for Mobile Facility Location in [29]. It yields an approximation factor of 8 and is based on LP-rounding. This result was recently improved by Ahmadian et al. in [3] who developed a local search based algorithm yielding an $(3+\varepsilon)$ approximation (with $\varepsilon > 0$).

1.5.2 Models of Computation and Locality

The initially developed algorithms for the Facility Location problem, as well as for most other classical problems, are sequential (e.g., [1, 34, 35, 41]): they are executed on a single processor that has knowledge of the entire problem instance. Later on, new computation models with multiple processors working together in parallel were introduced [43, 57]. Here, not only additional processors were added, but at the same time the information each processor was provided with was limited to only a small portion of the whole input. This limit on knowledge together with the use of multiple processors was

motivated by the emergence of large computer networks. Due to the huge size of such a network and its perpetually changing properties, it is impossible for each of its members to know everything about the entire network (i.e., the entire input for the given problem).

The focus of this thesis lies to a high degree on local algorithms, which nicely reflect the requirements of large (i.e., arbitrarily scalable) networks. Local algorithms are executed in a distributed way: Each node of the network represents a single processor and its knowledge about the entire input is strongly limited (i.e., it knows only about its direct neighbors and the information they have stored). Together, all nodes have the information about the entire network. All nodes are executing the same algorithm in parallel. During the algorithm's execution, nodes can communicate with each other and thus gain additional information. Still, one can assume that at the end of the execution, each single node knows, because of the small number of communication rounds, only a tiny fraction of the entire input. The actual solution computed by a local algorithm is just the sum of all the outputs computed by each node in the network. As an example, each facility can be represented by a single node and the node's output is whether it is open or not. Other nodes represent clients and each such node's output is the information to which facility node it is connected to. Together, all the outputs represent a solution to the Facility Location problem.

The $\mathcal{LOCAL}$ model, as popularized by Linial [43] and Peleg [57], is very common in modeling locality (as described above) in computer networks. Here, in each round all nodes execute some arbitrary complex computation in parallel and then are allowed to send a single message of arbitrary size to each node they share an edge with. This process is repeated over and over until the algorithm terminates. The number of rounds until termination is achieved determines the algorithm's runtime (the processing time by each node within one round is of no interest). A generalization of this model is the $\mathcal{CONGEST}$ model, where the size of the messages that are send each round is limited. Most often the limit is set to $O(\log n)$ bits, where n is the number of nodes. This is motivated by the fact that $\Theta(\log n)$ bits are required to encode a node's ID in a network of size n. The $\mathcal{CONGEST}$ model will be used in Section 4 where a distributed primal-dual algorithms for Facility Location is presented.

A different way to model locality is to consider the weights of the edges of the given network and define locality using them. To do so, we consider an arbitrary graph G with non-negative edge weights which represents the given network. Its weights can for example represent the latencies between nodes. We use G to construct a complete weighted graph G' with the same set of nodes as G. In order to define the weights for each edge $v, w \in G'$ we compute the length of the shortest weighted path between v, w in the graph G and assign it to $\{v, w\} \in G'$. We now say that two nodes v and w are neighbors, if the weight of the edge they share in G' is smaller than some fixed value

r. This models the fact that it is unreasonable for two nodes to communicate with each other if the latency between them is too high. The weights in G' can also be interpreted as Euclidean distances. Setting the value r to 1 can be used to simulate a unit disc graph. This approach is used in Section 3 where the focus lies on sensor networks.

Recently, in [27], Fraigniaud, Korman and Peleg initiated a new branch of computational complexity theory that deals with locality. They formulate a hierarchy of complexity classes for decision problems using the $\mathcal{LOCAL}$ model and allowing the algorithms to use at most a constant number of communication rounds.

1.5.3 Dynamics

The concept of dynamics can be introduced in every one of the aforementioned Facility Location variants. By dynamics we refer to changes of the properties of our input graph that occur over time. For example, new nodes can be added to the graph or old ones can be removed. Also, the weights of the edges, the nodes, or even both can change. The main difficulty when dealing with such dynamics is that the changes to the graph are not known beforehand but are revealed piece by piece as time goes by. This means that our algorithms have to update and maintain a feasible solution of good quality for each time step without knowing the future. Each new solution is not computed from scratch (which would mean that only a sequence of static/offline problems is considered), but is strongly related to prior solutions. In this thesis we will consider such dynamics from two different angles: The classical approach of *competitive* analysis for *online* problems (see Chapter 2) and an approach that focuses on the time needed to update the current solution with as few modifications as possible (see Chapter 3).

To be able to apply competitive analysis, the input has to be presented as a data stream (i.e., in each time step a new request presents itself and has to be dealt with). In the context of Facility Location such a stream could represent clients that arrive one by one over time and are required to be connected to an open facility upon their arrival (Chapter 2 considers such a scenario). Now, in order to evaluate a given online algorithm, we compare – for each possible problem instance – the cost of the solution it produces with the optimal cost produced by an offline algorithm. An upper bound for the ratio between the costs of such online and offline solutions for all possible problem instances determines the quality of our online algorithm.

The second approach is not too much concerned with the ratio between the online and the offline solution (even though this ratio is still important), but in the process of updating the current solution such that it fits the new requirements presented by the stream's current request. Consider the following example: To a given instance of a Facility Location problem, a good solution is computed and according to it, facilities

are opened and clients are connected. Now, this instance changes in some way (e.g., the weights of some edges change) such that the quality of our current solution drops significantly. Our objective is to update the current solution (e.g., open/close facilities or reconnect clients) in such a way that the new solution is of good quality again. The question that arises now is how fast a new solution can be computed in a distributed way (using only local information) and how much the old and the new solution differ from each other (i.e., how far did the changes, which were necessary to improve the old solution, spread in the network)? In Section 3.5, we use approach to analyze the effects of dynamic on a sensor networks with respect to the Facility Location problem. A similar kind of analysis, referred to as *quiescence analysis*, was used for the construction and maintenance of graph spanners by Elkin in [21].

CHAPTER 2

The Facility Location Problem in Dynamic Networks

2.1 Introduction

Consider a company that runs a distributed service on a computer network. In order to be able to provide this service, the company has to use a subset of the network's nodes as service providers. Since the nodes in the network do not belong to the company, they first have to be acquired by it in some way. While there is no possibility to buy them (they might not be for sale, or buying them might be too expensive), they can be *leased* for limited periods of time.

In our scenario, leasing a node works quite like renting a car. First, you have to pick the car you want to rent (i.e., the nodes that are going to be used to provide the service). Next, you have to decide from which date on you want to be using the car (i.e., the point in time you want the service to be accessible). You also have to choose, right at the moment you sign the contract with the car-renting firm, for how long you want to have the car. Finally, you pay up front, and can use the car, at least during the period it is rented, as much as you like. A very important aspect here is the irrevocability of your decisions; once you have signed the contract, there is no way to get your money back (even if you decide to never use the rented car at all). There are usually many options for how long a car can be rented (a single day, two days, one week, maybe even an entire month). Of course, the options become more expensive with increasing length of the time period (you will pay more for one week than for one day). But, they also become less expensive if the cost per day ratio is considered. For example, if you want to use the car for four consecutive days, it might be cheaper to choose the "one week" option, rather than rent it for four days separately (even though you will not be using the car during the week's remaining three days).

Going back to our network scenario, we introduce, next to the service providing nodes, client nodes that want to use the service. These client nodes issue requests that can be fulfilled by service providing nodes (i.e., nodes that are currently leased by the company). The cost of satisfying such a request is proportional to the distance (i.e., latency) between the client that issues the request and the leased node that fulfills it. Similar to the original Facility Location problem, the company wants to lease nodes as seldom as possible, while making sure that currently leased nodes are not too far away from request issuing nodes. Additionally, the company has to also decide on the lease types that are used (there are, like in the car rental example, various leases of different costs and durations). The main difficulty here is the unpredictable behavior of the client nodes: It is not known beforehand which nodes, and at what point in time, will request the service. Thus, it might happen that the company buys long and expensive leases for some nodes, just to realize later on that no more requests are issued in subsequent steps in the proximity of those leased nodes. Or, the other way round, it buys short leases, just to notice later that having bought a single longer lease would have been way less expensive.

The problem described above is known as *Facility Leasing*. It is one of many infrastructure leasing problems that were considered by Anthony and Gupta in [6]. In this chapter, we state and analyze an online algorithm for this problem. Our algorithm provides a means to decide online (i.e., without knowing the future) which nodes, with which lease type, and at what point in time, should be leased. We show that the cost of the solution computed by our algorithm is not too far away from the optimal solution (i.e., a solution computed by an optimal algorithm that knows in advance when nodes will issue requests, and thus can plan accordingly). More precisely, we show that our competitive factor is polylogarithmic in the length of the longest lease type. This is a complementary result to the work of Nagarajan and Williamson in [52], who show an competitive factor that is polylogarithmic in the number of clients respectively the number of time steps.

In the remainder of this chapter's introduction, we give a formal definition of the Facility Leasing problem (see Subsection 2.1.1). Then, in Subsection 2.1.2, our results are presented. Subsection 2.1.3 covers related work that is strongly connected to the Facility Leasing problem presented in this chapter. At last, Subsection 2.1.4 presents an overview of the approach we use for our algorithm's analysis.

2.1.1 Problem Definition & Notation

This subsection presents a formal definition of the Facility Leasing problem, for which we gave an intuitive explanation in the previous subsection. Two variants of Facility Leasing exist. It can either be formulated as an online or as an offline problem. In the

online version, we have, as mentioned above, no information about the future, and thus have to deal with the arriving clients' requests in a classical online manner. The offline version is quite different. Here, all information about future client requests is known right from the start. But even with this additional knowledge, the offline version is still a challenging problem. When solving it, not only are you required to solve the original Facility Location problem, but you also have to deal with the aspect of time and the leasing of facilities.

From now on, we will refer to the *online* version of Facility Leasing as FACILITYLEASING. This will enable us to distinguish the online from the offline variant, and also allow us to talk about further variants of online Facility Leasing, like FACILITYLEASING$_1$, FACILITYLEASING$_2$ and 2-FACILITYLEASING (which we will introduce later on), in a more elegant way.

The FACILITYLEASING problem, as presented by Nagarajan and Williamson [52], is defined as follows: We are given, as is typical for Facility Location problems, a set F of m facilities and a set D of n clients. Our goal is to minimize the costs of serving the clients' demands by opening facilities (which incurs costs) and assigning each client to a nearby open facility (which incurs costs that are proportional to the distance between the client and the facility). However, in contrast to the classical Facility Location model, there is a notion of (discrete) time. Clients do not arrive all at once. Instead, at time $t \in \mathbb{N}$ a subset $D_t \subseteq D$ of clients appears and these clients have a demand for the current time t only. Note that these subsets form a partition of D, i.e., a client arrives exactly once. Often, we will use $H_q := \sum_{i=1}^{q}(|D_i| / (\sum_{j=1}^{i} |D_j|))$ to describe a sequence of arriving clients. Another difference to the classical model is that opening a facility is not simply a binary decision. Instead, in order to open a facility, one is required to determine the point in time during which the facility is going to be opened and one of the K different lease types that is to be used to open it. Each lease type k has length l_k and these lengths $l_1, l_2, \ldots, l_K$ are considered as a part of the input. We use $l_{\max} := \max_{1 \leq k \leq K}(l_k)$ to denote the maximal lease length. Consider a facility i opened at time t using lease type k. This facility is open for the l_k time steps during the interval $[t, t + l_k - 1]$ (let I_t^k denote this interval). Now, a client j arriving at time step t' can only be assigned to the facility i if i is open at j's arrival (i.e., $t' \in I_t^k$).

A solution has to assign each client to an eligible facility. Each time we open a facility $i \in F$ for the next l_k time steps using a lease type $k \in K$, a cost of f_i^k is charged. Moreover, assigning a client $j \in D$ to a facility $i \in F$ incurs a connection cost of c_{ij} (independent of the lease type used to open i). These connection costs can be assumed to correspond to the distance between facility i and client j. Clients and facilities reside in a metric space, such that the connection costs satisfy the following triangle inequality: $\forall\, i, i' \in F, j, j' \in D:\ c_{i'j} \leq c_{ij} + c_{ij'} + c_{i'j'}$.

ILP formulation of FACILITYLEASING

$$\min \quad \sum_{(i,k,t)\in F} f_i^k y_{ikt} + \sum_{(j,t)\in D} \sum_{(i,k,t')\in F: t\in I_{t'}^k} c_{ij} x_{ikt',jt}$$

$$\begin{aligned}
\text{s.t.} \quad \sum_{(i,k,t')\in F: t\in I_{t'}^k} x_{ikt',jt} &\geq 1 && (j,t)\in D \\
y_{ikt'} - x_{ikt',jt} &\geq 0 && (i,k,t')\in F, (j,t)\in D \\
x_{ikt',jt} &\in \{0,1\} && (i,k,t')\in F, (j,t)\in D \\
y_{ikt'} &\in \{0,1\} && (i,k,t')\in F
\end{aligned}$$

Figure 2.1: The integer linear program describing the online Facility Leasing problem.

Figure 2.1 shows the linear programming (LP) formulation of this problem (while 2.2 shows its dual). To ease the formulation of the LP and the analysis, we sometimes abuse the notation and write $(j,t) \in D$ for a client j appearing at time t. Similarly, we write $(i,k,t) \in F$ and refer to this triple as a single facility instead of a (potential) facility opened at time t using lease type k. The first sum in the objective function for FACILITYLEASING represents the costs incurred by opening facilities. Here, the indicator variable y_{ikt} tells us whether the facility i is opened at time step t with lease type k. The remaining part of the objective function represents the costs incurred by connecting each client to a facility. The variable $x_{ikt',jt}$ indicates whether a client j that arrived at time step t is connected to facility i opened at time step t' with lease type k. While the first primal constraint guarantees that each client is connected to at least one facility, the second makes sure that each client is only connected to a facility that is open during the time step of the clients arrival.

Given an instance I of the just described problem and a corresponding solution S, we refer to the total cost of the solution by $\text{cost}(S)$. We are mainly interested in the online version of the problem (FACILITYLEASING), i.e., when a client j appears at time t we have to connect it to an open facility without any knowledge of the future. Especially, we do not know whether in the following time steps more clients appear in j's proximity (which would favor a longer lease type) or not (which would encourage a shorter lease type). We measure the quality of algorithms by their *competitiveness*: Let $A(I)$ denote the solution of an algorithm A for problem instance I and $O(I)$ an optimal solution to I. Then, the competitiveness of A is defined as $\sup_I \frac{cost(A(I))}{cost(O(I))}$. We seek algorithms yielding a small competitive ratio.

Dual of the ILP for FACILITYLEASING

$$\max \sum_{(j,t)\in D} \alpha_{jt}$$

$$\begin{aligned}
\text{s.t.} \quad \alpha_{jt} - \beta_{ikt',jt} &\leq c_{ij} \quad (i,k,t') \in F, (j,t) \in D \\
\sum_{(j,t)\in D} \beta_{ikt',jt} &\leq f_i^k \quad (i,k,t') \in F \\
\beta_{ikt',jt} &\geq 0 \quad (i,k,t') \in F, (j,t) \in D \\
\alpha_{jt} &\geq 0 \quad (j,t) \in D
\end{aligned}$$

Figure 2.2: The dual integer linear program describing the online Facility Leasing problem.

2.1.2 Results

We introduce the first online algorithm for the FACILITYLEASING problem with a competitive ratio that does neither depend on the number of time steps nor on the number of clients requesting the service. The previous result by Nagarajan and Williamson [52] yields a competitive factor of $O(K \log n)$, where n is the number of clients and (assuming that at least one client arrives each time step) also an upper bound on the number of time steps. Their competitive factor can be expressed as $O(\log(l_{\max}) \log n)$ with $l_{\max}$ denoting the length of the longest lease type (since $\log(l_{\max})$ can be seen as an upper bound for K). Our algorithm allows us to replace the $\log n$ factor of Nagaraja and Williamson with $l_{\max}$ in general and with $\log(l_{\max})$ for many "natural" special cases of client arrival sequences. These cases include, e.g., all instances where the number of clients arriving in each time step varies only by a constant factor, or is non-increasing, or is polynomially bounded in $l_{\max}$. Thus we have an $O(l_{\max} \log(l_{\max}))$-competitive algorithm in general and, which is our main contribution, an $O(\log^2(l_{\max}))$-competitive algorithm for the arrival sequences described above. To be more precise, the approximation factor of our algorithm is $O(\log(l_{\max}) H_{l_{\max}})$, where $H_{l_{\max}} := \sum_{i=1}^{l_{\max}} (|D_i| / (\sum_{j=1}^{i} |D_j|))$ with D_i denoting the set of clients arriving in time step i. This means that our approximation guarantee depends not on the absolute numbers of clients arriving in each time step, but on the relationship (as defined by $H_{l_{\max}}$) between those numbers. While our algorithm is not a direct improvement on Nagarajan and Williamson [52] (since we state our approximation factor using $l_{\max}$ and not K), our algorithm has the advantage that its competitive ratio is not dependent on the number of time steps.

2.1.3 Related Work

In [6] Anthony and Gupta propose and analyze various infrastructure leasing problems. Here, instead of obtaining some infrastructure for a single step or for infinitely long, the infrastructure is leased for a specific number of steps (which depends on the lease type) and can only be used during this time. Also, their problems are offline, i.e., the entire input is known from the start. They show that these kinds of problems have a connection to multi-stage stochastic optimization problems. One of the problems they consider is the metric uncapacitated Facility Location problem with K different lease types. For this offline leasing problem they present a $O(K)$ approximation algorithm obtained from an algorithm by Swamy and Shmoys for the K-stage stochastic facility location problem [65]. This upper bound of $O(K)$ was improved by Nagarajan and Williamson in [52], who presented a 3-approximation algorithm based on [34]. As already mentioned, they are also the first to consider the online variant of Facility Leasing (i.e., FACILITYLEASING).

It is important to note, that FACILITYLEASING is a generalization of the two following problems introduced by Meyerson: The *Online Facility Location* problem [49] and the *Parking Permit* problem [50]. We will discuss these two problems and their connection to FACILITYLEASING in the following paragraphs.

Parking Permit. Meyerson describes the Parking Permit problem with the following example: There are two possibilities for us to get to our working place. We can either walk, or go by car. The option we choose depends on the current weather (we go by car if it is a rainy day; otherwise, we always choose to walk). Thus, our actions are determined by something that we not only cannot influence, but also do not know in advance. Whenever we drive to work, we need to have a valid parking permit. There are various permits that vary in cost and duration (with the ones that last for a longer period of time tending to cost less per day than the shorter ones). Each day we choose to drive and do not have a valid permit, we have to buy a new one. The decision of which of the various permits to buy is a tricky one, since we do not know how often we will actually be able to make use of this permit again. If we use the car quite often during the next few days, buying a more expensive, but longer lasting permit, seems like a good idea. Otherwise, buying a one day permit is a better option.

Meyerson, in [50], shows a lower bound of $\Omega(K)$ on the competitive factor of deterministic algorithms for this problem, where K is the number of permits. He also shows that a simple greedy algorithm achieves this factor. For the case that randomization is allowed, he presented an algorithm with a competitive factor of $O(\log n)$ in expectation and showed that this factor cannot be improved. This Parking Permit problem is a special case of the FACILITYLEASING problem: Let us assume that there is only a single

facility. We simulate each day that we drive to work with a time step in which a client appears on top of the facility (distance between client and facility is zero). Each day we walk is represented by a time step in which no client appears (these are days on which we are not required to have a valid parking permit). Buying a permit is then equivalent to leasing this single facility with a lease type corresponding to the bought permit. Due to the simplicity of the considered instance (there is only a single facility), the question, at which position a facility shall be opened, does not arise. Also, the aspect of connecting clients and their distance to the used facility, are irrelevant. Only the question "which type of permit (i.e., lease) should be bought?" remains.

Online Facility Location. The second problem by Meyerson, Online Facility Location, is also a special case of FACILITYLEASING. Here, both problems deal with opening facilities and connecting clients to them. The difference is that in Online Facility Location facilities are opened permanently. This can be simulated by a FACILITYLEASING instance in which there is only one lease with an "infinite" length (e.g., choose the number of clients as the lease length of this single lease type). A very well written comprehensive survey of results concerning Online Facility Location and related problems is presented by Fotakis in [25].

Building upon the work of Meyerson, Fotakis presents various results for Online Facility location in [22]. He shows a lower bound of $\Omega(\frac{\log n}{\log \log n})$ and also that this bound is tight. This lower bound holds not only for deterministic, but also for randomized algorithms. A further algorithm, also by Fotakis, with an approximation factor of $O(\log n)$ can be found in [24]. Although this algorithm's runtime bound is weaker than the one from [22], it is a lot simpler than [22] and thus easier to implement. It is elegantly formulated with the help of the primal and the dual formulation of the Facility Location problem. Surprisingly, the primal-dual method (i.e., showing that the gap between the primal solution computed by the algorithm and a dual solution is not too large) is not used in its analysis. Later on, in [52], Nagarajan and Williamson apply the primal-dual method to show the same upper bound of $O(\log n)$ for this algorithm. They use this analysis as an intermediate step to show the competitive factor of $O(K \log(n))$ for their algorithm for the FACILITYLEASING problem.

An algorithm that is especially designed for the Euclidean plane was published by Anagnostopoulos et al. in [5]. It has an approximation factor of $O(\log n)$ and can be applied to various models (e.g., facilities can be either placed anywhere on the plane, or only on positions of already arrived clients, or only at fixed locations).

Incremental Facility Location. For optimization problems that are related to clustering, Charikar et al. introduced the framework of incremental algorithms in [14]. It

is used to incrementally and efficiently perform hierarchical clustering in online scenarios. *Incremental Facility Location*, which was first considered in [23], is directly derived from this framework. It is very similar to the Online Facility Location problem. The main difference is the additional option of "merging" facilities with each other: Whenever a facility i is opened, we are allowed to close some facility i' and then reconnect each client j (that used to be connected to i') to the newly opened i. After this merge, we are no longer required to pay the opening costs of i', but now have to pay the connection costs of connecting j to i (instead of paying for the connection between j and i'). The idea here is that, while clients can be reconnected, it is always guaranteed that clients that were at some point connected to the same facility will remain grouped together (i.e., always be connected to the same facility; their facility can, of course, change over time). Fotakis showed in [23] that a competitive factor of $O(1)$ can be achieved for this problem.

