UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Local, distributed approximation algorithms for geometric assignment problems

Dissertation

by

Sebastian Degener

International Graduate School Dynamic Intelligent Systems and Faculty of Computer Science, Electrical Engineering and Mathematics
Department of Computer Science and Heinz Nixdorf Institute
University of Paderborn, Germany

January 2010

Zusammenfassung

Wir betrachten eine Gruppe von autonomen Robotern, die in einem unbekannten Gelände ausgesetzt werden. Es gibt keine zentrale Steuerung und die Roboter müssen sich selbst koordinieren. Zentrale Herausforderung dabei ist, dass jeder Roboter nur seine unmittelbare Nachbarschaft sieht und auch nur mit Robotern in seiner unmittelbaren Nachbarschaft kommunizieren kann. Daraus ergeben sich viele algorithmische Fragestellungen. In dieser Arbeit wird untersucht, wie in einem solchen Szenario Zuweisungsaufgaben gelöst werden können, so dass sich trotz der lokalen Einschränkungen global beweisbar gute Lösungen ergeben. Dabei werden im ersten Teil der Arbeit Roboter zu Schätzen zugewiesen, die im Gelände gefunden wurden. Im zweiten Teil der Arbeit werden dynamische Rollenzuweisungen innerhalb des Roboterteams vorgenommen. Dabei müssen die Zuweisungen mit der Zeit geändert werden, da die Roboter sich bewegen. Es werden jeweils untere Schranken gezeigt, sowie lokale Approximationsalgorithmen beschrieben und analysiert.

Reviewers:

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

Contents

CHAPTER 1

Introduction

Consider a set of robots that are deployed to an unknown environment, for instance an ocean or an outer planet. Since there is no global control, the robots have to organize their actions by themselves. Among the problems that have to be considered is the maintenance of good infrastructures, exploration of the environment and assignment of tasks within the team, always under the restriction of limited energy. While there are already considerable results achieved in most single fields, there are not as many results in the field of assigning tasks to the robots. This thesis aims at filling this gap, since assigning tasks to individual entities seems to be a fundamental field of problems when dealing with cooperative agents.

The main difficulty in the described scenario arises from the fact that each robot has only a limited view on the world, namely a limited viewing radius which is determined by a geometric ball. This is a natural restriction for robot teams deployed to areas unaccessible to humans. Each robot has to take decisions based on this information only and is restricted to communication with robots located within the geometric ball as well. Nevertheless, we will design and analyze strategies that globally lead to provably good solutions. This is in contrast to many state-of-the-art approaches in distributed settings, which rely on good heuristics that work well on instances occurring in practice. However, we focus on rigorous analysis of our algorithms under worst-case assumptions. This is favorable, because the detailed circumstances of an application are often not known in advance and furthermore we want to get a structural insight into the problems.

Local algorithms Local algorithms gain increased interest recently, since computing entities are becoming smaller and smaller nowadays and therefore portable. Thus, we have to deal with dynamics. Furthermore, because the computing entities are getting smaller they are also getting cheaper and more of them can be deployed. Thus, distributed algorithms have to be scalable. Dynamics and scalability are a motivation to consider local algorithms, since it is unrealistic to have all information at hand, especially in an environment that dynamically changes. Additionally, communication over long distances is usually slow and expensive. But local algorithms can deal with these circumstances. Since the algorithms use only little information, they are mostly easy to describe, understand and implement. They usually pose few requirements on the executing nodes in terms of computing power, sensing capabilities and storage. Despite the easy description, it is often hard to analyze them and to prove that they are good from a global point of view. The problem is rather the interaction between the nodes than the data structures and the design of the local algorithms running on the nodes. This is for example evident in the second part of the thesis, where we first analyze a global algorithm, which is difficult mainly due to the interaction between the data structures, the algorithm and mobility. While for the local algorithm that we consider afterwards, the worst-case order of interaction is crucial to consider.

External and internal assignment There are two major fields of problems that have to be addressed: external and internal tasks. External tasks arise from the environment. For instance, there might be treasures (which could also be victims from catastrophes that need to be rescued or objects that need to be monitored for instance) that are found and that have to be processed by several robots. In this case the robots have to be assigned to the treasures. See Figure 1.1 for an illustration. On the other hand, there are internal tasks. Here, different roles have to be taken by members of the team (e.g. providing a costly service for other robots). Ultimately, each robot should decide on its own which role it takes, based on its local information. These role assignments might need to be changed due to the movement of the robots. An example is shown in Figure 1.2. Hence, we denote problems where robots are assigned to external entities as *external assignment* problems and problems where assignments take place only within the robot team as *internal assignment* problems.

This thesis consists of two parts, dedicating one to each field. Both parts start with 'bad news', such as lower bounds concerning the locality, the dynamics as well as complexity theoretic aspects of computation. Then, we present 'good news' in terms of local approximation algorithms for the respective problem at hand. In the second part, the conceptually simpler problem of finding a good assignment in a dynamic environment with global view and central control is considered first. The gained insights and developed techniques are afterwards applied to a completely distributed setting with local view. At the end of each part we conclude and give an outlook on possible future research.

Our local algorithms and our model Often local algorithms are considered in graphs, where the notion of locality is the hop distance. Then an algorithm is called k-local if only nodes in a hop distance of at most k are considered. However, by our geometric motivation it is natural to assume that a geometric ball is the proper notion of locality.

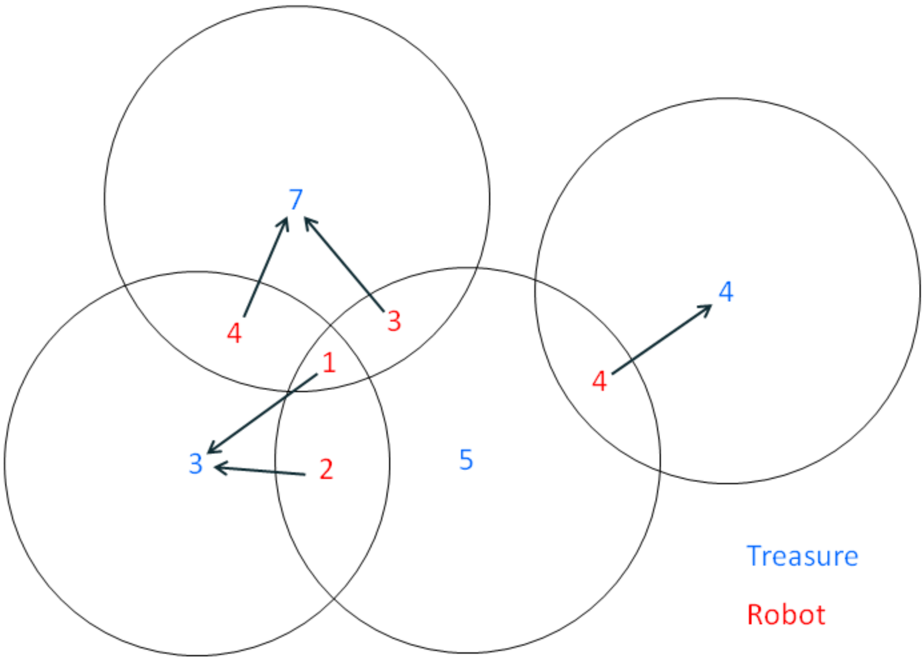


Figure 1.1.: An example instance for the external assignment problem. The circles around the treasures indicate the robots that can be assigned to them due to the uniform viewing radius of the robots. For the given assignment three out of four treasures can be unearthed, since the robot strength sums up at least to the corresponding treasure weight.

From a theoretical point of view this is interesting, since the additional information at hand can be used to solve our problems. This gives an insight into the structure of the problems.

All presented algorithms in this thesis are oblivious, deterministic and the runtime in terms of communication rounds is only a small function in n, where n is the number of robots. It has been pointed out that those local algorithms can be easily transformed into self-stabilizing algorithms by an efficient roll back compiler mechanism, which is able to handle transient failures [AV91]. This means that started form an arbitrary state, the algorithm will lead to a recovery by returning to a state that is desired by the designer. Hence, our algorithms are robust in the sense that slight perturbations in the input, such as temporary wrong measurements, will not harm the results. For a recent survey on related subjects see [LSW09].

Our algorithms need no global information, not even the total number of nodes n is required to execute them. In the first part dedicated to external assignment, our algorithm is stated in a synchronous round model to simplify the description. Therefore all robots

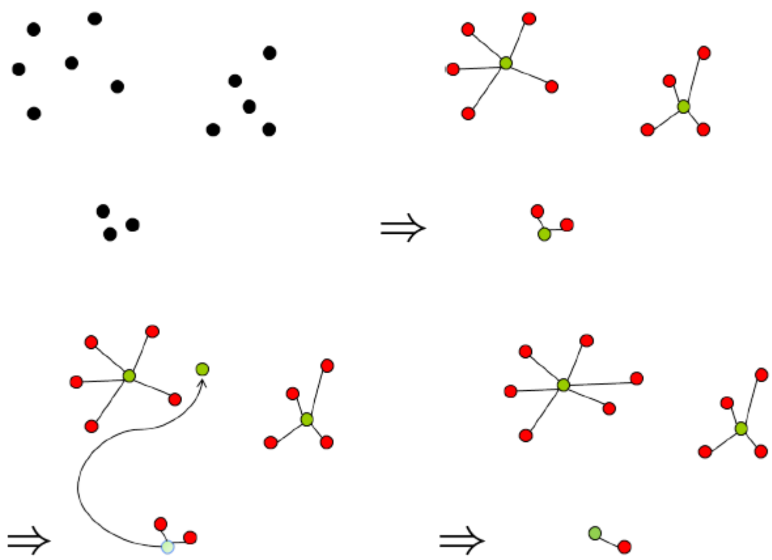


Figure 1.2.: An example instance for the internal assignment problem. The green circles are the chosen robots that take a predefined role. The red robots make use of the green robots. Movement of robots leads to a reassignment of the providing robots.

act in parallel in synchronous rounds. Standard techniques are available to deal with the problem of applying our synchronous algorithm to an asynchronous setting via α synchronizers, where a node basically waits for all its neighbors to acknowledge that they completed a round before proceeding to the next round [Awe85]. The distributed algorithm in the second part of the thesis concerning internal assignment is applied directly to the more natural asynchronous round model. There, nodes wake up in an asynchronous manner and a round is counted, as soon as each robot was active at least once. We assume idealized robots that are capable of exact measurements within their perfectly round geometric ball, that the robots have no physical extend and communication is reliable. We abstract from physical robots to capture the locality constraints that we are mainly interested in. Sometimes we refer to the abstract robots as objects, points or nodes.

Our main contributions The main contributions in the first part of the thesis are several complexity theoretic results showing the computational hardness of the problem and a local constant factor approximation algorithm with constant factor resource augmentation for the UNEARTH TREASURE problem. With this algorithm, robots can assign themselves locally to treasures such that they are capable of unearthing them and the number of treasures unearthed is maximized. Due to its design, the algorithm can be applied to various other objective functions in this context. As a building block, we introduce a local constant factor approximation algorithm for the maximum weighted independent set problem[1] on bounded degree graphs.

In the second part of the thesis, the two main contributions are a kinetic data structure solving the classical FACILITY LOCATION problem under dynamics as well as a local algorithm for the same problem. From a technical point of view, the most important feature of the kinetic data structure is a technique to respond to changes of the location of the robots in poly-logarithmic time. For the local algorithm we are able to prove that changes in the environment lead to changes in the role assignment in the local neighborhood which is upper bounded by a geometric ball with constant radius.

Bibliographic note Parts of this thesis were previously published in [BDKP09] for the first part and in [DGL08a, DGL08b, DGL08c, DKP10] for the second part:

- [BDKP09] O. Bonorden, B. Degener, B. Kempkes, and P. Pietrzyk. Complexity and approximation of a geometric local robot assignment problem. In *Algosensors*, 2009.

- [DGL08b] B. Degener, J. Gehweiler, and C. Lammersen. The kinetic facility location problem. In *Proceedings of the 24th European Workshop on Computational Geometry*, pages 251–254, 2008.

- [DGL08c] B. Degener, J. Gehweiler, and C. Lammersen. The kinetic facility location problem. In *Proceedings of the 11th Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 378–389, 2008.

- [DGL08a] B. Degener, J. Gehweiler, and C. Lammersen. Kinetic facility location. *Algorithmica*, 2008.

- [DKP10] B. Degener, B. Kempkes, and P. Pietrzyk. A local, distributed constant-factor approximation algorithm for the dynamic facility location problem. In *24th IEEE International Parallel and Distributed Processing Symposium*, 2010. to appear.

[1] Throughout this thesis we call a set of nodes in a graph independent, iff their is no pair of nodes that share an edge. We call a set maximal independent, iff it is independent and no node can be added without violating the independence constraint. We consider a maximal independent set as a maximum independent set, iff there is no independent set with larger cardinality. Similar, the maximum weighted independent set is an independent set where the sum of the node weights in the set has the largest cardinality.

In [BDKP09], the problem described in Chapter 2 is introduced, the lower bounds presented in Chapter 3 are summarized and the algorithm in Chapter 4 is presented. [DGL08b] describes the problem defined in Chapter 6 from the global point of view. In [DGL08a, DGL08c] the solution to the global problem is given. Chapter 8 elaborates on this. In [DKP10] the local variant of the problem defined in Chapter 6 is introduced. Chapter 7 on lower bounds for the problem and Chapter 9, where the local algorithm is presented are also based on this paper. The relevant technical related work is presented in the respective chapters.

Part I.

External assignment

CHAPTER 2

Introduction to external assignment

We are given a group of robots which is deployed arbitrarily to a Euclidean space, in which treasures are hidden. Being equipped with sensors, the robots are able to detect treasures positioned within a given viewing range. The treasures can have different weights and the robots can have different amounts of strength. The task is to assign the robots to treasures in such a way that the number of treasures which are unearthed by the robots is maximized. A treasure is unearthed if the amount of strength of the robots assigned to it sums up to at least the weight of the treasure. One constraint must be kept for the assignment: A robot may only be assigned to a treasure if the treasure is within the robot's viewing range. The viewing range of all robots is equal. Note that the robots do not actually move before the assignment is computed. Hence, we are dealing with a static scenario. For an illustration of the problem refer to Figure 1.1 in the previous chapter.

Throughout this chapter, we will adhere to the figurative description of treasures which have to be unearthed. Nevertheless, in an application these treasures could for example also be tasks which have to be handled by teams of robots. Another application would be the assignment of mobile robotic sensors to objects that need a certain number of guards to be properly monitored. These are just a few examples for applications of autonomous robot teams or mobile robotic sensors with the objective at hand. Remember that our main challenge is that all acting entities have only their own local view and no knowledge about the global state of the environment.

We present complexity results for the problems at hand - though stated in a traditional global fashion - capturing the hardness of locality, which is modeled through the constraint that a robot may only be assigned to a treasure in its viewing range. Furthermore, we present a distributed local approximation algorithm using *resource augmentation*. This is a well established tool for algorithmic analysis: An algorithm is provided with some additional resources, but is still compared to an optimal algorithm without these.

We distinguish two scenarios of the problem in this part: In the *heterogeneous* scenario each robot holds an integer value indicating its individual strength, while in the *homogeneous* scenario every single robot has a strength of 1.

2.1. Our contribution

We explore the complexity of calculating the optimal solution for the heterogeneous and the homogeneous scenario in different variants if all knowledge about the treasures and robots is at hand. Because we show that most variants are $\mathcal{NP}$-complete, the calculation of the optimal solution is also impracticable in a local and distributed way. So we present a local, distributed approximation algorithm that uses resource augmentation and provides a constant factor approximation of the optimal solution. The algorithm has a runtime of $\mathcal{O}(\log^* n)$[1] communication rounds. As a building block, we use an algorithm which we consider to be of independent interest, because it is the first distributed local algorithm for the maximum weighted independent set problem on Δ-bounded degree graphs (Lemma 4.4). It has a Δ^Δ-approximation ratio, which is constant in our case.

2.2. Related work

This part of the thesis deals with allocating robots to points in a Euclidean space. A related area is the one of Multi-Robot Coordination, especially Multi-Robot Task Allocation (MRTA) (for an overview, see [GM04]). Here, robots need to be allocated to some kind of task, e.g. an emergency handling or exploration task. For each task, every robot has an estimated fitness and performance cost, which depend on the concrete task and which need not be constant. The utility for a robot to handle one task is often defined as the difference between its fitness and cost. The goal is to assign the robots to the tasks such that all tasks are handled and the overall utility is maximized. Due to the nature of the problems, distributed solutions are often favored. Most approaches to MRTA use experiments or simulations to evaluate the strategies ([LZ05], [BMSS05], [OMS01], [TP07]), but some theoretical results exist as well. In [LBK$^+$05], a two-approximation algorithm is given for allocating exploration tasks to robots. Here, a robot fulfills a task by traveling to a destination point indicated by the task. The cost to be minimized is the overall traveled distance.

For our local approximation algorithm, we need to compute a maximal independent set on unit disk graphs locally. This has recently been shown to be possible in time $\mathcal{O}(\log^* n)$ [SW08], which is asymptotically optimal [Lin92]. Furthermore, we use a local variant of an approximation algorithm for the maximal independent set problem on bounded degree graphs [GPS87]. The main idea of this algorithm is to compare the IDs

[1] The function $\log^* n$ is the iterated logarithm and defined as $\log^* n := \begin{cases} 0 & \text{if } n \leq 1; \\ 1 + \log^*(\log n) & \text{if } n > 1 \end{cases}$. Note that the function tends to infinity, but extremely slowly.

of adjacent nodes and iteratively collapse them. The position of a bit where two IDs differ is concatenated with the value of the bit. Since this shortens the ID exponentially in each iteration, IDs have a constant length after $\mathcal{O}(\log^* n)$ iterations and are all different. After some refinement this yields a maximal independent set. This algorithm has a runtime of $\mathcal{O}(\log \Delta(\Delta^2 + \log^* n))$, which is an improvement of a coloring based algorithm given in [CV86] with complexity $\mathcal{O}(\log^* n + 3^\Delta)$. However, it is possible to compute a maximal independent set in only $\mathcal{O}(\Delta^2 + \log^* n)$ rounds [Lin92]. Since Δ is bounded by a constant in our case, all algorithms have a runtime of $\mathcal{O}(\log^* n)$. The authors in [SW08] also elaborate on this approach, when designing their algorithm for unit disk graphs.

We furthermore present a local distributed Δ^Δ approximation algorithm for the maximum weighted independent set problem. The best global approximation algorithm for this problem yields a Δ-approximation [STK03]. Note that we assume that global coordinates are not available to the robots. While our complexity results would still be valid, our algorithm would be a lot simpler and it would run in constant time using the approach of [KMW05] for computing a maximal independent set. They define a global grid, in which generalized unit disk graphs can easily be colored and turned into maximal independent sets in constant time.

2.3. Formal problem definition

We define different variants of the UNEARTH TREASURES problem. All of these variants accept the same kind of input, which is modeled by two sets T and R and an integer v. The set $T = \{t_1, \ldots, t_n\}$ represents n treasures, the set $R = \{r_1, \ldots, r_m\}$ represents m robots. The viewing range of the robots is represented by v. Since v is equal for all robots, exactly all robots positioned in the sphere around a treasure of radius v can be assigned to the treasure. We call this the viewing range v of the treasure. With every treasure $t_i \in T$ and every robot $r_j \in R$ a position $p(t_i)$ respectively $p(r_j)$ in the d-dimensional Euclidean space is associated. Each of the $n+m$ positions is modeled as a d-tuple. Additionally, we associate an integer value $w(t_i)$ with treasure t_i representing its weight and an integer value $s(r_i)$ with robot r_i representing its strength. The function $assign : R \rightarrow T$ is defined in such a way that for all $r_i \in R$ the implication $(assign(r_i) = t_j \Rightarrow \|p(r_i) - p(t_j)\| \leq v)$ is true. This means that it only assigns a robot to a treasure, if the treasure is in the robot's viewing range. The function $unearth : (T, R, assign) \rightarrow \mathbb{N}$ computes the number of treasures that can be unearthed if the assignment $assign$ is employed. A treasure t_j counts as being unearthed under $assign$ if $\sum_{i|assign(r_i)=t_j} s(r_i) \geq w(t_j)$.

We define the decision problem UNEARTH TREASURES by a function $k : \mathbb{N} \rightarrow \mathbb{R}$ with $k(n) \leq n$. This function tells us how many of the n treasures are supposed to be unearthed. In general, the decision problem of UNEARTH TREASURES with the function $k(n)$ is formulated as follows: Given two arbitrary sets T and R and an integer v as defined above, does a function *assign* exist with $unearth(T, R, assign) \geq k(n)$? We only consider k which are computable in polynomial time and which are monotonically increasing. In Section 4 we consider the optimization variant of the problem.

Note that we formally classify a scenario as homogeneous, if for all $r_i \in R$ the equality $s(r_i) = 1$ is true. Otherwise, we call it heterogeneous.

2.4. Organization of the part external assignment

In Chapter 3 we deal with the UNEARTH TREASURES problem in the heterogeneous and the homogeneous scenario. For both scenarios, we prove the complexity of different variants of the problem, i.e. for different functions $k(n)$. Then we present a local approximation algorithm using resource augmentation in Chapter 4. We conclude in Chapter 5 by discussing some open problems.

CHAPTER 3

The complexity of external assignment

This chapter deals with the complexity of finding a solution for the different variants of the UNEARTH TREASURES problem. In order to keep the results as general as possible, we consider a centralized version in two dimensions. In Section 3.1, the problem is analyzed within the more general heterogeneous scenario, while in Section 3.2 we investigate whether a restriction of the strength-values of the robots to 1 simplifies the problem. Table 3.1 shows an overview of the results presented in this chapter.

3.1. The heterogeneous scenario

In this section we will analyze four variants of the UNEARTH TREASURES problem in the heterogeneous scenario. First we show that providing a solution when just one treasure is supposed to be unearthed ($k(n) = 1$) is in $\mathcal{P}$. Then, for $k(n) = c$ with a constant $c > 1$, we prove that the problem becomes weak $\mathcal{NP}$-complete. Thus, deciding whether a constant number of treasures can be unearthed is already hard. The third variant we analyze is $k(n) = n$, i.e. the question if all treasures can be unearthed. This variant turns out to be strongly $\mathcal{NP}$-complete. Finally, we show that the problem stays strongly $\mathcal{NP}$-complete even if we relax the constrains from $k(n) = n$ to $k(n) \in \Omega(n^{\varepsilon})$ and $0 < \varepsilon \leq 1$ for each $k(n)$.

3.1.1. Unearthing a single treasure

As a warm-up we start with a simple observation for the simplest problem one can imagine in this scenario. We just want to know, whether it is possible at all to unearth a single treasure.

Theorem 3.1. *The variant of* UNEARTH TREASURES *with $k(n) = 1$ in the heterogeneous scenario is in $\mathcal{P}$.*

$k(n)$	homogen.	proof sketch
general	strong $\mathcal{NP}$-complete	reduction from PL. INDEP. SET
1	$\mathcal{O}(n \cdot m)$	
c	$\mathcal{O}(n^c \cdot m^3)$	multiple net flows
$[c, n^\varepsilon]$	unknown	
$[n^\varepsilon, n - n^\varepsilon]$	strong $\mathcal{NP}$-complete	add treasures, reduction from general $k(n)$
$[n - n^\varepsilon, n - c]$	unknown	
$n - c$	$\mathcal{O}(n^c(n+m)^3)$	multiple net flows
n	$\mathcal{O}((n+m)^3)$	netflow
$k(n)$	heterogeneous	proof sketch
general	strong $\mathcal{NP}$-complete	see $k(n) = n$
1	$\mathcal{O}(n \cdot m)$	
c	weak $\mathcal{NP}$-complete $\mathcal{O}((n \cdot m \cdot \max \text{str.})^c)$	reduction from PARTITION dynamic program
$[c, n^\varepsilon]$	$\mathcal{NP}$-complete	reduction from PARTITION
$[n^\varepsilon, n - n^\varepsilon]$	strong $\mathcal{NP}$-complete	add treasures, reduction from $k(n) = n$
$[n - n^\varepsilon, n - c]$	strong $\mathcal{NP}$-complete	see $[n^\varepsilon, n - n^\varepsilon]$
$n - c$	strong $\mathcal{NP}$-complete	see $[n^\varepsilon, n - n^\varepsilon]$
n	strong $\mathcal{NP}$-complete	reduction from 3-SAT

Table 3.1.: Complexity Classes

Proof. It is easy to check for each treasure one after the other whether it is satisfied or not by assigning all adjacent robots to this treasure. A trivial algorithm runs in time $\mathcal{O}(n \cdot m)$, where n is the number of treasures and m the number of robots. □

3.1.2. Unearthing a constant number of treasures

By reduction from PARTITION to UNEARTH TREASURES with $k(n) = c$, we show that unearthing a constant number of treasures is $\mathcal{NP}$-complete. Still, there exists a pseudo-polynomial algorithm to solve this problem, which uses a dynamic programming technique similar to the one usually used to solve the PARTITION problem, which we will present afterwards.