A streaming algorithm that relaxes the constrains in the Incremental Facility Location problem was introduced in [26] by Fotakis. Here, clients can be reconnected not only to facilities which are merged with the facility they are connected to, but also to the currently closest open facility. The focus here is to compute a good approximation while only storing very limited amount of data. For this problem, Fotakis presents a memoryless (it only requires to stores information regarding the set of currently open facilities) $O(1)$ competitive algorithm.

2.1.4 Structure of the Chapter

In Section 2.2, we will show how to modify FACILITYLEASING in such a way that it is, on one hand, way easier to work with, and on the other hand, a good solution to this simplified problem can still be used as an intermediate step to compute a good solution to the original problem (i.e., FACILITYLEASING). There are two different modifications needed to construct the required simplification. One restricts the lease lengths, while the other restricts the points in time at which facilities can be leased.

Our algorithm, which works on the simplified model (called 2-FACILITYLEASING), is presented in Section 2.3 and later on analyzed in Section 2.4. The analysis of the approximation factor consists of two parts. We first show that the contribution of clients can be used to bound the costs of the algorithm's solution. Then we argue that scaling down this contribution yields a feasible dual solution (which in turn yields the desired approximation guarantee). Finally, the chapter ends with Section 2.5, where possible future work is described.

2.2 Reduction to a Simplified Model Variant

The goal of this section is to show that we can transform a problem instance I of our online Facility Leasing problem FACILITYLEASING into an instance I' of the slightly simplified problem variant 2-FACILITYLEASING. In 2-FACILITYLEASING, all lease lengths are a power of two and a lease of type k may only start each l_k time steps.

2.2.1 Rounding the Lease Length

We consider the problem variant FACILITYLEASING$_1$ of FACILITYLEASING that only allows lease types whose lengths are a power of two. Consider an instance I of FACILITYLEASING having K leases $k \in \{1,2,\ldots,K\}$. From I we construct an instance I' by rounding the lease lengths l_k to the next larger power of two. That is, the lease lengths l'_k for $k \in \{1,2,\ldots,K\}$ are defined as $l'_k := 2^{\lceil \log l_k \rceil}$. Other than that, I and I' are identical. Note that I' is an instance of both FACILITYLEASING and FACILITYLEASING$_1$.

Let O denote an optimal solution to I and O' an optimal solution to I'. First note that $\text{cost}(O') \leq \text{cost}(O)$, as any solution to I yields a solution to I' having the same cost. Indeed, whenever a facility of lease type k in the solution to I is opened, opening a facility of the same lease type k (but of lease length $l'_k \geq l_k$) yields a feasible solution to I'. Now, consider any solution S' to I'. From this, we can build a feasible solution S to I as follows: Whenever a facility of lease type k is opened in S', we open two consecutive facilities of the same lease type in S. Since $2l_k \geq l'_k$, this yields a feasible solution to I. Moreover, we obviously have the relation $\text{cost}(S) = 2\,\text{cost}(S')$. We use these observations to prove the following lemma.

Lemma 2.1. *Given a c-approximation algorithm A' for the* FACILITYLEASING$_1$ *problem, we can construct a $2c$-approximation algorithm A for the* FACILITYLEASING *problem. Similarly, any c-competitive algorithm for the* FACILITYLEASING$_1$ *problem yields a $2c$-competitive algorithm for the* FACILITYLEASING *problem.*

Proof. Given a problem instance I for FACILITYLEASING we let A build a problem instance I' for FACILITYLEASING$_1$ by rounding the lease lengths to powers of two as described above and invoke A' on I'. The resulting solution S' is used to build solution S by opening two consecutive facilities for each facility opened in S' for I (see above). We have

$$\text{cost}(S) \leq 2\,\text{cost}(S') \leq 2c\,\text{cost}(O') \leq 2c\,\text{cost}(O).$$

Note that this can be done in an online fashion, by mimicking the behavior of A' but opening two consecutive facilities whenever A' opens one. ■

2.2.2 Restricting the Start of Leases

Let us consider the problem variant FACILITYLEASING$_2$ of FACILITYLEASING that allows to open facilities only at times t that are a multiple of their corresponding lease length. That is, a facility of type k may only open at times t with $t \equiv 0 \mod l_k$. Note that we do allow to open facilities belated, such that one may not be able to take advantage of the full lease length one has paid for.

First of all, consider a problem instance I of FACILITYLEASING. Obviously, any such I may be considered as a problem instance I' of FACILITYLEASING$_2$ without any changes. Moreover, given a solution S' to I', we can interpret it directly as a solution S to I. Thus, we have $\text{cost}(S) = \text{cost}(S')$. On the other hand, given optimal solutions O to I and O' to I', we have $\text{cost}(O') \leq 2\,\text{cost}(O)$. Indeed, any solution S to I yields a feasible solution S' to I' with $\text{cost}(S') \leq 2\,\text{cost}(S)$ as follows: Consider any facility of type $k \in \{1,2,\ldots,K\}$ opened in S at time t. For this facility, we open up to two consecutive facilities of the same lease type k in S'. The first facility is opened at time $t_1 := t - (t \mod l_k)$, the second at time $t_2 := t_1 + l_k$ if necessary. This yields a feasible solution S' to I' with cost $\text{cost}(S') \leq 2\,\text{cost}(S)$. We get the following lemma.

Lemma 2.2. *Given a c-approximation algorithm A' for the* FACILITYLEASING$_2$ *problem, we can construct a $2c$-approximation algorithm A for the* FACILITYLEASING *problem. Similarly, any c-competitive algorithm for the* FACILITYLEASING$_2$ *problem yields a $2c$-competitive algorithm for the* FACILITYLEASING *problem.*

Proof. Algorithm A is identical to A'. Run on a problem instance I of FACILITYLEASING, which we interpret as an identical instance I' of FACILITYLEASING$_2$, we get a solution S' to I' which can be immediately interpreted as a solution S to I (see above). We get

$$\text{cost}(S) = \text{cost}(S') \leq c\,\text{cost}(O') \leq 2c\,\text{cost}(O).$$

■

2.2.3 Combining Both Variants

The model variant 2-FACILITYLEASING of our online facility location problem FACILITYLEASING combines the simplifications of both FACILITYLEASING$_1$ and FACILITYLEASING$_2$. That is, we allow only lease lengths that are a power of two and restrict the starting points of facilities of type k to multiples of l_k. By combining the results from Lemma 2.1 and Lemma 2.2, one easily gets

Corollary 2.3. *Given a c-approximation algorithm A' for the* 2-FACILITYLEASING *problem, we can construct a* $4c$*-approximation algorithm A for the* FACILITYLEASING *problem. Similarly, any c-competitive algorithm for the* 2-FACILITYLEASING *problem yields a* $4c$*-competitive algorithm for the* FACILITYLEASING *problem.*

2.3 Algorithm description

Let us describe our online algorithm for the 2-FACILITYLEASING problem. In the beginning, all facilities are closed. At the arrival of the client set D_t at time t, our algorithm assigns these clients to open facilities to satisfy their demands, opening new facilities if necessary. The costs charged for this step comprise the corresponding connection cost c_{ij} for assigning the clients $j \in D_t$ to open facilities $i \in F$ and the opening cost of newly opened facilities (of a certain lease type). Remember that in this simplified problem variant, we open facilities using lease type k only at times t that are a multiple of the corresponding lease length l_k. That is, only at times t with $t \equiv 0 \mod l_k$. Especially, when we say we open facility $(i,k,t) \in F$, we can only make use of it up to the the next time $t' > t$ that is a multiple of l_k. This means we may open a facility belatedly, such that we cannot take advantage of the full lease length we paid for. Note that for a fixed facility i and lease type k we get a partition of the time horizon into intervals of length exactly l_k. These intervals can be identified with the corresponding facility i of lease type k which may serve clients in this interval. Now, for any time t we get exactly one interval I_i^k for each facility i and lease type k that can be used by a client appearing at time t. It remains to specify which pair (i,k) is chosen to satisfy a client's demand.

Our algorithm is based on an approximation algorithm by Jain and Vazirani [34] for the classical facility location algorithm. Their algorithm uses a primal-dual approach to compute a 3-approximation, and we make use of a similar approach in each single time step. In each time step t our algorithm operates in two phases, similar to the algorithm from [34] for the static facility location problem. In the first phase, the clients essentially bid towards the facilities (or more exactly, towards the intervals I_i^k). In the second phase, we use the triangle inequality to choose a cheap subset of facilities (intervals I_i^k) to actually open and assign clients to. In contrast to [34], we have to cope with the problem to build a good solution for a facility location problem starting from a partial solution (earlier arrived clients). This is similar to Nagarajan and Williamson [52], however, our subproblem is much more complex, as we consider all newly arrived clients simultaneously (instead of one after the other).

First Phase For each client $j \in D_{\leq t} := \bigcup_{t' \leq t} D_{t'}$ that arrived at time t or before, we introduce a potential α_{jkt} that starts at zero and is continuously increased (concurrently and at the same rate for each client). Each of these potentials is reset

to zero in each round. To simplify notation, let us define $(x)_+ := \max(x,0)$. For any facility i of lease type k we maintain the invariant $f_i^k \geq \sum_{j \in D_{\leq t}} (\alpha_{jkt} - c_{ij})_+$ (INV1). Whenever equality is reached for some facility i and lease type k, we temporarily open i using lease type k. As soon as $\alpha_{jkt} \geq c_{ij}$ for a client $j \in D_{\leq t}$ and a (temporarily or permanently) open facility i of lease type k, we stop increasing α_{jkt}. If $j \in D_t$ (i.e., j is a newly arrived client), we connect j to i and furthermore set $\hat{\alpha}_j := \alpha_{jtk}$ (these $\hat{\alpha}_j$ correspond to the dual variables α_{jt} in the ILP from Figure 2.2, the t given implicitly by the relation $j \in D_t$). As a second invariant, we ensure that in no time step t' an $\alpha_{jkt'}$ is increased beyond $\hat{\alpha}_j$ (INV2).

Second Phase In this phase we build K different conflict graphs, one for each lease type k. The nodes of the graph for lease type k are given by temporarily and permanently opened facilities i of lease type k. There is an edge between two nodes i and i' if and only if there is some client $j \in D_{\leq t}$ with $\alpha_{jkt} > \max(c_{ij}, c_{i'j})$. We say that the facilities i and i' are in conflict. Now, for each conflict graph we compute a maximal independent set (MIS) and open the facilities in the MIS permanently (while closing the remaining temporarily opened facilities). If for a client $j \in D_t$ (i.e., newly arrived clients) the facility i it was connected to during the first phase is not in a member of a MIS, j is reconnected to a neighbor of i that is a member of a MIS (i.e., permanently open).

2.4 Analysis

Here we prove that the algorithm described in the previous section is $(3+K)H_{l_{\max}}$-competitive with respect to 2-FACILITYLEASING, the simplified online problem variant described in Section 2.2. Remember that $l_{\max}$ denotes the maximum lease length. Moreover, note that it is sufficient to consider the first $l_{\max}$ time steps: at time $l_{\max}$ all facilities must be closed, since for any $k \in \{1,2,\ldots,K\}$ we have $l_{\max} \equiv 0 \mod l_k$. Let us partition the time horizon into rounds $\tau_i := \{(i-1)l_{\max}, \ldots, il_{\max} - 1\}$ of length $l_{\max}$. By the above observation, these rounds yield independent subproblems, each of length $l_{\max}$. We continue to show that the solution of our algorithm is $(3+K)H_{l_{\max}}$-competitive in each such round. As the costs over all rounds are additive, this yields that it is $(3+K)H_{l_{\max}}$-competitive for the complete problem.

Our analysis follows the typical idea of the analysis of a primal-dual algorithm, similar to, e.g., the analysis of Jain and Vazirani [34]. The values $\hat{\alpha}_j$ computed by our algorithm correspond to the dual variables of the ILP formulation in Figure 2.2. First of all, we show that the sum of all $\hat{\alpha}_j$ times $(3+K)$ is an upper bound for the cost of the solution produced by our algorithm (Lemma 2.4). Next, we consider the $\hat{\alpha}_j$ as a (possibly

infeasible) solution to the dual program of the FACILITYLEASING ILP. We prove that by scaling this solution down by a suitable factor, we get a feasible solution to the dual program (Lemma 2.7). By the weak duality theorem, multiplying both factors yields the final competitive factor (Theorem 2.8).

2.4.1 Upper Bounding the Solution

The following lemma upper bounds the cost of the solution produced by our algorithm by $(3+K)\sum_{j\in D}\hat{\alpha}_j$. The basic idea is to exploit the triangle inequality to show that $3\sum_{j\in D}\hat{\alpha}_j$ is a bound on the total connection cost and that our algorithm ensures that each $\hat{\alpha}_j$ is used at most K times to cover the complete costs for opening facilities.

Lemma 2.4. *The cost of the primal solution produced by our algorithm can be bounded from above by* $(3+2K)\sum_{j\in D}\hat{\alpha}_j$.

Proof. We bound the connection costs of the clients and the opening costs of facilities separately. The $\hat{\alpha}_j$ value of a client j is computed in step t of j's arrival during the first phase of our algorithm. During this phase, j is either connected to a facility i that was already (permanently) opened at time $t' < t$ or one that was temporarily opened at the current time t. In both cases the client's $\hat{\alpha}_j$ value was set in such a way that it can cover at least the distance c_{ij} between i and j. If i remains in one of the MISs computed in the second phase of our algorithm, this guarantees that j is assigned to a facility i' such that $\hat{\alpha}_j$ is an upper bound on the client's connection costs $c_{i'j}$. Otherwise, if i is no longer in any MIS at the end of phase two, Proposition 2.5 (see below) exploits the metric property of our facility location problem and yields that j is assigned to a facility i' such that $3\hat{\alpha}_j$ is an upper bound on the client's connection costs $c_{i'j}$.

Now, consider the facility costs and fix a facility i of lease type k that is permanently opened at some time t by our algorithm. As (i,k) is opened permanently at time t in the second phase, it must have been temporarily opened in the first phase. Thus, by definition of the algorithm, invariant INV1 must hold with equality, that is $f_i^k = \sum_{j\in D_{\leq t}}(\alpha_{jkt} - c_{ij})_+$. Consider these *bids* $(\alpha_{jkt} - c_{ij})_+$ of clients $j \in D_{\leq t}$ to facility i of lease type k. Note that all non-zero bids of clients j at the current time are guaranteed to be used by facility (i,k) only, as (i,k) must have been in the corresponding MIS for lease type k. Moreover, note that for a single client that arrived at time t, all its bids given to (and used by) facilities of type k sum up to at most $2\hat{\alpha}_j$, as any $\alpha_{jkt'}$ with $t' > t$ stops increasing as soon as a corresponding open facility (or $\hat{\alpha}_j$) is reached. Together, this yields that the total costs for opening facilities of type k in the solution produced by our algorithm is upper bounded by $\sum_{j\in D}\hat{\alpha}_j$. As there are K different lease types, together with the bound on the connection costs we get the lemma's statement. ■

The following proposition exploits the triangle inequality of our metric facility location problem and can be proven completely analogue to [34, Lemma 5].

Proposition 2.5. *For each client j that is reconnected in the second phase to a facility i, we have $\hat{\alpha}_j \geq \frac{1}{3}c_{ij}$.*

Proof. Let i' be the facility that client j was connected to in the first phase of the algorithm. There must be a client j' that is responsible for the conflict between facility i and i'. We have that $\alpha_{j'} \geq c_{i'j}$, $\alpha_{j'} \geq c_{ij}$ and $\alpha_j \geq c_{i'j}$. Let s_i resp. $s_{i'}$be the points in time where i resp. i' are temporarily opened. We know that $\alpha_{j'} \leq \min(s_i, s_{i'})$ since j' was contributing to both facilities, and that $\hat{\alpha}_j \geq s_{i'}$ since j was connected to i'. Plugging this information into the triangle inequality $c_{ij} \leq c_{i'j} + c_{i'j'} + c_{ij'}$ yields the proposition. ■

2.4.2 Scaling the Dual Variables for Feasibility

For the second part of the proof, it remains to scale down the dual solution represented by the $\hat{\alpha}_j$ such that we obtain a feasible solution. Before we do so, we need another proposition based on the triangle inequality. In spirit, it is similar to [52, Lemma 5], but has a slightly more involved proof due to the fact that we have to consider multiple $\hat{\alpha}_j$'s that increase simultaneously.

Proposition 2.6. *Given a client l that arrived in time step t and a facility i of type k for any client j that arrived before time t, we have $\alpha_{jkt} - c_{ij} \geq \hat{\alpha}_l - 2c_{ij} - c_{il}$.*

Proof. Showing that $\alpha_{jkt} + c_{ij} + c_{il} \geq \hat{\alpha}_l$ proves the proposition. Since for $\alpha_{jkt} \geq \hat{\alpha}_l$ the statement trivially holds, we assume the contrary. This means that client j reached an open facility i' (and thus its α_{jkt} stopped increasing) before $\hat{\alpha}_l$ was fixed (i.e., α_{lkt} stopped increasing). Since α_{lkt} stops increasing once it is large enough to cover the distance between i' and l and this distance is at most $\alpha_{jkt} + c_{ij} + c_{il}$, the proposition follows. ■

Before we continue with the Lemma 2.7 and its proof, let us define N_t to be the number of clients that have arrived until time t (i.e., $N_t := |D_1| + |D_2| + \ldots + |D_{t-1}|$) and note that

$$N_t = N_t \frac{|D_t|}{|D_t|} = \sum_{j \in D_t} \frac{N_t}{|D_t|} = \frac{N_t}{|D_t|} \sum_{j \in D_t} 1. \tag{2.1}$$

For the prior defined series H_q, it holds that

$$\sum_{t=1}^{t^*} 2h_t \sum_{t'=1}^{t-1} \sum_{j \in D_{t'}} c_{ij} = \sum_{t=1}^{t^*} 2 \sum_{j \in D_t} c_{ij}(H_{t^*} - H_t), \tag{2.2}$$

which can be easily seen by observing that $\sum_{i=1}^{q}\sum_{j=1}^{i-1} x_j h_i = \sum_{i=1}^{q} x_i(H_q - H_i)$ holds for any series and any coefficients x_i. Given these tools, we are now ready to formulate and prove Lemma 2.7, which essentially shows that we get a feasible solution to the dual program of our problem if we scale the $\hat{\alpha}_j$ by a factor of $\frac{1}{H_{l_{\max}}}$. To ease notation in the following, whenever we consider a facility i of lease type k at time t^*, any time steps t we speak of are assumed to lie in the the corresponding time interval of (i,k,t^*) (all other time steps are of no interest with respect to the constraints of the dual program).

Lemma 2.7. *For any facility i and lease type k at time t^* we have*

$$\sum_{t=1}^{t^*}\sum_{j\in D_t}(\hat{\alpha}_j/2H_{t^*} - c_{ij}) \leq f_i^k\ ,$$

where $H_q := \sum_{i=1}^{q} \frac{|D_i|}{\sum_{j=1}^{i}|D_j|}$.