Theorem 3.2. *The variant of* UNEARTH TREASURES *with $k(n) = c$ in the heterogeneous scenario is $\mathcal{NP}$-complete for any constant $c > 1$.*

Proof. It is easy to see that all variants of UNEARTH TREASURES are in $\mathcal{NP}$. To show the $\mathcal{NP}$-hardness we reduce the $\mathcal{NP}$-complete PARTITION (see [Kar72]) to UNEARTH TREASURES with $k(n) = 2$. We only need to consider $k(n) = 2$, since deciding if $k(n) = c$ with $c > 2$ treasures can be unearthed is at least as difficult as deciding whether two treasures can be unearthed.

In the PARTITION problem we are given a finite set $A = \{a_1, a_2, \ldots, a_m\}$ of m positive integers which sum up to $2b$, $b \in \mathbb{N}$. Using this set A, we construct an instance of UNEARTH TREASURES in the following way: We position two treasures t_1 and t_2 with $w(t_1) = w(t_2) = b$ in the plane, so that their viewing ranges intersect. Into this intersection we place m robots $r_1, r_2, \ldots, r_m$ with $s(r_i) = a_i$ for all $1 \leq i \leq m$. An assignment for the robots yields a solution for the PARTITION problem and the reduction takes polynomial time in m. □

The theorem above states that the problem at hand is $\mathcal{NP}$-complete. From the proof also easily follows that the problem cannot be approximated within $2 - \varepsilon$ for $\varepsilon > 0$ unless $\mathcal{NP} = \mathcal{P}$: In the given instance both treasures need to be unearthed to achieve a $(2 - \varepsilon)$-approximation, but this would imply solving PARTITION exactly. This could also be done with several copies of one PARTITION instance in the same UNEARTH TREASURES instance. This technique is sometimes referred to as gap producing.

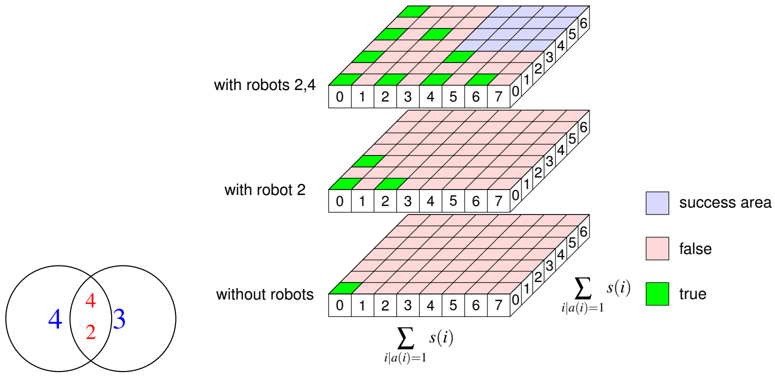
Considering the fact that r_i with strength $s(r_i)$ can be interpreted as a team of $s(r_i)$ robots where every robot has strength one and which have to stay together due to a common device, a unary coding of $s(r_i)$ would make sense. Thus, from a practical point of view, it is reasonable to ask whether there exists an pseudo-polynomial algorithm for the UNEARTH TREASURES problem with $k(n) = c$, as is the case for PARTITION. The proof for the following theorem provides such an algorithm.

Theorem 3.3. *There is a pseudo-polynomial time algorithm for the* UNEARTH TREASURES *problem with* $k(n) = c$.

Proof. We use a dynamic programming technique which is similar to the one usually used to solve the PARTITION problem.

As a first step, we identify the treasures that are candidates for being satisfied in a final solution. There are $\binom{n}{c}$ possible combinations. Since c is a constant and $\binom{n}{c} < \frac{n^c}{c!}$, the number of possible combinations is polynomial in n. We define a dynamic program to deal with each combination.

The dynamic program receives c treasures $t_1, \ldots t_c$ and the robot set $R = \{r_1, \ldots, r_m\}$ as input. It iteratively creates $m + 1$ c-dimensional arrays with boolean entries. We name these arrays $Arr_0, Arr_1, \ldots Arr_m$. First, we give a definition for array Arr_i $(0 \leq i \leq m)$ and then describe an efficient way to compute Arr_i using Arr_{i-1}.



(a) problem (b) matrix P

Figure 3.1.: Dynamic Program

The array Arr_i is defined with the help of the set $R_i := R_{i-1} \cup \{r_i\}$ with $R_0 := \emptyset$. The dimension j $(1 \leq j \leq c)$ of Arr_i represents the treasure t_j. Every dimension of every array has length $\sum_{l=1}^{m} s(r_l)$. The value of the entry $Arr_i[x_1, x_2, \ldots, x_c]$ is only set to *true* if a function *assign* : $R_i \rightarrow \{t_1, \ldots t_c\}$ (as defined in Section 2.3) exists, such that for all t_j $(1 \leq j \leq c)$ the sum of the strength of all robots assigned to t_j using *assign* is equal to x_j.

The values of array Arr_i can be computed easily with the help of array Arr_{i-1}. The entry $Arr_i[x_1, \ldots, x_c]$ is *true*, if the entry $Arr_{i-1}[x_1, \ldots, x_c]$ is *true* or if $Arr_{i-1}[x_1, \ldots, x_k - s(r_i), \ldots, x_c]$ is *true* and r_i is in viewing range of treasure t_k. For Arr_0 only the entry $Arr_0[0, \ldots, 0]$ is *true*, while all others are *false*.

If Arr_m contains at least one entry $Arr_m[x_1, x_2, \ldots, x_c]$ with value *true*, such that $w(t_k) \leq x_k$ for all $0 \leq k \leq c$, all c treasures can be unearthed. Figure 3.1 shows the arrays created by our dynamic problem for a simple instance with two treasures with weight 4 and 3 and two robots with strength 2 and 4.

Our dynamic program has runtime $\mathcal{O}((d \cdot m)^c \cdot m)$ where $d := max(\{s(r_1), \ldots s(r_m)\})$. Since the dynamic program is called a polynomial number of times, the entire algorithm has a polynomial runtime, if d is bounded by a polynomial in $n + m$. $\square$

3.1.3. Unearthing a polynomial number of treasures

In this section we show that the problem is strongly $\mathcal{NP}$-complete for a broad range of parameters. Now we are not dealing with a single problem, but we are actually dealing with an infinite set of problems and therefore with a range of special cases. We first prove that UNEARTH TREASURES is $\mathcal{NP}$-complete for $k(n) = n$ and then we reduce this variant to the class of variants were $k(n) \in \Omega(n^{\varepsilon})$.

Theorem 3.4. *The variant of* UNEARTH TREASURES *with* $k(n) = n$ *in the heterogeneous scenario is* strongly $\mathcal{NP}$*-complete.*

Proof. It is easy to see that any variant of UNEARTH TREASURES is in $\mathcal{NP}$, thus we just need to prove that it is strongly $\mathcal{NP}$-hard. The proof is a reduction from the strongly $\mathcal{NP}$-complete problem 3-SATISFIABILITY (3-SAT). Given a set of clauses we will construct an instance of UNEARTH TREASURES in the Euclidean plane where all treasures can be unearthed if and only if all clauses can be satisfied.

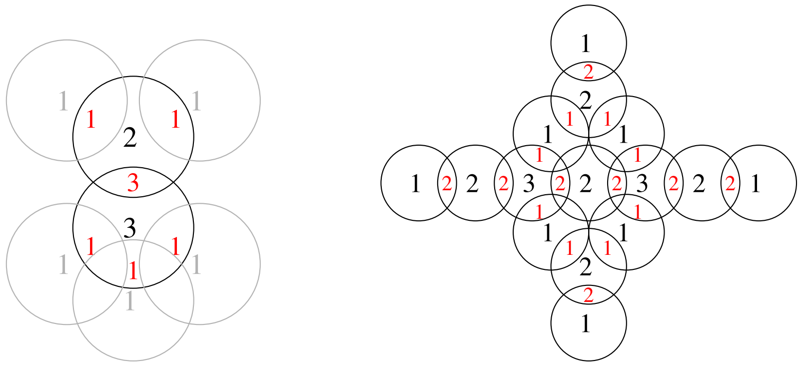
We will describe how to create treasures and robots from the 3-SAT formula and where to place them. To help with the description, we introduce three different constructions: The *clause-gadget*, the *variable-gadget*, and the *connection-gadget*.

Clause-gadget For every single clause C_k in the 3-SAT formula, one treasure, referred to as the C_k-treasure, with weight 1 is created. Since every clause in 3-SAT consists of exactly three literals, for each C_k-treasure three robots with strength 1 are placed in such a way that they are in range of the C_k-treasure, but not in range of any other treasure. Each of these robots represents one of the clause's literals and is called a *(x_i, k)-literal-robot*, where x_i specifies the literal the robot represents and k the clause it belongs to. We will name a construction consisting of one C_k-treasure and the three corresponding (x_i, k)-literal-robots a *C_k-clause-gadget*. Apparently, after we are finished with constructing the clause-gadgets from a 3-SAT formula containing t clauses we have t treasures and $3t$ robots in the Euclidean plane.

Variable-gadget The next step in the construction of an UNEARTH TREASURES instance from the 3-SAT formula is to create the *variable-gadgets*. For every variable X_i appearing in the 3-SAT formula a single *X_i-variable-gadget* is created. It consists of two treasures, referred to as the *x_i-treasure* and the *$\bar{x}_i$-treasure*, and a certain number of robots. These two treasures' positions are chosen in such a way that the two circles defined by their viewing ranges intersect with each other.

Let us assume the literal x_i occurs in p clauses in the SAT formula, while $\bar{x}_i$ occurs in q clauses. This means that we have to set the weight of the x_i-treasure to p and the weight of the $\bar{x}_i$-treasure to q. We also place a single robot in the intersection of the two spheres and set its strength to the maximum of p and q. This robot is referred to as the *X_i-robot*. Next, we place p robots with strength one in range of the x_i-treasure, but out of range of the $\bar{x}_i$-treasure. These robots represent the clauses that contain the literal x_i. We call them *(x_i, k)-variable-robots*, with x_i representing the literal the robot stands for and k specifying the clause. The same is done for the $\bar{x}_i$-treasure: We place q robots, called $(\bar{x}_i, k)$-variable-robots, with strength 1 in its range, but outside of x_i-treasure's range. This concludes the creation of a variable-gadget.

Connection-gadget A variable-gadget derived from the variable X_i has to be connected to every single clause-gadget that is derived from a clause C_k containing either the literal x_i or $\bar{x}_i$. To create such a connection the *(i, k)-connection-gadget* is introduced, with i



(a) example for an x_i-variable-gadget two times x_i, three times $\bar{x}_i$

(b) crossing

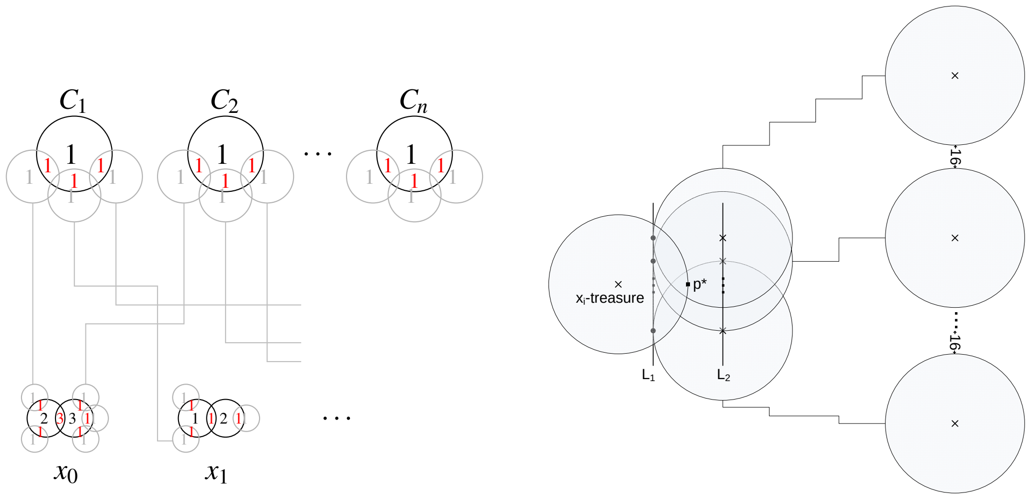
Figure 3.2.: Construction of treasure map for a 3-SAT instance

referring to X_i and k to C_k. We will first describe a naive idea how to create such a connection. Later we will show how this idea can be implemented in a practical way.

To create the (i,k)-connection-gadget that connects the X_i-variable-gadget and the C_k-clause-gadget, we first have to check whether clause C_k contains the literal x_i or $\bar{x}_i$. (If neither literal x_i, nor $\bar{x}_i$ is contained in C_k, the (i,k)-connection-gadget is not created.) Let C_k contain the literal l, with $l = x_i$ or $l = \bar{x}_i$. We place a treasure with weight 1 in such a way that the (l,k)-literal-robot and the (l,k)-variable-robot are the only robots that can unearth it and call it the (l,k)-*connection-treasure*. Thus, our (i,k)-connection-gadget consists of the (l,k)-literal-robot, the (l,k)-variable-robot and the (l,k)-connection-treasure.

Geometrical constraints can make placing the (l,k)-connection-treasure between the (l,k)-literal-robot and the (l,k)-variable-robot in the way described above impossible. In order to still be able to connect a variable-gadget with a clause-gadget, even if those two gadgets are far away from each other, the connection-gadget has to be modified. Instead of placing just one treasure in range of the the (l,k)-literal-robot and the (l,k)-variable-robot, we arrange a chain of treasures with weight 1 between those two robots. This chain's treasures are placed in such a way that the ranges of two neighboring treasures intersect. In every one of these intersection a single robot with strength 1 is placed.

Arrangement of the Gadgets By now, we have constructed a graph with variable-gadgets and clause-gadgets as nodes and connections-gadgets as edges. Next, we describe the arrangement of these elements. An example for such an arrangement is provided by Figure 3.3(a): In the upper part of the figure we arrange the clause-gadgets in a row, while the variable-gadgets are also positioned in a row, but on the figure's lower part. Since the connection-gadgets represent our graph's edges, they are displayed as lines connecting the clause- and the variable-gadgets. These lines (that are actually rows of robots and treasures) are placed on a grid with a lattice constant of 16 times viewing range v.



(a) Construction of treasure map for a 3-SAT instance: example for $(x_0 \vee x_1 \vee x_i) \wedge (\bar{x_0} \vee \dots) \wedge \dots$

(b) Arrangement of variable treasures and the corresponding connection gadgets

Figure 3.3.: Positioning of treasures in the plane

Since we cannot guarantee that described construction leads to a planar graph, we also need *crossings* of connection-gadgets for the two dimensional space. A crossing of two connection-gadgets is shown in Figure 3.2. It is important to note that a line (connection-gadget) can only overlap another line on a single point and that the maximal number of lines that can cross each other at a single point of the grid is limited by two. The reason we choose $16v$ for the lattice constant is to have enough space for such crossings. The area covered by the grid that we placed the lines (connection-gadgets) on is quadratic in the number of all the variable-robots from all variable-gadgets. The number of crossings and the longest connection-gadget are also linear.

Geometrically correct construction of a variable-gadget Assuming that the number of clauses in the 3-SAT formula is t, it might be necessary to connect a variable-gadget to up to t clause-gadgets with the help of up to t connection-gadgets. Creating such connections involves a careful choice of coordinates for the robots and treasures involved in those connections:

Let K be the set of clauses containing x_i and $|K| = s$. Recall that there are s (x_i, k)-variable-robots with $k \in K$ within range of the x_i-treasure. Every (x_i, k)-variable-robot must be positioned in such a way that it is in range of the x_i-treasure and the (x_i, k)-connection-treasure, while not being in range of any other (x_i, l)-connection-treasure with $k \neq l$. It is also important that the coordinates of the robots's position have a size polynomial in s. To achieve this, we need the following construction: Choose an arbitrary position p^* such that the distance between p^* and the x_i-treasure is exactly the treasure's viewing range v. Now, assume that the coordinates of the x_i-treasure are $(0,0)$, while the coordinates of p^* are $(0,v)$. Let L_1 be the line segment between the points $(-\frac{1}{2}v, \frac{1}{2}v)$ and $(\frac{1}{2}v, \frac{1}{2}v)$ and L_2 the line segment between the points $(-\frac{1}{2}v, \frac{3}{2}v)$ and $(\frac{1}{2}v, \frac{3}{2}v)$. We now place the variable-robots on L_1 in such a way that the distance between two of these robots is at least $\frac{s}{v}$. The connection treasures are placed on L_2 in the same way. Additionally (x_i,k)-connection-treasure is placed in such a way that it has minimal distance to the (x_i,k)-variable-robot. See also Figure 3.3 (b).

Satisfiability vs. unearthing treasures In this paragraph we show that all treasures in an instance of UNEARTH TREASURES created from a 3-SAT formula according to the instructions presented this far can only be unearthed if and only if all clauses from the 3-SAT formula can be satisfied.

Consider a 3-SAT formula and an assignment A of TRUE/FALSE values for the formula's variables. We create an UNEARTH TREASURE instance from the formula as described above and assign the robots to the treasures according to the assignment A. If the variable X_i is set to TRUE by A, the X_i-robot is assigned to unearth the x_i-treasure. In the case that A assigns FALSE to X_i, the X_i-robot is sent to unearth the $\bar{x}_i$-treasure. The assignment for the rest of the robots follows directly from the X_i-robots' assignment. We will explain the assignment for the $X_i = true$ case (the $X_i = false$ is analogous) explicitly:

Assigning the X_i-robot to the x_i-treasure allows all (x_i,k)-variable-robots (with $k \in 1,\ldots,n : x_i \in C_k$) to unearth their (x_i,k)-connection-treasures, while the $(\bar{x}_i,l)$-variable-robots (with $l \in 1,\ldots,n : \bar{x}_i \in C_l$) are forced to team up together and unearth the $\bar{x}_i$-treasure. Since every (x_i,k)-variable-robot was free to unearth its connection-treasure, the (x_i,k)-literal-robots are free to unearth their corresponding C_k-treasures and thus satisfy the C_k clauses containing the literal x_i. On the other hand the $(\bar{x}_i,l)$-literal-robots are needed to unearth all $(\bar{x}_i,l)$-connection-treasures, and thus they are not available to unearth the C_l-treasures containing the $\bar{x}_i$ literal. This means that the the C_l-clauses are not satisfied by the $\bar{x}_i$ literal and that other literals are needed to satisfy them. □

The result for $k(n) = n$ is remarkable in the light of Theorem 3.6 in the next chapter, where we show that the corresponding problem in the homogeneous scenario is in $\mathcal{P}$. However, now we use the construction above to show that the problem remains strongly $\mathcal{NP}$-complete for a broad range of parameters. This is a straight forward generalization of the previous theorems, where we considered some special cases. As mentioned before, we are not dealing with a single problem, but we are actually dealing with an infinite set of problems. More formally, we prove that the following theorem holds for every fixed ε:

Theorem 3.5. *The variant of* UNEARTH TREASURES *with fixed* $k(n) \in \Omega(n^{\varepsilon}), (0 < \varepsilon \leq 1)$ *in the heterogeneous scenario is strongly* $\mathcal{NP}$*-complete.*

Proof. We reduce the UNEARTH TREASURES problem with $k(n) = n$ to the variant where only $f(n) \in \Omega(n^{\varepsilon})$ treasures have to be unearthed. So we are given an instance I of UNEARTH TREASURES with n treasures and are asked whether it is possible to unearth $k(n) = n$ of them. For a constant $0 < \varepsilon \leq 1$, we create an instance $\overline{I}$ with $\overline{n}$ treasures from I such that iff it is possible to unearth $f(\overline{n})$ of the treasures in $\overline{I}$ it is possible to unearth $k(n) = n$ of the treasures in I. To achieve this, we extend I with t treasures which cannot be reached by any robot and therefore cannot be unearthed. This means that in $\overline{I}$ we have $\overline{n} = n + t$ robots. If we choose t in a way that $f(\overline{n}) = k(n)$ and therefore $f(n+t) = n$, $f(\overline{n})$ treasures in $\overline{I}$ can be unearthed if and only if $f(n)$ treasures can be unearthed in I. Moreover, t must be greater than or equal to 0 (we cannot add a negative number of treasures) and polynomial in n, because otherwise the reduction would not be polynomial (since f is computable in polynomial time, this also holds for a t polynomial in n). So now we still need to prove that such a t exists.

There exists a t' such that $f(n+t) = f(n) + t'$, where $t \geq 0$ if and only if $t' \geq 0$, because f is monotonically increasing. Since $f(n) \leq n$ (we cannot unearth more than all treasures) and $f(n) + t' = k(n) = n$, $t' \geq 0$ and therefore also $t \geq 0$. Moreover, because $f(n) \geq 0$, $t' \leq n$. To see that t is polynomial in n, note that $f(n+t) \geq c \cdot (n+t)^{\varepsilon}$ for large n and a constant c, because $f(n) \in \Omega(n^{\varepsilon})$. Additionally, $f(n+t) = f(n) + t' \leq 2n$. Putting these two inequalities together, $c \cdot (n+t)^{\varepsilon} \leq 2n$ and therefore $t \leq (\frac{2n}{c})^{\frac{1}{\varepsilon}} - n$. Since c and ε are constants, it follows that t is polynomial in n. $\square$

3.2. The homogeneous scenario

In the homogeneous scenario, all robots have the same strength value set to 1. In this section we analyze whether adding this constraint simplifies the problem in some of the variants for the function k. This is the case: The problem is in $\mathcal{P}$ if $k(n)$ or $n - k(n)$ is constant (where the case $k(n) = 1$ follows directly from the heterogeneous scenario, but this is not the case for other c). It is strongly $\mathcal{NP}$-complete for general k and for every k with $k(n) \in \Omega(n^{\varepsilon_1}) \wedge k(n) \in \mathcal{O}(n^{\varepsilon_2})$, $0 < \varepsilon_1 \leq \varepsilon_2 < 1$.

Note that the homogeneous scenario is equivalent to a scenario where robots are splittable resources and can be allocated to treasures partially. This makes sense if the weight of a treasure corresponds to the energy the robots must spend to unearth it, and robots can help unearthing several treasures until they have no energy left. The equivalence of the scenarios follows from the proof of Theorem 3.6: If we have an assignment where robots are assigned partially to treasures, we can build a flow network like shown in the proof omitting all non-satisfied treasures. The assignment corresponds to a maximum flow in the network with fractional flow on some (assignment) edges. Since all capacities in the network are integer, there also exists an integer maximum flow in the network. This flow also satisfies all treasures (otherwise it would not be a maximum flow) and can be computed in polynomial time.

We now first show the algorithms with polynomial running time, then we prove the $\mathcal{NP}$-hardness claims.

3.2.1. Variants of UNEARTH TREASURES which are in $\mathcal{P}$

In this section we show that some variants of the considered problem are indeed easier in the homogeneous scenario.

Theorem 3.6. *The variants of* UNEARTH TREASURES *with $k(n) = n$ (can all treasures be unearthed?), with $k(n) = n - c$ (can all but c treasures be unearthed?), and with $k(n) = c$ in the homogeneous scenario are in $\mathcal{P}$ for all $c \geq 0$.*

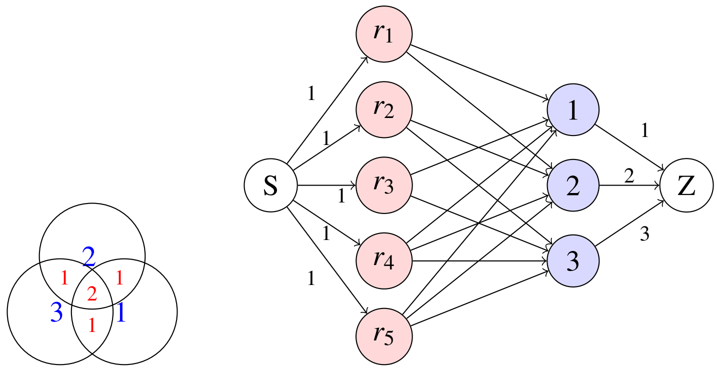
Proof. We model an instance of the robot problem for $k(n) = n$ as a flow network. There is one source node and one sink node. Additionally, we create one node for each robot and one for each treasure. We have an edge from the source node to each robot node with an upper bound of 1 for the flow on this edge. We call these edges *robot edges*. There is also an edge from each robot node to those treasure nodes which are reachable from the respective robot. These edges are called *assignment edges*. From each treasure node we create an edge to the sink. Here, the capacity of the edge is set to the weight of the treasure to be unearthed. We call these edges *treasure edges*.

Since standard flow problems with integer capacities can be solved in polynomial time, the maximum flow f of this network can also be computed in polynomial time. Moreover, this maximum flow is integer on each edge [CLRS01]. Due to the capacities on the robot edges, in a maximum flow there is at most one assignment edge from each robot node with a flow of 1. All other assignment edges have a flow of 0. So the flow on the assignment edges corresponds to an assignment of robots to treasures. All treasures can be unearthed if and only if in a maximum flow all treasure edges are saturated, which can be checked in polynomial time.

For the other cases, delete all $\binom{n}{c}$ respectively $n - \binom{n}{c}$ possible combinations of c respectively $n - c$ treasures ($\binom{n}{c} < \frac{n^c}{c!}$ is polynomial in n) and formulate the corresponding network for each combination. $\square$

Note that the technique used above does not result in a polynomial time algorithm for any c larger than a constant. That is due to the fact that $\binom{n}{f(n)}$ is larger than any polynomial as n tends to infinity if $f(n)$ is larger than a constant and smaller than $\frac{n}{2}$. (If $f(n)$ is larger than $\frac{n}{2}$, similar arguments apply for $n - c$ due to symmetry).

Lemma 3.7. *For $c < f(n) < \frac{n}{2}$ and a constant c, the following holds: $\lim_{n\to\infty} \binom{n}{f(n)} > p(n)$ for any polynomial p.*



(a) problem instance (b) network for flow algorithm

Figure 3.4.: Example for Flow Algorithm

Proof. It is well known that $\binom{n}{f(n)} \geq (\frac{n}{f(n)})^{f(n)}$ holds. We consider two cases: $f(n) \leq \sqrt{n}$ and $f(n) > \sqrt{n}$. For $f(n) \leq \sqrt{n}$ the term $(\frac{n}{f(n)})$ is larger than $n^{\frac{1}{2}}$. If this is raised to any power which is not bounded by a constant, it cannot be a polynom. On the other hand, if $f(n) > \sqrt{n}$ (and is smaller than $\frac{n}{2}$), then $(\frac{n}{f(n)})$ can be bounded from below by 2. Since the exponent is at least $\sqrt{n}$, this results in a super-polynomial function as well. □

3.2.2. Unearthing an arbitrary number of treasures

So far we have only shown that there are special cases in the homogeneous scenario which are in $\mathcal{P}$. This was motivated by the special case $k(n) = n$ which is strongly $\mathcal{NP}$-complete in the heterogeneous scenario, but in $\mathcal{P}$ in the homogeneous scenario. In this section we show that deciding the UNEARTH TREASURES problem with an arbitrary $k(n)$ is also $\mathcal{NP}$-complete in the homogeneous scenario. We can assume that the function $k(n)$ is part of the input. This is equivalent to getting an integer k as input and being asked whether at least k treasures can be unearthed.

Theorem 3.8. *The general variant of* UNEARTH TREASURES *with an integer k as part of the input is strongly $\mathcal{NP}$-complete in the homogeneous scenario.*

Proof. It is obvious that the problem is in $\mathcal{NP}$, thus we just prove that it is strongly $\mathcal{NP}$-hard by reducing the $\mathcal{NP}$-complete problem PLANAR INDEPENDENT SET to it. Given a planar graph G and an integer k', we construct the input for UNEARTH TREASURES in the following way: For each node in G we create a treasure and set its weight to the node's degree, while for each edge e, we create a robot which can only reach the treasures corresponding to the nodes adjacent to e. If these two treasures have to be placed too far apart from each other, we use the construction of a connection-gadget to link them (see proof of Theorem 3.4). Thus, a set of treasures T and a set of robots R is constructed. To make things easier to explain, we assume that for every connection-gadget only one of its robots is assigned to unearth a treasure corresponding to a node, while the remaining robots unearth the treasures which belong to the connection-gadget. Therefore, unearthing the treasure corresponding to $u \in G$ makes unearthing treasures which correspond to u's neighbor nodes impossible. Thus, unearthing treasures that correspond to a node is the same as building an independent set in G. Let l be the number of treasures in all created connection-gadgets and $k := l + k'$. This means that an algorithm with input T and R which can decide whether k treasures can be unearthed also decides whether an independent set with cardinality k' exists in G.

Note that allowing the robots to behave in a different way than we assumed above (i.e assigning two robots belonging to the same connection-gadget to treasures corresponding to nodes) does not increase the number of treasures that can be unearthed, since for every treasure corresponding to a node that is additionally unearthed due to this behavior, at least one treasure belonging to a connection-gadget cannot be unearthed. □

3.2.3. Unearthing a polynomial number of treasures

So far, we have shown, that in general UNEARTH TREASURE is hard to decide. However, we know that the hardness of the problem depends on the number of treasures that are unearthed by an optimal algorithm. The hardness result in the last subsection applies to a quite special construction. This does not necessary tell us anything for typical sets of parameters. In this subsection we show that the problem is strongly $\mathcal{NP}$-complete for a broad range of parameters, similar to the heterogeneous scenario (Section 3.1.3).

Theorem 3.9. *The variant of* UNEARTH TREASURES *in the homogeneous scenario is strongly $\mathcal{NP}$-complete for any function k with $k(n) \in \Omega(n^{\varepsilon_1})$ and $k(n) \in \mathcal{O}(n^{\varepsilon_2})$*, $(0 < \varepsilon_1 \leq \varepsilon_2 < 1)$.

Proof. We first reduce the UNEARTH TREASURES problem with general $k(n)$ to a variant with $k'(n) = \frac{n}{2}$ and then we reduce the variant with $k'(n) = \frac{n}{2}$ to the variant where $f(n)$ treasures have to be unearthed, $f(n) \in \Omega(n^{\varepsilon_1})$ and $f(n) \in \mathcal{O}(n^{\varepsilon_2})$.

For the first reduction, we are given an instance I of UNEARTH TREASURES with n treasures and are asked whether it is possible to unearth $k(n)$ of them. From I, we create an instance $\overline{I}$ with $\overline{n} \geq n$ treasures and ask whether we can unearth $f(\overline{n}) = \frac{\overline{n}}{2}$ treasures.

If $k(n) < \frac{n}{2}$, we add $s = n - 2k(n)$ to the n treasures from I, which are independent of the old treasures, and the same number of robots which can unearth exactly these treasures. s is positive and polynomial in n. In the new instance, the question is whether we can unearth $f(\overline{n}) = f(n+s) = f(2n-2k(n)) = n - k(n) = k(n) + s$ treasures. This number of treasures can be unearthed if and only if $k(n)$ treasures in I can be unearthed.

If $k(n) > \frac{n}{2}$, we add $t = 2k(n) - n$ new treasures that cannot be reached by any robot. Again, $t \geq 0$ and polynomial in n, since $k(n) \leq n$. Here, $f(\overline{n}) = f(n+t) = f(2k(n)) = k(n)$. Since $k(n)$ treasures in I can be unearthed if and only if $k(n)$ treasures in $\overline{I}$ can be unearthed, this concludes the first reduction.

For the second reduction, we are given an instance I of UNEARTH TREASURES with n treasures and are asked whether it is possible to unearth $k'(n) = \frac{n}{2}$ of them. For two constants $0 < \varepsilon_1 \leq \varepsilon_2 < 1$, we create an instance $\bar{I}$ with $\bar{n}$ treasures from I such that iff it is possible to unearth $f(\bar{n})$ of the treasures in $\bar{I}$ it is possible to unearth $k'(n) = \frac{n}{2}$ of the treasures in the original instance. To achieve this, we extend the original instance with s treasures and s robots to unearth them and additionally t treasures which cannot be reached by any robot and therefore cannot be unearthed. This means that in our new instance we have $\bar{n} = n+s+t$ robots. If we choose s and t in a way that $f(\bar{n}) = k'(n)+s$ and therefore $f(n+s+t) = \frac{n}{2}+s$, this question is equivalent to the question whether $\frac{n}{2}$ treasures in the original instance can be unearthed. Moreover, s and t must be greater than or equal to 0 (we cannot add a negative number of treasures) and polynomial in n, because otherwise the reduction would not be polynomial. So now we still need to prove that such s and t exist and can be computed in polynomial time:

There exists a t' such that $f(n+s+t) = f(n+s)+t'$, where $t \geq 0$ if and only if $t' \geq 0$, because f is monotonically increasing. Now let $s = (\frac{n}{2})^{\frac{1}{\varepsilon_2}} - n$. $s \geq 0$ iff $n \geq 2^{\frac{1}{1-\varepsilon_2}}$. Moreover, since ε_2 is constant, s is polynomial in n. $f(n+s)+t' = f(n+s+t) = \frac{n}{2} + s = -\frac{n}{2} + (\frac{n}{2})^{\frac{1}{\varepsilon_2}}$. Additionally, since $f(n) \in \mathcal{O}(n^{\varepsilon_2})$, $f(n+s) = f(\frac{n}{2}^{\frac{1}{\varepsilon_2}}) \leq c_2\frac{n}{2}$ for some constant c_2 and large n. Because $\frac{1}{\varepsilon_2} > 1$, it follows that $t' \geq 0$ and therefore $t \geq 0$ for large n. Moreover, because $f(n+s) \geq 0$, $t' \leq \frac{n}{2} + s = -\frac{n}{2} + (\frac{n}{2})^{\frac{1}{\varepsilon_2}}$.

To see that t is polynomial in n, note that $f(n+s+t) \geq c_1 \cdot (n+s+t)^{\varepsilon_1}$ for large n and a constant c_1, because $f(n) \in \Omega(n^{\varepsilon_1})$. Additionally, because $f(n) \leq n$, $f(n+s+t) = f(n+s)+t' \leq n+s+t'$. Putting these two inequalities together, $c_1 \cdot (n+s+t)^{\varepsilon_1} \leq n+s+t'$ and therefore $t \leq (\frac{n+s+t'}{c_1})^{\frac{1}{\varepsilon_1}} - n - s$. Since c_1 and ε_1 are constants and s and t' polynomial in n, it follows that t is polynomial in n. s can obviously be computed in polynomial time. This is also true for t, because f can be computed in polynomial time. □

Now, we have systematically studied the complexity of UNEARTH TREASURE. Most of the variants considered are $\mathcal{NP}$-complete. However, we still want to deal with the problem. Therefore, in the next chapter we present the main result of this part, a local approximation algorithm.

CHAPTER 4

A local approximation algorithm with resource augmentation

This section describes a local, distributed algorithm which uses resource augmentation and computes a constant factor approximation for the UNEARTH TREASURES problem. In order to be able to execute our algorithm, the robots must be able to perform computations and to communicate with other robots within distance $c \cdot v$, where c is a constant which we will set to 6. Furthermore, every robot has exact information about the position and strength/weight of every robot/treasure within distance $c \cdot v$. The communication is performed in synchronous rounds. In each round, every robot can send an $\mathcal{O}(\log n)$ number of bits to all robots within distance $c \cdot v$. Only after a round is finished, a robot can react to the information it received in the previous round. Moreover, robots and treasures have a unique ID which can also be communicated to neighbors. We assume that treasures have the same capabilities as robots. (A possible explanation for this is that each treasure was located by a robot which now shares the same position as the treasure and performs its communication.) To avoid trivialities, we demand that each robot has a treasure within its viewing range (to achieve this, a robot with no treasures within distance v switches itself off at the very beginning). The constant factor resource augmentation we use in our algorithm allows the robots and treasures to communicate with each other if the distance between them is at most $c \cdot v$ (not only v), and permits assigning a robot to a treasure if they are within distance $c \cdot v$ of each other (and not just v).

The algorithm we propose consists of two parts. In the first part, local clusters of treasures and robots are built. Then, using resource augmentation as described above, the algorithm computes an assignment of robots to treasures for each cluster. In the second part, the algorithm chooses some clusters and, using the assignments computed in the first part for these chosen clusters, creates an assignment for all robots.

The number of rounds used by our algorithm is $\mathcal{O}(\log^* n)$. Since the runtime in wireless networks is dominated by the time needed for communication and since the computation which robots perform in each round is time efficient, we think that the number of communication rounds is a good measure to determine our algorithm's quality.

4.1. Basic Definitions

This paragraph presents definitions which are required in the remaining part of this chapter.

Given an instance (T,R,v) of the UNEARTH TREASURES problem, we construct the *treasure graph* $\bar{G} = (\bar{V},\bar{E})$ as follows: For each treasure $t_i \in T$ we define a node $v_i \in \bar{V}$. An edge $\{v_i, v_j\} \in \bar{E}$ is created, if the treasures $t_i, t_j \in T$ corresponding to v_i and v_j are within distance $2v$ of each other.

A *valid clustering* of an UNEARTH TREASURES instance is a set of subsets $C_i \subseteq R \cup T$ which we call *clusters* such that

1. each treasure and each robot is in at least one cluster
2. in each cluster C_i, exactly one treasure is marked as cluster-center c_i
3. robots belong to a cluster C_i, iff they are in at most distance $3v$ from the cluster-center c_i
4. treasures belong to a cluster C_i, iff they are in at most distance $2v$ from the cluster-center c_i
5. each cluster-center c_i is contained only in its own cluster C_i.

The *cluster-graph* $\widehat{G} = (\widehat{V},\widehat{E},w)$ with $w : \widehat{V} \to \mathbb{N}$ contains one node v_i for each cluster-center c_i of a fixed valid clustering of an instance of UNEARTH TREASURES. Each node v_i has a weight w_i, which is the value of an approximation for UNEARTH TREASURES in its cluster. Iff there is a robot in the considered instance that can reach two cluster-centers c_i and c_j, there is an edge $\{v_i, v_j\}$ in $\widehat{E}$.

Given a graph $G = (V,E)$, a subset V' of V is called *independent*, iff there are no two nodes n_i, n_j in V' such that $(n_i,n_j) \in E$. V' is called an *(inclusion-)maximal* independent set, iff there is no independent V'' with $V' \subsetneq V''$.

For a graph $G = (V,E)$ and a function *weight* that assigns a real number to each node, a subset V' of V is called *maximum weighted independent set*, iff V' is independent and there is no independent set V'' with $\sum_{v' \in V'} \text{weight}(v') < \sum_{v'' \in V''} \text{weight}(v'')$.

Algorithm 1 LOCALUNEARTHTREASURES(INPUT: treasures and robots within distance $6v$ of me)

{*This algorithm is executed locally by a treasure and described from its view*}
$\bar{G}_{local}$:= treasure-graph induced by treasures within distance $2v$ of me
Use $\bar{G}_{local}$ to compute whether I am in INCLUSION-MAXIMAL INDEPENDENT SET (MIS) of $\bar{G}$
if NOT in MIS {*I am no cluster-center*} **then**
  wait for assignment by cluster-center
else
  myCluster := robots within distance $3v$ and treasures within distance $2v$ of me
  T_c := treasures in *myCluster*; R_c := robots in *myCluster*
  assignment := ALLTOALL(T_c, R_c) in *myCluster*
  myWeight := number of unearthed treasures in *assignment*
  $\widehat{G}_{local}$:= cluster-graph induced by cluster-centers in local $6v$-neighborhood of me with respective weights
  inFinalSet := LOCALMWIS($\widehat{G}_{local}$)
  if *inFinalSet* **then**
    tell robots in cluster to use *assignment* (OUTPUT)

4.2. A description of the algorithm

In this section, we provide an intuition for how and why the algorithm works. A formal description is given in Algorithm 1. This pseudocode is written from the view of a single treasure, so that the input for the algorithm is the set of robots and treasures which are within the augmented viewing range of the treasure. At the time of the algorithm's termination, each robot knows the treasure it is assigned to.

In the first step of the algorithm, a valid clustering of the robots and treasures is built. This can be achieved by choosing treasures as cluster-centers and assigning all robots and treasures which are in the corresponding distance of a cluster-center c_i (point 3 and 4 of the definition of a valid clustering) to the cluster C_i. Cluster-centers are chosen by computing an inclusion-maximal independent set on the treasure-graph. Each treasure which is in the independent set marks itself as cluster-center (see Lemma 4.1 for why an inclusion-maximal independent set yields a valid clustering of robots and treasures). Since we have a valid clustering, a treasure and a robot belonging to the same cluster have a Euclidean distance of at most five times the viewing range. This means that if we use resource augmentation to increase the viewing range by a factor of five, we have a special situation in each cluster: Any robot can be assigned to any one treasure. This property is necessary to use the algorithm ALLTOALL (Algorithm 2). This algorithm receives a set of robots and a set of treasures in which all robots can reach all treasures as input.

Algorithm 2 ALLTOALL(INPUT: T,R)

L_1 := treasures sorted by weight, lowest first
L_2 := robots sorted by strength, lowest first
for all treasures t_i in L_1 **do**
  if a single robot r_j can satisfy t_i **then**
    satisfy t_i by assigning r_j to t_i and delete r_j from L_2 and t_i from L_1
while there are treasures in L_1 **do**
  take first treasure t from L_1
  while t is not satisfied AND there are robots in L_2 **do**
    assign first robot from L_2 to t
    if t is satisfied **then**
      delete t from L_1
return assignment of robots to treasures

It computes a solution for these two sets, so that the number of treasures unearthed is at least one half of the number of treasures unearthed in an optimal solution (see Lemma 4.3). Now, in each cluster the cluster-center can apply ALLTOALL to compute a solution for its cluster. But having an approximation for each cluster does not directly lead to an approximation for the whole problem instance, since robots can be in more than one cluster and two different cluster-centers might therefore assign these robots to different treasures. Therefore, we select some of the clusters for the final solution, so that we still have a constant factor approximation, but each robot is only in one cluster of the final solution. To do this, we approximate a maximum weighted independent set on the cluster-graph (Algorithm 3) and select the clusters in this independent set for the final solution. This computation is efficient (see Lemma 4.6), because the cluster-graph has a constant degree (see Lemma 4.2). Ultimately, robots can only belong to one cluster in the final solution and will therefore be assigned to at most one treasure.

4.3. Clustering of robots and treasures: Correctness

Now we have a closer look at how the clustering of robots and treasures is computed. The idea is to select treasures as cluster-centers in such a way that a valid clustering is generated and letting all treasures and robots which are in the according distance of a cluster-center belong to its cluster. The next lemma shows how treasures can be selected as cluster-centers.

Lemma 4.1. *Any inclusion-maximal independent set on the treasure-graph $\bar{G}$ of an instance of* UNEARTH TREASURES *corresponds to a selection of treasures as cluster-centers which yield a valid clustering.*

Algorithm 3 LOCALMWIS(INPUT: $G_{local} = (V,E,w)$ $\{$*in local neighborhood of me*$\}$)

$\{$*This algorithm is executed locally by a node and described from its point of view*$\}$
while (NOT marked as deleted) and (NOT assigned to MWIS) **do**
  Use G_{local} to compute whether I am in UNW. INCLUSION-MAXIMAL INDEPENDENT SET(MIS) of G
  if NOT in MIS **then**
    wait this round
  else
    if (for all neighbors in cluster-graph) *myWeight* $>$ neighbors.*myweight* **then**
      assign myself to MWIS
      mark neighboring nodes as deleted
    else
      mark myself as deleted
return whether assigned to MWIS

Proof. Since the clusters are constructed by the choice of cluster-centers, obviously each cluster has a cluster-center. Furthermore, the distances from treasures and robots to cluster-centers are also kept due to the construction of the clusters. We still have to show that each treasure and each robot is in at least one cluster (Proposition 1) and that each cluster-center is only in its own cluster (Proposition 2).

For the first proposition, assume that there is one treasure t which is not in a cluster. It therefore exists no cluster-center within distance $2v$ of t. Therefore, there is no edge between t and a cluster-center in the treasure-graph. So adding t to the independent set increases the size of it and therefore the independent set was not inclusion-maximal. Now assume that there is one robot r which is not in a cluster. It therefore exists no cluster-center within distance $3v$ of r. Since there is at least one treasure in the viewing range v of r, this treasure cannot be within distance $2v$ of a cluster-center. This constitutes the contradiction.

For the second proposition, consider two cluster-centers c_1 and c_2. Since they both are in the inclusion-maximal independent set of the treasure-graph $\bar{G}$, there is no edge $\{c_i, c_j\}$ in $\bar{G}$. This means that c_i and c_j are in distance more than $2v$ of each other and therefore neither of them is in the other's cluster. □

To be able to compute a maximum weighted independent set on the cluster-graph efficiently, we need the cluster-graph to have a constant degree. Like Lemma 4.2 states, this is guaranteed if the clustering of robots and treasures is valid.

Lemma 4.2. *A valid clustering of an* UNEARTH TREASURES *instance yields a cluster-graph with at most degree* $\Delta = 48$.

Proof. Since each cluster-center is only in its own cluster, the distance between two cluster-centers is more than $2v$. This implies that each cluster-center has a circle with a radius v surrounding itself which does not intersect with the according circles of other cluster-centers. We call this circle the *exclusive area* of a cluster-center. Furthermore, two cluster-centers which share an edge in the cluster-graph can be at most in distance $6v$ of each other. Now consider a cluster-center c. All its neighbors in the cluster-graph must be in distance $6v$ around c. This means that the exclusive areas of the neighbors of c are in distance at most $7v$ and therefore in an area of size $\pi \cdot (7v)^2 = \pi \cdot 49v^2$ around c. Since each cluster-center has an exclusive area of size πv^2, there can be no more than $\frac{\pi \cdot 49v^2}{\pi v^2} = 49$ cluster-centers and therefore 48 neighbors of c in this area. □

Note that in the proof of Lemma 4.2 we made use of the fact that we are dealing with a geometric setting. As we will see this is the only place in the analysis, where we actually use geometry. Furthermore, the only geometric property used is the fact that for objects with an extend or volume only a bounded number of objects fits into a fixed sized neighborhood. Therefore any space offering this property will work for this setting as well. In general, metrics with this property are called *doubling metrics*, because their doubling dimension is constant. The *doubling dimension* $\dim(X)$ of a metric (X,d) is at most α, if every set of diameter D can be covered by 2^α sets of diameter $D/2$.

4.4. Choosing clusters for a final solution: Correctness

As soon as the cluster-centers have been defined, each cluster-center computes an approximation for its own cluster. To achieve that each robot can be assigned to each treasure, the viewing range of each robot and treasure is increased by a factor of 6 (5 would be sufficient here, but we need 6 later). The next lemma states that the solution computed via ALLTOALL by a cluster-center is a two-approximation.

Lemma 4.3. *Given an instance of* UNEARTH TREASURES *where each robot can reach all treasures,* ALLTOALL *computes a 2-approximation.*

Proof. We compare the solution computed by ALLTOALL with an optimal solution. We show that for each treasure satisfied by ALLTOALL, there is at most one additional treasure satisfied in the optimal solution. In the first loop, each satisfied treasure is satisfied by a single robot. In an optimal solution this robot might have been assigned to another treasure, which is the only one which might become unsatisfiable by this action.