Proof. Remember INV1 of our algorithm (see algorithm description in Section 2.3). It states that the sum of bids towards a facility at any point in time does never exceed its opening costs. Thus, for any time step t^* we have

$$\begin{aligned}
f_i^k &\geq \sum_{j\in D_{\leq t^*}}(\alpha_{jkt^*} - c_{ij})_+ \\
&= \sum_{l\in D_{t^*}}(\hat{\alpha}_l - c_{il})_+ + \sum_{j\in D_{<t^*}}(\alpha_{jkt^*} - c_{ij})_+ \\
&\geq \sum_{l\in D_{t^*}}(\hat{\alpha}_l - c_{il}) + \sum_{j\in D_{<t^*}}(\alpha_{jkt^*} - c_{ij}) \\
&= \sum_{l\in D_{t^*}}(\hat{\alpha}_l - c_{il}) + \sum_{t=1}^{t^*-1}\sum_{j\in D_t}(\alpha_{jkt^*} - c_{ij}) \\
&\geq \sum_{l\in D_{t^*}}(\hat{\alpha}_l - c_{il}) + \sum_{t=1}^{t^*-1}\sum_{j\in D_t}(\hat{\alpha}_{l^*} - c_{il^*} - 2c_{ij}) \qquad \text{(Prop. 2.6 and } l^* := \arg\max(\hat{\alpha}_l - c_{il})\text{)} \\
&= \sum_{l\in D_{t^*}}(\hat{\alpha}_l - c_{il}) + \sum_{t=1}^{t^*-1}\sum_{j\in D_t}(\hat{\alpha}_{l^*} - c_{il^*}) - 2\sum_{t=1}^{t^*-1}\sum_{j\in D_t} c_{ij} \\
&= \sum_{l\in D_{t^*}}(\hat{\alpha}_l - c_{il}) + (\hat{\alpha}_{l^*} - c_{il^*})\frac{N_{t^*}}{|D_{t^*}|}\sum_{j\in D_{t^*}} 1 - 2\sum_{t=1}^{t^*-1}\sum_{j\in D_t} c_{ij} \\
&= \sum_{l\in D_{t^*}}(\hat{\alpha}_l - c_{il}) + \frac{N_{t^*}}{|D_{t^*}|}\sum_{j\in D_{t^*}}(\hat{\alpha}_{l^*} - c_{il^*}) - 2\sum_{t=1}^{t^*-1}\sum_{j\in D_t} c_{ij} \\
&\geq \sum_{l\in D_{t^*}}(\hat{\alpha}_l - c_{il}) + \frac{N_{t^*}}{|D_{t^*}|}\sum_{j\in D_{t^*}}(\hat{\alpha}_j - c_{ij}) - 2\sum_{t=1}^{t^*-1}\sum_{j\in D_t} c_{ij}
\end{aligned}$$

$$= \left(1 + \frac{N_{t^*}}{|D_{t^*}|}\right) \sum_{j \in D_{t^*}} (\hat{\alpha}_j - c_{ij}) - 2 \sum_{t=1}^{t^*-1} \sum_{j \in D_t} c_{ij}$$
$$= \left(\frac{N_{t^*} + |D_{t^*}|}{|D_{t^*}|}\right) \sum_{j \in D_{t^*}} (\hat{\alpha}_j - c_{ij}) - 2 \sum_{t=1}^{t^*-1} \sum_{j \in D_t} c_{ij}.$$

The above inequality holds for each $t \in \{1, \ldots, t^*\}$. Dividing each such inequality by $\frac{N_t + |D_t|}{|D_t|}$ yields the following set of inequalities:

$$\frac{|D_{t^*}|}{N_{t^*} + |D_{t^*}|} f_i^k \geq \sum_{j \in D_{t^*}} (\hat{\alpha}_j - c_{ij}) - 2 \frac{|D_{t^*}|}{N_{t^*} + |D_{t^*}|} \sum_{t=1}^{t^*-1} \sum_{j \in D_t} c_{ij}$$
$$\frac{|D_{t^*-1}|}{N_{t^*-1} + |D_{t^*-1}|} f_i^k \geq \sum_{j \in D_{t^*-1}} (\hat{\alpha}_j - c_{ij}) - 2 \frac{|D_{t^*-1}|}{N_{t^*-1} + |D_{t^*-1}|} \sum_{t=1}^{t^*-2} \sum_{j \in D_t} c_{ij}$$
$$\vdots \qquad \vdots \qquad \vdots$$
$$\frac{|D_2|}{N_2 + |D_2|} f_i^k \geq \sum_{j \in D_2} (\hat{\alpha}_j - c_{ij}) - 2 \frac{|D_2|}{N_2 + |D_2|} \sum_{t=1}^{1} \sum_{j \in D_t} c_{ij}$$
$$\frac{|D_1|}{N_1 + |D_1|} f_i^k \geq \sum_{j \in D_1} (\hat{\alpha}_j - c_{ij}) - 0.$$

Adding up these t^* inequalities yields

$$\left(\frac{|D_1|}{N_1 + |D_1|} + \ldots + \frac{|D_{t^*}|}{N_{t^*} + |D_{t^*}|}\right) f_i^k \geq \sum_{t=1}^{t^*} \sum_{j \in D_t} (\hat{\alpha}_j - c_{ij}) - \sum_{t=1}^{t^*} 2 \frac{|D_t|}{N_t + |D_t|} \sum_{t'=1}^{t-1} \sum_{j \in D_{t'}} c_{ij}.$$

Due to Inequality (2.2) we have

$$\begin{aligned} H_{t^*} f_i^k &\geq \sum_{t=1}^{t^*} \sum_{j \in D_t} (\hat{\alpha}_j - c_{ij}) - \sum_{t=1}^{t^*} 2 \sum_{j \in D_t} c_{ij} (H_{t^*} - H_t) \\ &= \sum_{t=1}^{t^*} \sum_{j \in D_t} (\hat{\alpha}_j - 2 H_{t^*} c_{ij}) + \sum_{t=1}^{t^*} \sum_{j \in D_t} 2 c_{ij} \left(H_t - \frac{1}{2}\right) \\ &\geq \sum_{t=1}^{t^*} \sum_{j \in D_t} (\hat{\alpha}_j - 2 H_{t^*} c_{ij}). \end{aligned}$$

Dividing by $2H_{t^*}$ yields $\sum_{t=1}^{t^*} \sum_{j \in D_t} \left(\frac{\hat{\alpha}_j}{2H_{t^*}} - c_{ij}\right) \leq \frac{f_i^k}{2} \leq f_i^k$. ■

Finally, by combining our results from Corollary 2.3, Lemma 2.4 and Lemma 2.7, and using that our time horizon is at most $l_{\max}$ (i.e., $t^* \leq l_{\max}$), the weak duality theorem implies a competitive factor depending on the series H_k.

Theorem 2.8. *Our online algorithm is at most* $4(3+2K)H_{l_{max}}$*-competitive for the* FACILITYLEASING *problem. Here, the series* H_q *is defined by*

$$H_q := \sum_{i=1}^{q} \frac{|D_i|}{\sum_{j=1}^{i} |D_j|}$$

and describes the relationship between the number of clients that arrive in each step.

The following corollaries bring the competitive factor guaranteed by Theorem 2.8 into a more concrete and compact form.

Corollary 2.9. *Our algorithm is at most* $4(3+2K)l_{max} = O(\log(l_{max})l_{max})$*-competitive for the* FACILITYLEASING *problem.*

Corollary 2.10. *If for each round, the number of clients at any time t does vary by at most a constant factor, is non-increasing, or bounded from above by a polynomial in* l_{max}*, the competitive factor of our algorithm becomes at most* $O(K\log(l_{max})) = O(\log^2(l_{max}))$*-competitive for the* FACILITYLEASING *problem.*

While Corollary 2.10 arguably covers the most interesting and realistic cases, it seems probable that one can in fact construct an instance where the bound given in Corollary 2.9 is tight. Based on the convergence behavior of the series H_k, instances where the number of arriving clients increases at least exponentially seem the most difficult and challenging for an online algorithm.

2.4.3 Randomization

Our competitive bounds can also be written as $O(Kl_{\max})$ and $O(K\log(l_{\max}))$, respectively. Meyerson [50] showed a lower bound of $\Omega(K)$ on the competitive ratio of deterministic online algorithms for the Parking Permit problem, a special case of FACILITYLEASING. For randomized algorithms, a lower bound of $\Omega(\log K)$ is proven together with an algorithm with a runtime matching this bound. As these bounds immediately carry over to our model, one may hope, using the randomization techniques from [50], to improve our bounds to $O(\log(K)l_{\max})$ and $O(\log(K)\log(l_{\max}))$ by replacing the K factor with $\log(K)$. To the best of our knowledge, no work containing this approach has been published yet.

We would like to present a short intuition of how a randomized algorithm for FACILITYLEASING might look like. These are no actual results, but rather ideas we had when we wanted to tackle this problem. Thus, they are explained in a very informal manner.

First, note that the K factor in the approximation stems from the fact that each lease type is considered almost independently from the other lease types. This can best be seen in Lemma 2.4 where we use the contribution $\hat{\alpha}_j$ of a client j multiple, namely K, times in order to give an upper bound on the cost of our solution. Our aim should be to (i) only use the $\hat{\alpha}_j$ value $\log(K)$ times, and to (ii) find a different process to determine which lease type to use (in the deterministic case we just use a lease if it was fully paid for; now we have to choose one out of K possibilities and base our choice on the amount of contribution to each lease). To achieve (i), the clients need to split up their contribution $\hat{\alpha}_j$ to each facility among the K different lease types. Splitting the $\hat{\alpha}_j$ value evenly between the lease types can be shown to not work out. Splitting that is dependent on the costs f_i^k of the lease for a given facility also does not work. The process mentioned in (ii) should also use some kind of randomization, since we have a lower bound of $\Omega(K)$ for deterministic algorithms for the Parking Permit problem (note that it is possible that randomization is required at some other place to "overcome" this bound).

An idea to deal with both (i) and (ii) is to adapt the randomized algorithm for the Parking Permit problem by Meyerson Meyerson [50]. Using his approach, we are able to make sure that each $\hat{\alpha}_j$ is only used $\log(K)$ many times in expectation. The contribution to the leases is split in a similar way as is the payment to the parking permits in Meyerson's algorithm. This means that the rate at which the contribution is assigned to leases depends on two things: First, the lower the cost of the lease, the more contribution it receives. Second, the more contribution is already assigned to the leases in rounds before, the more contribution will be given to this lease in the current round. A facility is now opened once the ratios of current contributions to the lease cost, summed up over all lease types, add up to 1. The actual lease type our facility is opened with is now chosen by a random experiment, where the probability of choosing a certain lease type is equal to its ratio mentioned above. All these steps will ensure that each $\hat{\alpha}_j$ is used, in expectation, at most $O(\log(K))$ times.

Now, all what remains to be done is to scale the $\hat{\alpha}_j$ values (i.e., divide by $O(\log(n))$ (or $O(\log(l_{\max}))$) such that the resulting values yield a feasible primal solution. We did not manage to analyze this step. The main difficulty here is due to Proposition 2.6, which is not applicable for our randomized algorithm. This proposition is used to bound the growth of the $\hat{\alpha}_j$ using the fact that a client j has to stop increasing this value once an open facility is reached. In the randomized setting, we do not know which lease was chosen by our random experiment and thus cannot say with certainty when the $\hat{\alpha}_j$ will stop to grow. Thus, one has to find a technique that uses facilities that are almost opened (the ratio of contribution is almost 1) to bound the $\hat{\alpha}_j$'s growth.

2.5 Conclusion & Future Work

We gave the first algorithm for the online Facility Leasing problem FACILITYLEASING that has a time-independent competitive factor of $O(l_{\max}\log(l_{\max}))$ in general and $O(\log(l_{\max})^2)$ in many common cases. The competitive factor can be upper bounded by $O(H_{l_{\max}}\log(l_{\max}))$, where $H_{l_{\max}}$ is defined by the series $H_t := \sum_{i=1}^{t}\frac{D_i}{\sum_j^i D_j}$. The $D_i \in \mathbb{N}$ represent the number of clients arriving at time step i. For an exponential increase in the number of arriving clients, e.g., $D_i = 2^i$, we have $H_t = \Theta(t)$. We conjecture that, based on this observation, it is possible to build an instance that shows that our upper bound is tight for our algorithm. Instances featuring such an exponential increase in arriving clients seem to have a unique hardness for an online algorithm: At any time t the number of arriving clients essentially matches the total number of clients that arrived up to now. Thus, in each single time step, we have to solve a problem that is as hard as the complete problem up to the current time. It remains an interesting problem, whether such instances are inherently difficult to handle for online algorithms, or whether this conjectured lower bound is merely limited to our online algorithm.

Disregarding our algorithm (which expresses the competitive factor using the length of the longest lease type), the best known competitive factor for FACILITYLEASING is $O(K\log(n))$ due to [52]. It is still an open question whether this result can be improved without expressing the factor with the help of $l_{\max}$. The best existing lower bound for FACILITYLEASING is $\Omega\left(\log(n)/\log(\log(n)) + K\right)$ for deterministic algorithm and $\Omega\left(\log(n)/\log(\log(n)) + \log(K)\right)$ for randomized algorithms. This is due to FACILITYLEASING being a generalization of the Online Facility Location problem (with the lower bound of $\Omega\left(\log(n)/\log(\log(n))\right)$) and the Parking Permit problem (with the lower bound of $\Omega(K)$for deterministic and a lower bound of $\Omega(\log(K))$ for randomized algorithms). Closing this rather large gap remains an interesting challenge.

CHAPTER 3

The Facility Location Problem in Sensor Networks

3.1 Introduction

In the previous chapter, we considered Facility Location from the viewpoint of a centralized algorithm (i.e., an algorithm that is executed on a single processor). Being able to use such centralized algorithms in computer networks is not always possible. As the given networks can be very large and highly dynamic, an entity executing a centralized algorithm might require too much time to gather all information that is necessary to solve the problem. Also, by the time the information is gathered and a solution is computed, the network might have changed so much that the quality of the (possibly outdated) solution is no longer sufficient.

In this chapter, we addresses the issue stated above by introducing distributed, especially *local*, approximation algorithms for the Facility Location problem. Such local algorithms are executed in parallel on each node of the network. We call them *local*, since each node performs its computations based on information regarding only its local neighborhood. A node's local neighborhood can, e.g., be defined to include nodes that are at most a constant number of hops away from it (i.e., the number of edges on a shortest path between them is constant). Once a local algorithm terminates, we have many (one for each node) partial solutions to the problem at hand. Together, all these solutions form the actual output (i.e., the global solution to our problem). With this distributed approach, information about the network is no longer required to be propagated over long distances (since everything that is far away from a node is irrelevant for its computations). This aspect of locality makes the size of the network less significant. Also, perpetual changes in the network structure are less problematic. Only in the proximity of the change are required to update their solutions, while all the other nodes of the network

remain unaffected by the change. The downside of this local approach is that the quality of our solution is often lower than the quality of a solution computed by a centralized algorithm. Nevertheless, we show that the two algorithms presented in this chapter still yield good (i.e., constant factor) approximations.

In the remainder of this chapter's introduction, we first explain why our algorithms are especially suited for use in sensor networks. This is done in Subsection 3.1.1, where we list properties of such networks and describe how our algorithms take them into account. We give, in Subsection 3.1.2, a formal definition of the distributed variant of the Facility Location problem at hand. It is quite different from the problem we studied in the previous chapter, as it is no longer an online problem, has a different cost function, and is executed in a distributed computation model. In Subsection 3.1.3, we state our results concerning our approximation algorithms. Related work is handled in Subsection 3.1.4, where we especially focus on Facility Location problems formulated for a distributed setting. At last, Subsection 3.1.5 gives an overview of the way our local algorithms and their analysis are described in the remainder of the chapter.

3.1.1 Relevance for Sensor Networks

The focus of the work presented in this chapter are algorithms designed especially with sensor networks in mind. In the context of such networks, the Facility Location problem arises if the objective is to provide a costly (e.g., energy intensive) distributed service to the network. To be able to provide such a service, we have to choose a subset of nodes as service providers (*facilities*) and assign the remaining nodes (*clients*) to close by service providing nodes. Examples for such services are the maintenance of a distributed database, the gathering of measurement data, control of the network and various energy intensive computations.

The Facility Location variant that we formulate in this chapter reflects the properties of wireless sensor networks. We assume these networks to contain a very large (arbitrarily scaling) number of nodes with very limited capabilities. Another property of these networks are the ongoing changes they are subjected to (e.g., nodes joining/leaving the network or changing their position). Only local algorithms can be used in our sensor network scenario, as the considered networks are arbitrary scalable, highly dynamic, and consist of nodes with very limited capabilities. We formulate two such algorithms that produce solutions, which are guaranteed to be constant factor approximations, by assigning either the role *facility* or the role *client* to each sensor node. Note that this is different from the previous chapter, where we have two fixed stets of client and facility nodes and only have to determine which of the facility nodes should be opened. The computed solutions are guaranteed to be constant factor approximations of an optimal

solution.

As already mentioned, our algorithms are designed to cope with the various challenges presented by wireless sensor networks. These challenges and the way our algorithms deal with them are presented next.

Limited communication capabilities: One can expect two sensor nodes to be able to communicate with each other only if the geographical distance between them is small. Additionally, the messages exchanged between the nodes are highly limited in size. Furthermore, a node is only aware of other nodes, and their positions, if they are close by. Our algorithms take these constraints into account and only use this strongly limited information for their execution.

Dynamics: Since the network is subjected to ongoing changes, the quality of the current solution can deteriorate, even if all nodes execute the algorithm correctly. A global algorithm can, after detecting changes to the network graph, just restart the computation on every single node in order to reestablish a solution of good quality. In our case, such an approach is infeasible and also undesired: a restart request issued by a node can require a large amount of time to be propagated to all other nodes (during this time a lot of other changes might occur). Also, recomputing a (possibly totally different) solution for the entire network, just because a small part of it changed, can be too costly. In order to deal with this, nodes locally detect such changes to the network and adapt their own local solution accordingly. These adaptations only affect nodes that are in close vicinity of the change (i.e., are not propagated through the entire network). Thus, only a small part of the network is affected, while the remaining nodes never even notice that the global solution changed.

Fault-Tolerance: Another aspect of sensor nodes is their susceptibility to incorrect behavior. Nodes might stop executing the algorithm, just to resume its execution some time later. They might also produce incorrect output until a technician repairs them. If such behavior occurs, we want our algorithms to deal with it in a local manner. No restart of the entire algorithm should be necessary and the adaptations to the solution should only affect nodes in the proximity of the malfunctioned node.

Energy consumption: In the original version of the metric facility location problem, the costs of connecting a client to a facility are linear in the distance c between them. This is not necessarily the case in the setting of wireless networks. Here, the costs can be modeled by a polynomial c^p, where the value of the path loss exponent p varies depending on the environment the sensor network is deployed

in. This polynomial is a simplified way to represent the energy costs necessary to maintain a stable radio signal with increasing distance. According to [59], possible values for p can be 2 in the (idealized) free space, 2.7 up to 3.5 in urban areas, and even as large as 4 up to 6 when the nodes are obstructed by buildings. We take the path loss exponent into account by including it in the cost function.

Measurement errors: Obviously, information about the distances between nodes are required for the computation of a good solution. These information are not necessarily available to the sensor nodes (especially in a dynamic scenario). Thus, the distances need to be measured first. These measurements can be erroneous and it is reasonable to assume (see [28]) that the measurement error increases with the distance. Thus, we introduce the possibility to model this error and show its effect on the approximation factor.

Asynchronous execution: Finally, in a large scale network one cannot assume that the execution of an algorithm can be performed in a synchronous way. Thus, our algorithms allow the nodes to act asynchronously. A symmetry problem can arise if the local neighborhood of neighboring nodes is equal and the nodes act exactly at the same time (see 3.4 for a detailed explanation). This situation can lead to infinite loops, where the nodes never stop switching their role. We deal with this problem with the help of randomization.

3.1.2 Problem Definition & Notation

We are given a complete, undirected, weighted graph $G = (V,E)$ as input. Each edge $\{i, j\}$ in G is weighted with a nonnegative value $c(i, j) \in \mathbb{R}_{\geq 0}$ that represents the distance (i.e., the costs of the connection) between node i and j. These weights satisfy the triangle inequality (i.e., $\forall\, i, j, k$ we have $c(i, j) \leq c(i,k) + c(k, j)$), are symmetric, and $c(i, j) = 0$ iff $i = j$. Also, there are two values $d_i, f_i \in \mathbb{R}_{\geq 0}$ associated with each node i.

The objective is to assign one of two roles to each node. A node must either become a *facility* or a *client*. We say that a node *opens* (resp. *closes*) if it changes its role to facility (resp. client). The value f_i represents the costs of opening node i and d_i represents the demand of a node with the client role. The assignment (partition of V into the sets F and C) has to be chosen in such a way that the objective function $\sum_{i \in F} f_i + \sum_{j \in C} d_j \cdot c(j,F)^p$ is minimized. Here, F (resp. C) represents the set of nodes with the facility (resp. client) role (thus, we have $C = V \setminus F$). The value $c(j,F) = \min_{i \in F}\{c(j,i)\}$ describes the distance between client j and its nearest facility i, while $p > 0$ is the exponent representing the exponentially increasing communication costs. Since an arbitrary number of clients can use a single facility, we deal with the *uncapacitated* facility location problem.

In order to represent errors that might occur when nodes measure distances between each other, we introduce relative measurement errors bounded by the parameter μ with $1 \leq \mu$. In other words, a node i's measurement may yield a distorted distance $\hat{c}(i,j)$ to node j instead of the exact distance $c(i,j)$ where $1/\mu \cdot c(i,j)^p \leq \hat{c}(i,j)^p = \hat{c}(j,i)^p \leq \mu \cdot c(i,j)^p$. We use the $\hat{c}(i,j)^p$ values for the execution of our algorithms.

Execution model. The model used for the execution of our algorithm is based on the $\mathcal{CONGEST}$ model, which was already mentioned in the introductory Section 1.5.2. It was introduced by Peleg (see [57]) and is commonly used to model the execution of distributed algorithms on graphs. Here, algorithms are executed in synchronous send-receive-compute cycles. In one cycle, each node sends a message to each of its neighbors in the graph. Note that the messages sent to each neighbor by a single node are not required to contain the same information. Once all nodes have sent their messages, they receive a single message from each of their neighbors. After all the messages have been received, every node is allowed to spend an arbitrary amount of time for computation (i.e., computation is for free and we are only interested in the number of communication rounds). The end of the computation by all nodes marks the start of a new send-receive-compute cycle.

The message size in the $\mathcal{CONGEST}$ model is bounded. We limit the size of the messages used in our algorithms to $O(\log n)$ bits, where $n = |V|$. The same limitations are also used in [30, 51, 54]. This bound is reasonable, because it allows the nodes to send their IDs in single messages. Due to this constraint, we restrict the values f_i, d_i, and $c(i,j)$ to be at most polynomial in n such that they can be represented with $O(\log(n))$ bits (i.e., to be able to send them in a single message).

We adapt the $\mathcal{CONGEST}$ model to suite our requirements as follows. First, we limit the communication range of the nodes by only allowing the nodes i and j to communicate with each other if the weight $c(i,j)$ of the edge $\{i,j\}$ is smaller than $O(r_{max})$ where $r_{max} := \max_{i \in V}\{f_i/d_i\}$. Second, again to better reflect the abilities and limitations of sensor nodes, we allow the nodes to execute the algorithm in asynchronous rounds. In such a round every node can be active multiple times. Nodes are not required to (but still may) act at the same time. A round is completed as soon as all nodes have been active at least once. Note, that if all nodes would be required to act at the same time, a round in our model would be equal to a round in the standard $\mathcal{CONGEST}$ model. Nodes do not know, nor is it of any importance to them, when such an asynchronous round start or ends. These rounds are only introduced for the sake of the runtime analysis of our algorithms. Further, as is the case with all computation models we use in this thesis, we abstract from message transmission delays and consider computation time within a single round as instant. This means that a node which becomes active at time t receives

all messages that have been sent before t (but not the message sent exactly at time t). Furthermore, all computation is done instantly and messages are sent immediately.

In order to execute our algorithms, nodes require information only about other nodes within $O(r_{max})$ distance. One can assume that, at least in the sensor networks scenario, r_{max} is independent of the number of nodes. Thus, the execution of our algorithms only requires nodes to able to communicate with nodes within constant (i.e., independent of n) distance.

3.1.3 Results

The results presented in this chapter are published in [2] and are a continuation of our work from [18]. We present two similar approximation algorithms for the Facility Location problem. Both algorithms are based on the approach by Mettu and Plaxton presented in [48]. The first algorithm runs in expected $O(\log_{1+\varepsilon} n)$ rounds and yields a $\mu^4(1+4\mu^2(1+\varepsilon)^{1/p})^p$ approximation, while the second has an expected running time of $O(\log^2_{1+\varepsilon} n)$ rounds and an approximation factor of $\mu^4(1+2(1+\varepsilon)^{1/p})^p$, where $\varepsilon > 0$ is an arbitrarily small constant.

These two algorithms are particularly suitable for sensor networks. They are fast (expected running time is polylogarithmic in expectation) and very simple to implement (once the radius value mentioned above is computed, the nodes only need to perform a relatively easy operation to ensure a specific property is satisfied). Since expecting the nodes to act in a synchronized manner might not always be appropriate, the used model allows for an asynchronous execution. To represent the fact that distances between nodes cannot always be measured accurately, the parameter μ is introduced. It relates the resulting approximation factor to the relative measurement errors. Robustness is also added to the algorithms; if during the algorithms' execution some nodes do not execute the algorithms correctly for some rounds, the computed solution will (at the price of an increased number of rounds) still have the desired approximation factor (i.e., no restart of the algorithm is required). The algorithms can also deal with changes that might occur when nodes change their positions or are removed/added to the network. We show that such a change only forces our algorithms to update the role of nodes close by (i.e., nodes far away from this change are not affected).

Even though our algorithms were developed with sensor networks in mind (i.e., problem instances where the given metric is Euclidean), all results are valid for general metrics.

3.1.4 Related Work

In the following subsection, we present results that are strongly related to our two local approximation algorithms for Facility Location in sensor networks: in [54], Pandit et al. present an algorithm yielding a 7-approximation and a running time of $O(\log(n))$ where n is the number of nodes. Their approximation factor can be improved to $(3+\varepsilon)$ by changing the incrementation factor used in their algorithm from 2 to $(1+\varepsilon)$ and bounding the difference between the highest and the lowest facility cost by a polynmial in n. A similar result to [54] is presented by Blelloch et al. in [10]. Instead of the $\mathcal{CONGEST}$ model, they use a PRAM model, as they are more interested in the number of operations performed by their algorithms than the aspect of communication/locality. They achieve a $(3.722+\varepsilon)$ approximation with a running time of $O(\log^2_{1+\varepsilon}(n))$ rounds by parallelizing the greedy algorithm of Jain et al. [35]. This is later improved by the same authors to a $(1.861+\varepsilon)$ approximation with an increased running time of $O(\log^4(n))$.

In our opinion, our algorithms are more fitting for sensor networks than the algorithms listed above. This is due to their simplicity, asynchronous execution, and robustness when dealing with errors/changes to the network. The main difference is that they do not need to be restarted when the network changes, but instead have a natural distributed way to update their solution in a very local manner (i.e., only affecting close by nodes).

In [55], Pandit et al. present a $\log^*(n)$ rounds algorithm that, contrary to ours, requires a *Unit Disc Graph* and the graph's geometrical representation for its execution. This algorithm is very well suited for sensor networks in which the nodes posses additional geometric data regarding their positions.

The approach by Mettu and Plaxton [48], on which our algorithms are based on, has been successful in a lot of other settings as well: in the kinetic setting [17], in game theoretic settings [53], for algorithms working in sublinear time [7], for the uniform facility location problem [30] (i.e., opening each facility incurs the same costs), and when confronted with perpetual changes to the problem instance [18]. Compared to [18], we improve the approximation factor and introduce a more realistic model of computation. The model used in [18] guarantees that at any point in time only a single node has been active. This means that there is no need to resolve conflicts between nodes in [18], which is not the case in our problem formulation.

3.1.5 Structure of the Chapter

This chapter presents two local approximation algorithms for Facility Location. Starting in Section 3.2, we introduce the *radius* and the *invariant*. These two concepts are key building blocks for both of our algorithms. They are also used for the analysis of our algortihms' approximation factors. Then, in Section 3.3 we describe a distributed

Maximal Independent Set algorithm, which is used as a subroutine by our algorithms. Section 3.4 is the most important part of this chapter as both algorithms and their runtime analysis are presented here. In Section 3.5, we discuss how our algorithms deal with ongoing changes in the network. Finally, in Section 3.6 we end this chapter with concluding remarks and possible future work.

3.2 The Radius and the Invariant

In this section, we introduce the *radius* and the *invariant*. These two concepts are used by both our distributed approximation algorithms defined in Section 3.4. The radius is a value used to determine how beneficial it is to assign the facility role to a node, whereas the invariant describes a specific condition of a node. This condition depends on the node's current role and radius value, but also on the roles and radius values of its neighboring nodes. Now, the general idea of our algorithms is to assign roles to the nodes (i.e., compute a solution to our Facility Location problem) in such a way that the invariant is fulfilled at all nodes. Once all invariants are fulfilled, the resulting role assignment is a good (i.e., constant factor) approximation of an optimal role assignment (see Theorem 3.3).

The next paragraphs describe the radius and the invariant in a detailed way, and also show a relationship between the invariant and the quality of solutions for the Facility Location problem.

Radius. For each node i we define a *radius* value which represents how beneficial it is for i to become a facility (the smaller the value is, the better it is to open the node). The radius of node i is computed based on i's opening costs f_i and on the distances between i and other nodes (the radius is completely independent of the roles assigned to nodes). Nodes with high opening costs tend to have a high radius value, while nodes with a lot of other nodes close by tend to have small radius values. Our algorithms try to find a balance between opening and closing nodes (preferring nodes with a small radius value to be opened, but also making sure that nodes are not opened too close to each other). The radius is computed using the following definition.

Definition 3.1 (Radius)**.** *Given $r \in \mathbb{R}$ and a node i we define the set $B(i,r)$ to contain all nodes j such that $c(i,j) \leq r$. The radius ρ_i of a node i is the unique number satisfying*

$$\sum_{j \in B(i,\rho_i^{1/p})} d_j \cdot (\rho_i - c(i,j)^p) = f_i \,,$$

while the distorted radius $\hat{\rho}_i$, which takes measurement errors into account, satisfies

$$\sum_{j \in B(i,\hat{\rho}_i^{1/p})} d_j \cdot (\hat{\rho}_i - \hat{c}(i,j)^p) = f_i \,,$$

where, in order to define the ball $B(i,\hat{\rho}_i^{1/p})$, the values $\hat{c}(i,j)$ are used.

The best way to get an intuition for the radius computation is to consider Figure 3.1. Here, a simple example is given where all nodes are placed on the Euclidean plane, have the same demands, there is no measurement error, and the exponent $p = 1$.

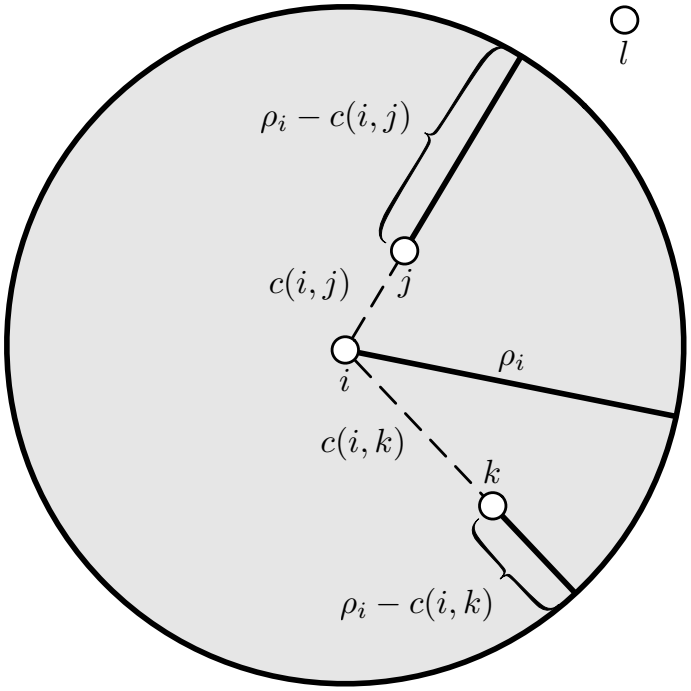

Figure 3.1: Example of the radius definition ($\mathbb{R}^2$, $p = 1$, $d_j = 1$). The radius of node i is the value ρ_i such that the distances to the circular line of all nodes within the circle around i with radius ρ_i, namely $\rho_i + (\rho_i - c(i,j)) + (\rho_i - c(i,k))$, sum up to f_i.

Our algorithms do not use the radius values ρ_i and $\hat{\rho}_i$ directly. Instead, they work with the radius value r_i which is defined to be $\hat{\rho}_i$ rounded to the next power of $(1+\varepsilon)$ (where $\varepsilon > 0$). In order to simplify the description and analysis of our algorithms in Section 3.4 we will use the value $\hat{r}_i = \mu^{2/p} \cdot r_i^{1/p}$.

We have $\frac{1}{\mu(1+\varepsilon)} \cdot r_i \leq \rho_i \leq \mu \cdot r_i$. Also note that for every problem instance with n nodes, there exists a polynomial P such that $1/P(n) < r_i < P(n)$ for all nodes i. Therefore, the number of distinct rounded radius values is logarithmic in n and can be sent using at most $\mathcal{O}(\log_{1+\varepsilon} n)$ bits (even $\mathcal{O}(\log\log_{1+\varepsilon} n)$ is possible if only the exponent is

encoded). Later on, we will use this property to bound the number of rounds that our algorithms require.

Invariant. Our algorithms defined in Section 3.4 use the *invariant* to compute their role assignments. Intuitively, the invariant is a means to allow node i to judge whether the current role assignment (in i's local neighborhood) is good or not. We define the invariant in the following way:

Definition 3.2 (Invariant)**.** *The invariant is said to be fulfilled if and only if the following conditions that depend on the node's role are satisfied:*

- ***Facility Role.*** *If node i is a facility (i.e., $i \in F$), then no other facility j, $j \neq i$ with the following properties exists:*
  - $r_j \leq r_i$
  - $\hat{c}(i,j) \leq 2 \cdot \mu^{2/p} \cdot r_i^{1/p} (= 2 \cdot \hat{r}_i)$
- ***Client Role.*** *If node i is a client (i.e., $i \in C$), then at least one facility j with the following property exists:*
  - $r_j \leq r_i$
  - $\hat{c}(i,j) \leq 2 \cdot \mu^{2/p} \cdot r_i^{1/p} (= 2 \cdot \hat{r}_i)$

The conditions described in the invariant's definition depend on the node's current role. In the case that node i is a facility, we have to check other nodes with the facility role in i's proximity. If among them there exists a facility j with a smaller radius than the radius of i (i.e., a more (at least according to the radius) beneficial facility), then the current solution is (from an intuitive point of view) not good, closing i and connecting it to j might significantly improve the solution. In the other case (where i is a client), we again have to check close by nodes with the facility role. If at least one among them has a smaller radius than i, everything is fine, since i can be connected to it. Otherwise, if all of them have a higher radius, the solution might be not be quite good, since changing the role of i to facility and the roles of close by facilities to clients might improve the solution significantly.

Assume that you are given a role assignment for all nodes in a graph. Consider an arbitrary node i. If i's invariant is not fulfilled, it is always possible to fulfill it by changing only i's role. In a distributed algorithm, each node can try to take care of its own invariant (and fulfill it whenever necessary). The problem one has to deal with here, is that a node changing its role in order to fulfill its invariant, might violate the (until now fulfilled) invariants of other nodes. In Section 3.4, we discuss this problem and show that our distributed algorithms compute solutions (in polylogarithmic time) where the invariant is fulfilled for all nodes.

Approximation guarantee. In this paragraph we show that any solution where the invariant is fulfilled at all nodes has costs very close to those of an optimal solution. This is done with the help of Theorem 3.3. Its proof is very close to the work of Mettu and Plaxton in [48]. Our main contribution here is the introduction of measurement errors and the changes in the cost function.

Theorem 3.3. *Let F_{opt} be an arbitrary optimal solution and F_{inv} be a solution where, for each facility and client, the invariant is satisfied. Then, the cost of F_{inv} can be bounded as follows*

$$\text{cost}(F_{inv}) \leq \mu^4(1+2(1+\varepsilon)^{1/p})^p \,\text{cost}(F_{opt}) \; .$$

Proof. Analogously to [48], we introduce a *charge* value for each node i which is dependent on a solution F, i.e.,

$$\text{charge}(i,F) = c(i,F)^p + \sum_{j \in F} \max\{0, \rho_j - c(i,j)^p\} \; .$$

Note that given a solution F, the sum over all charges is equal to the costs of the solution F, namely

$$\sum_{i \in V} d_i \,\text{charge}(i,F) = \sum_{i \in F} f_i + \sum_{j \in C} d_j \cdot c(j,F)^p \; .$$

The theorem is proven by showing that for a solution F_{inv} satisfying the invariants (in particular a solution computed by our algorithms) and for any solution F (in particular an optimal solution) it holds that

$$\text{charge}(i,F_{inv}) \leq \mu^4(1+2(1+\varepsilon)^{1/p})^p \,\text{charge}(i,F) \; .$$

This is done with the help of the following lemmas:

Lemma 3.4. *Let i be a node, let F be a set of facilities, and let j be a facility. If $c(i,j) = c(i,F)$, then* $\text{charge}(i,F) \geq \max\{\rho_j, c(i,j)^p\}$.

Proof. We distinguish two cases:

- $i \in B(j, \rho_j^{1/p})$, here $\text{charge}(i,F) \geq c(i,j)^p + \max\{0, \rho_j - c(i,j)^p\} \geq \rho_j \geq c(i,j)^p$
- $i \notin B(j, \rho_j^{1/p})$, here $\text{charge}(i,F) \geq c(i,j)^p \geq \rho_j$.

■

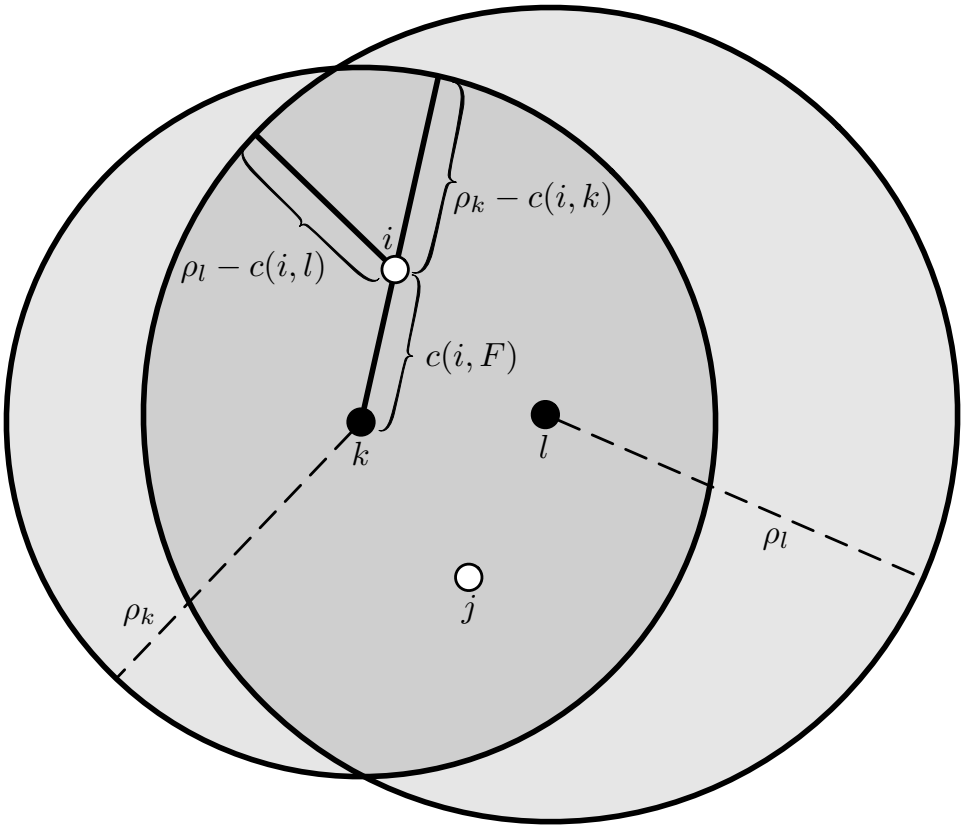

Figure 3.2: Example of the charge definition ($\mathbb{R}^2$, $p = 1$). Here, i and j are client nodes and k and l are facility nodes. The charge of node i is the sum of the distance $c(i,F)$ of i to its closest facility (namely k) and the distances of i to the circular line of all facility circles in F where i lies in (namely k and l, i.e., the charge of i is equal to $c(i,F) + (\rho_k - c(i,k)) + (\rho_l + (c(i,l)))$.

Lemma 3.5. *Let i be a node and let F_{inv} be a set of facilities such that all invariants are satisfied. If $j \in F_{inv}$ with $i \in B(j, \rho_j^{1/p})$, then* $\text{charge}(i, F_{inv}) \leq \rho_j$.

Proof. First, we prove by contradiction that a node cannot be contained in the balls of two different facilities. Assume there are two facilities $j,k \in F_{inv}, j \neq k$, such that $i \in B(j, \rho_j^{1/p})$ and $i \in B(k, \rho_k^{1/p})$. Without loss of generality, assume $\rho_j^{1/p} \geq \rho_k^{1/p}$. Then, the invariant at j must be violated, since there is another facility k within distance $c(j,k) \leq c(j,i) + c(i,k)$ (by the triangle inequality) and

$$\begin{aligned} c(j,i) + c(i,k) &\leq \rho_j^{1/p} + \rho_k^{1/p} \leq 2\rho_j^{1/p} \\ &\leq 2\mu^{1/p} r_j^{1/p} \\ &= 2\hat{r}_j \,. \end{aligned}$$

Thus, $i \in B(j, \rho_j^{1/p})$ and i is in no other facility's ball. Then,

$$\begin{aligned}\text{charge}(i, F_{inv}) &= c(i, F_{inv})^p + \sum_{j \in F_{inv}} \max\{0, \rho_j - c(i,j)^p\} \\ &= c(i, F_{inv})^p + \rho_j - c(i,j)^p \leq \rho_j \,,\end{aligned}$$

since $c(i, F_{inv})^p \leq c(i,j)^p$. ■

Lemma 3.6. *Let i be a node and let F_{inv} be a set of facilities such that all invariants are satisfied. For any facility $j \in F_{inv}$ with $i \notin B(j, \rho_j^{1/p})$ we have* $\text{charge}(i, F_{inv}) \leq c(i,j)^p$.

Proof. We distinguish two cases. In the first case, let there be no facility $k \in F_{inv}$ with $i \in B(k, \rho_k^{1/p})$. It follows that for all facilities $l \in F_{inv}$: $c(i,l) \geq \rho_l^{1/p}$. Thus,

$$\text{charge}(i, F_{inv}) = c(i, F_{inv})^p \leq c(i,j)^p \,.$$

In the second case, let $k \in F_{inv}$ be such that $i \in B(k, \rho_k^{1/p})$. This implies that $c(i,k) \leq \rho_k^{1/p}$. We know that $\hat{c}(j,k) > 2\mu^{2/p} \max\{r_j^{1/p}, r_k^{1/p}\}$ because of the invariants. This yields

$$\begin{aligned}c(i,j) &\geq c(j,k) - c(k,i) \\ &\geq \frac{1}{\mu^{1/p}} \cdot \hat{c}(j,k) - \rho_k^{1/p} \\ &\geq \frac{1}{\mu^{1/p}} 2\mu^{2/p} r_k^{1/p} - \mu^{1/p} r_k^{1/p} \\ &\geq \mu^{1/p} r_k^{1/p} \\ &\geq \rho_k^{1/p} \\ &\geq \text{charge}(i, F_{inv})^{1/p} \,.\end{aligned}$$

■

Using lemmas 3.4, 3.5, and 3.6, for each node we compare the charge of solution F_{inv} to its charge of an optimal solution F_{opt}:

Lemma 3.7. *For any node i and any set of facilities F_{inv}, where all invariants are satisfied, we have*

$$\text{charge}(i, F_{inv}) \leq \mu^4 (1 + 2(1+\varepsilon)^{1/p})^p \cdot \text{charge}(i, F_{opt}) \,.$$

Proof. Let $j \in F_{opt}$ such that $c(i,F_{opt}) = c(i,j)$. By the invariant there exists a facility $k \in F_{inv}$ such that $\hat{c}(j,k) \leq 2\mu^{2/p} r_j^{1/p}$ and $r_k \leq r_j$. We distinguish two cases, either $i \in B(k,\rho_k^{1/p})$, or $i \notin B(k,\rho_k^{1/p})$.

In the first case, by Lemma 3.5, we have $\text{charge}(i,F_{inv}) \leq \rho_k$. Thus, by Lemma 3.4,

$$\begin{aligned}\text{charge}(i,F_{inv}) &\leq \rho_k \leq \mu r_k \leq \mu r_j \\ &\leq \mu^2(1+\varepsilon)\rho_j \leq \mu^2(1+\varepsilon)\cdot\text{charge}(i,F_{opt})\,.\end{aligned}$$

In the second case (as depicted in Figure 3.3), by Lemma 3.6, we have

$$\begin{aligned}\text{charge}(i,F_{inv}) &\leq c(i,k)^p \\ &\leq \Big(c(i,j)+\mu^{1/p}\hat{c}(j,k)\Big)^p \\ &\leq \Big(c(i,j)+2\mu^{3/p} r_j^{1/p}\Big)^p \\ &\leq \Big(c(i,j)+2\mu^{4/p}(1+\varepsilon)^{1/p}\rho_j^{1/p}\Big)^p \\ &\leq \mu^4(1+2(1+\varepsilon)^{1/p})^p\cdot\text{charge}(i,F_{opt})\,.\end{aligned}$$

The last estimation follows by distinguishing whether ρ_j or $c(i,j)^p$ is the biggest element of the term. ■

This concludes the proof of Theorem 3.3. ■

Note that if all invariants hold, $\mu = 1$, and $p = 1$, then the solution is a $(3+\varepsilon)$-approximation, which is an improvement to the 17-approximation of [18].

3.3 Computation of Maximal Independent Sets

Here, we restate an inclusion maximal independent set (MIS) algorithm (see [4, 44, 57]) which was originally introduced by Luby in [44]. This distributed algorithm calculates a MIS in expected $O(\log n)$ communication rounds and forms a building block of our algorithms.