The second loop iterates over all treasures. There are only two possibilities for how the loop can be left. Either all treasures are satisfied, leading to an optimal solution, or there are no robots left to assign. In this case, observe that the treasures with the smallest values are satisfied. We claim that the algorithm assigns at most twice the needed amount of robot strength to each of these treasures, thus the total power of the robots can at most satisfy the double number of treasures in an optimal solution. This follows from the fact that all wasted robot power belongs to a single robot, because no robots are assigned after the treasure is satisfied. Assume for the sake of contradiction that this robot has more power than is needed to satisfy this single treasure. Then it would have been assigned to the treasure in the first loop of the algorithm. □

As soon as the weights of each cluster-center in form of an approximation in each cluster have been calculated, we have to choose the clusters that will be part of the final solution. This is done by approximating a maximum weighted independent set on the cluster-graph. For neighboring cluster-centers to be able to communicate with each other, it is necessary to increase the viewing range of cluster-centers by a factor of 6 to $6v$.

Lemma 4.4. *Given a (node-)weighted graph G and a maximum degree* Δ*,* LOCALMWIS *computes a* Δ^{Δ}*-approximation of the maximum weighted independent set of G.*

Proof. First of all, we prove that the while-loop terminates after at most $\Delta + 1$ rounds, where Δ is the degree of the graph. If a node u is in the inclusion-maximal independent set (MIS) in one round, either u will mark itself or all its neighbors as deleted. If u is not in the MIS, one of its neighbors is. So either u will be marked as deleted by its neighbor or one of its neighbors will be marked as deleted. This leaves no neighbors for deletion after Δ rounds. Furthermore, note that the final set is indeed independent: In each round the nodes that are chosen to the maximal independent set are independent by definition. If they assign themselves to the final set, they mark their neighbors as deleted, which subsequently cannot be assigned to the final set in later rounds.

To show that we get a Δ^{Δ}-approximation, note that from each node which is not in the final set we can define a path of *dominating nodes* to a node chosen to the final set. A dominating node of a node u is a neighbor of u with a larger weight. For any given node which is not in the final set, the next node on the path of dominating nodes is either the one that marked it as deleted or, if the node marked itself as deleted, a neighboring node with a higher value. This neighbor must exist, since otherwise the node would not have marked itself as deleted. After at most Δ hops from each node, a node in the final set is reached, because there are at most $\Delta + 1$ rounds and all remaining nodes in the last round (with no degree) are chosen to the final set. Now consider an arbitrary node u in the final set and all nodes which are not in the final set whose paths lead to u. These are at most Δ^{Δ} nodes, because this is an upper bound on the number of nodes in a Δ-neighborhood of a Δ-degree graph. All those nodes have a smaller value than u. Therefore the value of the chosen node is at least $\frac{1}{\Delta^{\Delta}}$ times the value of a set to which all other nodes would have been chosen. This applies to all nodes in the final set. □

4.5. Putting it all together

Now we have all preliminaries for proving the correctness of the algorithm.

Theorem 4.5. *Algorithm* LOCALUNEARTHTREASURES *computes a constant-factor approximation of* UNEARTH TREASURES *using factor-6 resource augmentation.*

Proof. According to Lemma 4.3, the weight of a cluster is a two-approximation of an optimal solution in the cluster. Furthermore, the choice of clusters for a final solution is approximized with a factor of Δ^Δ (Lemma 4.4), where Δ is the degree of the cluster-graph and therefore constant (Lemma 4.2). Furthermore, each robot can only be assigned to one treasure within the same cluster and therefore it needs to travel at most distance $5v$. Treasures must be able to communicate with robots and treasures in their own cluster and additionally their neighbors in the cluster-graph. All these robots and treasures are within distance at most $6v$. □

Theorem 4.5 states the correctness of the algorithm. We still need to analyze its complexity. Note that two different kinds of runtime can be employed. On the one hand, we can count the number of local computations that the robots and treasures perform and on the other hand we can count the communication rounds. In a wireless network, the costly part concerning time and energy consumption is the communication between robots and therefore we use the number of communication rounds for measuring the complexity of our algorithms. This leads to a complexity of $\mathcal{O}(1)$ of the algorithm ALLTOALL, since this algorithm uses no communication besides the final broadcast of the solution. However, it is worth noting that the local computation for ALLTOALL is still efficient in the worst-case: If all treasures and robots are in one cluster, our algorithm takes $\mathcal{O}(n\log n + m\log m)$ local computations. Since the runtime of ALLTOALL dominates the runtime of all local computations, this is also an upper bound on the total local processing time. Furthermore note that in each communication round at most $\mathcal{O}(\log n)$ bits are transmitted.

Lemma 4.6. *Algorithm* LOCALMWIS *terminates in* $\mathcal{O}(\log^* n)$ *communication rounds.*

Proof. According to the proof of Lemma 4.4, the while-loop in LOCALMWIS is executed at most $\Delta + 1$ times. In each round of the loop, an inclusion-maximal independent set is computed. This can be done using the algorithm from [GPS87] in $\mathcal{O}(\log^* n)$ rounds. Line 7 can again take Δ time, the remaining of the loop takes constant time. Since Δ is also constant, the combined running time is $\mathcal{O}(\log^* n)$. □

Theorem 4.7. *Algorithm* LOCALUNEARTHTREASURES *terminates in* $\mathcal{O}(\log^* n)$ *communication rounds.*

Proof. In line 3 of the algorithm, an inclusion-maximal independent set on a unit disk graph is computed. This can be done using [SW08] in $\mathcal{O}(\log^* n)$ communication rounds. Since LOCALMWIS also takes $\mathcal{O}(\log^* n)$ communication rounds due to Lemma 4.6 and the remaining of the algorithm takes constant time, the overall running time is $\mathcal{O}(\log^* n)$. □

4.6. Generalizations

We claimed that our technique can be applied to other objective functions as well. In this section, we will shortly demonstrate this for an easy example. As a motivation in our scenario, we assume that robots are specialized and have different skills. The treasures are of different kinds and each kind requires a set of skills to be unearthed. For instance, to unearth a treasure that is buried deep in the ground, one needs a robot that is capable of digging a hole. For a heavy treasure a special transportation robot might be necessary. Depending on its specification a robot might have a set of skills. For ease of description, we assume that there are only two skills in total: skill a and skill b.

Formally, we consider the UNERATH TREASURE problem as defined in in Section 2.3 for the homogeneous scenario. Furthermore, each treasure requires at most one robot of each skill. Therefore, there are treasures that need one robot with skill a, treasures which need one robot with skill b, and treasures that require one robot with skill a and one robot with skill b.

We apply algorithm LOCALUNEARTHTREASURE (see Algorithm 1) and replace the subroutine ALLTOALL (see Algorithm 2) by a greedy algorithm that runs in three steps: First all treasures that only require skill a get robots which have skill a, as long as there are such robots. Then all treasures that only require skill b get the respective robots. Then the rest of the treasures are assigned. Observe, that this yields a two-approximation, because for each wrong assigned robot still one treasure is unearthed and at most one treasure in an optimal solution is lost. The rest of the analysis of the algorithm LOCALUNEARTHTREASURE remains unchanged.

CHAPTER 5

Conclusion and open questions concerning external assignment

We presented the problem UNEARTH TREASURES, a simple assignment problem, revealing the difficulties of solving external assignment task in a local fashion. We studied the complexity of this problem in detail and provided a quite general framework to compute approximate solutions locally.

Furthermore, we introduced an algorithm that computes a constant factor approximation for the maximum weighted independent set on bounded degree graphs locally in time $\mathcal{O}(\log^*(n))$. Note, that the runtime is of the same order as the lower bound given by [Lin92] for computing inclusion-maximal unweighted independent sets locally. But, here we were facing several additional difficulties. First of all, even from a global point of view computing on bounded degree graphs a maximum independent set instead of a maximal independent set changes the problem. While the latter admits a simple greedy algorithm, the former is $\mathcal{NP}$-complete and does not even admit a PTAS unless $\mathcal{NP}$=$\mathcal{P}$ [BK99]. Therefore having a constant-factor approximation is the best we can hope for. Furthermore, checking whether an independent set is maximal can be easily checked locally, while this is obviously not true for maximum independent sets. The weighted case seems to be even harder on the first glance. Consider a path of nodes, where the node weights increase from one end to the other. The local view of all nodes is identical. They do not know, if they should join the set, because they have more weight than their left neighbor or if they should not join the set because they have less weight than their right neighbor. Note, that we overcame the problem by breaking the symmetry first.

However, several problems remained open about external assignment. Concerning the complexity analysis in Section 3, there are still some functions $k(n)$ left, for which the complexity of the corresponding variant of the UNEARTH TREASURES problem was not covered. This is the case for the homogeneous scenario where the function $k(n)$ grows faster than a constant, but not as fast as a polynomial yet (see also Table 3.1 in Chapter

3).

An important characteristic of the local algorithm presented in Chapter 4 is its use of resource augmentation. A question that arises is, whether resource augmentation is needed to be able to compute a constant factor approximation for the UNEARTH TREASURES problem in polynomial time. If this is the case, it is reasonable to improve the factor for resource augmentation. Even a $(1+\varepsilon)$-augmentation might be possible. This is not the case for the approximation factor: Here, the analysis of the PARTITION problem shows a lower bound of 2.

Another arising question is, how far the constants of our algorithm can be improved. Especially the approximation factor of Δ^Δ for the local weighted maximum independent set problem should be significantly improved, since a Δ-approximation for global algorithms is known. Extensions to higher dimensions seem straight forward.

However, it might also be worthwhile to study other metrics. Geometry is used only in Lemma 4.2. As pointed out earlier, the arguments in the lemma hold for any doubling metric. This covers many spaces, for instance for constant d, the space $\mathbb{R}^d$ under any L_p norm has a constant doubling dimension. It has also been argued that distance metrics induced by peer-to-peer networks or Internet latencies have doubling dimension.

More important is the fact, that the objective function is quite easy. Hence the lower bounds hold also for more general objective functions. On the other hand, in the proof of the central Theorem 4.5 it is only used that Lemma 4.3 guarantees a two-approximation. Therefore, for any objective function, where an assignment with a (global) constant-factor approximation is feasible, it seems to be straight forward to apply our technique.

Part II.

Internal assignment

CHAPTER 6

Introduction to internal assignment

In the first part of the thesis we were concerned about *external* assignment tasks, while in this part of the thesis we are dealing with *internal* assignment tasks. Hence, we are not longer concerned with entities outside the team. Rather we seek for (dynamic) assignments of different roles within the team. The reason is that in the domain of wireless sensor networks it is often crucial to save energy in order to maximize the lifetime of the network. Often these networks are clustered and each cluster contains a node referred to as cluster-head that has to fulfill a special, possibly highly energy consuming task, while the other nodes operate in an energy-saving mode, but their energy consumption depends on their distance to the closest cluster-head. This can be seen as a hierarchical organization, where the upper layer offers the lower layer a certain service, such as a routing infrastructure. Each node can act as a cluster-head, but, at any time, cost arises for each node that is set up or maintained as a cluster-head. This additional overhead for a cluster-head is caused by a higher energy consumption due to message passing, setting up communication channels, or maintaining equipment or other costs, for instance for storing of routing tables, storing the data etc. Since each node should be able to access a service as fast as possible, there is also a cost for each non-cluster-head, or client for short, namely the energy to communicate with the cluster-head which depends on the distance to the nearest cluster-head. Now, to decrease the total cost for the system, nodes are allowed to change their status[1] from cluster-head to client or vice versa. Note that in particular, in our scenario where teams of autonomous robots deployed to an unknown terrain have to handle certain tasks, the above described hierarchical communication system can be applied. Finding the proper subset of robots that should act as cluster-head, while the team is moving through the terrain, is the problem that we want to consider here. We assume that each node has its own certain costs for being a cluster-head and for being a client a cost-factor for the distance to the closest facility as well. It turns out

[1] We use the terms 'role' and 'status' interchangeable. The terms 'system' and 'configuration' denote the set of robots and their positions as well as their current roles at given point of time.

that the problem described above is well known and studied in several fields of operations research and computer science and is known as FACILITY LOCATION problem.

In its original version we are presented with a set of possible warehouse locations and a set of customer locations. Our objective is to decide on which of these possible warehouse locations we want to actually build warehouses. Since maintaining a warehouse incurs high costs, we want to build as few as possible. On the other hand, every customer prefers to be located as close to a warehouse as possible, since costs rise with the distance to the nearest warehouse. This means that we are looking for a placement of the warehouses that minimizes the sum of the costs caused by the customers and the warehouses. Since this problem is $\mathcal{NP}$-complete, approximations of the optimal solution are of interest.

However, usually the static variant of the problem is considered from a global view. Here, we are interested in dynamic versions, as well as distributed and local variants, since these are required in the decentralized settings we are dealing with. Since the ultimate goal are distributed algorithm, where each acting robot has only information about its local neighborhood, we cannot apply any centralized solution, where we need a central instance that is not only able to see the entire team, but also able to compute a solution and distribute it to the robots. In our scenario this is entirely out of scope. Although, we seek -and eventually present- local solutions, we start our studies with a global framework. This way we can deal with dynamics first and develop techniques that we use later on in the local setting.

We seek at a given point of time for a constant-factor approximation of the optimal solution at that point of time. The main goal is to reconstruct such a solution as fast as possible if we cannot guarantee it anymore.

Next we will state our main contribution in this part both for the global scenario and the local scenario in Section 6.1. We present relevant related work in Section 6.2, before giving a formal definition of the FACILITY LOCATION problem as considered by us in Section 6.3. We briefly review an important approach from the literature in Section 6.4 where some of our algorithmic ideas come from and which is due to Mettu and Plaxton [MP00]. The organization of the rest of the second part of the thesis is given in Section 6.5. We finish this introduction with an overview over the main differences between our global and local algorithm in the following chapters in Section 6.6.

6.1. Our contribution

The global scenario Although, we require local strategies in our distributed robot scenario, we will first develop a global algorithm that is able to deal with dynamics. This will provide us with the necessary insight on dynamics to develop a distributed, local algorithm later on.

The kinetic data structure framework (confer to Subsection 8.1.1) is well suited to deal with objects moving in a Euclidean space. We present a kinetic Data Structure for the FACILITY LOCATION problem for a set of n points with given trajectories in $\mathbb{R}^d$, where d is a constant. At any point of time, each point is either a facility or a client. The cost that arises for a facility persists during the entire time it is open. Analogously, a client permanently has to pay some cost for its connection to a facility. Our kinetic data structure maintains a subset of the moving points as facilities such that, at any time, the sum of the maintenance cost for the facilities and the connection cost for the clients is at most a constant factor larger than the current optimal cost.

The challenge is to construct a kinetic data structure whose underlying combinatorial structure is stable. To be able to ensure this, we keep up the invariant that, on the one hand, for each client there exists a facility in a certain local neighborhood and, on the other hand, no facility has another facility in a certain local neighborhood. The problem is now that restoring the invariant at one point (by changing the status of the point from facility to client or vice versa) can lead to a violation of the invariant at many other points. Our main technical contribution is a technique that allows us to restore the invariant in poly-logarithmic time.

The local scenario The main drawback of a global solution is that we cannot apply it in our distributed setting, where each node has to take its action by itself and only with the limited information given by its local neighborhood. Based on the methods developed in the global case, we introduce a simple distributed algorithm that is executed by each node and is used to determine whether the node should act as a facility or a client in the current situation. Since the distances between the nodes change over time, the algorithm constantly reevaluates its decision and, if necessary, changes the node's role in order to reestablish the approximation. Taken as a whole, the decisions of all the nodes yield a constant factor approximation of the FACILITY LOCATION problem.

An important property of our algorithm is the fact that each node only requires local information to be able to execute it: For each node p_i a certain value is computed. In order to compute this value, p_i requires only information about nodes that are within constant distance of p_i (i.e. a distance independent of the total number of nodes). In addition to that value, p_i requires information about the current role of all nodes p_j which are in a distance from p_i bounded from above by a constant. That value, the d_j's and the current roles of p_i's neighbors p_j is all that is necessary for p_i to determine its own role.

We introduce a characterization for a broad range of motion patterns to add dynamics to our local scenario. We will use the *dynamics parameter* to count the changes of the relative movement of pairs of robots. It turns out to be a powerful tool for the analysis of our algorithm and possibly for future algorithms in dynamic environments as well.

We describe and analyze this algorithm showing that, although the decisions of the nodes are based on local information, the system stabilizes in a solution that yields a global $\mathcal{O}(1)$- approximation. Furthermore, we prove that the process of finding the approximation only requires $\mathcal{O}(\log n)$ communication rounds, and -most important- that changes in the role or the radius of a node p_i only affect nodes within constant distance of p_i.

6.2. Related work

General FACILITY LOCATION The FACILITY LOCATION problem has been extensively studied in combinatorial optimization and operations research [CG99, GK98, JMS02, JV01, MYZ02, MP00]. In general, the problem is known to be $\mathcal{NP}$-complete. The first global constant factor approximation for the static setting was presented in [STA97]. For the Euclidean case, there exists a randomized PTAS [KR07]. However, the FACILITY LOCATION problem has also been investigated in other settings, for instance in dynamic settings [Ind04]. Nevertheless, this thesis is based on [DGL08a, DGL08b, DGL08c, DKP10] and no other algorithms are known for the kinetic or the local setting so far. Unfortunately, it seems that the only known $(1+\varepsilon)$-approximation given in [KR07] cannot be translated to neither the kinetic nor the local setting, since the authors use the Arora-scheme including dynamic programming techniques, which does not well comply with kinetization or local algorithms.

Kinetic data structures The kinetic data structure framework was introduced and applied on the convex hull problem by Basch et al. [BGH97]. Later kinetic data structures for measuring various descriptors of the extent of a set of points, including the diameter, width, or smallest bounding box, have been designed [AGHV97, AHPV04]. Several further algorithms that use the kinetic data structure framework have been developed, e.g., algorithms for kinetic collision detection [AdBPS09, BEG$^+$04, Her04, KSS00], kinetic planar subdivisions [AEG98, ABdB$^+$99, AGMV00], kinetic range searching [AAE03, BGZ97], kinetic *kd*-trees [AdBS07, AGG02], and kinetic connectivity for unit disks, rectangles, and hypercubes [GHSZ01, HS01]. Only some results are known for problems related to clustering, which the FACILITY LOCATION problem belongs to. For instance, Gao et al. [GGH$^+$01] provided a kinetic data structure to maintain an expected constant factor approximation for the minimal number of centers to cover all points for a given radius. The centers that they considered are a subset of the moving nodes, whereas Bespamyatnikh et al. [BBKS00] studied k-center problems for $k = 1$ in the kinetic data structure framework, where the centers are not necessarily located at the moving points. Another algorithm for the kinetic k-center problem can be found in [GGN04]. Hershberger [Her03] proposed a kinetic algorithm for maintaining a covering of the moving points in $\mathbb{R}^d$ by unit boxes such that the number of boxes is always within a factor of

3^d of the optimal static covering at any instance. Czumaj et al. [CFS07] presented a kinetic data structure for the Euclidean MaxCut problem. For other work on kinetic data structures, we refer to the survey by Guibas [Gui98].

Har-Peled [HP04] considered the k-center problem in a mobile setting different from the kinetic data structure framework. Instead of handling events, a static set is provided, which ensures a constant factor approximation at all times. However, a set of size $k^{\mu+1}$ is required, where μ is the degree of the polynomial of the trajectories.

Distributed and local algorithms Recently, a 7-approximation for the FACILITY LOCATION problem on a complete bipartite graph in a distributed setting was introduced in [PP09b] using a linear programming approach. Different to our approach they do not focus on locality issues, but rather on the size of transmitted messages, limiting them to $\mathcal{O}(\log n)$ bits. A uniform variant of the problem with $f_i = d_i = 1$ for all nodes was also considered in distributed and static settings: In [MW05], a $\mathcal{O}(k(mn)^{1/k}\log(m+n))$-approximation in $\mathcal{O}(k)$ communication rounds is achieved, with m and n being the number of facilities and clients. In [GLS06], three communication rounds are needed to calculate a $\mathcal{O}(1)$-approximation in the uniform setting. For both algorithms, the number of bits transferred each round is bounded by $\mathcal{O}(\log n)$ and both algorithms cannot easily be transferred to a dynamic setting. The same applies to the non-uniform variant introduced in [MW05], where the results are harder to compare to ours. In contrast to our constant factor approximation, in their work, there is always a dependency of the approximation factor on m and n like in the uniform case, but they do not require the distances to be a metric.

Some papers use different notions of locality. [FR07] considers the hop-distance and [PP09a] presents a distributed algorithm for the static FACILITY LOCATION problem in unit disk graphs. Contrary to our distributed algorithm, global coordinates are required to achieve a constant factor approximation.

While we present a rigorous analysis, there are also heuristic approaches, e.g.[KSW05]. For a recent survey on placing facilities in wireless mobile networks see [WS08].

[GLS06] applies the approach of Mettu and Plaxton [MP00], which is also a major building block in this work (confer to Section 6.4), both for the kinetic and the local algorithm. This approach has been successful in lot of other settings as well, for instance, in game theoretic settings [PT03] and for algorithms working in sub-linear time [BCIS05]. However, the metric FACILITY LOCATION problem has never been considered before in a dynamic and local setting.

6.3. Formal problem definition

We consider a special case of the metric FACILITY LOCATION problem defined in the following way: We are given a metric space $M(P, D(t))$ in which P represents a set of nodes and the function $D(t) : P \times P \rightarrow \mathbb{R}_{\geq 0}$ represents the distance between two nodes $p_i, p_j \in P$ at time $t \in \mathbb{R}_{\geq 0}$. We introduce dynamics to our problem by allowing the distances between nodes to change continuously over time.

For each point $p_i \in P$, there exists a positive *maintenance cost* $f_i \in \mathbb{R}$, that has to be paid at time t if p_i is a facility, and a positive *demand* $d_i \in \mathbb{R}$, which is the cost-factor that is multiplied with the distance to the closest facility when the p_i is a client. Note that both the maintenance cost and the demand of a point are provided as input and do not change over time.

Every node can fulfill one of two roles: It can either be a *facility* or a *client*. The role of a node is not fixed and can change over time. This means that, at any given time, we can subdivide the set P into the sets $F(t)$ and $G(t)$, where $F(t)$ contains all the nodes with the facility role at time t, while $G(t)$ contains all the client nodes. If the node p_i is a client, it causes costs $d_i \cdot D(t)(p_i, F)$, where $D(t)(p_i, F) := min_{p_k \in F}\{D(t)(p_i, p_k)\}$ is the distance between p_i and the closest facility to p_i at time t. Otherwise the node p_i is a facility and causes a cost of f_i.

Our goal is now to assign and maintain one of the two roles to each node in such a way that the function

$$cost(F) := \sum_{p_i \in F} f_i + \sum_{p_j \in G} d_j \cdot D(p_j, F)$$

is minimized at any point of time t. Note that changing from being a facility to client and the other way around does not incur any cost. Furthermore, the cost for the facility does not arise only once, but persists permanently for the entire time, the facility is open. However, our algorithms aim at providing that for any snapshot point of time, the given set of facilities yields a constant factor approximation for the corresponding static problem instance. If this is not the case, we want to quickly reconstruct a good solution.

Moreover, we want algorithms that enables the nodes to switch their roles, so that the subdivision into the sets $F(t)$ and $G(t)$ induced by their roles keeps the value of the $cost(t)$ function as small as possible during the changes of the metric function. The major goal is to reconstruct a good solution as fast as possible.