The algorithm (for a pseudocode description refer to Algorithm 1) works as follows on an arbitrary graph: When node i awakes, it *marks* itself with probability of $1/(2\deg(i))$, where $\deg(i)$ is the number of i's neighbors. The next time i awakes, it checks whether a neighbor has *joined* the MIS in the meantime and decides to stay out of the MIS if this is the case. If not, then i decides to join the MIS if there is no other marked neighbor of a higher degree. When calculating the degree, only neighbors are considered which have not yet decided. If two neighboring nodes with same degree are marked, then the

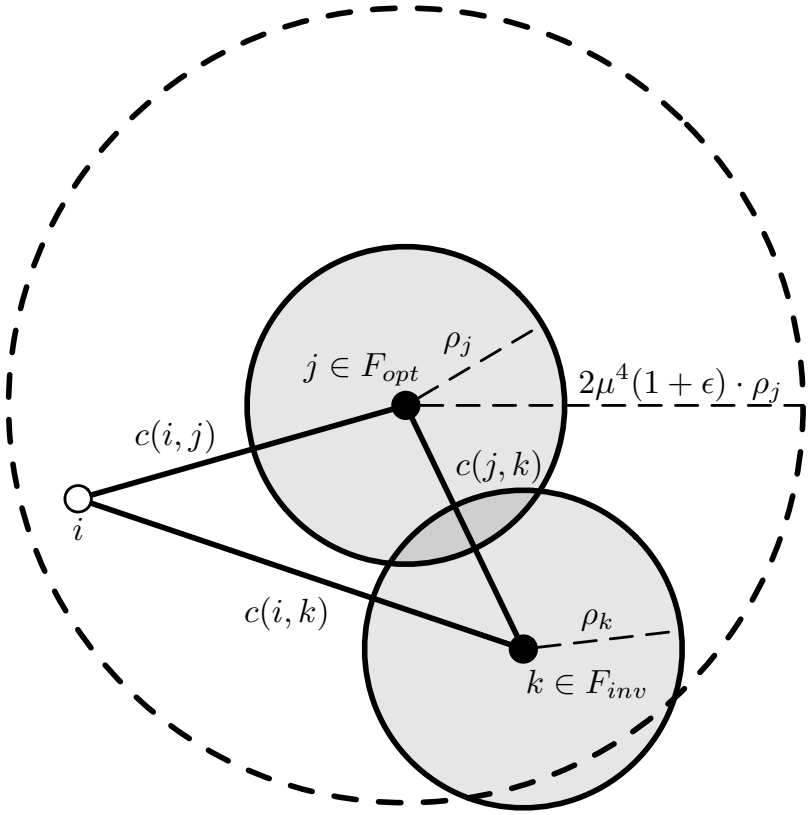

Figure 3.3: Comparison of F_{inv} and F_{opt} in Lemma 3.7. The charge of i in a solution with all invariants fulfilled is not too far away from the charge of i in an optimal solution.

node with highest ID is selected. The decision to join or to stay out of the MIS is final and each node stops executing the algorithm as soon as it has decided.

Note that if some nodes start executing the algorithm belatedly, then the MIS is still calculated in expected $O(\log n)$ communication rounds after the last node has started since previous decisions can only cause a faster termination. This is important for the interleaved calculation of an MIS in Algorithm 3.

3.4 Approximation Algorithms

Due to Theorem 3.3, we know that a solution yields a good approximation if the invariants are fulfilled for every node. What remains to be shown is that our algorithms compute such a solution in a polylogarithmic number of asynchronous rounds.

First note that if the invariant of node i is violated, i can always remedy this by changing its own role. Thus, a straightforward idea for a distributed approximation algorithm would be to allow each node to change its role if its invariant is violated (and by doing so

Algorithm 1 CREATEMIS(G) executed by each node i (Luby's algorithm)

```
1: if ∃ neighbor j of i with j ∈ MIS then
2:    Not-in-MIS ← Not-in-MIS ∪{j};
3:    terminate;
4: else
5:    G' ← subgraph of G induced by {j ∈ V | j ∉Not-in-MIS};
6:    if state(i) =marked then
7:       if ∃ j ∈ G' with state(j) = marked and
         (deg_G'(j) > deg_G'(i) or deg_G'(j) = deg_G'(i) and id(j) > id(i)) then
8:          state(i) ← unmarked;
9:       else
10:         MIS← MIS∪{i};
11:         terminate;
```

possibly violate the invariants of other nodes). As a result, all nodes would switch back and forth between being a client or a facility until they eventually reach a role assignment where all invariants are fulfilled. This is essentially the idea used in [18]. The reason why this approach worked in [18] is that the model used only allowed a single node to be active (i.e., able to change its role) at any point in time. However, this approach is no longer feasible in our (more realistic and more general) model, where nodes can execute the algorithm simultaneously. For example, imagine a problem instance with only two nodes. Both have the facility role and the same radius and are positioned in such a way that they mutually violate each other's invariant. If both become active at the same time, they will both become clients. Now, both their invariants are violated again so they both change their role to facility and everything is repeated. This process never reaches a state where all invariants are fulfilled.

Therefore, we present two algorithms that can deal with this problem and show their approximation factors, expected running time in communication rounds and how they cope with dynamic scenarios. In order to simplify the presentation, we will set $\mu = p = 1$ and refer to the $(\mu^4(1+4\mu^2(1+\varepsilon)^{1/p})^p)$ (respectively, $(\mu^4(1+2(1+\varepsilon)^{1/p})^p)$) approximation algorithm as the $(5+\varepsilon)$ (respectively, $(3+\varepsilon)$) approximation algorithm. These two algorithms are presented in the following two sections.

3.4.1 $(5+\varepsilon)$-Approximation in $O(\log_{1+\varepsilon} n)$ Rounds

As we could see above, nodes that have the same radius, are close to each other, and act at the same time can cause undesired behavior. To be able to avoid this, we first define the *radius graph* as follows:

Definition 3.8. *A* radius graph $G_r = (V_r, E_r)$ *for radius* r *is defined by the nodes* $V_r = \{i \in V | \hat{r}_i = r\} \subseteq V$ *and edges* $E_r = \{\{i, j\} | i, j \in V_r \wedge \hat{c}(i, j) \leq 2r\}$.

A node belongs to exactly one radius graph. We will prevent the nodes with the same radius $\hat{r}$ to influence each other by calculating a MIS on each radius graph $G_{\hat{r}}$. The algorithm proceeds in three steps:

1. Every node i calculates its radius $\hat{r}_i$ (see Section 3.2 for the radius definition) and sends it to all neighbors (nodes at a distance of at most $2 \cdot \hat{r}_{max}$). Now, each node is aware of its neighbors and their radii in the specific radius graph $G_{\hat{r}}$ it belongs to.

2. For each radius graph the nodes compute a maximal independent set (MIS) using Algorithm 1.

3. If a node is not member of a MIS, it becomes a client. Otherwise, whenever a node becomes active, it checks whether its invariant is fulfilled and, if necessary, changes its role such that its invariant is fulfilled again.

As mentioned before, the radius (as well as all other information required to execute the algorithm) can be sent in a single message. A pseudo code description is given by Algorithm 2.

Algorithm 2 FIVEAPPROXIMATION executed by each node $i \in V$

```
r̂_i ← CALCULATEROUNDEDRADIUS i;
CREATEMIS(G_{r̂_i});
if i ∈ Not-in-MIS then
  become client;
else if i ∈ MIS then
  if i is client and not INVARIANT then
    become facility;
  else if i is facility and not INVARIANT then
    become client;
```

The expected number of rounds required to compute a solution, for which the invariant is fulfilled for each node, is $O(\log_{1+\varepsilon} n)$. Step (1) requires a constant number of rounds, while step (2) is finished after $O(\log n)$ communication rounds in expectation (see Section 3.3). Finally, step (3) requires $O(\log_{1+\varepsilon} n)$ rounds. To see this, note that a node i changing its role to facility can only violate invariants of nodes that have a strictly larger radius than $\hat{r}_i$ (nodes with strictly smaller radius are never affected by node i's

role changes and the MIS computation guarantees that nodes with equal radius are not affected either). With each round, the invariants of the nodes with the next higher radius become (and remain) fulfilled (starting with the nodes with the smallest radius). Since there are $O(\log_{1+\varepsilon} n)$ distinct radius values (due to us using radius values rounded to the next power of 2), the claim follows.

Theorem 3.9. *When the invariant holds for all nodes that are in the MIS and all nodes not in the MIS are clients, then the resulting set of facilities yields a* $\mu^4(1+4\mu^2(1+\varepsilon)^{\frac{1}{p}})^p$*-approximation.*

Proof. Since only a node that is a member of the MIS can be a facility, the invariant is fulfilled for every facility. The invariant of clients that are not in the MIS is not necessarily fulfilled. Consider a client i with a violated invariant. There must be another client j with the same radius in distance $2\hat{r}_i$ which is part of the MIS, otherwise i itself would be part of the MIS. Since the invariant of j is fulfilled, there is a facility k with a smaller or equal radius in distance $2\hat{r}_j$ of j. Thus, due to the triangle inequality, facility k is at a distance of at most 4 times $\mu^2\hat{r}_i$ of i. Application of Lemma 3.7 yields the desired approximation. ■

3.4.2 $(3+\varepsilon)$-Approximation in $\mathcal{O}(\log^2_{1+\varepsilon} n)$ Rounds

This section presents an algorithm with the improved approximation factor of $\mu^4(1+2(1+\varepsilon)^{1/p})^p$, but increased running time of $O(\log^2_{1+\varepsilon} n)$ communication rounds. In contrast to the algorithm before, we want to guarantee that the invariant is fulfilled at every single node, and not just the nodes belonging to one of the MISs.

In order to describe the algorithm, the following states are introduced: undecided, marked, MIS/facility, or no-MIS/client. We say a node has *decided* if it is either in state MIS/facility or no-MIS/client. Also, if a node i is in the state undecided or marked and all neighbors j of i with a smaller radius and within distance $c(i,j) \leq 2r_i$ have decided, then we say this node is *playing*.

Theorem 3.10. *All nodes have decided after* $O(\log_{1+\varepsilon} n \cdot \log n)$ *communication rounds (in expectation). The resulting solution is a* $\mu^4(1+2(1+\varepsilon)^{1/p})^p$*-approximation.*

Proof. The state diagram in Figure 3.4 and Algorithm 3 illustrate the nodes' behavior. At the beginning, each node i is in the state undecided and calculates its radius $\hat{r}_i$. Once a node enters the state MIS/facility (resp. no-MIS/client) it never changes its state again and has the facility (resp. client) role in the final solution. A node that is in the marked or undecided state changes its state to no-MIS/client if by assuming the client role its invariant is fulfilled (i.e., the change of its role to client fulfills its invariant). A node i,

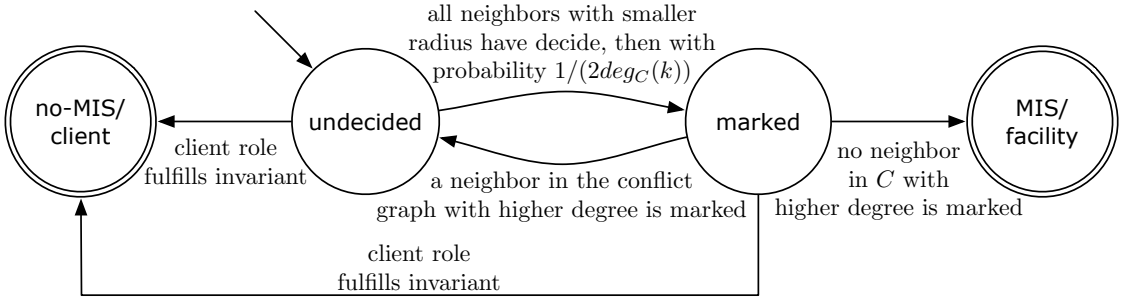

Figure 3.4: A state diagram showing all possible states and their transitions in Algorithm 3.

that cannot fulfill its invariant by assuming the client role, starts computing a MIS once all nodes j with $\hat{r}_j < \hat{r}_i$ within distance $2 \cdot \hat{r}_i$ of i have decided. The MIS is computed on a *conflict graph* consisting of all playing nodes which is a subgraph of the radius graph defined in Section 3.4.1.

Definition 3.11. *A* conflict graph $C_r(t) = (V_r(t), E_r(t))$ *for radius r at the beginning of round t is defined by the vertices* $V_r = \{i \in V | r_i = r \wedge i \text{ is playing at beginning of round } t\} \subseteq V$ *and edges* $E_r = \{\{i, j\} | i, j \in V_r \wedge c(i, j) \leq 2r\}$.

Once a node enters the MIS, it changes its state to MIS/facility. Otherwise (if one of its neighbors in the conflict graph joined the MIS and is therefore a facility), its state is changed to no-MIS/client.

We prove by induction over the radius values that once a node has decided to change its state to MIS/facility (i.e., it becomes a facility) or no-MIS/client (i.e., it becomes a client), its invariant will never be violated again.

All nodes with the smallest radius $\hat{r}_{small}$ start playing and compute a MIS (which takes $O(\log n)$ expected rounds). If a node of this radius enters the MIS, it is a facility. No other node with this radius within distance $2r_{small}$ will ever enter the MIS, and therefore its invariant will always be fulfilled. Each node j which is not in the MIS is within distance $2\hat{r}_{small}$ of a node contained in the MIS and therefore within distance $2\hat{r}_{small}$ of a facility node. Thus, such a node's invariant is, and will remain, fulfilled.

Consider all nodes with radius $\hat{r}' > \hat{r}_{small}$. Assume that in some round the nodes with radius less than $\hat{r}'$ have decided and do not change their roles anymore. Then, we have two cases for node i with radius $\hat{r}'$. First, there is a facility j with radius less $\hat{r}'$ within distance $2\hat{r}'$ (i.e., assuming the client role fulfills the invariant of i). By the induction hypothesis facility j will never change its state again and thus the invariant of node i will not be violated in future rounds. Second, there is no facility j within distance $2\hat{r}'$. Here, using the same argumentation as for the nodes with the smallest radius (i.e., nodes with radius $\hat{r}'$ compute a MIS on their conflict graph), we can state that the invariant of i is

Algorithm 3 THREEAPPROXIMATION executed by each node $i \in V$

$\hat{r}_i \leftarrow$ CALCULATERADIUSi;
if state(i) = undefined or (state(i) = client and not INVARIANT) or
(state(i) = facility and not INVARIANT **then**
  state(i) $\leftarrow$ undecided;
if INVARIANT **then**
  state(i) $\leftarrow$ client;
$C \leftarrow$ conflict graph $G_{\hat{r}_i}(t)$;
if state(i) = marked **then**
  if $\exists\ j \in C$ with state(j) = marked and ($\deg_C(j) > \deg_C(i)$ or
  $\deg_C(j) = \deg_C(i)$ and $\mathrm{id}(j) > \mathrm{id}(i)$) **then**
    state(i) $\leftarrow$ undecided;
  else
    state(i) $\leftarrow$ facility;
if state(i) = undecided and $\forall\ j \in V$ with $\hat{r}_j < \hat{r}_i$ and $c(j,i) \leq 2\hat{r}_i$ and
state(j) $\in$ facility, client **then**
  with probability $\frac{1}{2\deg_C(i)}$: state(i) $\leftarrow$ marked;

fulfilled after $O(\log n)$ rounds. Since there are $O(\log_{1+\varepsilon} n)$ different radius values, the expected number of communication rounds is $O(\log_{1+\varepsilon} n \cdot \log n)$. ■

3.5 Dealing with Dynamics

Until now, we only considered a static scenario where the distances between nodes, facility costs and client demands do not change over time. In this section we introduce a dynamic scenario where such changes are possible. Starting with a problem instance and a role assignment such that the invariant is satisfied for each node, we modify the instance (e.g., by changing the distance between two nodes) such that the invariant is violated at a single node (we call this "triggering an event at a node"). We want to enable our algorithm to deal with such a change and analyze its effect on the roles of other nodes. To be able to react to such changes, the algorithms have to be slightly extended.

- Regarding the $(5+\varepsilon)$-algorithm, we have two cases to consider. First, assume the change did not invalidate the MIS on the nodes' radius graph. If the affected node is part of the MIS, the node needs just to correct its invariant (which will possibly trigger role changes of other MIS nodes). Otherwise, the node does nothing. Second, the change invalidates (by adding or removing edges in the radius graph)

the MIS. Here, nodes at which the MIS property is violated change their states to unmarked (and thus start executing the $(5+\varepsilon)$-algorithm again).

- The modification of the $(3+\varepsilon)$-algorithm is simpler: here, whenever the invariant of a *decided* node (i.e., in state MIS/facility or no-MIS/client) is violated, it just needs to change its state back to undecided.

The following lemma, which is an adaptation of the proof in [18], states that, given a solution satisfying the invariants of all nodes, both modified algorithms will compute a new solution with the desired approximation factor. Also, only nodes with a small distance to the node where the change occurred can have a different role in this new solution.

Lemma 3.12. *A node i can only be affected by an event if it is triggered at a node which is at most in distance $4\mu^{3/p}(1+\frac{1}{\varepsilon})r_i$ from i.*

Proof. Let k be the node at which an event is triggered. Let $e_i \cdot r_i$ be the maximal range around k in which nodes with radii at most $r_i = (1+\varepsilon)^i \cdot r_k$ are affected by the state change of k. First, we give an upper bound for e_0 which implies that $r_0 = (1+\varepsilon)^0 \cdot r_k = r_k$. Nodes with radius $< r_0$ cannot be affected by the state change of k, as the invariant only depends on nodes with smaller or equal radii. If k changes its state from closed to opened, then this decision cannot violate the invariant of other nodes with radius r_0 because then the invariant of k would have been fulfilled (note that we used the fact that the distorted distances are symmetric). If k changes its state from opened to closed, then this change affects only nodes with radius r_0 within distance $\leq 2\mu^{3/p} r_k$ since they might underestimate their distance to k by a factor of $\mu^{1/p}$ when checking their invariant. Therefore, we have $e_0 = 2\mu^{3/p}$.

Now we describe the step from e_{i-1} to e_i and claim $e_i \leq \frac{e_{i-1}}{1+\varepsilon} + 4\mu^{3/p}$ for $i > 0$. By definition of e_{i-1}, k nodes with radii at most $r_{i-1} = (1+\varepsilon)^{i-1} \cdot r_k$ can be at a distance of at most $e_{i-1} \cdot r_k$ from k. Let m be such a node and let l be a node with radius $r_i = (1+\varepsilon)^i \cdot k$ that changes its role due to a role change of m. This node l must be within distance $2\mu^{3/p} r_i$ of m. If l needs to be opened, then m must have closed. No invariant of nodes with radius r_i is affected. If l needs to be closed, then m must have opened. Another node n with the same radius r_i might have to open and n can be in distance of at $2\mu^{3/p} r_i$ of l. Again, the opening of n cannot violate the invariant of a node with the same radius. Therefore, $c(k,n) \leq e_{i-1} \cdot r_k + 2 \cdot 2\mu^{3/p} r_i \leq (\frac{e_{i-1}}{1+\varepsilon} + 4\mu^{3/p}) r_i$ and thus $e_i \leq \frac{e_{i-1}}{1+\varepsilon} + 4\mu^{3/p}$.

Finally, the recurrence can be solved,

$$\begin{aligned} e_i &\leq \frac{e_{i-1}}{1+\varepsilon} + 4\mu^{3/p} \\ &= \frac{e_0}{(1+\varepsilon)^j} + \sum_{j=0}^{i-1} \frac{4\mu^{3/p}}{(1+\varepsilon)^j} \\ &= \frac{e_0}{(1+\varepsilon)^i} + 4\mu^{3/p} \frac{1-\frac{1}{(1+\varepsilon)^i}}{1-\frac{1}{1+\varepsilon}} \\ &= 4\mu^{3/p} \cdot \frac{1+\varepsilon}{\varepsilon} - \frac{4\mu^{3/p}\frac{1+\varepsilon}{\varepsilon} - e_0}{(1+\varepsilon)^i} \\ &\leq 4\mu^{3/p}\left(1+\frac{1}{\varepsilon}\right) . \end{aligned}$$

■

Notice that a node i can only be affected by a change occurring at node j, if $c(i,j) \in O(\hat{r}_i)$. Also, $\hat{r}_i \in O(f_i/d_i)$. This means that, if f_i and d_i are independent of the number of nodes (which is a reasonable assumption in a sensor networks scenario) the area of effect of a change is independent of the number of nodes. Thus, the solution computed by our algorithms is, in practical scenarios, very stable (e.g., the removal/addition of a node from/to the network will not trigger role changes on almost all of the other nodes).

3.6 Conclusion & Future Work

We converted the sequential algorithm by Mettu and Plaxton [48] to be applicable in a distributed sensor nodes scenario. The two resulting algorithms have constant approximation factors ($(3+\varepsilon)$ respectively $(5+\varepsilon)$) and polylogarithmic running times ($O(\log^2 n)$ respectively $O(\log n)$). One of these algorithms retains the sequential algorithm's approximation factor up to an arbitrarily small constant ε. This is arbitrarily close to the best approximation factor one can achieve using our approach (computing the radius value and switching the roles until the invariant is satisfied at all nodes). Since both algorithms are designed for the same model and use very similar operations, we can choose which one of them we want to used based only on the desired runtime/approximation guarantee. Although better approximations can be achieved in distributed settings (e.g., an approximation of 1.86, as explained in Section 3.1.4), our algorithms adapt to changes in the network in a very efficient way (since they are formulated as a simple infinite process). This is in contrast to all other distributed algorithms known to us (e.g., see [10, 11, 54]), since they have to be restarted each time the network changes.

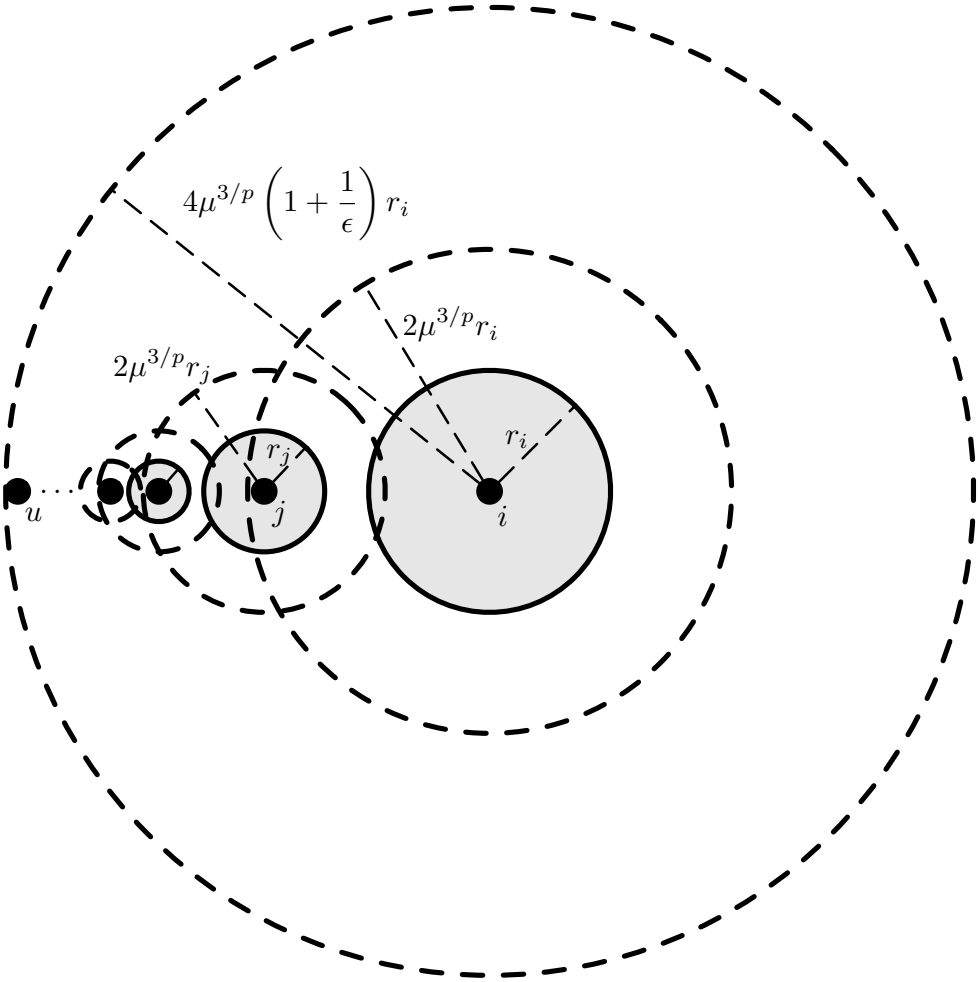

Figure 3.5: Illustration of Lemma 3.12. Node *i* can only be affected by an event which is triggered at a node within distance: $2\mu^{3/p}r_i + 2\mu^{3/p}r_j + \ldots + 2\mu^{3/p}r_u \leq 4\mu^{3/p}\left(1+\frac{1}{\varepsilon}\right)$.

Such a restart can be very costly (e.g., it requires a lot of time and/or energy) and is undesired, maybe even infeasible, in a sensor networks scenario.

Concerning dynamics, we only analyzed and bounded the distance between the location at which a change in the network occurred and the nodes affected by this change (i.e., nodes that changed their role because of the change in the network). Another aspect one could try to analyze is the number of nodes that are affected by it. Here, limiting the problem to include only graphs that can be embedded into the Euclidean plane might be helpful. With this restriction, we showed in [18] (for a predecessor algorithm to the two algorithms presented in this chapter) that at most $O(\log^2 n)$ nodes are affected by a change. Another interesting challenge is to find bounds on the number of role changes performed as a result of network dynamics.

CHAPTER 4

The Facility Location Problem in Distributed Settings

4.1 Introduction

This Chapter, similar to Chapter 3, also considers the Facility Location problem in a distributed scenario, nevertheless its focus is quite different. Our objective here is to design an distributed algorithm with an approximation ratio that is as close as possible to the best achievable ratio. Aspects concerning the applicability in wireless sensor networks (e.g., dynamics, asynchronous execution, locality) are, contrary to Chapter 3, no longer of high importance. In order to be able to achieve an approximation factor that is better than the ones of our algorithms presented in the previous chapter, we allow a higher running time (but still require that our algorithm terminates in sub-linear number of rounds).

Later in this chapter it will become apparent that, in order to achieve a high quality approximation, it is necessary to develop a selection mechanism that allows us to determine which facilities (from a set of "eligible" facilities) we have to open. Constructing such a selection mechanism is by itself an interesting problem that can be considered independently from the Facility Location problem. Thus, a major part of this chapter is dedicated to the description and analysis of the selection mechanism that we developed.

The remainder of this introductory section will be structured as follows. First, we define the Facility Location problem that is studied in this chapter (see Subsection 4.1.1). This definition is slightly different from the one in Chapter 3, as we use a complete *bipartite* graph instead of a complete graph. Here, we do not assign the roles of *client* or *facility* to a node, but instead have a fixed set of client and facility nodes; we thus only have to decide which facility node we want to open (one could say that we asign the roles *open* or *closed* to each facility). We continue the introduction with Subsection

4.1.2 where we present our results. Subsection 4.1.3 states related work that is especially important for our distributed setting (i.e., work with focus on approximation quality). Finally, in Subsection 4.1.4 we give an overview about the structure of this entire chapter.

4.1.1 Problem Definition & Notation

We consider the *metric uncapacitated Facility Location* problem in a distributed setting. Here, we are given a complete bipartite graph $G = F \cup C$ consisting of a set of *facilities* F and a set of *clients* C. To each facility $i \in F$ an opening cost $f_i \in \mathbb{R}_{\geq 0}$ is assigned. Each edge $\{i, j\}$ in G is weighted with the value $c_{ij} \in \mathbb{R}_{\geq 0}$ that represents the costs of connecting client j with facility i. The objective is to determine a subset of the facilities to be opened and connect every client to at least one open facility in such a way that the sum of the opening costs and connection costs is minimized. The linear program representation of this Facility Location problem is as follows:

Facility Location IP

$$
\begin{array}{lrcll}
\text{minimize} & \displaystyle\sum_{i \in F} f_i y_i + \sum_{i \in F, j \in C} c_{ij} x_{ij} & & & \\
\text{subject to} & \sum_{i \in F} x_{ij} & \geq & 1 \qquad j \in C & (1) \\
& y_i - x_{ij} & \geq & 0 \qquad i \in F, j \in C & (2) \\
& x_{ij} & \in & \{0,1\} \quad i \in F, j \in C & \\
& y_i & \in & \{0,1\} \quad i \in F &
\end{array}
$$

The variable y_i indicates whether facility i is open ($y_i = 1$) or closed ($y_i = 0$). The other indicator variable x_{ij} has the value 1 if the client j is connected to facility i, and 0 otherwise. The constraints (1) guarantee that each client is connected to at least one facility, while the constraints (2) make sure that a client can only be connected to an open facility. The problem we consider is *metric*, since the values c_{ij} are required to satisfy the triangle inequality (i.e., $\forall i, j, i', j' : c_{ij} \leq c_{ij'} + c_{j'i'} + c_{i'j}$), and *uncapacitated*, since an arbitrary number of clients can be connected to an open facility. Furthermore, for the sake of presentation we assume all f_i, c_{ij} to be normalized such that the smallest non-zero value is 1.

The dual program of the Facility Location problem will be used in the description of

our approximation algorithm. It is formulated in the following way:

Dual of the Facility Location LP

$$
\begin{array}{lrcll}
\text{maximize} & \sum_{j \in C} \alpha_j & & & \\
\text{subject to} & \alpha_j - \beta_{ij} & \leq & c_{ij} \quad i \in F, j \in C & (3) \\
& \sum_{j \in C} \beta_{ij} & \leq & f_i \quad i \in F & (4) \\
& \beta_{ij} & \geq & 0 \quad i \in F, j \in C & \\
& \alpha_i & \geq & 0 \quad j \in C &
\end{array}
$$

Intuitively, a α_j variable can be seen as the amount the client j is willing to pay for being connected to a facility. From the point of view of a facility i, α_j is the sum of c_{ij} (the amount j pays for a connection to i) and β_{ij} (the amount it pays for opening i).

Our algorithm is executed in the $\mathcal{CONGEST}$ model (see [57]). A variant of this model was already described in Section 3.1.2 in the previous chapter. Here, we give a short reminder of its formulation and point out the difference between the model used here and the model from the previous Chapter. In the $\mathcal{CONGEST}$ model, algorithms are executed in synchronous send-receive-compute rounds. In a single round, each node sends a message to each of its neighbors in the graph. Note that the messages sent to each neighbor by a single node are not required to contain the same information. Once all nodes have sent their messages, they receive a single message from each of their neighbors. After all the messages have been received, every node is allowed to spend an arbitrary amount of time for computation (i.e., computation is for free and we are only interested in the number of communication rounds). The end of the computation by all nodes marks the start of a new send-receive-compute round. The message size in the $\mathcal{CONGEST}$ model is bounded. We will limit the size of the messages used in our algorithm to $O(\log(n))$ bits. The same limitations were also used in [51], [30] and [54]. This bound is reasonable, because it allows the nodes to send their ID in a single message. Due to this constraint on message size, we restrict the values f_i and c_{ij} in such a way that it is possible to represent them with $O(\log(n))$ bits (i.e., to be able to send them in a single message). Now, the difference between the model used here and the one from Chapter 3 is the lack of asynchronous execution and the use of a bipartite graph.

The fact that the graph our algorithm is executed on is a complete bipartite graph of clients and facilities, means that within a single round each client can communicate with all facilities and each facility can communicate with all clients. Since nodes of the same partition cannot communicate directly with each other, but instead have to use nodes of the other partition as relays, a node requires $\Omega(n)$ rounds in order to gather all information about the graph. In order to see this, consider a single client. It knows the opening cost of all facilities and the distances between them and itself. Each distance/opening

cost is encoded with $\Theta(\log n)$ bits. As there can be $\Theta(n)$ facilities in the graph and only a single message of size $O(\log n)$ bits can be sent in a single round, sending all the information requires $\Omega(n)$ rounds. Note that without a limit on the message size our problem could be trivially solved in four rounds (all information about the graph is gathered at a single node in two rounds, an optimal solution is computed and distributed in two more rounds). We want to stress that, although in our algorithm the nodes communicate with all their neighbors in each round, it is possible to restate the algorithm such that nodes i and j where c_{ij} is "large" (more than $\max_{j\in C}(\min_{i\in F}(c_{ij}+f_i))$) never communicate with each other.

4.1.2 Results

We present a distributed approximation algorithm for the metric uncapacitated facility location problem. It is based on a greedy algorithm by Jain et al. [35] (from here on referred to as *GreedyFL*) and yields a guaranteed (not in expectation) $(1.861+\varepsilon)$-approximation in $O(n^{3/4}\log^2_{1+\varepsilon}(n))$ rounds (with high probability) with $\varepsilon > 0$. Our algorithm is executed in the $\mathcal{CONGEST}$ model on a complete bipartite graph. Although it is strongly related to the *GreedyFL* algorithm, there are (due to the parallel execution) new challenges regarding the selection process that occurs when multiple facilities are eligible for opening. The difficulty is that we require the cost of the solution computed by our algorithm to be always at most $(1.861+\varepsilon)$ times worse than an optimal solution (in contrast to similar work by Blelloch et al. [10], where the approximation factor is increased by factor 2). Moreover, unlike other work (e.g., [56], where the approximation factor is given "in expectation"), our algorithm provides a worst-case guarantee on the approximation factor.

4.1.3 Related Work

The following results concerning the Facility Location problem can be found in the distributed scenario: In [54] Pandit et al. present an algorithm yielding a 7-approximation and a running time of $O(\log(n))$. Their algorithm is a parallel version of the primal-dual algorithm by Jain et al. [34] and is – like our algorithm – executed in the $\mathcal{CONGEST}$ model on a complete bipartite graph with message size limited to $O(\log(n))$ bits. Pandit et al. assume that the difference between the opening costs of the cheapest and the most expensive facility can be arbitrary large. It is reasonable to drop this assumption, since they (as well as we) require that the facility opening costs and distances between clients and facilities are encoded with $O(\log(n))$ bits. By dropping this assumption and modifying the algorithm and analysis by Pandit et al. in a small way (changing a factor from 2 to $(1+\varepsilon)$), it is possible to achieve a $(3+\varepsilon)$ approximation factor in $O(\log_{1+\varepsilon}(n))$

rounds. Further improvement of the factor with this approach is not possible, since the approximation factor of [34], which Pandit et al. parallelized, is 3. This means that their algorithm, while faster than ours, has a worse approximation factor.

In [56] Pandit et al. all presented a technique that can be used to execute greedy Facility Location algorithms (like *GreedyFL*) in parallel in polylogarithmic time. Although they consider a similar problem as we do, their results are quite different from ours: Their approximation factor is $O(1)$ in expectation (their algorithm can produce an arbitrary bad solution, even though this is very unlikely), while we can guarantee an approximation factor of $(1.861+\varepsilon)$ in the worst case. Also, although they do not state their exact approximation factor, the expected approximation factor achieved with their technique cannot, to the best of our knowledge, be decreased below $4c$, where c is the approximation factor of the sequential algorithm (i.e., ≈ 7.444 if used with *GreedyFL*).

A similar result to [56] was presented by Blelloch et al. in [10]. Instead of the $\mathcal{CONGEST}$ they use a PRAM model and achieve a $(3.722+\varepsilon)$ approximation with running time $O(\log^2_{1+\varepsilon}(n))$ by parallelizing the *GreedyFL* algorithm. In order to choose which facilities to open, they use a technique introduced by Rajagopalan and Vazirani [58]. This technique can also be used to execute greedy facility location algorithms (in a distributed manner and polylogarithmic time) and yields a guaranteed approximation factor of $2c$, where c is again the approximation factor of the corresponding sequential algorithm (i.e. ≈ 3.722 if used with *GreedyFL*). Later, they develop a technique for computing fast "nearly" maximal independent sets in [11]. This technique can be used to execute the greedy algorithm by Jain et al. [35] that yields approximation factor of 1.861 (which matches the approximation guarantee of the original sequential algorithm by Jain et al.). Although the algorithm based on [11] guarantees a very good approximation, its running time of $O(\log^4(n))$ is slightly higher than the running time of the algorithms listed above.

Recently, in [9], Berns et al. presented a distributed algorithm for metric Facility Location that runs on a complete graph in the $\mathcal{CONGEST}$ model. Its running time is exceptionally fast, as it subpolylogarithmic ($O(\log\log n * \log^* n)$). For its approximation factor the authors show an upper bound of $O(1)$ (the actual approximation factor is, to the best of our knowledge higher than the factors of the algorithms discusses above).

Other results that also use the bound of $O(\log(n))$ on the message complexity are [51] and [30]. In [51] Moscibroda et al. show that in $O(k^2)$ communication rounds a $O(k(n_F\rho)^{1/k}\log(n_F+n_C))$ approximation in $O(k^2)$ communication rounds (n_F and n_C are the number of facilities, respectably clients, and ρ a coefficient dependent on instance parameters) can be achieved in the more general *non-metric* Facility Location problem.

4.1.4 Structure of the Chapter

In Section 4.2, we give a detailed description of our approximation algorithm (see Subsection 4.2.1) and prove its approximation factor by generalizing the techniques and results of [35] by Jain et al. (see Subsection 4.2.2). Our main contribution is Section 4.3. Here, we deal with the problem of selecting facilities to be opened that arises due to the parallel execution of the *GreedyFL* algorithm. We present a selection mechanism in Subsection 4.3.1 and show that it terminates in sublinear time in Subsection 4.3.2.

4.2 The Distributed Approximation Algorithm

Our parallel algorithm is based on the *GreedyFL* algorithm by Jain et al. [35]. As stated above, it is executed in the $\mathcal{CONGEST}$ model on a complete bipartite graph. We will first describe the algorithm and then give an analysis of its approximation factor and running time.

4.2.1 Algorithm Description

Due to the completeness of the bipartite graph, in a single synchronous round each client can send to and receive a message from all facilities. The same applies analogously to each of the facilities. During the algorithm's execution, clients and facilities can be in various states: clients can be *not connected* or *connected*, while facilities can be *closed*, *currently-paid*, or *open*. Intuitively, *currently-paid* facilities compete with each other for the permission to change their state to *open*. All clients start in the *not connected* state and all facilities in the *closed* state. When a client j changes its state to *connected*, it is assigned to an *open* facility i ($\varphi(j) := i$) and never changes its state again. A *closed* facility can only change its state to *currently-paid*, while a *currently-paid* facility can either become *closed* or *open*. Once a facility is *open*, it never changes its state. The algorithm terminates, once all clients are *connected*. The solution returned by our algorithm is the set of all *open* facilities and the assignment φ. Each facility knows now whether it is open or not and each client knows the facility it is assigned to.

Algorithm 4 EXECUTED BY CLIENT j

1: $\forall i \in F$: SEND$[\alpha_j, status(j)]$
2: Execute Algorithm 7 until there are no *currently-paid* facilities left
3: **if** $status(j) =$ *not connected* **then**
4: Compute $\alpha_j := (1+\varepsilon)\alpha_j$

The algorithm is executed in a loop and each passing of the loop is referred to as a *phase*. The behavior of a client during a phase is described in Algorithm 4, while the behavior of a facility is described in Algorithm 5. Facilities and clients interact with each other by transmitting messages; to each SEND operation there exists a corresponding RECEIVE operation.

The idea of the algorithm is to assign to each client j a variable α_j initialized with 1. This variable represents the amount a client is willing to pay for being connected to a facility and for opening this facility in the current phase (i.e., the α_j variable of the dual program). Each *not connected* client j increases this variable in every phase by multiplying it with $(1+\varepsilon)$ where ε is an arbitrary small constant greater than 0 (fixed at the start of our algorithm). After α_j is increased, all clients send their current α_j to all facilities.

Algorithm 5 EXECUTED BY FACILITY i

$\forall j \in C:$ RECEIVE$[\alpha_j, status(j)]$
if $(status(i) = closed)$ **then**
  $U := \{j | j \in C \wedge status(j) = not\ connected\}$
  Compute $coveredCost := \sum_{j \in U} \max(0, \alpha_j - c_{ij})$
  if $(f_i \leq coveredCost)$ **then**
    $status(i) := currently\text{-}paid$
Execute Algorithm 6 until there are no *currently-paid* facilities left

Let U be the set of all *not connected* clients and $\beta_{ij} := \max(\alpha_j - c_{ij}, 0)$. We say that client j *contributes* to facility i, if $\beta_{ij} > 0$ (the client pays c_{ij} for being connected to and β_{ij} for the opening of facility i). Upon receiving the α_j variables, every *closed* facility i computes $\sum_{j \in U} \beta_{ij}$, which represents the amount *not connected* clients are willing to contribute to the payment of opening facility i. If the contribution to the opening of facility i (i.e. $\sum_{j \in U} \beta_{ij}$) reaches a point such that $f_i \leq \sum_{j \in U} \beta_{ij}$, it changes its status to *currently-paid*.

Opening all the *currently-paid* facilities could result in a very bad approximation, since it is possible that a single *not connected* client j is contributing to an arbitrary number of *currently-paid* facilities (i.e., $\beta_{i,j}, \beta_{i',j} > 0, i \neq i'$) and opening just one of these facilities would be sufficient. Thus, a selection procedure is needed to determine which *currently-paid* facilities are going to be permanently opened (change their status to *open*) and which of them will remain closed in this phase (changing their status from *currently-paid* to *closed*). Furthermore, it is required to determine which client is connected to which facility. All of this is achieved by the interaction of Algorithm 6 (from the point of view of facility i) and Algorithm 7 (from the point of view of client j). In each iteration of Algorithm 6 and 7, a subset of *currently-paid* facilities is chosen to be

Algorithm 6 EXECUTED BY FACILITY i

$R_i :=$ uniformly distributed random number in $[0,1]$
$\forall j \in C:$ SEND$[R_i, status(i)]$
$\forall j \in C:$ RECEIVE$[S_j, status(j)]$
Define $T_i := \max_I(S_j)$ where $I := \{j | j \in C, status(j) = not\ connected, \alpha_j \geq c_{ij}\}$
$thisRoundOpened_i := false$
if $(T_i = R_i \wedge status(i) = currently\text{-}paid)$ **then**
  $status(i) :=$ open
  $thisRoundOpened_i := true$
$\forall j \in C:$ SEND$[status(i), thisRoundOpened_i]$
$\forall j \in C:$ RECEIVE$[status(j)]$
$I' := \{j | j \in C, status(j) = not\ connected, \alpha_j \geq c_{ij}\}$
if $(status(i) = currently\text{-}paid \wedge \sum_{j \in I'} \max(0, \alpha_j - c_{ij}) < f_i)$ **then**
  $status(i) = closed$

opened. The development and analysis of these two algorithms is the main contribution of this chapter, as they are responsible for selecting the facilities (which are going to be opened) in a fast and efficient way. The algorithms operate by generating a random number at each facility and then opening a facility, if its number is the highest one in its neighborhood (the neighborhood of a facility i consists of all facilities i', such that there is at least one client that contributes to i and i'). Only $O(\log(n))$ bits are used to represent the generated random number, which is sufficient to guarantee that, with high probability, no two facilities generate the same number. The main objective of Algorithm 6 and 7 is to establish the following fact:

Fact 4.1. *In every phase, after the iterative execution of Algorithm 6 and 7 terminates, the following is true for all facilities:*

(i) No facility is currently-paid.

(ii) If facility i is open, *then* $\sum_{j \in U'} \beta_{ij} \geq f_i$, *where* $U' = \{j | \varphi(j) = i\}$.

(iii) If facility i is closed, *then* $\sum_{j \in U} \beta_{ij} < f_i$, *where* $U = \{j | state(j) =$ not connected$\}$.

Intuitively, (ii) means that a facility is only opened, if all clients connected to it can (together) pay for its opening, and (iii) means that, after a phase ended, there is no *closed* facility to which clients contribute enough to pay for its opening. The fact that a *closed* facility with $\sum_{j \in U} \beta_{ij} \geq f_i$ cannot enter the next phase, as implied by (iii), is curcial for (the later introduced) Lemma 4.3.