We will use the following naming conventions to facilitate talking about the problem we just described: A node being a facility will be referred to as *open*, while one being client will be called *closed*. Analogously, we say that a node *opens* when it changes its role from client to facility, respectively *closes* when it changes its role from facility to client. When appropriate, we will omit the index t representing the time and refer to $D(t)$, $F(t)$ and $G(t)$ as D, F and G in order to improve on the readability. Furthermore, we use the function $D(p_i, p_j, t)$ to describe the Euclidean distance between p_i and p_j at

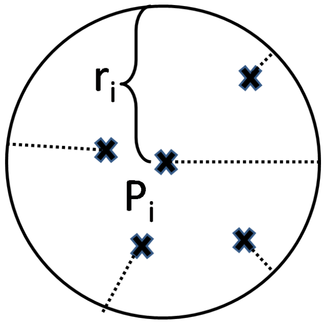


Figure 6.1.: Illustration of $\sum_{p_j \in B(p_i, r_i^{\mathrm{MP}})} d_j \cdot (r_i^{\mathrm{MP}} - D(p_i, p_j))$. The dotted lines weighted with the d_j's correspond to the distances summed up to f_i.

point of time t. We abbreviate this by $D(p_i, p_j)$, when clear from context, as well as we abbreviate $D(p_i, P, t) := \min_{p_j \in P}\{D(p_i, p_j, t)\}$, with $D(p_i, P)$.

6.4. The Mettu & Plaxton approach and radii

In [MP00], Mettu and Plaxton presented a simple greedy method for the static FACILITY LOCATION problem. This method defines in time $\mathcal{O}(n^2)$ a subset of a given point set P as facilities that leads to a total cost which is at most a factor of 3 larger than the optimal cost. The main idea of the algorithm is crucial for the remainder of this work. For this reason, we introduce the algorithm briefly and discuss its significant properties. They introduce so called *radii* and we use an approximation of those radii later on. Therefore we start by defining those radii.

6.4.1. Radius associated with a point

For a point $p_i \in P$ and a non-negative value r, we define $B(p_i, r)$ to be the ball with center p_i and radius r. Given such a ball $B(p_i, r)$, we let $weight(B(p_i, r))$ denote the sum of the demands of all the points in P that are located in the ball $B(p_i, r)$, i.e., we define

$$weight(B(p_i, r)) := \sum_{p_j \in P \cap B(p_i, r)} d_j \,.$$

For each point $p_i \in P$, we define the value r_i^{MP} to be the radius of the ball with center p_i that is used in [MP00] and satisfies

$$\sum_{p_j \in B(p_i, r_i^{\mathrm{MP}})} d_j \cdot (r_i^{\mathrm{MP}} - D(p_i, p_j)) = f_i \,.$$

For an illustration see Figure 6.1. Observe that the sum on the left side of the equality is continuous and strictly monotonically increasing with r_i^{MP}. Hence, there exists a unique value r_i^{MP} satisfying the equality. Moreover, at any point of time t and for any point $p_i \in P$, the radius r_i^{MP} ranges from $\frac{\min_{p_j \in P} f_j}{n \cdot \max_{p_j \in P} d_j}$ to $\frac{\max_{p_j \in P} f_j}{\min_{p_j \in P} d_j}$.

An intuitive explanation for the limits of a radius is that the higher the contribution of a point p_j to the sum is the smaller is the value of r_i^{MP}. More precisely, the lower limit of the range is met if

1. $f_i = \min_{p_j \in P} f_j$,
2. all the points in P are at the same position, and
3. the demands of all the points are uniform, such that $d_\ell = \max_{p_j \in P} d_j$ for any ℓ, $1 \leq \ell \leq n$.

Conditions 1 and 2 cause that the contribution of each point $p_j \in P$ to the sum is as high as possible. The upper limit of the range is met if

1. $f_i = \max_{p_j \in P} f_j$,
2. p_i is the only point in the ball with radius r_i^{MP} and center p_i, and
3. $d_i = \min_{p_j \in P} d_j$.

In this case, conditions 1 and 2 cause that the contribution of each point $p_j \in P \backslash p_i$ to the sum is 0.

The value r_i^{MP} is the value that we will refer to, when we talk about a *radius*. Later we will also define r_i^{KFL} and r_i^{LFL}, and call them in the corresponding context *radius* as well.

Now, we have all preliminaries to discuss the algorithm as presented by Mettu and Plaxton.

Algorithm 4 METTU-PLAXTON(P, t_0)

```
calculate the radius r_i^MP(t_0) for each point p_i(t_0) ∈ P(t_0)
sort all points in ascending order according to their radii
let p_1(t_0), p_2(t_0), ..., p_n(t_0) be the sorted sequence
for i = 1 to n do
    if there is no facility in B(p_i(t_0), 2 · r_i^MP(t_0)) then
        open facility at p_i(t_0)
```

The steps of the method developed by Mettu and Plaxton are listed in Algorithm 4. Let t_0 be a fixed point of time. Then, we can use algorithm METTU-PLAXTON to compute a set of facilities at this specific point of time t_0 for the global algorithm. In particular, we will apply a modified version of algorithm METTU-PLAXTON on the initial positions of the input points to get an initial set of facilities. No such step is necessary for the local algorithm. Based on the modified version, we will present a kinetic data structure with poly-logarithmic update time.

Note that we cannot apply the original Mettu-Plaxton algorithm to obtain a kinetic data structure with poly-logarithmic update time. The reason is that similar to maintaining an exact solution for the FACILITY LOCATION problem, keeping up the solution provided by algorithm METTU-PLAXTON is not stable. That means, a slight perturbation of the input might result in $\Omega(n)$ status changes (also confer to Chapter 7), whereas we are looking for stable solutions, where only a poly-logarithmic number of changes occur upon on an event.

For the local algorithm no explicit initialization phase is necessary, however, similar concepts apply also in the local algorithm.

As announced above, we will use modified definitions of the radius introduced in [MP00]. These are slightly different for the global and the local algorithm.

6.5. Organization of the part internal assignment

In this chapter we gave an introduction to the part of internal assignment and defined the FACILITY LOCATION problem. Now, we will compare the two approaches to the problem, that we use in the rest of the thesis. But, before actually introducing the algorithms, like in the first part we start with 'bad news' about the problem in Chapter 7 in terms of complexity, and lower bounds on the number of required changes and the crucial bounds on locality. The 'good news' in terms of approximation algorithms start with an introduction of the kinetic data structure framework that we use for the global algorithm in Chapter 8. The algorithm will be described and analyzed. Then we proceed to the model for the local algorithm, its description and analysis in Chapter 9. Finally we will conclude the part on internal assignment and discuss some open questions in Chapter 10.

6.6. Similarities and differences between the global and the local algorithm

Before we proceed to the lower bounds and the algorithms, we first describe what the main differences between the two settings are. We use the term *global algorithm* when we refer to the kinetic setting. The local setting is distributed and therefore harder by concept. However:

- The kinetic data structure has to manage points. For instance, sophisticated data structures are necessary to store the n^2 distances with poly-logarithmic overhead only. In a distributed setting this is not required, since each point just keeps its own data structure with at most n distances.

- The kinetic data structure has to manage explicitly the events. This happens implicitly in the local model. Actually the notion of events is quite different there, since they are only used as means for analysis and not for the design of the algorithm.

- There is a central control with a sophisticated algorithm in the kinetic setting. For the local algorithm the same ideas apply. However, the algorithm is just a simple control loop per point. The difficulty is rather the analysis than the algorithm itself.

- The kinetic data structure framework is well established. Therefore certain 'rules' apply. For instance, the points get an initial input of trajectories that are described by bounded degree polynomials. In the local case we introduce our own (more general) motion model.

- In the kinetic setting the order in which points are invoked is crucial. In the local case each node just acts, when it is its turn.

- For the global algorithm there is no difference whether two events are close to each other or far apart. If nodes that are far apart from each other change their relative position, the algorithm has to take care of this. The main feature of the local algorithm is that events have only an effect on their local neighborhood.

- Keeping track of the location of points is difficult in the kinetic setting as well. We have to model were the points are at a certain point of time. In the local case we just assume that the points reside at given locations and are aware of it (at least relative to its local neighborhood, which is sufficient). An additional problem with the analysis of the kinetic algorithm is the fact that we only use an approximation of the exact location of a point.

- The global algorithm applies only to the Euclidean case due to the kinetic data structure framework. We also describe the local algorithm for the Euclidean case. However, there will be a subsection devoted to non-Euclidean generalizations.

- We have a central property called radius, that has to be maintained. This is easy to do for a single node. Therefore this is not a problem for the local algorithm, but for the global algorithm an own data structure is required for this.

- There is no explicit initialization for the local case. We just launch the algorithm at all nodes and analyze the number of rounds until a stable state is reached for the first time. The nodes themselves are not aware of the fact, that they are in the initialization phase. In contrast, in the global case the initialization requires a runtime of order n^2 which is a major part of the entire runtime of the algorithm.

- In the global algorithm, we have a constant factor approximation at all times. In the local algorithm, we have to wait until the algorithm stabilizes before we can guarantee that we have constant factor approximation of the respective static setting.
- For kinetic data structures the trajectories of points have to be given as input, such that the time of events can be precomputed. It is a rather strong assumption that the movement of points in known in advance. This is not necessary for the local algorithm. The points move and the trajectory is only of interest for the analysis.
- For the local algorithm we consider only the number n of robots as input, since we assume that they are identically constructed. For the global algorithm we take additionally the f_i and d_i as input.

But there are similarities in the approach as well:

- Both algorithm pursue to keep two invariants, which are quite similar. However, in order to fulfill the requirements of the data structures the invariants for the kinetic setting are more complicated.
- When the invariants hold, this yields a global constant factor approximation for the given static setting.

The main difference between the two setting is that we have to restore our data-structure fast in the global setting and in the local setting it is crucial to argue why the order of activation of nodes is not important.

CHAPTER 7

The complexity of internal assignment

In this chapter, we deal with 'bad news'. There is already a lot known about the complexity theoretic limits, since FACILITY LOCATION is a prominent problem. However, besides the complexity of computation, there are other aspects that are of concern as well.

First, the ultimate goal is to have the viewing range of the robots to be as limited as possible. We will limit this viewing range to a constant, when dealing with our local algorithm and achieve a constant factor approximation. In this chapter we ask ourselves, whether we can get an improved approximation factor when allowing a larger viewing range (which is not the case due to [GK98]), or whether we can even limit the viewing range further, without getting a worse approximation factor (see Theorem 7.1).

Second, when dealing with dynamics, stable solutions get disturbed eventually. We want changes at one part of the system to stay in a bounded neighborhood and not to affect distant parts of the system. However, we will show that this is not possible for exact FACILITY LOCATION (see Theorem 7.2). This gives another reason, to use approximation algorithms instead of exact solutions. Note that this even holds if $\mathcal{P}$=$\mathcal{NP}$. Limiting changes to a local neighborhood is also the only way to accomplish fast updates on an existing solution. Furthermore, we cannot just apply the Mettu and Plaxton algorithm, because then the affected neighborhood is not bounded either (see Theorem 7.3).

Third, we do not want too many nodes to be affected by the role change of a single robot. Once again, for exact FACILITY LOCATION this turns out to be impossible (also see Theorem 7.2).

Fourth, the number of required rounds until a constant factor approximation can be achieved is of interest. In this respect, our algorithms can compete with state of the art constant-factor approximation algorithms for the non-uniform metric FACILITY LOCATION problem, since so far all these algorithms require $\Omega(\log n)$ rounds. Finding lower bounds in this model is still an important open problem according to [MW05]. We do not tackle this problem in this thesis, since we are mainly interested in questions concerning locality.

Last, we do not want our solution to change too often in total. This obviously depends on the motion pattern of the robots. For the global algorithm, we use the standard motion pattern of kinetic data structures, which are bounded degree polynomials. For the local algorithm we use a generalization of that motion model. In Section 7.3 we also state a lower bound from the literature, which fits to both our models.

7.1. The locality

Our local algorithm in Chapter 9 shows that, in order to compute a constant factor approximation, it is sufficient for each node to have information about other nodes within a constant distance from itself. It is known that (unless $\mathcal{NP} \subset \mathcal{DTIME}(n^{\mathcal{O}(\log\log n)})$) it is not possible (even if each node is allowed to see the entire graph) to achieve a better than constant approximation for general metrics in polynomial time [GK98]. (In the Euclidean case a (randomized) PTAS, which is based on the Arora scheme [Aro98], is known [ARR98]). Since our results for the local algorithm hold in general metrics, relaxing the locality constraints will not result in an improved approximation. Now the question arises whether the approximation factor changes when the locality is restricted such that nodes are only able to see other nodes within a distance smaller than constant. The following theorem states that we lose in the approximation factor when locality is restricted in such a way.

Theorem 7.1. *A distributed algorithm ALG limited to information about nodes within distance $1/f(n)$ (in a general metric) can at best achieve an $\Omega(f(n))$-approximation for functions f with $f(n) \in \Omega(1)$.*

Proof. Consider a star graph constructed in such a way that the distance from the center node p_0 to all other nodes is $\varepsilon/f(n)$ for an $\varepsilon > 1$. This star graph is now used to induce a metric M on the nodes. The metric is created by defining the distance between any two nodes as the shortest path in the star graph between them. We set $d_0 = f_0 = 0$ for the center node p_0 and $d_i = f_i = 1$ for the remaining nodes. Since an optimal algorithm OPT operating on M opens p_0 and keeps all other nodes closed, the resulting costs are 0 for the center node and $(n-1)\cdot\varepsilon/f(n)$ for the remaining nodes. See Figure 7.1.

Now, consider the metric M^∞ where all distances are set to infinity. Here, we also assign the $d_0 = f_0 = 0$ to p_0 and $d_i = f_i = 1$ to the remaining nodes. When operating on M^∞ any reasonable algorithm ALG is forced to open every single node, since closing at least one node raises the costs to infinity. Because all nodes are more than $1/f(n)$ away from each other and only have information about nodes within distance $1/f(n)$, ALG cannot distinguish the metric M from the metric M^∞ and therefore is forced to also open every single node when working on M. The ratio of the local algorithm ALG to the

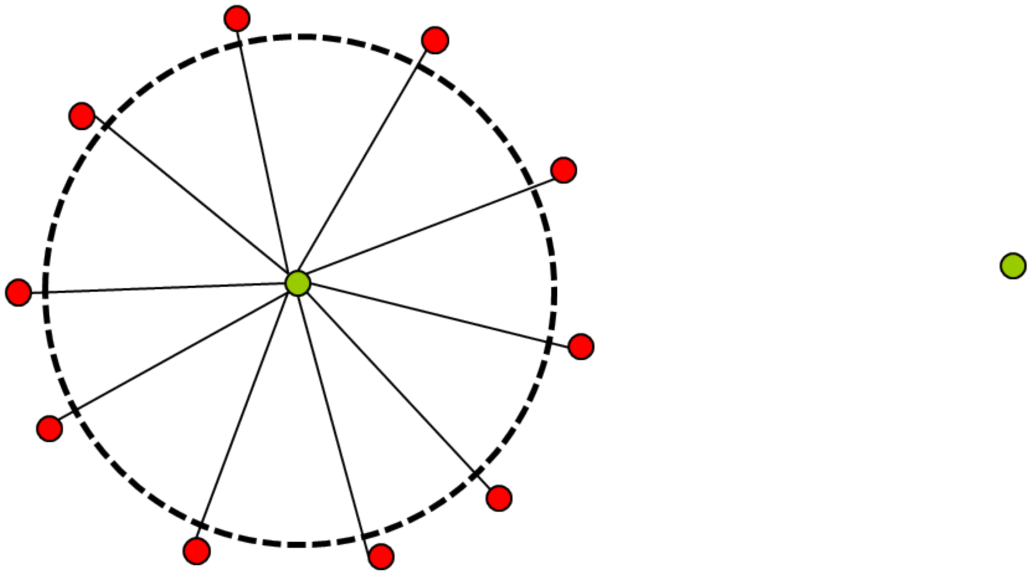


Figure 7.1.: In the graph on the left side all optimal clients are just outside of the viewing range. In the graph on the right side the other nodes are arbitrary far away. The two graphs cannot be distinguished locally.

optimal algorithm OPT is

$$\frac{\text{ALG}}{\text{OPT}} = \frac{n-1}{0+(n-1)\cdot\frac{\varepsilon}{f(n)}} = \frac{f(n)}{\varepsilon}$$

and thus $\text{ALG} \geq \frac{f(n)}{\varepsilon} \cdot \text{OPT}$. □

This proves our results concerning locality, which we will obtain in Chapter 9 to be tight.

7.2. Exact FACILITY LOCATION and the Mettu & Plaxton algorithm

We show that it is not possible to bound the distance between a node p_i and the nodes which are affected by p_i's change of role without using approximation. That is, we have to approximate, not only because we require a polynomial runtime, but also due to our locality constraints. The next theorem states that in contrast to our local approximation algorithm, where changing the role of a single node affects only its local neighborhood of constant size, an exact solution needs to change the role of nodes that are in linear distance from a single node which moves.

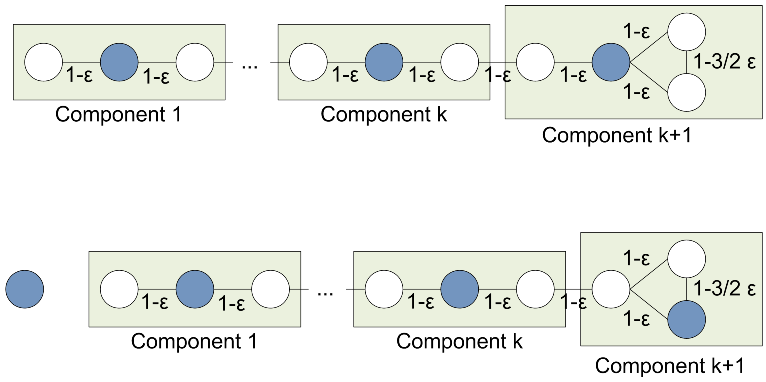


Figure 7.2.: Graph for the proof of Theorem 7.2

Theorem 7.2. *There exists a placement of nodes in the Euclidean plane such that the movement of a single node forces a solution for* exact FACILITY LOCATION *to perform* $\Omega(n)$ *role changes. In particular, there are nodes which change their role in distance* $\Omega(n)$ *from the moving node.*

Proof. Consider a graph in the Euclidean plane with n nodes where $n = 3k+4$ for some $k \in \mathbb{N}$. Let $d_i = f_i = 1$ for all i. The graph is a horizontal line where the distance between all nodes is $1-\varepsilon$ for some small $\varepsilon > 0$, except for the nodes at the very right. Here, the last two nodes are arranged in such a way that they are in distance $1-\varepsilon$ to the third-last node and in distance $1-\frac{3}{2}\varepsilon$ to each other (and in distance larger than 1 to all other nodes, see Figure 7.2). We will show in the remainder of the proof that moving the leftmost node to the left for more than ε will change the exact solution for the FACILITY LOCATION problem in $\Omega(n)$ nodes and hence covers an area which has a linear stretch.

We now first show the optimal solution for the original graph. For the sake of analysis, we group the nodes in k components of 3 nodes each plus a component consisting of the 4 nodes at the right end. We show that choosing exactly the middle nodes in each component yields an optimal solution: This solution has facility costs of $k+1$ and client costs of $(2k+3)\cdot(1-\varepsilon)$, since each of the $k+1$ facilities produces a cost of 1 and each other node a cost of $1-\varepsilon$. It is not possible to reduce the number of facilities, since in each component there needs to be at least one facility. Furthermore, the client costs can at most be reduced by $\frac{1}{2}\varepsilon$ for choosing the connection to the right instead of another one plus $1-\varepsilon$ times the number of additional facilities. Since opening additional facilities produces a cost of 1, this would increase the overall costs by ε for each additional facility. Furthermore, using the cheaper connection between the two rightmost nodes increases the number of facilities in this component by one. Since the number in the other components cannot be decreased, this would increase the overall costs by $\frac{\varepsilon}{2}$. Thus the described choice of facilities is optimal.

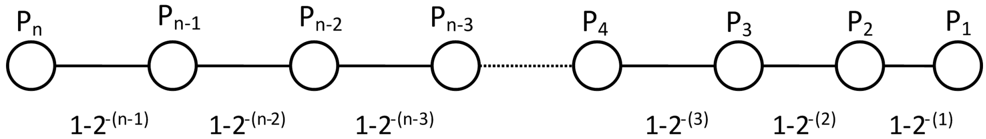


Figure 7.3.: An instance, where applying the Mettu and Plaxton algorithm will lead to a linear number of changes, after the movement of a single node.

Now we show that we have indeed $\Omega(n)$ role changes when moving the left node to the left. Eventually this node has to open. We arrange the remaining nodes in $k+1$ components again, but shift groups one unit to the right. The component on the very right now consists only of three nodes. With the same argument as above we need at least one facility in each component and it is cheapest to open the middle node in each except for the right one. Here it is now cheapest to open one of the nodes to the very right, since this does not increase the facility costs, but decreases the connection costs. The minimal costs in this scenario are therefore $(k+2)+(2k+1)\cdot(1-\varepsilon)+1-\frac{3}{2}\varepsilon$. For this operation, $2k$ nodes in the components have to change their role plus the node on the left side plus one of the two nodes on the right side. □

A similar argument holds for the algorithm of Mettu and Plaxton (confer to Algorithm 4 in Section 6.4).

Theorem 7.3. *There exists a placement of nodes in the Euclidean plane such that the movement of a single node results in $\Omega(n)$ role changes when the* METTU-PLAXTON *algorithm is applied. In particular, there are nodes which change their role in distance $\Omega(n)$ from the moving node.*

Proof. Consider a graph in the Euclidean plane with n nodes. Let $d_i = f_i = 1$ for all i. The graph is a horizontal line where the distance between all nodes is $1-\frac{1}{2^i}$, where the nodes are labeled from 1 to n. See Figure 7.3 for an illustration. Note that the nodes cover a linear stretch, since the distance between two neighbors is at least $\frac{1}{2}$. Observe that the radii decrease from left to right, with the smallest radius being $r_1^{\mathrm{MP}} = \frac{3}{4}$. Each node is the only node in its radius additionally to its right neighbor, except for node p_1. Lets consider algorithm METTU-PLAXTON. The nodes are handled from right to left. The first node to open is p_1. The nodes p_2 and p_3 are not opened. For p_2 it is obvious that p_1 is in its double radius. For p_3 observe that p_2 is contained in r_3^{MP} and $D(p_1,p_2) < D(p_2,p_3)$. The next point to open is p_4, because its distance to p_1 is $3-(\frac{1}{2^3}+\frac{1}{2^2}+\frac{1}{2^1}) > 2$, while the radius is at most 1. Therefore p_1 cannot be in its double radius. For symmetry reasons, this continues for the rest of the graph and every third node is opened.

Now, move p_1 far enough to the right. Rerunning algorithm METTU-PLAXTON will open p_1 and p_2. For symmetry reasons it will also open every third node again. But those nodes are shifted one node to the left now. Therefore $\Theta(n)$ facilities will close and $\Theta(n)$ facilities will open, covering a linear stretch. □

7.3. The effect of dynamics

Our algorithms change the set of facilities due to motion. How often the set changes depends on the motion pattern. Thus, we will analyze how often changes occur in dependency of the given motion pattern at hand. Here, we want to recall a known lower bound for changes of the solution, where the only motions are linear movements of robots only in one dimensional space enforcing $\Omega(n^2)$ changes.

In [GGH$^+$01] it is shown that there exists a set of n nodes moving linearly on the real line that forces any c-approximate cover to change $\Omega(n^2/c^2)$ times. Setting all f_i and c_i to 1 yields that any c-approximation of the FACILITY LOCATION problem in this setting has to change $\Omega(n^2/c^2)$ times as well. The proof is completely analogous to [GGH$^+$01] and we therefore omit it here. We just want to mention the main idea of the proof. We have $\frac{n}{2c}$ (for some c) many positions on the real line, each of these positions is the starting position for a group of robots. The speed of the single points are assigned in such a way that they all have pairwise a mutually distance that is large enough after some time. Again after some time they meet at the positions again. This procedure iterates $\Omega(n^2/c^2)$ times. Whenever the points are far apart, any c-approximate solution has open facilities at all points. When the points are at the predefined positions, in an optimal solution only one point is open at each of those positions. This forces the any constant factor approximation to change $\Omega(n^2/c^2)$ times and therefore $\Omega(n^2/c^2)$ motion-induced changes of the solution are inevitable.