Note that after the algorithm's termination the cost of the solution computed by our algorithm lies in the interval $[\sum_{j \in C} \alpha_j/(1+\varepsilon), \sum_{j \in C} \alpha_j]$. This is due to the following

Algorithm 7 EXECUTED BY CLIENT j

1: $\forall i \in F:$ RECEIVE$[R_i, status(i)]$
2: Define $S_j := \max_J(R_i)$ where $J := \{i | i \in F, status(i) = currently\text{-}paid, \alpha_j \geq c_{ij}\}$
3: $\forall i \in F:$ SEND$[S_j, status(j)]$
4: $\forall i \in F:$ RECEIVE$[status(i), thisRoundOpened_i]$
5: $J' := \{i | i \in F, status(i) = \text{open}, \alpha_j \geq c_{ij}, thisRoundOpened_i = true\}$
6: **if** $(status(j) = not\ connected \wedge J' \neq \emptyset)$ **then**
7: $status(j) = connected$ {There is exactly 1 element in J'}
8: $\varphi(j) = i$ $(i \in J')$
9: $\forall i \in F:$ SEND$[status(j)]$

facts: Each client is connected to exactly one facility and this facility is *open*. Since $\beta_{ij} = \max(\alpha_j - c_{ij}, 0)$ a client j contributes to the opening of a facility i only after α_j covered the costs c_{ij} of connecting i to j. Also, since we require that only after $f_i \leq \sum_{j \in U} \beta_{ij}$ facility i changes its state to *open*, the opening of i is completely paid for and since U is the set of all *not connected* clients, it is made sure that a client contributes to the opening of at most one facility.

The running time of the entire algorithm is bounded by $O(n^{3/4} \log^2_{1+\varepsilon}(n))$ rounds: The number of phases is $O(\log_{1+\varepsilon}(n))$, since in the worst case a client has distance $O(poly(n))$ to its nearest facility and this facility has opening costs of $O(poly(n))$ (the representation of c_{ij} and f_i is limited by $O(\log(n))$ bits). Thus, increasing the $\alpha_j \geq 1$ value iteratively by multiplying it with $(1+\varepsilon)$, where $\varepsilon > 0$, results in $O(\log_{1+\varepsilon}(n))$ rounds. Later, we will show that with high probability Algorithm 6 and 7 terminate after $O(n^{3/4} \log(n))$ rounds. The value chosen for ε not only determines the runtime, but also the approximation factor, which is $1.861(1+\varepsilon)^2$.

4.2.2 Analysis of the Approximation Factor

Intuitively, the *GreedyFL* algorithm can be seen as continuously and simultaneously increasing the α_j values. This allows *GreedyFL* to open a facility i in the exact moment it is paid for (i.e., when $\sum_{j \in U} \beta_{ij} = f_i$); a property which is essential for proving its approximation factor. In contrast, our algorithm increases the α_j values in discrete steps and thus, in general, it may occur that $\sum_{j \in U} \beta_{ij} \geq f_i$. The goal of this section is to show that – even though our algorithm may yield a very different solution from the sequential algorithm – the costs of the optimal and our solution do not differ too much. Namely, we show that our approximation factor is at most $(1+\varepsilon)^2$ times greater than the approximation factor of *GreedyFL*. Our proof uses similar arguments as, and is based on, the technique by Jain et al. ([35]). Still, it turns out that in order to apply the results

of Jain et al., we need to get a deeper insight into the structural properties of the linear programs involved.

The α_j and β_{ij} values computed by our algorithm can be interpreted as a solution to the dual program of the Facility Location program presented in Subsection 4.1.1. These values have two properties that are of importance to us. First, they do not necessarily represent a feasible dual solution: our algorithm opens a facility when $\sum_{j \in U} \beta_{ij} \geq f_i$ (summing over all *not connected* clients) and not, as required by constraint (4), over all the clients. Second, the cost of this (most probably infeasible) dual solution (i.e., $\sum_{j \in C} \alpha_j$) is exactly equal to the cost of the primal solution computed by our algorithm. In order to see this, consider a facility i and the set C^i of all clients that are connected to i in our primal solution. We can use the α_j values of the clients in C^i to pay for their connection costs, and additionally for opening facility i (i.e., show that $\sum_{j \in C^i} \alpha_j = f_i + \sum_{j \in C^i} c_{ij}$). Since we have $\alpha_j = \beta_{ij} + c_{ij}$ for $j \in C^i$, the connection costs of the clients are paid for and we can use the remainder (i.e., the β_{ij} values) to pay i's opening costs, as our algorithm made sure that $f_i = \sum_{j \in C^i} \beta_{ij}$. It is save to consider each facility, and the clients that are connected to it, in a separate way (since a client withdraws all its contribution to other facilities once it is assigned to one).

Now, if we choose γ such that $\alpha_j^* := \frac{\alpha_j}{\gamma}$ and $\beta_{ij}^* := \alpha_j^* - c_{ij}$ constitute a feasible dual solution, the Duality Theorem yields that our solution is by a factor of at most γ more expensive than an optimal solution. For their original, sequential algorithm Jain et al. show that, as long as you choose a value greater than 1.861 for γ, the α_j^* and β_{ij}^* values are a feasible solution (see [35]). They denoted the used technique as dual fitting with a factor-revealing LP. We will construct such a factor revealing program (see *FacRev-Dist*) for our algorithm and establish a structural connection between it and the LP by Jain et al. (see *FacRev-Jain*) for their sequential algorithm (*GreedyFL*).

Next, we want to explain how to find an appropriate γ with the help of the factor-revealing LP. Consider a single facility i and assume, that w.l.o.g. $\alpha_j \geq \gamma c_{ij}$ holds only for the first k clients and that $\alpha_1 \leq \alpha_2 \leq \ldots \leq \alpha_k$. Our objective now is to find the minimum value for γ such that $\sum_{j=1}^{k} \max(\frac{\alpha_j}{\gamma} - c_{ij}, 0) \leq f_i$ holds for all possible problem instances (i.e., all possible c_{ij} and f_i values). This is in fact a minimization linear program with a single constraint and variables f_i, α_j's and c_{ij}'s. Note that, since we want γ to be applicable to all possible instances, f_i and c_{ij}'s are not fixed values any more. Starting with this program, we will construct our factor-revealing LP by adding additional constraints to it. As it is now, our linear program is unbounded. But, since there are dependencies between the α_j variables and the f_i and c_{ij} variables (since α_j's are computed by the algorithm base on the input), we can add constraints to the program which represent those dependencies. These additional constraints are chosen is such a way that our algorithm guarantees that they are always satisfied. If we choose them

wisely, our linear program becomes bounded and we can find a non-trivial value for γ. The two constraint types we add are listed in Lemma 4.2 and 4.3. We also change our program into a maximization program (without changing the cost of its optimal solution) by replacing the current cost function (minimize γ) and the $\sum_{j=1}^{k} \max(\frac{\alpha_j}{\gamma} - c_{ij}, 0) \leq f_i$ constraint with the cost function (maximize $(\sum_{j=1}^{k} \alpha_j)/(f_i + \sum_{j=1}^{k} c_{ij})$). This formulation yields the following family (since we have to bound γ for each possible integer value of k) of linear[1] programs, referred to as the factor-revealing LP.

FacRev-Dist

$$
\begin{array}{rrcll}
\text{maximize} & z_k = \frac{\sum_{j=1}^{k} \alpha_j}{f_i + \sum_{j=1}^{k} c_{ij}} & & & \\
\text{subject to} \quad \alpha_j & \leq & \alpha_{j+1} & \forall j \in \{1,\ldots,k-1\} & (1^{**}) \\
\alpha_j & \leq & (1+\varepsilon)(\alpha_l + c_{ij} + c_{il}) & \forall j,l \in \{1,\ldots,k\} & (2^{**}) \\
\sum_{l=j}^{k} \max(\alpha_j - (1+\varepsilon)c_{il}, 0) & \leq & (1+\varepsilon) f_i & \forall j \in \{1,\ldots,k\} & (3^{**}) \\
\alpha_j, c_{ij}, f_i & \geq & 0 & \forall j \in \{1,\ldots,k\} & (4^{**})
\end{array}
$$

The original factor-revealing LP for the sequential stetting looks as follows:

FacRev-Jain

$$
\begin{array}{rrcll}
\text{maximize} & z_k = \frac{\sum_{j=1}^{k} \alpha_j}{f_i + \sum_{j=1}^{k} c_{ij}} & & & \\
\text{subject to} \quad \alpha_j & \leq & \alpha_{j+1} & \forall j \in \{1,\ldots,k-1\} & (1^{*}) \\
\alpha_j & \leq & \alpha_l + c_{ij} + c_{il} & \forall j,l \in \{1,\ldots,k\} & (2^{*}) \\
\sum_{l=j}^{k} \max(\alpha_j - c_{il}, 0) & \leq & f_i & \forall j \in \{1,\ldots,k\} & (3^{*}) \\
\alpha_j, c_{ij}, f_i & \geq & 0 & \forall j \in \{1,\ldots,k\} & (4^{*})
\end{array}
$$

Note that constraints (1^*) resp. (1^{**}) are only used to sort the α_j values in order to ease the formulation of constraints (3^*) resp. (3^{**}).

The following two Lemmas show that the constraints added to the factor-revealing LP are always satisfied during our algorithm's execution. They resemble similar lemmas to those used in the proof for the sequential algorithm (cf. [35]).

Lemma 4.2. *For any two clients j, j' and a facility i, we have $\frac{\alpha_j}{(1+\varepsilon)} \leq \alpha_{j'} + c_{ij'} + c_{ij}$.*

Proof. If $\alpha_j/(1+\varepsilon) \leq \alpha_{j'}$ the lemma obviously holds. Thus, let us assume the opposite. Let t be the phase when our algorithm connects client j' to some facility i' and s the phase i' was opened. Facility i' was either opened before j' was connected to it ($s < t$) or opened exactly in the same phase client j' was connected to it ($s = t$). Since $\alpha_j > \alpha_{j'}$,

[1] Note that this formulation is not a *linear* program, but can be transformed easily into one.

we know that client j was connected in phase $u \geq t+1$. At time $s \leq t$ the contribution of client j was not enough to cover the distance between j and i', since otherwise j would be connected in phase s with i'. This means that once the contribution of j reaches $c_{i'j}$, client j will be connected to i'. Since the contribution is increased by multiplying the current α value with $(1+\varepsilon)$, it cannot become greater than $(1+\varepsilon)c_{i'j}$, hence $\alpha_j \leq (1+\varepsilon)c_{i'j}$. Thanks to the triangle inequality we have $\alpha_j/(1+\varepsilon) \leq c_{i'j} \leq c_{i'j'} + c_{ij'} + c_{ij} \leq \alpha_{j'} + c_{ij'} + c_{ij}$. ■

Lemma 4.3. *For every client j and facility i, the execution of the algorithm guarantees that* $\sum_{l=j}^{k} \max(\alpha_j - (1+\varepsilon)c_{il}, 0) \leq (1+\varepsilon)f_i$.

Proof. To prove by contradiction, we assume that the inequality does not hold. Thus, we have $\sum_{l=j}^{k} \max(\alpha_j - (1+\varepsilon)c_{il}, 0) > (1+\varepsilon)f_i$. Since we ordered the clients according to their α value, we know that for $l \geq j$ we have $\alpha_l \geq \alpha_j$. Let t be the phase client j was connected to a facility. By the assumption facility i is fully paid for (i.e. i's current payment is $\geq f_i$) in phase $s < t$. There must be at least one client l, with $j \leq l \leq k$ and $\alpha_j - (1+\varepsilon)c_{il} > 0$ which was connected to i in a phase $q < t$. For this client l, we have $\alpha_l < \alpha_j$, since the algorithm will stop increasing α_l when l is connected to a fully paid facility (see Fact 4.1). ■

The following theorem quantifies the correlation between our (*FacRev-Dist*) and the original factor-revealing LPs (*FacRev-Jain*).

Theorem 4.4. *For any $k \in \mathbb{N}$, the optimal solution to* FacRev-Dist *is bounded by* $(1+\varepsilon)^2 1.861$*, and thus the approximation factor of our algorithm is also* $(1+\varepsilon)^2 1.861$.

Proof. Jain et al. showed that z_k can be at most 1.861 in *FacRev-Jain*. In our modified version z_k can be at most $(1+\varepsilon)^2 1.861$: Fix a problem instance by setting the c_{ij} variables and the f_i variable to arbitrary values ≥ 0 and consider the following two propositions.

Proposition 4.5. *In an optimal solution $\alpha := (\alpha_1, \alpha_2, \ldots, \alpha_k)$ to* FacRev-Jain*, $\forall j \in C$ at least one of the two properties is true:*

(i) $\alpha_j = (f_i + \sum_{q \in Q_j} c_{iq})/|Q_j|$, *where* $Q_j := \{q | q \in C \wedge \alpha_j - c_{iq} \geq 0 \wedge \alpha_j \leq \alpha_q\}$

(ii) $\exists l \in C : \alpha_l = (f_i + \sum_{q \in Q_l} c_{iq})/|Q_l|$ *and* $\alpha_j = \alpha_l + c_{ji} + c_{il}$

Proof. Note that the term $(f_i + \sum_{q \in Q_j} c_{iq})/|Q_j|$ is fixed. Since we deal with a maximization problem, which is bounded (see constraint (3^*) of *FacRev-Jain*), each α_j is bounded. Since α represents an optimal solution, for each α_j there must be at least one constraint that bounds α_j and is also tight. If this tight constraint is of the form $\sum_{l=j}^{k} \max(\alpha_j - c_{il}, 0) \leq f_i$, (i) holds.

Otherwise, there must be at least one tight constraint of the form $\alpha_j \leq \alpha_{k_1} + c_{ij} + c_{ik_1}$, i.e., $\exists k_1 \in C : \alpha_j = \alpha_{k_1} + c_{ij} + c_{ik_1}$. This argument can be analogously applied to k_1: either $\alpha_{k_1} = (f_i + \sum_{q \in Q_{k_1}} c_{iq})/|Q_{k_1}|$, or there exists a k_2 such that $\alpha_{k_1} = \alpha_{k_2} + c_{ik_1} + c_{ik_2}$. Like this, we can build a recursion for $\alpha_j = \alpha_{k_1} + c_{ij} + c_{ik_1}$, which terminates as soon as we reach a variable α_{k_t} with $\alpha_{k_t} = (f_i + \sum_{q \in Q_{k_t}} c_{iq})/|Q_{k_t}|$ and thus (i) holds for k_t.

Note that for every α_j there must be such a terminating sequence, since constraint (3^*) is the only constraint which gives an absolute upper bound on the variables. Otherwise, the α_j would be unbounded. The recursion yields $\alpha_j = \alpha_{k_t} + c_{ik_1} + c_{ik_1} + c_{ik_2} + c_{ik_2} + \ldots + c_{ik_{t-1}} + c_{ik_t}$, implying $\alpha_j \geq \alpha_{k_t} + c_{ij} + c_{ik_t}$. Since also $\alpha_j \leq \alpha_{k_t} + c_{ij} + c_{ik_t}$ (constraint (2^*)), we have $\alpha_j = \alpha_{k_t} + c_{ij} + c_{ik_t}$, proving (ii). ■

Proposition 4.6. *Given an optimal solution* $\alpha := (\alpha_1, \alpha_2, \ldots, \alpha_k)$ *to* FacRev-Jain *and an optimal solution* $\alpha' := (\alpha'_1, \alpha'_2, \ldots, \alpha'_k)$ *to* FacRev-Dist*, we have* $\forall j \in C : \alpha'_j \leq (1+\varepsilon)^2 \alpha_j$.

Proof. Since the constraints in the linear program *FacRev-Dist* are a relaxation of the constraints of *FacRev-Jain*, the solution α is also a feasible solution of *FacRev-Dist*. Based on Proposition 4.5, we know that to any α_j at least one of the following cases applies:

(i) $\alpha_j = (f_i + \sum_{q \in Q_j} c_{iq})/|Q_j|$, i.e., the constraint $\sum_{l=j}^{k} \max(\alpha_j - c_{il}, 0) \leq f_i$ is tight. Notice that for α'_j satisfying the constraint $\sum_{l=j}^{k} \max(\alpha_j - (1+\varepsilon)c_{il}, 0) \leq (1+\varepsilon) f_i$ we have, due to the set $\{c_{il} | \alpha_j > c_{il}\}$ containing the set $\{c_{il} | \alpha'_j > (1+\varepsilon)c_{il}\}$, $\alpha'_j \leq (1+\varepsilon)\alpha_j$.

(ii) According to Proposition 4.5 we know that $\exists l \in C : \alpha_l = (f_i + \sum_{q \in Q_l} c_{iq}/|Q_l|$ and $\alpha_j = \alpha_l + c_{ji} + c_{il}$. Constraint (2) of *FR-Dist* guarantees that $\alpha'_j \leq (1+\varepsilon)(\alpha'_l + c_{ji} + c_{il})$. Since we can apply case (i) for l, we have $\alpha'_l \leq (1+\varepsilon)\alpha_l$, which yields $\alpha'_j \leq (1+\varepsilon)((1+\varepsilon)\alpha_l + c_{ji} + c_{il}) \leq (1+\varepsilon)^2(\alpha_l + c_{ji} + c_{il})$.

■

Given any instance of the facility location problem, we know that the optimal solution to the *FacRev-Jain* LP is bounded by 1.861 and that, by Proposition 4.6, the solution to *FacRev-Dist* for the same instance is at most by a factor of $(1+\varepsilon)^2$ larger. Since the problem instance was chosen arbitrarily, the theorem's statement follows. ■

4.3 Facility Selection Mechanism

In this section we are only concerned with a single *phase* of our algorithm (i.e., the loop in which Algorithm 6 and 7 are invoked and which establishes Fact 4.1). We show that it terminates after $O(n^{3/4}\log(n))$ rounds with high probability.

To simplify notation, F and C do not refer to all facilities respectively all clients (as was the case before). Instead, let F represent the set of all facilities that are in the *currently-paid* state and C denote the set of all *not connected* clients contributing to a *currently-paid* facility. All the other clients and facilities effectively do not take part in the execution of Algorithm 6 and 7 in the considered phase and thus can be ignored. We consider the bipartite Graph $G = (F \cup C, E)$. There is an edge in E between $i \in F$ and $j \in C$ if and only if client j contributes to i (i.e., $\alpha_j - c_{ij} > 0$). Furthermore, we assume that there are no isolated nodes (i.e., $\deg(v) \geq 1$ for all $v \in F \cup C$). Let $n = |F| + |C|$. By $G_F := (F, E_F)$ we denote the *Facility Graph*, where there is an edge $\{i, i'\}$ between two facilities $i, i' \in F$ if and only if they share a common client in G. We use $\deg(\cdot)$ and $N(\cdot)$ to denote the degree and neighborhood of a node in G respectively. If the neighborhood includes the node itself, we write $N^+(\cdot)$. Similarly, we use $\deg_F(\cdot)$, $N_F(\cdot)$, and $N_F^+(\cdot)$ to denote the corresponding properties in G_F.

In each execution of Algorithm 6 and 7 the set F of *currently-paid* facilities shrinks (they change their status to *closed* or *open*). Also, with every facility changing its state from *currently-paid* to *open*, clients that are connected to those newly opened facilities are removed from C. Removing these clients form C can potentially cause *currently-paid* facilities to change their state to *closed* (they lose contributing clients and their opening costs are not paid for anymore). Since it cannot be guaranteed that a *currently-paid* facility which loses a contributing client is no longer fully paid for, applying a distributed MIS (maximal independent set) algorithm (e.g., Luby's MIS algorithm [44]) on G_F does not help to solve our problem: a facility that loses a contributing client (i.e., a facility that is not part of the MIS) might still be fully paid for. If such a facility is not opened in the current phase (i.e., (iii) of Fact 4.1 does not hold), clients contributing to it will raise their α_j values in the next phase, which in turn could violate the constraints of the *FacRev-Dist* LP. Since shrinking F by removing *currently-paid* facilities that are not paid for only decreases the runtime of Algorithm 6 and 7, we will assume (in order to consider the worst case) that facilities never change their status from *currently-paid* to *closed* as the result of clients changing their state to *connected* (i.e., losing contributing clients).

4.3.1 Simplified Problem Description

In order to alleviate describing the shrinking process of the set of *currently-paid* facilities, we simplify the analysis of Algorithm 6 and 7 by omitting the information about the values of c_{ij} and f_i and simulating the distributed execution of the algorithms by the algorithm *FacilitySELECT*. Its input is the bipartite graph G and it possesses global knowledge.

Algorithm 8 *FacilitySELECT*

1: **while** $(F \neq \emptyset)$ **do**
2: $I := \emptyset$ (*Subsequently, determine* $I \subseteq F$ *such that* $\{i,i'\} \notin E_F$ *for all* $i,i' \in I$))
3: **for all** $i \in F$: **do**
4: Generate a uniformly distributed random value $r_i \in [0,1]$
5: **if** $(r_i > \max_{i' \in N_F(i)} r_{i'})$ **then**
6: Add i to I
7: Eliminate $N^+(I)$ from G

As the execution of *FacilitySELECT* progresses, F, C, E, and n change. We denote these sets and n in round t as F_t, C_t, E_t and n_t. However, most of the time we consider only the effect of a single iteration of the while loop on the graph. In these cases, we omit the time parameter and simply use F, C, E, and n to refer to the corresponding values in the considered iteration.

4.3.2 Time Required to Shrink the Bipartite Graph

This paragraph analyzes the time required to remove all nodes from G. It is easy to see that a fixed facility $i \in F$ is selected with probability $1/(\deg_F(i)+1)$ by this strategy. First, we show that, in expectation, *FacilitySELECT* removes at least $|E|/|F|$ clients and at least $\max\{|F|, |E|/|F|\}$ edges in a single iteration of the while loop.

Lemma 4.7. *The expected number of clients removed by* FacilitySELECT *in a single iteration of the while loop is at least* $|E|/|F|$.

Proof. For a facility $i \in F$, let X_i denote a random variable counting the number of clients $j \in C$ removed because of i being added to I or not. That is,

$$X_i = \begin{cases} \deg(i) & \text{if } i \in I \\ 0 & \text{otherwise} \end{cases} .$$

Since I is an independent set in G_F, the total number of clients removed in one iteration can be described by the random variable $X := \sum_{i \in F} X_i$. Using the inequality $\deg_F(i)+1 \leq |F|$, we compute:

$$E(X) = \sum_{i \in F} E(X_i) = \sum_{i \in F} \frac{\deg(i)}{\deg_F(i)+1} \geq \sum_{i \in F} \frac{\deg(i)}{|F|} = \frac{|E|}{|F|} .$$

■

Lemma 4.8. *The expected number of edges removed by* FacilitySELECT *in a single iteration of the while loop is at least* $\max\{|F|, |E|/|F|\}$.