We will show in Chapter 9 that so called events in our local algorithm only affect local neighborhoods and that the number of events can be upper bounded by $\mathcal{O}(x \cdot \log(n))$, where x is the dynamics parameter which will be introduced there. Since all nodes move linearly in the lower bound example, for the dynamics parameter holds $x = \mathcal{O}(n^2)$ in this example. Hence Theorem 9.9 in Chapter 9, which states that the number of events is upper bounded by $\mathcal{O}(x \log n)$ for any motion pattern is asymptotically tight up to the factor of $\log(n)$. The number of rounds until stable configurations can be reached is also only $\mathcal{O}(n^2 \log^2 n)$ in this scenario due to Corollary 9.10. Since the movement is linear and therefore in particular a bounded degree polynomial, the total processing time of the kinetic data structure as stated in Theorem 8.22 in Chapter 8 is also tight up to polylogarithmic factors.

In this chapter we have seen the minimal effort to maintain a constant-factor approximation for FACILITY LOCATION problem locally under motion. The viewing range has to be at least constant, exact solutions are not stable and even approximate solutions have to change at least a minimum number of $\Omega(n^2/c^2)$ times. For the remainder of the thesis it is left to show that those bounds can be (nearly) reached by approximation algorithms. Therefore, we show in the next chapter how to deal with dynamics first, before we proceed to our fully distributed local algorithm.

Chapter 8

The global scenario (A kinetic data structure)

This chapter is devoted to the global algorithm in the kinetic data structure framework. We start with a general introduction into the framework in Section 8.1. Then we introduce in Section 8.2 an important property that is used in our algorithm which is based on the term *radius* that we already discussed in Section 6.4. Then we describe our global algorithm in detail in Section 8.3 and finally analyze it in Section 8.4.

8.1. The model

When designing the global algorithm for our problem, we were looking for a proper framework to deal with moving objects. Although kinetic data structures are not designed for distributed algorithms, it is a good starting point for our investigations and our local algorithm will base on the ideas that we develop in this chapter. Therefore, we start with an overview over this model.

8.1.1. Kinetic data structures

The kinetic data structure framework is well-suited to maintain a combinatorial structure (an approximation of a solution for the FACILITY LOCATION problem in our case) of continuously moving objects and it is common in the field of computational geometry [AHPV04, BGH97, Gui98]. In this framework, we are given a set of objects and a *flight plan*, i.e., each object moves continuously along a known trajectory. Furthermore, at any point of time, it is possible to change the flight plan by performing a so-called *flight plan update*, which means that one object changes its trajectory. The main idea is now that the continuous motion of the objects is utilized in a way that updates of the current solution take place only at discrete points of time and can be computed fast. As a result, a lot of computational effort can be saved by maintaining the kinetic data structure compared to handling just a series of instances of the corresponding static problem. To guarantee that the required properties of the combinatorial structure are satisfied at any point of time, a kinetic data structure ensures that *certificates* which have to be defined for each problem are always valid. This indicates that the combinatorial structure is maintained. Whenever a certificate fails, we call this an *event*, and an *update* is required to restore the certificates and therefore the combinatorial structure. To be able to handle each of these events at the correct time, an *event queue* is maintained. In the event queue all events are stored with time stamps. They are precomputed on basis of the trajectories that are given as in input.

There are four important properties to measure the quality of a kinetic data structure. The worst-case amount of time to process an update is called *responsiveness*. The second and third property are *compactness* and *kinetic locality*. The compactness is given by the ratio between the maximum number of certificates ever present to guarantee the required properties of the combinatorial structure and the number of the moving objects. An object might be involved in several events in the event queue. The kinetic locality addresses the maximum number of events in the queue, for any one object. As a result, the kinetic locality is a measure for how easily flight plan updates can be performed. Is is important to note that the term 'locality' is used in different way in the context of kinetic data structures than it is used in the rest of this thesis. Usually 'kinetic locality' is just called 'locality' in the context of kinetic data structures, but in order to separate it from our notion, we will refer to it by 'kinetic locality'.

The fourth property, the *efficiency* of a kinetic data structure, is the worst case ratio between the number of total events processed by the kinetic data structure and the minimum number of events that would have been sufficient to maintain a solution for the given kinetic problem for an arbitrary instance. For a more detailed description of these concepts, the reader is referred to [BGH97, Gui98]. We say that a kinetic data structure is responsive, compact, kinetic local, and efficient, respectively, if the associated value is at most poly-logarithmic in the size of the input. Thus, some additional overhead is allowed, but it is limited to logarithmic factors, which is a common convention in the domain of kinetic data structures.

8.2. The special radii

In this section, we present the essential ideas which our kinetic data structure (and later the local algorithm) is based on and introduce some basics and notations used throughout this part of the thesis. At first, we introduce a new radius associated with a point that it is much easier to maintain when the points move than the radius r_i^{MP} defined in Section 6.4. Compared to the previous definition, the new radii of the points depend on cubes instead of balls. The key idea of our kinetic data structure is to use a set of nested cubes around each point and to update the kinetic data structure each time a point enters or leaves a cube of another point.

8.2.1. Definition of the special radii

8.2.1.1. Cubes

Similar to the definition of balls, for a point $p_i \in P$ and a non-negative value r, we define $C(p_i, r)$ to be the axis-parallel cube whose center is the point p_i and whose side length is $2r$. Given such a cube $C(p_i, r)$, we let $weight(C(p_i, r))$ denote the sum of the demands of all the points in P that are located in the cube $C(p_i, r)$, i.e., we define

$$weight(C(p_i, r)) := \sum_{p_j \in P \cap C(p_i, r)} d_j \,.$$

Note that the cube $C(p_i, r)$ is a ball with radius r with respect to the L_∞-metric. In the following, we will refer to the value r of a cube $C(p_i, r)$ as the radius of the cube, i.e. the double radius of a cube is equal to its side length.

8.2.1.2. Radius Associated with a Point

Our kinetic data structure maintains for each point $p_i \in P$ an approximation of r_i^{MP} as defined in Section 6.4, which we denote by r_i^{KFL}. We define r_i^{KFL} to be the value 2^{k^*}, such that $k^* = k_0 + \lceil \log_2(4\sqrt{d}) \rceil$ and k_0 is the minimum integer k with $\log_2(\frac{\min_{p_j \in P} f_j}{n \cdot \max_{p_j \in P} d_j}) \leq k \leq \log_2(\frac{\max_{p_j \in P} f_j}{\min_{p_j \in P} d_j})$, for which $weight(C(p_i, 2^{k_0})) \geq f_i \cdot 2^{-k_0}$ holds. We will prove that, at any point of time t, the special radius r_i^{KFL} of any point $p_i \in P$ is a constant factor approximation for the value r_i^{MP}. Similarly we will define the radius for the local algorithm in Section 9.2. Note that our choice of the radius leads to discretization of the radius. While r_i^{MP} takes values form a continuous range this not the case for r_i^{KFL}, where the set of potential radii is known in advance. Each of the potential value differs form the next potential radius by a factor of two. Therefore, we get only a logarithmic number of potential radii for each point. The exact upper and lower bounds as stated here, follow from the range that the radius takes as explained in Subsection 6.4.1 and the proof that we will see for Lemma 8.3 later on.

In the following, we prove in Lemma 8.1 that, at any point of time t, the special radius r_i^{KFL} of any point $p_i \in P$ is a constant factor approximation for the value r_i^{MP}.

The proof is based on two results obtained in [BCIS05]. For the uniform FACILITY LOCATION problem, the authors in [BCIS05] gave lower and upper bounds for the value r_i^{MP} and showed how to approximate r_i^{MP} by counting the number of points in a certain ball around p_i. We generalize their two results to the non-uniform case, by considering a weighted sum instead of a simple sum:

Lemma 8.1. *For any point of time t and for each $p_i \in P$, we have*

$$\frac{f_i}{weight(B(p_i, r_i^{MP}))} \leq r_i^{MP} \leq \frac{2 \cdot f_i}{weight(B(p_i, \frac{r_i^{MP}}{2}))} .$$

Proof. From the definition of r_i^{MP} follows $\sum_{p_j \in B(p_i, r_i^{\mathrm{MP}})} d_j \cdot r_i^{\mathrm{MP}} \geq f_i$, so that

$$r_i^{\mathrm{MP}} \geq \frac{f_i}{\sum_{p_j \in B(p_i, r_i^{\mathrm{MP}})} d_j} = \frac{f_i}{weight(B(p_i, r_i^{\mathrm{MP}}))} .$$

Furthermore, we get

$$\begin{aligned} f_i &= \sum_{p_j \in B(p_i, r_i^{\mathrm{MP}})} d_j \cdot \left(r_i^{\mathrm{MP}} - D(p_i, p_j)\right) \\ &\geq \sum_{p_j \in B(p_i, \frac{r_i^{\mathrm{MP}}}{2})} d_j \cdot \left(r_i^{\mathrm{MP}} - D(p_i, p_j)\right) \\ &\geq \frac{r_i^{\mathrm{MP}}}{2} \cdot \sum_{p_j \in B(p_i, \frac{r_i^{\mathrm{MP}}}{2})} d_j \\ &= \frac{r_i^{\mathrm{MP}}}{2} \cdot weight(B(p_i, r_i^{\mathrm{MP}}/2)) , \end{aligned}$$

where the second inequality follows from the fact that $r_i^{\mathrm{MP}} - D(p_i, p_j) \geq 0$ for all $p_j \in B(p_i, r_i^{\mathrm{MP}})$ and $B(p_i, \frac{r_i^{\mathrm{MP}}}{2}) \subseteq B(p_i, r_i^{\mathrm{MP}})$. □

The above lemma helps to calculate an approximation of the radius by counting the weighted sum of points in an orthogonal range around a point. Those operations are supported by our range trees, which are the main data structures and which we will explain in Section 8.3.2. Therefore we can use this method to calculate the radius r_i^{KFL} that we will use. However, we still need to show that our discretization is still a good enough approximation of the radius r_i^{MP} that was originally used. That is what the next lemma states, which is once again a generalization of the uniform case considered in [BCIS05].

Lemma 8.2. *At point of time t, let k_1 be the minimum integer k with $\lceil \log_2(\frac{\min_{p_j \in P} f_j}{n \cdot \max_{p_j \in P} d_j}) \rceil \leq k \leq \lfloor \log_2(\frac{\max_{p_j \in P} f_j}{\min_{p_j \in P} d_j}) \rfloor$, such that $weight(B(p_i, 2^k)) \geq f_i \cdot 2^{-k}$. Then $\frac{1}{2} \cdot r_i^{MP} \leq 2^{k_1} \leq 2 \cdot r_i^{MP}$ holds.*

Proof. Due to the choice of k_1, we have $weight(B(p_i, 2^{k_1 - 1})) < f_i \cdot 2^{-(k_1 - 1)}$. It follows that, for any $r_i^{\mathrm{MP}} < 2^{k_1 - 1}$, we get

$$\begin{aligned} weight(B(p_i, r_i^{\mathrm{MP}})) &\leq weight(B(p_i, 2^{k_1 - 1})) \\ &< f_i \cdot 2^{-(k_1 - 1)} \\ &< f_i \cdot \frac{1}{r_i^{\mathrm{MP}}} . \end{aligned}$$

Now, we have $r_i^{\mathrm{MP}} < \frac{f_i}{weight(B(p_i, r_i^{\mathrm{MP}}))}$, which is a contradiction to Lemma 8.1. Hence, $r_i \geq 2^{k_1 - 1}$ must be true, which proves the second inequality.

Furthermore, for any $r_i^{\mathrm{MP}} > 2^{k_1+1}$, we have

$$\begin{aligned} weight(B(p_i, r_i^{\mathrm{MP}}/2)) &\geq weight(B(p_i, 2^{k_1})) \\ &\geq f_i \cdot 2^{-k_1} \\ &> f_i \cdot \frac{2}{r_i^{\mathrm{MP}}}. \end{aligned}$$

In this case, it follows that $r_i^{\mathrm{MP}} > \frac{2 f_i}{weight(B(p_i, r_i^{\mathrm{MP}}/2))}$, which is again a contradiction to Lemma 8.1. Thus, we have $r_i^{\mathrm{MP}} \leq 2^{k_1+1}$, which proves the first inequality. □

Our algorithm uses the approach of [BCIS05], but we approximate the sum of the demands of all the points in a distance 2^k, for an integer k, by the sum of the demands of all the points in a cube instead of a ball with radius 2^k. This way we can use orthogonal range queries which are supported by our data structures. The following lemma shows that using a cube still gives a constant factor approximation of r_i^{MP}.

Lemma 8.3. *At point of time t, let k_0 be the minimum integer k with $\lceil \log_2(\frac{\min_{p_j \in P} f_j}{n \cdot \max_{p_j \in P} d_j}) \rceil \leq k \leq \lfloor \log_2(\frac{\max_{p_j \in P} f_j}{\min_{p_j \in P} d_j}) \rfloor$, such that $weight(C(p_i, 2^k)) \geq f_i \cdot 2^{-k}$. Then $\frac{1}{4\sqrt{d}} \cdot r_i^{MP} \leq 2^{k_0} \leq 2 \cdot r_i^{MP}$ holds.*

Proof. Let k_1 be defined as in Lemma 8.2. Then the radius of $C(p_i, 2^{k_0})$ is at most 2^{k_1}, since each point in P, that is located in $B(p_i, 2^{k_1})$, is also located in $C(p_i, 2^{k_1})$, so that we get $weight(C(p_i, 2^{k_1})) \geq f_i \cdot 2^{-k_1}$. Furthermore, the radius of $C(p_i, 2^{k_0})$ is larger than $\frac{1}{\sqrt{d}} \cdot 2^{k_1 - 1}$. The reason is that $weight(B(p_i, 2^{k_1-1})) < f_i \cdot 2^{-(k_1-1)}$ and $weight(C(p_i, \frac{1}{\sqrt{d}} \cdot 2^{k_1-1})) \leq weight(B(p_i, 2^{k_1-1}))$, so that we have

$$\begin{aligned} weight(C(p_i, 2^{k_1 - 1 - \log_2(\sqrt{d})})) &= weight(C(p_i, \frac{1}{\sqrt{d}} \cdot 2^{k_1-1})) \\ &< f_i \cdot 2^{-(k_1-1)} \\ &< f_i \cdot 2^{-(k_1 - 1 - \log_2(\sqrt{d}))}. \end{aligned}$$

Now, due to the fact that $2^{k_0} > \frac{1}{\sqrt{d}} \cdot 2^{k_1-1}$ and Lemma 8.2, the lemma follows. The maximum and minimum radius of $C(p_i, 2^{k_0})$ is illustrated in Figure 8.1. □

Due to Lemma 8.3, we have $2^{-\log_2(4\sqrt{d})} \cdot r_i^{\mathrm{MP}} \leq 2^{k_0} \leq 2 \cdot r_i^{\mathrm{MP}}$. In order to get $r_i^{\mathrm{MP}} \leq r_i^{\mathrm{KFL}}$, we set $r_i^{\mathrm{KFL}} = 2^{k^*} = 2^{k_0 + \lceil \log_2(4\sqrt{d}) \rceil}$. This implies $r_i^{\mathrm{MP}} \leq r_i^{\mathrm{KFL}} \leq 2^{3 + \lceil \log_2(\sqrt{d}) \rceil} \cdot r_i^{\mathrm{MP}}$.

Due to the restriction of the possible values for k, we have to consider only $\mathcal{O}(\log(nR))$ possible values for the radii, where $R := \frac{\max_{p_i \in P} f_i \cdot \max_{p_i \in P} d_i}{\min_{p_i \in P} f_i \cdot \min_{p_i \in P} d_i}$ and nR is the ratio between the upper and lower limit of the radii (see Subsection 6.4.1).

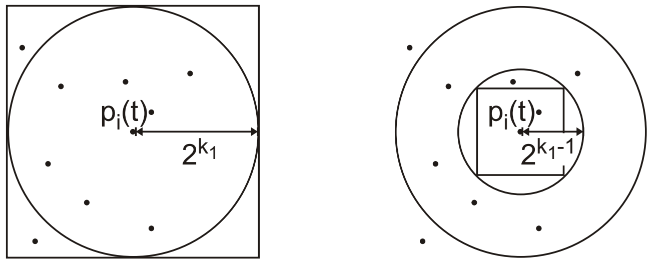


Figure 8.1.: Maximum and minimum radius of $C(p_i, 2^{k_0})$.

8.2.1.3. The ratio R

We have just defined R to be $\frac{\max_{p_i \in P} f_i \cdot \max_{p_i \in P} d_i}{\min_{p_i \in P} f_i \cdot \min_{p_i \in P} d_i}$. Note that it is necessary to consider R because the number of possible radii directly depends on this ratio and the number of radii is part of most of our bounds. On the other hand, all our bounds will only depend on $\mathcal{O}(\log(nR))$, and this term is only dominated by R if it is super-polynomial in n. So, for practical purposes, if the ratios of d_i's and f_i's do not differ that much, we can omit the parameter R in all our bounds. We will do so in Chapter 9, where we assume that we have structurally identical robots.

8.2.1.4. Walls around a Point and certificates

As stated above, there are $\mathcal{O}(\log(nR))$ potential radii for each point $p_i \in P$. This leads to a set of $\mathcal{O}(\log(nR))$ nested cubes for each point $p_i \in P$, each cube $C(p_i, 2^k)$ defined by a potential radius 2^k for each $k \in \{\lceil \log_2(\frac{\min_{p_j \in P} f_j}{n \cdot \max_{p_j \in P} d_j}) \rceil + \lceil \log_2(4\sqrt{d}) \rceil, \lceil \log_2(\frac{\min_{p_j \in P} f_j}{n \cdot \max_{p_j \in P} d_j}) \rceil + 1 + \lceil \log_2(4\sqrt{d}) \rceil, \ldots, \lfloor \log_2(\frac{\max_{p_j \in P} f_j}{\min_{p_j \in P} d_j}) \rfloor + \lceil \log_2(4\sqrt{d}) \rceil \}$. The side faces of the cube defined by $C(p_i, 2^k)$ form a wall around p_i, which we call $W_{i,k}$. Hence, there exists a set of $\mathcal{O}(\log(nR))$ walls for p_i. We use this set of walls to determine the points of time when an update of p_i in our kinetic data structure is required. These are our *certificates*: in each dimension we keep a sorted list of the walls and the points. When two of the items in one of the lists change their order, we call this an event. In general, an event occurs each time when any point crosses any wall of another point. Certainly not all events require an update operation, but each necessary update operation will be covered by an event. Therefore the number of events is an upper bound on the number of updates.

The technical details will be explained in Subsection 8.3.5.1.

8.3. Description of the global algorithm

Before going into detail on the description of the algorithm, we give an overview over the elements of the algorithm and how they interact in Subsection 8.3.1. Afterwards, we describe how the radius r_i^{KFL} as defined in the last section is computed in our scenario in Subsection 8.3.2, then we define in Subsection 8.3.3 an invariant which guarantees a constant factor approximation. The invariant is maintained over time. Afterwards, we show in Subsection 8.3.4 how the initialization can be done in order to get a first solution that has the desired properties, like yielding a constant factor approximation. Then we give in Subsection 8.3.5 a description of the kinetic data structure. It consists of two parts. One is an event queue, where we store when and where updates have to be performed in order to keep our invariant. The other one is an description how an update is performed. In the last section of this chapter (Section 8.4) follows an analysis of the kinetic data structure in terms of approximation quality, runtime and storage requirements.

8.3.1. How the parts of the model interact

Before starting the technical description of the algorithm, we want to sum up the main idea: First we compute an initial set of facilities such that the invariant (which guarantees a good structure) holds. Then the algorithm starts. We have points and walls around the points representing the potential radii. All points and walls are stored in sorted lists for each dimension. Due to motion, the sorting (which are our certificates) in the lists changes. When this happens, we call this an event. These events are stored in the event queue. The main algorithm processes the events one after the other. Upon an event an invariant might get violated. Then the set of facilities has to be updated, such that the invariant holds again.

Now, we show how we realize the idea above in an efficient way.

8.3.2. Computation of the special radii

In order to compute, at any point of time, the special radius associated with a point efficiently, we maintain two $(d+1)$-dimensional dynamic range trees denoted by T_1 and T_2. At any time, range tree T_1 is used to manage the current set of facilities, and T_2 stores the current set of clients. Apart from the fact that the two data structures contain different point sets, they are constructed in the same way. In the first d levels of the range trees, the points are handled according to their coordinates and in the $(d+1)$-st level according to their special radii. Additionally, with each node v in every binary search tree of the $(d+1)$-st level, we store the sum of the demands of all the points contained in the subtree of v. Besides the two range trees, we maintain a binary search tree T that contains, for each point in P, a pair consisting of the point's index and its current status (which is either open or closed). T is sorted according to the indices.

The dynamic data structure described in [BGZ97] supports all required properties of T_1 and T_2 efficiently. In particular, a range tree for a set of n points in $\mathbb{R}^{d+1}$ has size $\mathcal{O}(n\log^d(n))$ and can be built in $\mathcal{O}(n\log^{d+1}(n))$ time. We can maintain this data structure in $\mathcal{O}(\log^{d+1}(n))$ worst case time per insertion and deletion. Given any orthogonal range in $\mathbb{R}^{d+1}$, we can output the points inside this range in $\mathcal{O}(\log^{d+1}(n)+N)$ time, where N is the output size. Due to the additional information stored at each node in level $d+1$ of the range trees, we can also compute the sum of the demands of all the points in a certain range in $\mathbb{R}^{d+1}$ in $\mathcal{O}(\log^{d+1}(n))$ time. Finally, we can output the status of a given point in $\mathcal{O}(\log(n))$ time by querying T.

At any time t, the range trees rely on the relative position of the points in P. More precisely, the leaves of any binary search tree of any level ℓ, $1 \le \ell \le d$, in T_1 and T_2 store the points sorted according to their ℓ-th coordinate. We say that these points are sorted according to their ranks based on dimension ℓ. Now, the movement of the points in P is reflected by insert and delete operations on T_1 and T_2. At each point of time t, when any two points $p_i, p_j \in P$ change their ranks based on any dimension ℓ, we delete p_i and p_j from T_1 and T_2 and reinsert them according to their position at time t.

Note that we have to rely on the efficient range tree operations. This is the reason why we have to approximate the radius r_i^{MP} by r_i^{KFL}. Otherwise we could not use range trees, since they do not support queries on balls.

8.3.3. The invariant

The key idea of our kinetic data structure is to keep up one invariant consisting of the following conditions:

(a) for each closed point $p_i \in G$ there is an open point $p_j \in F$ with $r_j^{\mathrm{KFL}} \le r_i^{\mathrm{KFL}}$ in $C(p_i, 4\cdot r_i^{\mathrm{KFL}})$ and

(b) for each open point $p_i \in F$ there is no other open point $p_j \in F$ with $r_j^{\mathrm{KFL}} \le r_i^{\mathrm{KFL}}$ in $C(p_i, 2\cdot r_i^{\mathrm{KFL}})$.

The choice of conditions (a) and (b) enables our kinetic data structure to be stable. Moreover, the following proposition shows that, by keeping up conditions (a) and (b), we maintain a set of facilities that leads to a total cost which is at most a constant factor larger than the optimal cost. Thus, the goal of our kinetic data structure is to restore the invariant each time it is violated. Note that the invariant for one point p_i takes into account only nodes with a smaller radius. This enables us later on to ensure that if the invariant is not violated for all points with a radius bounded above by the same value, it will not be violated unless an event occurs. We will use this property when analyzing the algorithm in Section 8.4.

Proposition 8.4. *If the invariant is satisfied at time t, then we have*

$$cost(F) \leq (64d+1) \cdot cost(F_{\text{OPT}})\,.$$

Proof sketch. For each point $p_i \in P$, there is a facility $p_j \in F$ with radius $r_j^{\text{KFL}} \leq r_i^{\text{KFL}}$ in $C(p_i, 4 \cdot r_i^{\text{KFL}})$. Since $r_i^{\text{MP}} \leq r_i^{\text{KFL}} \leq 2^{3+\lceil \log_2(\sqrt{d}) \rceil} \cdot r_i^{\text{MP}}$, we get $D(p_i, p_j) \leq \sqrt{d} \cdot 4 \cdot r_i^{\text{KFL}} \leq \sqrt{d} \cdot 4 \cdot 2^{3+\lceil \log_2(\sqrt{d}) \rceil} \cdot r_i^{\text{MP}} \leq 64d \cdot r_i^{\text{MP}}$. Now, the proposition follows from the analysis in [MP00]. Details can be found in Section 8.4.1. □

8.3.4. Initialization

Let $p_i(t_0)$ denote the initial position of the point $p_i \in P$. To compute an initial set of facilities, such that the invariant is satisfied, we apply Algorithm 5, which is a modified version of Algorithm 4, on the point set $P(t_0)$. The modification is that, instead of following the exact sorted sequence of the $r_i^{\text{MP}}(t_0)$ values, we round each $r_i^{\text{MP}}(t_0)$ to one of the $\mathcal{O}(\log(nR))$ possible values for the special radii (i.e., its corresponding $r_i^{\text{KFL}}(t_0)$ value) and follow the sorted sequence of the rounded values. Note, that this is needed in the global case only.