Proof. First note that, since by Lemma 4.7 the expected number of clients eliminated is at least $|E|/|F|$ and there are no isolated clients, the expected number of edges removed is at least $|E|/|F|$. It remains to show that the expected number of edges removed is at least $|F|$. For a facility $i \in F$, let us define random variables Z_i counting the number of edges removed because of i being added to I or not. That is,

$$Z_i = \begin{cases} \sum_{j \in N(i)} \deg(j) & \text{if } i \in I \\ 0 & \text{otherwise} \end{cases} .$$

Since I is an independent set in G_F, the total number of edges removed by strategy *FacilitySELECT* can be described by the random variable Z defined as $Z := \sum_{i \in F} Z_i$. Using the inequality $\deg_F(i) + 1 \leq \sum_{j \in N(i)} \deg(j)$, we compute:

$$E(Z) = \sum_{i \in F} E(Z_i) = \sum_{i \in F} \frac{\sum_{j \in N(i)} \deg(j)}{\deg_F(i) + 1} \geq \sum_{i \in F} \frac{\sum_{j \in N(i)} \deg(j)}{\sum_{j \in N(i)} \deg(j)} = |F| \quad .$$

■

As an corollary, we get that, in expectation, the number of edges is reduced by at least its square root:

Corollary 4.9. *The expected number of edges removed by* FacilitySELECT *in a single iteration of the while loop is at least* $\sqrt{|E|}$.

Proof. Lemma 4.8 and the equality $|E| = |F| \cdot \frac{|E|}{|F|}$ imply $\max\{|F|, \frac{|E|}{|F|}\} \geq \sqrt{|E|}$. ■

Lemma 4.10. *If the number of edges is greater or equal* $n\sqrt{n}$*, the expected number of* clients *removed by* FacilitySELECT *in one iteration of the while loop is at least* $\sqrt{|C|}$*. If it is smaller or equal* $n\sqrt{n}$*, the expected number of removed* edges *is at least* $\sqrt{n\sqrt{n}}$.

Proof. For the first statement, we can bound the expected number of removed clients with the help of Lemma 4.7 as follows:

$$\frac{|E|}{|F|} \geq \frac{n\sqrt{n}}{|F|} \geq \frac{n\sqrt{n}}{n} = \sqrt{n} \geq \sqrt{|C|}$$

The second statement follows immediately from Corollary 4.9. ■

We call a round t *edge-heavy*, if in this round the number of edges is at least $n_t\sqrt{n}$. Otherwise, we refer to it as *edge-light*.

Lemma 4.11. *Consider the effect of several iterations of* FacilitySELECT*'s while loop starting at round* t *(for an arbitrary round* t*).*

(a) After $\frac{1}{2}\sqrt{n_t}$ *(not necessarily consecutive)* edge-heavy *rounds, at least* $\frac{1}{2\sqrt{2}}|C_t|$ *clients are removed in expectation.*

(b) After $\frac{1}{2}\sqrt{n_t\sqrt{n_t}}$ *(not necessarily consecutive)* edge-light *rounds, at least* $\frac{1}{2\sqrt{2}}|E_t|$ *edges are removed in expectation.*

Proof. We prove only the first statement, since the second can be proven analogously. Let Y denote a random variable counting the number of clients removed during $\frac{1}{2}\sqrt{n_t}$ edge-heavy rounds. If $Y < |C_t|/2$, the number of clients in each of the considered rounds was at least $|C_t|/2$. Thus, by Lemma 4.7, in each of these $\frac{1}{2}\sqrt{n_t}$ rounds at least $\sqrt{|C_t|/2}$ clients were removed in expectation. If $Y \geq |C_t|/2$, the number of removed clients is trivially lower bounded by $|C_t|/2$. We get:

$$\begin{aligned} E[Y] &\geq \frac{|C_t|}{2} \cdot Pr\left(Y \geq \frac{|C_t|}{2}\right) + \frac{1}{2}\sqrt{n_t} \cdot \sqrt{\frac{|C_t|}{2}} \cdot Pr\left(Y < \frac{|C_t|}{2}\right) \\ &\geq \frac{|C_t|}{2\sqrt{2}} \cdot Pr\left(Y \geq \frac{|C_t|}{2}\right) + \frac{|C_t|}{2\sqrt{2}} \cdot Pr\left(Y < \frac{|C_t|}{2}\right) = \frac{|C_t|}{2\sqrt{2}} \ . \end{aligned}$$

■

Lemma 4.12. *Consider the effect of several iterations of* FacilitySELECT*'s while loop starting at round t (for an arbitrary round t).*

(a) The probability of removing at least $\frac{1}{4\sqrt{2}}|C_t|$ *clients after* $\frac{1}{2}\sqrt{n_t}$ edge-heavy *rounds is at least a constant* > 0*.*

(b) The probability of removing at least $\frac{1}{4\sqrt{2}}|E_t|$ *edges after* $\frac{1}{2}\sqrt{n_t\sqrt{n_t}}$ edge-light *rounds is at least a constant* > 0*.*

Proof. We prove only the first statement. The second can be proven analogously.

By Lemma 4.11 $\frac{1}{2\sqrt{2}}|C_t|$ clients are removed after $\frac{1}{2}\sqrt{n_t}$ *edge-heavy* rounds in expectation. Let X define a random variable counting the number of clients removed after $\frac{1}{2}\sqrt{n_t}$ *edge-heavy* rounds. Now define μ to be the probability that more than $\frac{1}{4\sqrt{2}}|C_t|$ clients are removed. We can bound $E[X]$ in the following way:

$$\frac{1}{2\sqrt{2}}|C_t| \leq E[X] \leq \mu n + (1-\mu)\frac{1}{4\sqrt{2}}|C_t|$$

Solving this inequality for μ yields $\mu \geq c$, where c is a positive constant. ■

We are now ready to prove our main theorem.

Theorem 4.13. *The algorithm* FacilitySELECT *terminates with high probability after* $O(n^{\frac{3}{4}}\log(n))$ *iterations of the while loop.*

Proof. By Lemma 4.12, starting at iteration t we lose at least a constant fraction of either the clients or the edges with a non-zero, constant probability after at most $\frac{1}{2}\sqrt{n_t} + \frac{1}{2}\sqrt{n_t\sqrt{n_t}} \leq n_t^{3/4}$ rounds. Partition the number of iterations done by *FacilitySELECT* into phases of $n^{3/4}$ rounds. Let X_i denote a binary random variable indicating whether the i-th phase was *good* (we lost at least the above mentioned constant fraction of either clients or edges). We need about $2\log n$ good rounds to ensure that the problem size has become $O(1)$. Since we have $Pr(X_i = 1) \geq c$ for some positive constant c, we get for $X := \sum_{i=1}^{\frac{1}{c}2\log n} X_i$:

$$E[X] \geq 2\log(n) \ .$$

That is, in expectation it takes about $\frac{1}{c} \cdot 2\log(n)$ phases until *FacilitySELECT* terminates. Applying a standard Chernoff bound yields:

$$Pr\left(|X - E[X]| > \frac{E[X]}{2}\right) < 2^{-4E[X]/3} \ .$$

In other words, *FacilitySELECT* terminates with high probability after at most $\frac{1}{c}2\log(n)$ steps. ■

4.4 Conclusion & Future Work

We presented a parallel execution of a greedy sequential algorithm for the Facility Location problem and showed that we can preserve its approximation factor. However, there are other sequential Facility Location algorithms based on the greedy approach introduced by Jain et al. that yield (better) approximation factors of 1.61 [35] and 1.52 [45]. It might be also possible to execute them in the $\mathcal{CONGEST}$ model with a sublinear running time using the facility selection mechanism we introduced in this chapter.

Improving the runtime. A natural question that immediately arises is, whether it is possible to devise a faster mechanism than the one presented here. To answer this, one has to note that our algorithm chooses which facilities should be opened based solely on the information provided by the bipartite graph G described in Subsection 4.3.1. All other information (client contributions, distances, facility opening costs etc.) is discarded (see Section 4.3).

At this level of abstraction the lower bound for the runtime required to solve the problem of choosing facilities that are independent of each other is $\sqrt{n}$. To see this, consider a bipartite graph G^* with $\sqrt{n}$ facilities with degree of $\sqrt{n}-1$, and $\binom{\sqrt{n}}{2}$ clients with degree 2. Edges in G^* are chosen in such a way that any two facilities are in hop distance of 2 of each other (i.e., they are competing with each other to be opened in the current

round). Now, consider any algorithm that runs on G^* and solves our problem. During its first step it can choose only a single facility, since each facility is in conflict with all other facilities. This action will remove the chosen facility and $\sqrt{n}-1$ clients that were connected to it. The resulting graph will have a similar structure to G^*. Thus, in the second step, again only a single facility can be removed. As one can see, this process will repeat itself over and over for $\sqrt{n}$ time steps, since only one facility can be removed in a single step.

It is not clear to us whether a graph such as G^* can actually occur during the execution of our algorithm; the connection costs have to satisfy the triangle inequality which might prove to be a hindrance in the construction of a graph that exhibits the same behavior as G^* (i.e., only one facility can be removed each turn, as the other facilities have still enough contribution to be opened).

A very nice approach to improve the runtime was introduced by Blelloch et al. in [11]. They do not solve the exact problem we introduced in Section 4.3.1, but a relaxation thereof. In their problem formulation, a facility can be opened even if not all but a constant fraction of clients (that are necessary to pay the facility's opening costs) are assigned to it. Lets assume this fraction is $(1-\varepsilon')$, with $\varepsilon' > 0$. Opening not completely paid facilities will result in an increase of the approximation guarantee by the factor of $\frac{1}{(1-\varepsilon')}$. Blelloch et al. show that they can solve this kind of selection problem in a polylogarithmic number of steps (where the basis of the logarithm converges to 1 with a decreasing ε).

CHAPTER 5

Conclusion and Future Work

We considered the Facility Location problem from quite different points of view. It was studied as an online problem with the help of competitive analysis in Chapter 2. Chapter 3 and 4 deal, for the most part, with a static scenario. There, we presented distributed approximation algorithms with different properties. The algorithms presented in Chapter 3 were especially designed to be applicable in wireless sensor networks, where locality plays a major role. Our goal in Chapter 4 was to design a distributed algorithm with an approximation factor that is as close as possible to the best possible approximation factor achieveable by sequential algorithms. The next three sections list possible directions for future work.

Extending Our Results. A general idea for future work is to apply our techniques to different variants of the Facility Location problem (see Section 1.5 for an overview of such variants). As a first step, changing the objective function and the constraints (see Subsection 1.5.1), while retaining the respective model of computation, appears to be a reasonable approach. Consider, as an example, the Facility Leasing problem from Chapter 2. One can easily generalize the problem by adding the aspect of fault tolerance (requiring that each client is connected to a specific number of distinct open facilities, and not just a single one).

Something that appears to be very challenging is the addition of capacities (e.g., each facility can serve only a limited number of clients) to any of the problems we considered. Adding capacities changes the problem significantly. To the best of our knowledge, there are no distributed algorithm for this setting yet.

Combining Our Results. Another direction is to try to combine our results from Chapter 2 with the results from either Chapter 3 or Chapter 4. The idea here is to execute the online algorithm from Chapter 2 in a distributed manner. This seems to be

very promising, as the primal-dual approach used by our online algorithm (each client individually increases its α value, which is used to finance near facilities) is very similar to the approach used in Chapter 4.

Restricting the communication infrastructure. Yet another possible research direction is to change the model of computation that was used in the chapters dealing with a distributed scenario (i.e., Chapter 3 and 4). In those chapters we always considered a complete (bipartite) graph for the purpose of communication, allowing direct communication between each node (in Chapter 3) and each client with each facility (in Chapter 4). One idea is to consider two graphs G and G' with the same set of nodes. G is a complete (bipartite) graph that is used to represent an instance of the Facility Location problem. It is equal to the graphs we used throughout Chapter 2, 3 and 4. The other graph, namely G', represents the communication infrastructure that is used by the nodes. Now, nodes cannot communicate using the edges given by G (unlike Chapter 3 and 4, where the edges of G were used for communication); instead, they use only edges in G'. In this model, computing a good solution becomes more challenging, since nodes that have a close distance to each other in G (the weight of the edge connecting them is small) might be quite far away from each other in G'.

Bibliography

[1] K. Aardal, É. Tardos, and D. B. Shmoys. Approximation algorithms for facility location problems (extended abstract). In *Proceedings of the 29th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 265–274, 1997.

[2] S. Abshoff, A. Cord-Landwehr, B. Degener, B. Kempkes, and P. Pietrzyk. Local approximation algorithms for the uncapacitated metric facility location problem in power-aware sensor networks. In *Proceedings of the 7th International Symposium on Algorithms for Sensor Systems, Wireless Ad Hoc Networks and Autonomous Mobile Entities (ALGOSENSORS)*, pages 13–27, 2011.

[3] S. Ahmadian, Z. Friggstad, and C. Swamy. Local-search based approximation algorithms for mobile facility location problems. *CoRR*, abs/1301.4478, 2013.

[4] N. Alon, L. Babai, and A. Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of Algorithms*, 7(4):567–583, 1986.

[5] A. Anagnostopoulos, R. Bent, E. Upfal, and P. Van Hentenryck. A simple and deterministic competitive algorithm for online facility location. *Inf. Comput.*, 194 (2):175–202, 2004.

[6] B. M. Anthony and A. Gupta. Infrastructure leasing problems. In *Proceedings of the 12th International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 424–438, 2007.

[7] M. Badoiu, A. Czumaj, P. Indyk, and C. Sohler. Facility location in sublinear time. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 866–877, 2005.

[8] M.L. Balinski. On finding integer solutions to linear programs. In *In Proceedings of the IBM Scientific Computing Symposium on Combinatorial Problems*, pages 225–248, 1966.

[9] Andrew Berns, James Hegeman, and Sriram V. Pemmaraju. Super-fast distributed algorithms for metric facility location. In *Proceedings of the 39th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 428–439, 2012.

[10] G. E. Blelloch and K. Tangwongsan. Parallel approximation algorithms for facility-location problems. In *Proceedings of the 22nd Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 315–324, 2010.

[11] G. E. Blelloch, K. Tangwongsan, and R. Peng. Linear-work greedy parallel approximation algorithms for set covering and variants. In *Proceedings of the 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2011.

[12] P. Briest, B. Degener, B. Kempkes, P. Kling, and P. Pietrzyk. A distributed approximation algorithm for the metric uncapacitated facility location problem in the congest model. *CoRR*, arXiv/1105.1248, 2011.

[13] J. Byrka. An optimal bifactor approximation algorithm for the metric uncapacitated facility location problem. In *Proceedings of the 10th International Workshop, APPROX, and 11th International Workshop, RANDOM (APPROX-RANDOM)*, pages 29–43, 2007.

[14] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. *SIAM J. Comput.*, 33(6):1417–1440, 2004.

[15] F. A. Chudak and D. B. Shmoys. Improved approximation algorithms for a capacitated facility location problem. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 875–876, 1999.

[16] Fabián A. Chudak and David P. Williamson. Improved approximation algorithms for capacitated facility location problems. In *IPCO*, pages 99–113, 1999.

[17] B. Degener, J. Gehweiler, and C. Lammersen. Kinetic facility location. *Algorithmica*, 57(3):562–584, 2010.

[18] B. Degener, B. Kempkes, and P. Pietrzyk. A local, distributed constant-factor approximation algorithm for the dynamic facility location problem. In *Proceedings of the 24th IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, pages 1–10, 2010.

[19] E.D. Demaine, M.T. Hajiaghayi, H. Mahini, A.S. Sayedi-Roshkhar, S.O. Gharan, and M. Zadimoghaddam. Minimizing movement. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 258–267, 2007.

[20] Z. Drezner and H. W. Hamacher. *Facility location - applications and theory*. Springer, 2002.

[21] M. Elkin. A near-optimal distributed fully dynamic algorithm for maintaining sparse spanners. In *Proceedings of the 26th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 185–194, 2007.

[22] D. Fotakis. On the competitive ratio for online facility location. In *Proceedings of the 30th International Conference on Automata, Languages and Programming (ICALP)*, pages 637–652, 2003.

[23] D. Fotakis. Incremental algorithms for facility location and k-median. *Theor. Comput. Sci.*, 361(2-3):275–313, 2006.

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

[25] D. Fotakis. Online and incremental algorithms for facility location. *SIGACT News*, 42(1):97–131, 2011.

[26] D. Fotakis. Memoryless facility location in one pass. *ACM Transactions on Algorithms*, 7(4):49, 2011.

[27] P. Fraigniaud, A. Korman, and D. Peleg. Local distributed decision. In *Proceedings of the 52nd IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 708–717, 2011.

[28] H. Frey and R. Pillay. A feasibility check for geographical cluster based routing under inaccurate node localization in wireless sensor networks. In *Proceedings of the 17th GI/ITG Conference on Communication in Distributed Systems (KiVS)*, pages 145–156, 2011.

[29] Z. Friggstad and M. R. Salavatipour. Minimizing movement in mobile facility location problems. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 357–366, 2008.

[30] J. Gehweiler, C. Lammersen, and C. Sohler. A distributed O(1)-approximation algorithm for the uniform facility location problem. In *Proceedings of the 18th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 237–243, 2006.

[31] S. Guha and S. Khuller. Greedy strikes back: improved facility location algorithms. *Journal of Algorithms*, 31(1):228–248, 1999.

[32] S. Guha, A. Meyerson, and K. Munagala. Improved algorithms for fault tolerant facility location. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 636–641, 2001.

[33] K. Jain and V. V. Vazirani. An approximation algorithm for the fault tolerant metric facility location problem. In *Proceedings of the 3rd International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX)*, pages 177–183, 2000.

[34] K. Jain and V. 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, 2001.

[35] K. Jain, M. Mahdian, E. Markakis, A. Saberi, and V. V. Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP. *Journal of the ACM*, 50(6):795–824, 2003.

[36] Kamal Jain and Vijay V. Vazirani. Primal-dual approximation algorithms for metric facility location and k-median problems. In *FOCS*, pages 2–13, 1999.

[37] P. Kling, F. Meyer auf der Heide, and P. Pietrzyk. An algorithm for online facility leasing. In *Structural Information and Communication Complexity - 19th International Colloquium (SIROCCO)*, pages 61–72, 2012.

[38] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman. Analysis of a local search heuristic for facility location problems. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1–10, 1998.

[39] A.A. Kuehn and M.J. Hamburger. A heuristic program for locating warehouses. *Management Sci.*, 9:643–666, 1963.

[40] R. Levi, D. B. Shmoys, and C. Swamy. Lp-based approximation algorithms for capacitated facility location. In *Proceedings of the 10th International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 206–218, 2004.

[41] S. Li. A 1.488 approximation algorithm for the uncapacitated facility location problem. In *Proceedings of the 38th International Colloquium on Automata, Languages and Programming (ICALP (2))*, pages 77–88, 2011.

[42] J.-H. Lin and J. S. Vitter. epsilon-approximations with minimum packing constraint violation (extended abstract). In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing (STOC)*, pages 771–782, 1992.

[43] N. Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21(1):193–201, 1992.

[44] M. Luby. A simple parallel algorithm for the maximal independent set problem. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1–10, 1985.

[45] M. Mahdian, Y. Ye, and J. Zhang. Improved approximation algorithms for metric facility location problems. In *Proceedings of the 5th International Workshop, APPROX, and 11th International Workshop, RANDOM (APPROX-RANDOM)*, pages 229–242, 2002.

[46] M. Mahdian, Y. Ye, and J. Zhang. A 2-approximation algorithm for the soft-capacitated facility location problem. In *Proceedings of the 6th International Workshop, APPROX, and 12th International Workshop, RANDOM (APPROX-RANDOM)*, pages 129–140, 2003.

[47] A.S. Manne. Plant location under economies-of-scale-decentralization and computation. *Management Sci.*, 11:213–235, 1964.

[48] R. R. Mettu and C. G. Plaxton. The online median problem. In *Proceedings of the 41st IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 339–348, 2000.

[49] A. Meyerson. Online facility location. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 426–431, 2001.

[50] Adam Meyerson. The parking permit problem. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 274–284, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2468-0.

[51] T. Moscibroda and R. Wattenhofer. Facility location: distributed approximation. In *Proceedings of the 24th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 108–117, 2005.

[52] C. Nagarajan and D. Williamson. Offline and online facility leasing. In Andrea Lodi, Alessandro Panconesi, and Giovanni Rinaldi, editors, *Proceedings of the 13th International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 303–315, 2008. ISBN 978-3-540-68886-0.

[53] M. Pal and E. Tardos. Group strategy proof mechanisms via primal-dual algorithms. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 584–593, 2003.

[54] S. Pandit and S. V. Pemmaraju. Return of the primal-dual: distributed metric facility location. In *Proceedings of the 28th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 180–189, 2009.

[55] S. Pandit and S. V. Pemmaraju. Finding facilities fast. In *Proceedings of the 10th International Conference on Distributed Computing and Networking (ICDCN)*, pages 11–24, 2009.

[56] S. Pandit and S. V. Pemmaraju. Rapid randomized pruning for fast greedy distributed algorithms. In *Proceeding of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, pages 325–334, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-888-9.

[57] D. Peleg. *Distributed Computing: A locality-sensitive approach*, volume 5. SIAM Monographs on Discrete Mathematics and Applications, 2000.

[58] S. Rajagopalan and V. V. Vazirani. Primal-dual rnc approximation algorithms for set cover and covering integer programs. *SIAM J. Comput.*, 28(2):525–540, 1998.

[59] T. S. Rappaport. *Wireless Communications, Principles and Practice (Second Edition)*. Prentice Hall, 2002.

[60] R. Ravi and A. Sinha. Multicommodity facility location. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 342–349, 2004. ISBN 0-89871-558-X.

[61] David B. Shmoys. Approximation algorithms for facility location problems. In *Proceedings of the 3rd International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX)*, pages 27–33, 2000.

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

[63] J.F. Stollsteimer. A working model for plant numbers and locations. *J. Farm Econom.*, 45:631–645, 1963.

[64] C. Swamy and D. B. Shmoys. Fault-tolerant facility location. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 735–736, 2003.

[65] C. Swamy and D. B. Shmoys. Sampling-based approximation algorithms for multi-stage stochastic optimization. In *Probabilistic Methods in the Design and Analysis of Algorithms*, 2007.