Algorithm 5 MODIFIED-METTU-PLAXTON(P, t_0)

calculate the radius $r_i^{\text{KFL}}(t_0)$ for each point $p_i(t_0) \in P(t_0)$
let I_k be the set of indices of all the points with radius $2^{k+\lceil \log_2(4\sqrt{d}) \rceil}$
for $k \leftarrow \lceil \log_2(\frac{\min_{p_j \in P} f_j}{n \cdot \max_{p_j \in P} d_j}) \rceil$ **to** $\lfloor \log_2(\frac{\max_{p_j \in P} f_j}{\min_{p_j \in P} d_j}) \rfloor$ **do**
    for each $i \in I_k$ **do**
        if there is no facility in $C(p_i(t_0), 2 \cdot 2^{k+\lceil \log_2(4\sqrt{d}) \rceil})$ **then**
            open facility at $p_i(t_0)$

8.3.5. The kinetic data structure

This section addresses the design of our kinetic data structure for the FACILITY LOCATION problem. In particular, we describe how the event queue is structured and how an update of the kinetic data structure is processed.

8.3.5.1. Event queue

In order to maintain the invariant defined above, we have to update our kinetic data structure at certain points of time. More precisely, we perform an update each time a point p_j crosses a wall $W_{i,k}$, where $\lceil \log_2(\frac{\min_{p_j \in P} f_j}{n \cdot \max_{p_j \in P} d_j}) \rceil + \lceil \log_2(4\sqrt{d}) \rceil \leq k \leq \lfloor \log_2(\frac{\max_{p_j \in P} f_j}{\min_{p_j \in P} d_j}) \rfloor + \lceil \log_2(4\sqrt{d}) \rceil$, of another point p_i. For technical reasons we call the crossing of a wall p_j with a wall of p_i an event as well. It is just crucial that each possible point of time, where an invariant might get violated is captured by an event.

To keep track of these events, we use the following data structure: For each dimension ℓ, $1 \leq \ell \leq d$, we store all n points and all $\mathcal{O}(n \cdot \log(nR))$ wall faces that are orthogonal to the ℓ-th coordinate axis in a list sorted by the ℓ-coordinate. For each consecutive pair in each of the d lists, we keep up one certificate to certify the sorted order of the lists. We define the failure time of the certificate for any pair of consecutive objects to be the first future time when these objects change their ranks in their sorted list. The failure times of all certificates are maintained in one event queue.

In case that more than one event occurs at the same time, we handle them in an arbitrary order. Certainly, it is not the case that each event implicates that a point crosses a wall of another point (as, e.g., the change of the rank of two wall faces also causes an event), but definitely every crossing of a wall is discovered by a failure of at least one certificate. The event queue has the following complexity:

Lemma 8.5. *The event queue for the kinetic* FACILITY LOCATION *problem has size $\mathcal{O}(n\log(nR))$, can be initialized in $\mathcal{O}(n\log^2(nR))$ time, and can be updated in $\mathcal{O}(\log(nR))$ time. Provided that each trajectory can be described by a bounded degree polynomial, the total number of events is $\mathcal{O}(n^2\log^2(nR))$. A flight plan update involves $\mathcal{O}(\log(nR))$ certificates and requires $\mathcal{O}(\log^2(nR))$ time.*

Proof. Since there are $\mathcal{O}(n\log(nR))$ elements in the d lists, the initialization of these lists and of the event queue can be done in $\mathcal{O}(n\log(nR)\log(n\log(nR))) = \mathcal{O}(n\log^2(nR))$ time by sorting operations. In each following update we have to re-calculate the points of time when the two objects involved in the current event change their ranks with their two neighbors in the corresponding list. Thus, a constant number of events have to be updated in the event queue. Since the event queue contains $\mathcal{O}(n\log(nR))$ elements and we can use a min-heap to realize it, an update of an event requires $\mathcal{O}(\log(n\log(nR))) = \mathcal{O}(\log(nR))$ time. Furthermore, a flight plan update of a point causes a re-calculation of the points of time when the point and all its wall faces change their ranks with the associated neighbors in all d lists. Afterwards, the involved certificates are updated in the event queue. This can be accomplished in $\mathcal{O}(\log^2(nR))$ time.

In case that each trajectory can be described by a bounded degree polynomial and no flightplan update occurs, the upper bound on the total number of events is given as follows. For each pair of elements, an event occurs when the trajectories of the two elements cross each other. The number of cuts of two polynomials is bounded by the maximum degree of both polynomials. Hence, the total number of cuts of $\mathcal{O}(n\log(nR))$ bounded degree polynomials is $\mathcal{O}(n^2\log^2(nR))$. The upper bound on the space requirement is obvious. □

Note, that the term 'event' is part of the algorithm, as it is usually the case for kinetic data structures. In Chapter 9 we will use the term 'event' as well, since it occurs under similar circumstances. However, there it is only a means of analysis instead of part of the algorithm.

8.3.5.2. Handling an update

In this subsection, we describe how an event e, occurring at any point of time t, is handled (confer Algorithm 6)). As the first step, the event queue is updated as explained in Subsection 8.3.5.1. Then, we have to distinguish between the following three cases:

(i) Both objects involved in the considered certificate are faces of walls.

(ii) Both objects involved in the considered certificate are points.

(iii) One object involved in the considered certificate is a point and the other one is a face of a wall.

The handling of the three cases mainly depends on whether the invariant is violated or not. A point p_i violates the invariant at a point of time t in the following case: Either (a) p_i is closed but there is no facility with radius smaller than or equal to r_i^{MP} in $C(p_i, 4 \cdot r_i^{\text{KFL}})$, or (b) p_i is open but there is another facility with radius smaller than or equal to r_i^{MP} in $C(p_i, 2 \cdot r_i^{\text{KFL}})$. We assume that the invariant is satisfied by the time when e occurs and we describe in this section how the solution is changed so that the invariant holds again afterwards (see algorithm RESTORE). The proof of correctness can be found in Section 8.4.2.

In case (i), no point crosses the wall of another point. As a result, the invariant is still satisfied, so that handling e is finished.

In case (ii), the event indicates that a point p_i and another point p_j change their ranks based on a dimension ℓ, $1 \leq \ell \leq d$. This means that we have to update the position of p_i and p_j in the range trees T_1 and T_2. Since no point crosses a wall of another point, handling e is finished.

In case (iii), it might be that the invariant is violated. Let p_j be the first object involved in the considered certificate and p_i be the point whose wall is the second object involved in the considered certificate. Thus p_j crosses a wall of p_i. We update the radius $r_i^{\text{KFL}} = 2^{k^*}$, such that $k^* = k_0 + \lceil \log_2(4\sqrt{d}) \rceil$ and k_0 is the minimum integer k with $\log_2(\frac{\min_{p_j \in P} f_j}{n \cdot \max_{p_j \in P} d_j}) \leq k \leq \log_2(\frac{\max_{p_j \in P} f_j}{\min_{p_j \in P} d_j})$, for which $weight(C(p_i, 2^{k_0})) \geq f_i \cdot 2^{-k_0}$ holds. We will show that the new value of k_0 differs from its value before e by at most 1. Thus, there are three possible values for k_0, where each value can be tested by one range query on both T_1 and T_2. Afterwards, we test if p_i violates the invariant by using a range query on T_1. If this is the case, we change the status of p_i. As an effect of changing the radius or the status of one point, the invariant may be violated by many other points (e.g., their open facility has been closed). In the following, we will show how to deal with this problem (confer Algorithm 7).

Algorithm 6 KINETICFL(P, t_0, M)

MODIFIED-METTU-PLAXTON(P, t_0)
initialize event queue Q
while Q is not empty **do**
  $e \leftarrow$ dequeue(Q)
  update Q
  if e indicates that p_i and p_j change their ranks in any list for any i, j **then**
    update position of p_i and p_j in T_1 and T_2
  else
    if e indicates that p_j crosses a wall of p_i for any i, j **then**
      update $r_i^{\text{KFL}} \leftarrow 2^{k^*}$ in T_1 and T_2
      if p_i violates the invariant **then**
        change status of p_i
      if radius or status of p_i changed **then**
        RESTORE(p_i, k^*)

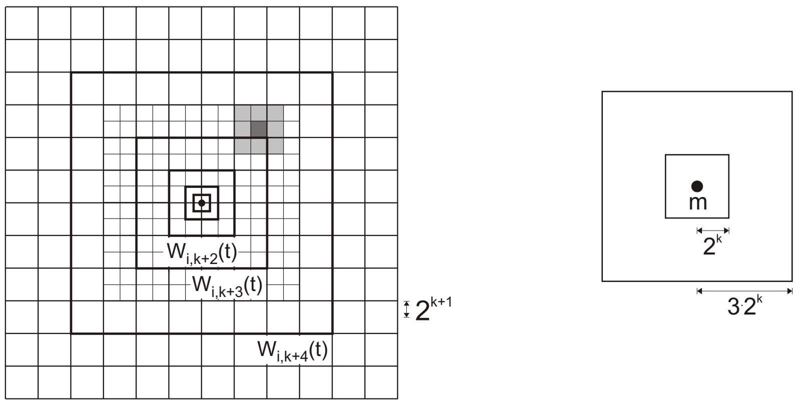
Algorithm 7 RESTORE(p_e, k^*)

for $k \leftarrow k^* - 1$ **to** $\lfloor \log_2(\frac{\max_{p_j \in P} f_j}{\min_{p_j \in P} d_j}) \rfloor + \lceil \log_2(4\sqrt{d}) \rceil$ **do**
  define cubes $S_1 := C(p_e, 4 \cdot 2^{k+1})$ and $S_2 := C(p_e, 6 \cdot 2^{k+1})$
  for each cubelet C with center m_C and radius 2^k in S_1 **do**
    if $\exists$ facility with radius $< 2^k$ in $C(m_C, 3 \cdot 2^k)$ **then**
      close all facilities with radius 2^k in C
  for each cubelet C with center m_C and radius 2^k in S_2 **do**
    if $\nexists$ facility with radius $\leq 2^k$ in $C(m_C, 3 \cdot 2^k)$ **then**
      open one point with radius 2^k in C (if existing)

Algorithm RESTORE Suppose that p_e is a point that triggered an event e at a point of time t and whose radius or status changed due to e. Let $r_e^{\text{KFL}} = 2^{k^*}$ be the updated radius of p_e. First, we restore the invariant at all points with radius 2^{k^*-1}, to ensure that no point with radius less than or equal to 2^{k^*-1} violates the invariant. Then we handle all points with radius 2^{k^*} that violate the invariant, then the ones with radius 2^{k^*+1}, ..., up to the biggest possible radius. Now, we describe the procedure in general for any radius 2^k.

We define the two cubes $S_1 := C(p_e, 4 \cdot 2^{k+1})$ and $S_2 := C(p_e, 6 \cdot 2^{k+1})$. Both cubes are divided into equally sized cubelets with radius 2^k. The left side of Figure 8.2 illustrates this decomposition in the plane.

To guarantee that no open point with radius 2^k violates the invariant, we perform the following test for each cubelet in S_1: Let m be the center point of the considered cubelet. If there is a facility with radius less than 2^k in $C(m, 3 \cdot 2^k)$, then close all facilities with radius 2^k in $C(m, 2^k)$. Note that there is at most one such facility, because otherwise the



(a) (b)

Figure 8.2.: Illustration of the decomposition into cubelets and the tested area for a cubelet. The shown decomposition is used during the iteration of algorithm RESTORE that restores the invariant at all points with radius 2^k. The cubes S_1 and S_2 are indicated by thick lines. For each cubelet in S_1 and S_2, we perform a test. The shaded area indicates the tested area for one cubelet in S_1. This area is magnified on the right side of the figure. The shaded area indicates $C(m, 3 \cdot 2^k)$ and the dark shaded area the tested cubelet $C(m, 2^k)$.

invariant would have been violated before event e. The considered area around a cubelet is illustrated in Figure 8.2.

In order to ensure that no closed point with radius 2^k violates the invariant neither, we test each cubelet in S_2 one after the other, whether there exists a facility with radius less than or equal to 2^k in $C(m, 3 \cdot 2^k)$. If this is not the case, then we open a point with radius 2^k in the cubelet (if there is such a point). No matter, whether we opened a point or not, it is guaranteed, that for each closed point p_j with $r_j^{\mathrm{KFL}} = 2^k$ in the cubelet, there is a facility in $C(p_j, 4 \cdot r_j^{\mathrm{KFL}})$.

8.4. Analysis of the global algorithm

We start the analysis with the proof of the central claim, that the kinetic data structure keeps a constant factor approximation of the static FACILITY LOCATION problem in Subsection 8.4.1 . Then we devote the main part of the section to the proof that the invariant is indeed maintained by our algorithm in Subsection 8.4.2 and finish the section with considerations about the runtime in Subsection 8.4.3.

8.4.1. The approximation factor of the global algorithm

In this section we prove the claim that our global algorithm indeed yields a (64d+1)-approximation. The analysis is basically the same as in [MP00]. Only a few adjustments to our scenario have been made. For completeness, we include below the full analysis.

Proposition 8.6. *For any point $p_i \in P$, there exists a point $p_j \in F$, such that $r_j^{KFL} \le r_i^{KFL}$ and $D(p_i, p_j) \le 64d \cdot r_i^{MP}$.*

Proof. Since, for each point $p_i \in P$, there is a facility $p_j \in F$ with radius $r_j^{\mathrm{KFL}} \le r_i^{\mathrm{KFL}}$ in $C(p_i, 4 \cdot r_i^{\mathrm{KFL}})$, we get $D(p_i, p_j) \le \sqrt{d} \cdot 4 \cdot r_i^{\mathrm{KFL}}$. Now, due to Lemma 8.3 and the definition of r_i^{KFL}, we have $D(p_i, p_j) \le \sqrt{d} \cdot 4 \cdot 2^{3+\lceil \log_2(\sqrt{d}) \rceil} \cdot r_i^{\mathrm{MP}} \le 64d \cdot r_i^{\mathrm{MP}}$. □

Proposition 8.7. *Let p_i and p_j be distinct points in F. Then we have $D(p_i, p_j) > 2 \cdot \max\{r_i^{MP}, r_j^{MP}\}$.*

Proof. Without loss of generality, $r_j^{\mathrm{KFL}} \le r_i^{\mathrm{KFL}}$. From the fact that the invariant is always restored after an event occurred, it follows that $p_j \notin C(p_i, 2 \cdot r_i^{\mathrm{KFL}})$. Thus, we have $D(p_i, p_j) > 2 \cdot r_i^{\mathrm{KFL}} \ge 2 \cdot r_i^{\mathrm{MP}}$ and $D(p_i, p_j) > 2 \cdot r_i^{\mathrm{KFL}} \ge 2 \cdot r_j^{\mathrm{KFL}} \ge 2 \cdot r_j^{\mathrm{MP}}$. □

For any point $p_j \in P$ and an arbitrary set of facilities $X \subseteq P$, let

$$charge(p_j, X) = D(p_j, X) + \sum_{p_i \in X} \max\{0, r_i^{\mathrm{MP}} - D(p_i, p_j)\}\,.$$

Note, that we will plug in for X later on. First we will plug in F for it and then the set of optimally placed facilities.

Proposition 8.8. *For an arbitrary set of facilities $X \subseteq P$, we get*

$$\sum_{p_j \in P} charge(p_j, X) \cdot d_j = cost(X)\,.$$

Proof. We get

$$\begin{aligned}
& \sum_{p_j \in P} charge(p_j, X) \cdot d_j \\
= & \sum_{p_i \in X} \sum_{p_j \in B(p_i, r_i^{\mathrm{MP}})} (r_i^{\mathrm{MP}} - D(p_i, p_j)) \cdot d_j \\
& + \sum_{p_j \in P} D(p_j, X) \cdot d_j \\
= & \sum_{p_i \in X} f_i + \sum_{p_j \in P} D(p_j, X) \cdot d_j\,.
\end{aligned}$$

□

Proposition 8.9. *Let $p_j \in P$ be a point, let $X \subseteq P$ an arbitrary set of facilities, and let $p_i \in X$. If $D(p_j, p_i) = D(p_j, X)$ then $charge(p_j, X) \ge \max\{r_i^{MP}, D(p_j, p_i)\}$.*

Proof. If $p_j \notin B(p_i, r_i^{\mathrm{MP}})$, then $charge(p_j, X) \geq D(p_j, p_i) > r_i^{\mathrm{MP}}$. Otherwise,

$$\begin{aligned} charge(p_j, X) &\geq (r_i^{\mathrm{MP}} - D(p_j, p_i)) + D(p_j, p_i) \\ &= r_i^{\mathrm{MP}} \geq D(p_j, p_i). \end{aligned}$$

□

Proposition 8.10. *Let $p_j \in P$ be a point, $p_i \in F$. If $p_j \in B(p_i, r_i^{MP})$, then $charge(p_j, F) \leq r_i^{MP}$.*

Proof. By Proposition 8.7, there is no open point $p_\ell \in F$ such that $i \neq \ell$ and $p_j \in B(p_\ell, r_\ell)$. Since $D(p_j, F) \leq D(p_j, p_i)$, the lemma follows from the definition of $charge(p_j, F)$. □

Proposition 8.11. *Let $p_j \in P$ be a point, $p_i \in F$. If $p_j \notin B(p_i, r_i^{MP})$, then $charge(p_j, F) \leq D(p_j, p_i)$.*

Proof. The correctness of the lemma follows immediately unless there is a point $p_\ell \in F$ such that $p_j \in B(p_\ell, r_\ell^{\mathrm{MP}})$. If such a point p_ℓ exists, then Propositions 8.7 and 8.10 imply $D(p_i, p_\ell) > 2 \cdot \max\{r_i^{\mathrm{MP}}, r_\ell^{\mathrm{MP}}\}$ and $charge(p_j, F) \leq r_\ell^{\mathrm{MP}}$, respectively. Furthermore, we have $D(p_j, p_i) \geq D(p_i, p_\ell) - D(p_j, p_\ell) > 2r_\ell^{\mathrm{MP}} - r_\ell^{\mathrm{MP}} = r_\ell^{\mathrm{MP}}$, which completes the proof of the proposition. □

Proposition 8.12. *For any point $p_j \in P$ and an arbitrary set of facilities $X \subseteq P$,*

$$charge(p_j, F) \leq (64d + 1) \cdot charge(p_j, X).$$

Proof. Let p_i be some point in X such that we have $D(p_j, p_i) = D(p_j, X)$. By Proposition 8.6, there exists a point $p_\ell \in F$ such that $r_\ell^{\mathrm{KFL}} \leq r_i^{\mathrm{KFL}}$ and $D(p_i, p_\ell) \leq 64d \cdot r_i^{\mathrm{MP}}$.

If $p_j \in B(p_\ell, r_\ell^{\mathrm{MP}})$, then $charge(p_j, F) \leq r_\ell^{\mathrm{MP}}$ by Proposition 8.10. The proposition follows since $r_\ell^{\mathrm{MP}} \leq r_\ell^{\mathrm{KFL}} \leq r_i^{\mathrm{KFL}} \leq \sqrt{d} \cdot 4 \cdot 2^{3 + \lceil \log_2(\sqrt{d}) \rceil} \cdot r_i^{\mathrm{MP}} \leq 64d \cdot r_i^{\mathrm{MP}}$ and Proposition 8.9 implies $charge(p_j, X) \geq r_i^{\mathrm{MP}}$.

If $p_j \notin B(p_\ell, r_\ell^{\mathrm{MP}})$, then $charge(p_j, F) \leq D(p_j, p_\ell)$ by Proposition 8.11. Thus,

$$\begin{aligned} charge(p_j, F) &\leq D(p_j, p_i) + D(p_i, p_\ell) \\ &\leq D(p_j, p_i) + 64d \cdot r_i^{\mathrm{MP}}. \end{aligned}$$

Since the ratio of $D(p_j, p_i) + 64d \cdot r_i^{\mathrm{MP}}$ to the maximum of r_i^{MP} and $D(p_j, p_i)$ is at most $64d + 1$, the proposition now follows by Proposition 8.9. □

Due to Propositions 8.8 and 8.12, we have $cost(F) \leq (64d+1) \cdot cost(X)$ for an arbitrary set of facilities $X \subseteq P$. Thus, the approximation factor is also true for an optimal set of facilities $X = F_{\mathrm{OPT}}$, which completes the proof of Proposition 8.4. We will use this later to formulate the central statement about the approximation factor in Theorem 8.20.

8.4.2. Global maintenance of the invariant

At first, we prove that the invariant is satisfied each time our kinetic data structure has handled an update. From this fact and Proposition 8.4 follows that our kinetic data structure maintains, at any point of time t, a set of facilities F such that $cost(F) = \mathcal{O}(cost(F_{\text{OPT}}))$. We start by proving that the invariant is satisfied as long as algorithm KINETICFL does not call algorithm RESTORE.

Proposition 8.13. *The invariant is satisfied after the first step of algorithm* KINETICFL.

Proof. Let t_0 be the point of time when the algorithm KINETICFL is started. Since algorithm MODIFIED-METTU-PLAXTON treats the points in ascending order according to their special radii and opens a point $p_i(t_0)$ with radius $r_i^{\text{KFL}}(t_0)$ only if there is no other open point in $C(p_i(t_0), 2 \cdot r_i^{\text{KFL}}(t_0))$, no open point violates the invariant. This is due to the fact that points with larger radii take care that their invariant is not violated by points with smaller radius, when it is their turn and that their invariant cannot be violated by points with a larger invariant, due to the construction of the invariant.

Furthermore, algorithm MODIFIED-METTU-PLAXTON does not open a point $p_i(t_0)$ with radius $r_i^{\text{KFL}}(t_0)$ if and only if there is another open point in the cube $C(p_i(t_0), 2 \cdot r_i^{\text{KFL}}(t_0)) \subseteq C(p_i(t_0), 4 \cdot r_i^{\text{KFL}}(t_0))$. Because this point has been treated earlier than $p_i(t_0)$, its radius is less than or equal to $r_i^{\text{KFL}}(t_0)$. Thus, there exists an open point with radius less than or equal to $r_i^{\text{KFL}}(t_0)$ in $C(p_i(t_0), 4 \cdot r_i^{\text{KFL}}(t_0))$. Hence, no closed point violates the invariant. □

Proposition 8.14. *Let e be any event such that algorithm* KINETICFL *does not change the radius or the status of any point. If the invariant is satisfied before e, then it holds after e as well.*

Proof. We have to consider two cases. In the first case two points cross or a wall crosses a wall of another point. Thus, no point crosses a wall of another point. This implies that no radius changes its value, and no invariant can be violated directly either, because then the wall at $2 \cdot r_i^{\text{KFL}}$ or $4 \cdot r_i^{\text{KFL}}$ of a point p_i has to be crossed. Thus the invariant is valid and the proposition holds.

Let t be the point of time when event e occurs. Then, in the second case, we have that a wall $W_{i,k}$ of a point p_i is crossed by another point p_j, but our algorithm does not change the radius or the status of p_i. It follows that p_i does not violate the invariant because otherwise our algorithm would have changed its status. Due to the fact that p_i is unchanged and only the wall $W_{i,k}$ is crossed at time t, no point in $P \backslash \{p_i\}$ violates the invariant. This completes the proof. □

Next, we prove that the updated radius of a point that triggered an event e differs only slightly from the value before e. More precisely, the radius r_i^{KFL} that we work with and the radius r_i^{LFL} that we use later in Chapter 9 will change at most by 1 upon an event.

Proposition 8.15. *Let e be an event at any time t, where any point $p_j(t) \in P(t)$ crosses any wall of any other point $p_i(t) \in P(t)$. Let $t' < t$ be any point of time after the latest point of time when p_i has been involved in one event. We get $r_i^{KFL}(t')/2 \le r_i^{KFL}(t) \le 2r_i^{KFL}(t')$.*

Proof. Let k_0', k_0 be the minimum integers k with $\log_2(\frac{\min_{p_j \in P} f_j}{n \cdot \max_{p_j \in P} d_j}) \le k \le \log_2(\frac{\max_{p_j \in P} f_j}{\min_{p_j \in P} d_j})$, for which we have $weight(C(p_i(t'), 2^{k_0'})) \ge f_i \cdot 2^{-k_0'}$ and $weight(C(p_i(t), 2^{k_0})) \ge f_i \cdot 2^{-k_0}$, respectively. Furthermore, let $W_{i,\ell}(t)$ be the wall that is crossed by $p_j(t)$. We have to consider the cases
(i) $p_j(t)$ leaves the cube $C(p_i(t), 2^\ell)$ and
(ii) $p_j(t)$ enters the cube $C(p_i(t), 2^\ell)$,
because otherwise p_i's radius does not change at all due to event e.

Case (*i*) Since the point of time t', p_j is the only point that has crossed a wall of p_i. It follows that $weight(C(p_i(t), 2^m)) < f_i \cdot 2^{-m}$, for any $m < k_0'$, and $weight(C(p_i(t), 2^{k_0'})) \le weight(C(p_i(t'), 2^{k_0'}))$. This implies $k_0 \ge k_0'$.

Because $p_j(t)$ has only crossed one wall of $p_i(t)$, we get $weight(C(p_i(t), 2^{k_0'+1})) \ge weight(C(p_i(t'), 2^{k_0'})) \ge f_i \cdot 2^{-k_0'} \ge f_i \cdot 2^{-(k_0'+1)}$, where the second inequality is given by the definition of k_0'. Thus, we have $k_0 \le k_0' + 1$, so that $k_0' \le k_0 \le k_0' + 1$.

Case (*ii*) Since $p_j(t)$ is the only point that has crossed a wall of $p_i(t)$ and $p_j(t)$ enters a cube with center $p_i(t)$, we have $weight(C(p_i(t), 2^m)) \ge weight(C(p_i(t'), 2^m))$, for all possible values of m. Hence, we get $k_0 \le k_0'$.

Recall that $p_j(t)$ crosses the wall $W_{i,\ell}(t)$. If $\ell \ge k_0'$, then $k_0 \ge k_0' - 1$ follows obviously. Now, let us assume that $\ell < k_0'$ and $k_0 = \ell - 1$. Due to this assumption, we have $weight(C(p_i(t), 2^{\ell-1})) \ge f_i \cdot 2^{-(\ell-1)}$. Since p_j is the only point that has crossed a wall of p_i, we also have $weight(C(p_i(t'), 2^\ell)) \ge f_i \cdot 2^{-(\ell-1)} \ge f_i \cdot 2^{-(\ell)}$. This implies $k_0' \le \ell$, which is a contradiction. Hence, we get $k_0' - 1 \le k_0 \le k_0'$.

Considering both cases, we get $k_0' - 1 \le k_0 \le k_0' + 1$. Now, the proposition follows due to the definition of the special radii. □

The following propositions show that the invariant is restored after each call of algorithm RESTORE.

Proposition 8.16. *Let p_e be a point that triggered an event e and whose radius or status changed due to e. Let $r_e^{KFL} = 2^{k^*}$ be the updated radius of p_e. If no point with radius less than or equal to 2^{k^*-2} violates the invariant before e, then this holds after e as well.*

Proof. Due to Proposition 8.15, the radius of p_e has been at least 2^{k^*-1} before e. While processing event e, we only change the status of points with radius larger than or equal to 2^{k^*-1}. These status changes cannot affect the invariant at points with radius less than or equal to 2^{k^*-2}. Thus, the proposition follows. □

Proposition 8.17. *Let p_e be a point that triggered an event e and whose radius or status changed due to e. Let $r_e^{KFL} = 2^{k^*}$ be the updated radius of p_e. If the invariant is satisfied before e and no open point with radius less than or equal to $2^{\ell-1}$ violates the invariant before running the outer **for**-loop of algorithm* RESTORE *for $k = \ell$, where $k^* - 1 \leq \ell \leq \lfloor \log_2(\frac{\max_{p_j \in P} f_j}{\min_{p_j \in P} d_j}) \rfloor + \lceil \log_2(4\sqrt{d}) \rceil$, then, after running this **for**-loop, no open point with radius 2^ℓ violates the invariant.*

Proof. The proof is by contradiction. Let us assume that after running the outer **for**-loop of algorithm RESTORE for $k = \ell$ there is an open point p_i with radius $r_i^{\text{KFL}} = 2^\ell$ that has another open point p_j with radius $r_j^{\text{KFL}} \leq r_i^{\text{KFL}}$ in $C(p_i, 2 \cdot r_i^{\text{KFL}})$. We consider the cases (i) $r_j^{\text{KFL}} = r_i^{\text{KFL}}$ and (ii) $r_j^{\text{KFL}} < r_i^{\text{KFL}}$:

Case (*i*) We have to consider the case that neither p_i nor p_j is opened while running the outer **for**-loop for $k = \ell$ and the case that at least one of p_i and p_j is opened during this **for**-loop. In the first case, it follows that p_i and p_j must have been open before running the outer **for**-loop for $k = \ell$. As a consequence, both points have been open before e or one point is p_e. Then either the invariant was violated before e or changing the status of p_e violated the invariant, a contradiction. In the latter case, we have opened p_i or p_j or both while running the outer **for**-loop for $k = \ell$. Without loss of generality, let us assume that we have opened p_i before we have opened p_j. Then we must have that $p_j \in C(m, 2^\ell)$ and $p_i \notin C(m, 3 \cdot 2^\ell)$ for one center m of a considered cubelet. It follows that $p_j \notin C(p_i, 2^{\ell+1}) = C(p_i, 2 \cdot r_i^{\text{KFL}})$, which is a contradiction.

Case (*ii*) Subcase $p_i \in S_1$: Due to the fact that $r_j^{\text{KFL}} < 2^\ell$, we have opened p_j before running the outer **for**-loop for $k = \ell$. It follows that $p_i \in C(m, 2^\ell)$ and $p_j \notin C(m, 3 \cdot 2^\ell)$ for one center m of a considered cubelet, because otherwise we either would have closed p_i or would not have opened p_i. As a consequence, $p_j \notin C(p_i, 2^{\ell+1}) = C(p_i, 2 \cdot r_i^{\text{KFL}})$, which is a contradiction.

Subcase $p_i \notin S_1$: Due to the fact that $r_j^{\text{KFL}} < 2^\ell$, we have opened p_j before running the outer **for**-loop for $k = \ell$. Hence, it must be located within S_2 of an earlier round. No cubelets of S_2 from earlier rounds are involved in a test when running the second inner **for**-loop for $k = \ell$. Hence, p_j cannot violate the invariant of p_i. □

Proposition 8.18. *Let p_e be a point that triggered an event e and whose radius or status changed due to e. Let $r_e^{KFL} = 2^{k^*}$ be the updated radius of p_e. If the invariant is satisfied before e and no closed point with radius less than or equal to $2^{\ell-1}$ violates the invariant before running the outer **for**-loop of algorithm* RESTORE *for $k = \ell$, where $k^* - 1 \leq \ell \leq \lfloor \log_2(\frac{\max_{p_j \in P} f_j}{\min_{p_j \in P} d_j}) \rfloor + \lceil \log_2(4\sqrt{d}) \rceil$, then, after running this **for**-loop, no closed point with radius 2^ℓ violates the invariant.*

Proof. The proof is by contradiction. Let us assume that after running the outer **for**-loop of algorithm RESTORE for $k = \ell$ there is a closed point p_i with radius $r_i^{\mathrm{KFL}} = 2^\ell$ that has no open point with radius less than or equal to r_i^{KFL} in $C(p_i, 4 \cdot r_i^{\mathrm{KFL}})$. We have to consider the cases (*i*) $p_i \in S_2$ and (*ii*) $p_i \notin S_2$.

Case (*i*) The assumption implies that $p_i \in C(m, 2^\ell)$ and there is an open point p_j in $C(m, 3 \cdot 2^\ell)$ for any center m of a considered cubelet, because otherwise we would have opened a point with radius 2^ℓ in $C(m, 2^\ell)$. Note that, in case there is no other point in $C(m, 2^\ell)$ except p_i, we would have opened p_i and $p_j = p_i$. As a consequence, $p_j \in C(p_i, 2^{\ell+2}) = C(p_i, 4 \cdot r_i^{\mathrm{KFL}})$, which is a contradiction.

Case (*ii*) Let t' be any point of time between the occurrence of e and the latest event before. Then there was an open point $p_j(t')$ with radius less than or equal to $r_i^{\mathrm{KFL}}(t)$ in the cube $C(p_i(t'), 4 \cdot r_i^{\mathrm{KFL}}(t'))$, because otherwise the invariant was violated before e.

First let us assume that $p_j = p_e$. Since $p_i(t) \notin S_2 = C(p_e(t), 6 \cdot 2^{\ell+1})$, we have $p_j(t) \notin C(p_i(t), 6 \cdot 2^{\ell+1})$. From $p_j(t') \in C(p_i(t'), 4 \cdot 2^\ell)$ and $p_j(t) \notin C(p_i(t), 6 \cdot 2^{\ell+1})$ follows that p_j must have crossed the wall $W_{i,\ell+3}(t'')$ at a time t'' with $t' < t'' < t$. This implies an event at time t'', which is a contradiction to the definition of t'. Thus, we have $p_j \neq p_e$.

Due to $p_i \neq p_e$, $p_j \neq p_e$, and $p_j(t') \in C(p_i(t'), 4 \cdot r_i^{\mathrm{KFL}}(t'))$, we have that $p_j(t) \in C(p_i(t), 4 \cdot r_i^{\mathrm{KFL}}(t))$ is also true. Thus, if p_i violates the invariant after e, then we must have closed p_j during processing e. We only close points with radius less than or equal to $r_i^{\mathrm{KFL}}(t)$ in S_1. Since $p_i(t) \notin S_2$ and $p_j(t) \in S_1$, we get $p_j(t) \notin C(p_i(t), 2 \cdot 2^{\ell+1}) = C(p_i(t), 4 \cdot r_i^{\mathrm{KFL}}(t))$, which is a contradiction. □

Now, we can combine the obtained results to the following lemma:

Lemma 8.19. *The invariant is satisfied after algorithm* KINETICFL *has handled an event.*

Proof. Due to Propositions 8.13 and 8.14, the invariant is satisfied as long as we do not call algorithm RESTORE. Now, we show that this is also true after processing algorithm RESTORE.

Let p_e be the point whose radius or status changed due to an event e, and let $r_e^{\mathrm{KFL}} = 2^{k^*}$ be its updated radius. Because of the precondition given above and Proposition 8.16 the lemma is true for all points p_i with radius $r_i^{\mathrm{KFL}} = 2^\ell$ where $\ell \leq k^* - 2$. Due to Propositions 8.17 and 8.18, it also follows for $\ell \geq k^* - 2$. Hence, it is true for all points. □

Due to Lemma 8.19 and Proposition 8.4, we get the following result:

Theorem 8.20. *The kinetic data structure for the* FACILITY LOCATION *problem in* $\mathbb{R}^d$ *maintains, at any point of time t, a subset* $F \subseteq P$ *such that*

$$cost(F) \leq (64d + 1) \cdot cost(F_{\mathrm{OPT}}) .$$

8.4.3. Complexity

In the remainder of the section, we analyze our kinetic data structure in terms of its complexity. Due to Lemma 8.5, we have already proven that our kinetic data structure is compact and kinetic local. Next we show that the requirement for responsiveness and efficiency is also fulfilled.

Lemma 8.21. *An update operation requires $\mathcal{O}(\log^{d+1}(n) \cdot \log(nR))$ time and $\mathcal{O}(\log(nR))$ status changes.*

Proof. Due to Lemma 8.5, the time to update the event queue is $\mathcal{O}(\log(nR))$. Except for algorithm RESTORE, all further steps to handle an event require $\mathcal{O}(\log^{d+1}(n))$ time. Next we examine the time needed for algorithm RESTORE. We consider the running time resulting for restoring the invariant at points with radius 2^k. The number of cubelets with radius 2^k in $C(p_e, 6 \cdot 2^{k+1})$ is 12^d. The query of open or closed points for one cubelet can be answered by T_1 and T_2 in time $\mathcal{O}(\log^{d+1}(n))$. Afterwards, there has to be at most one point inserted and deleted in T_1 and T_2. This requires $\mathcal{O}(\log^{d+1}(n))$ time. By summation over all radii, we get a total running time of $\mathcal{O}(\log^{d+1}(n) \cdot \log(nR))$.

There can exist at most one facility with radius 2^k in a cubelet with radius 2^k, because otherwise at least one facility would violate the invariant. Hence, the number of facilities with radius 2^k that are closed while running algorithm RESTORE is constant. Furthermore, we open at most one facility in each cubelet, so that the number of opened facilities with radius 2^k is also constant. Due to the fact that we handle $\mathcal{O}(\log(nR))$ radii, there are $\mathcal{O}(\log(nR))$ status changes per event. □

It follows from Lemma 8.5 and Lemma 8.21 that the total processing time is bounded by $\mathcal{O}(n^2 \log^{d+1}(n) \cdot \log^3(nR))$.

We summarize our results in the following theorem:

Theorem 8.22. *Let P be a set of n independently moving points in $\mathbb{R}^d$, where d is a fixed dimension. Then there exists a kinetic data structure for the* FACILITY LOCATION *problem that maintains, at any point of time t, a set $F \subseteq P$, such that we have $cost(F) \leq (64d+1) \cdot cost(F_{\text{OPT}})$. The kinetic data structure has a space requirement of $\mathcal{O}(n(\log^d(n) + \log(nR)))$, where $R = \frac{\max_{p_i \in P} f_i \cdot \max_{p_i \in P} d_i}{\min_{p_i \in P} f_i \cdot \min_{p_i \in P} d_i}$. Each event requires $\mathcal{O}(\log(nR))$ status changes and $\mathcal{O}(\log^{d+1}(n) \cdot \log(nR))$ update time. In case that each trajectory can be described by a bounded degree polynomial, the total number of updates is $\mathcal{O}(n^2 \log^2(nR))$, which results in a total processing time of $\mathcal{O}(n^2 \log^{d+1}(n) \cdot \log^3(nR))$. A flight plan update involves $\mathcal{O}(\log(nR))$ certificates and requires $\mathcal{O}(\log^2(nR))$ time.*

Note that according to the terms introduced in Section 8.1.1 our kinetic data structure is responsive, compact, kinetic local, and efficient.

This completes our considerations about the global scenario. The results are given in the theorem above. We will discus them and have an outlook in Chapter 10. Note that we only need a logarithmic number of status changes upon an event, while Theorem 7.2 states that for exact solutions are linear many changes necessary. The same holds for the unmodified Mettu and Plaxton algorithm due to Theorem 7.3. Furthermore, remember that the results in Section 7.3 show that any set of facilities on n linearly moving points needs to change $\Omega(n^2/c^2)$ in order to keep a c-approximation. Since our kinetic data structure processes a total number of events of $\mathcal{O}(n^2 \log^2(nR))$, the kinetic data structure for the FACILITY LOCATION problem is efficient.

So far, we know how to deal with dynamics in a centralized fashion. However, our robotic scenario, where the robots are limited to a local view only requires that our algorithms can be executed in a distributed way, each node gathering information about its immediate vicinity only. We will consider this case in the next chapter.

CHAPTER 9

The local scenario

So far, we have only considered a centralized algorithm. However, in our robot scenario, we want our robots to decide autonomously whether they should be a facility or not. Therefore, we need a distributed algorithm. Furthermore, we want our algorithm to be local, because we assume that each robot knows only its local environment and none of the robots is aware of the complete state of the system. Therefore the goal of this chapter to provide and analyze such an algorithm.

Main contribution of this chapter We introduce a simple distributed algorithm that is executed by each node and is used to determine whether the node should act as a facility or a client in the current situation. Since the distances between the nodes change over time, the algorithm constantly reevaluates its decision and, if necessary, changes the node's role in order to reestablish the approximation. Taken as a whole, the decisions of all the nodes yield a constant factor approximation of the FACILITY LOCATION problem in a stable solution. Here, we consider a solution (or configuration) to be stable, when under the given configuration no node decides to change its role anymore.

An important property of our algorithm is the fact that each node only requires local information to be able to execute it: For each node p_i the value r_i^{LFL}, referred to as radius, is computed. In order to compute its radius, p_i requires only information about nodes that are within constant distance of p_i (i.e. a distance independent of the total number of nodes). In addition to the radius, p_i requires information about the current role of all nodes p_j with $D(p_i, p_j)$ bounded from above by a constant. The radius, the d_j's and the current roles of p_i's neighbors p_j is all that is necessary for p_i to determine its own role. Note, that due to Theorem 7.1, we cannot restrict the radius of the considered ball any further.

We describe and analyze this algorithm showing that, although the decisions of the nodes are based on local information, the system stabilizes in a solution that yields a global $\mathcal{O}(1)$- approximation. Furthermore, we prove that the process of finding the approximation only requires $\mathcal{O}(\log n)$ communication rounds, and that changes in the role or the radius of a node p_i only affect nodes within constant distance of p_i. This also implies that upon an change all information that is used by a node stems only form its neighborhood that is bounded above by a constant, although it might require $\mathcal{O}(\log n)$ rounds until it reaches its final decision.

9.1. The model

While relying on the well established framework of kinetic data structures for the global algorithm, we have to define our own model for the local algorithm, since there is no corresponding model established for geometric local algorithms, yet. One of the differences is that in this framework the number of robots is the input that our results rely on. This means that the f_i and d_i are independent of n. This is a reasonable assumption, since it does not make any sense that the parameters of structurally identical robots should depend on the number of robots in the systems. Thus, formally the f_i and d_i are constant. A consequence of this is that the term R which is part of the theorems of the global algorithm is not needed here. Note that for a global algorithm all input should be regarded when stating runtime bounds and therefore the coding of the f_i and d_i is indeed of interest.

9.1.1. The communication model

In order to gather the information required to execute the algorithm, nodes need to communicate with each other.

For our robot scenario we envision indirect communication only. Here, each node constantly provides information about its role and radius to other nodes in its vicinity. This is possible using an idea inspired by the s-bots [MGB+03]. Those s-bots are robots that have 24 colored LEDs to represent their state (we assume that one can equip them with a larger number of 'communication bits'). Furthermore, they are equipped with cameras enabling them to acquire the information provided by other robots. Such a communication method could be adopted by nodes in sensor networks and is sometimes referred to as local broadcast.

9.1.2. The round model

The execution of the algorithm is embedded in an asynchronous round model. Concerning this model, a node's state changes as follows: Starting in an inactive state, the node's state changes to active after an arbitrary amount of time. Now, it determines whether it should change its role and acts according to this decision. After the node's role has been updated, it returns to the inactive state. This sequence of state changes is infinitely repeated by every node. Note that despite the asynchronous round model the movement of the nodes takes place in parallel and in a continuous fashion as described in the next subsection.

A round ends as soon as every single node has been active at least once since the end of the previous round. We assume that nodes spend most of the time in the inactive state and that at any point of time at most one node is in the active state.

As we need to define a relationship between the dynamics concerning the distance function D and the nodes' change of states, we introduce the term *event*. An event symbolizes a change in the distance function upon which our algorithm reacts and is defined formally later on. As the time between an event and the point in time until which all nodes became active at least once cannot be bounded, we assume that the activation frequency is high enough compared to the occurrence of events. More precisely, we suppose that at least $c \log n$ rounds take place between two events, where c is a constant hidden in the $\mathcal{O}$-notation in Theorem 9.7. Note, that the term 'event' is defined in different a way than in Chapter 8. Although in both cases, an event requires some action of the algorithm, in this chapter, an event is not part of the algorithm itself. It is rather a means of analysis. No robot realizes that an event occurred. However, the movement that triggers an event is similar in both settings.

9.1.3. Modeling dynamics

We allow the values of the distance function D to change arbitrarily over time. The only restrictions we place on the changes is that we require them to happen in a continuous way. Consider the time interval that starts at t_0 and ends at t_1. During this time the distances between the nodes will be changing continuously and thus, due to the execution of our algorithm, events will occur. In order to be able to bound the number of these events, we introduce a characterization, the *dynamics parameter*, which we will denote by x ($x \in \mathbb{N}$), of the distance function's changes. For each of the $\binom{n}{2}$ pairs of nodes $\{p_i, p_j\}$, we consider the value $D(p_i, p_j)$ as it changes over time. We say that at time t the pair of nodes $\{p_i, p_j\}$ is in *increasing* mode, if $D(p_i, p_j)$ is increasing, and in *decreasing* mode, if $D(p_i, p_j)$ is decreasing. If $D(p_i, p_j)$ is constant, we say that $\{p_i, p_j\}$ is in *constant* mode. The sum of all changes in the modes over all pairs of nodes during the time interval $[t_0, t_1]$ yields the value for the dynamics parameter x, where we assume that all pairs start in constant mode. If they do not start in constant mode, these are formally the first changes in the interval.

Obviously, x can be computed for any kind of continuous changes in the distance function D. If for example the set of nodes represents n robots moving on trajectories described by polynomials with a degree bounded by a constant and D models the distance between each pair of robots, then the movement parameter x is bounded by $\mathcal{O}(n^2)$. This is the case in the kinetic data structure framework presented in the global case. Therefore we are dealing with a generalization of the model of dynamics and here we are able to model more general motion patterns. Another example is the way-point model[1], where $x \leq m \cdot n^2$ (with m upper bounding the number of points each node travels to in the considered time interval). When trying to upper bound the number of occurring events, it is not important how exactly the distances between nodes change. It is only relevant how often the distances change from increasing to decreasing or vice versa. (If we for example consider two robots, it is irrelevant in which direction and speed the robots move away from each other. All that is needed to bound the number of events is the fact that the distance between them is increasing.) This makes the dynamics parameter x a useful abstraction for all continuous dynamic changes in the distance function D.

9.1.4. How the parts of the model interact

The general idea is that nodes wake up seldom enough, such that at each point of time in each local neighborhood only one robot is active. On the other hand the robots move slow enough, such that between two consecutive events at least $c \log n$ rounds take place. This way we are able to guarantee that we are always in a stable configuration before an event occurs. The motion pattern itself is described by the dynamics parameter, while the events are used to characterize the points of time, where a stable solution can become unstable. In a stable configuration, the solution is a constant-factor approximation for the corresponding static setting.

9.2. Description of the local algorithm

In the first section, we have already mentioned the radius and events. We now formally introduce these concepts, define an invariant similar to the one in the global setting and give a description of our algorithm.

Radius of a node Similar to r_i^{KFL}, we now define r_i^{LFL}, which is also crucial for our local algorithm. Once again, instead of the exact radius r_i^{MP}, we use an approximation for it, but a better one than in Chapter 8. The idea is again to round the radius to a value which is a power of 2. Because of the bounds for the original radius r_i^{MP}, this leads to $\mathcal{O}(\log n)$ possible values for our radius. This is similar to the approach used for the global scenario. However, here we do not need to approximate the radius any further since we may exploit the possibility of parallelization and do not need to fit the calculation of the radii to centralized data structures as the range trees. We restate the Lemma 8.2 here, because it is crucial for our approximation factor:

[1] In the way-point model each node is given a number of target points. All points move simultaneously on a straight line to their next target point. After reaching the target they proceed to next one.

Lemma 9.1. *Let k_1 be the minimum integer k with $\lceil \log_2(\frac{\min_{p_j} f_j}{n \cdot \max_{p_j} d_j}) \rceil \leq k \leq \lfloor \log_2(\frac{\max_{p_j} f_j}{\min p_j d_j}) \rfloor$, such that weight$(B(p_i, 2^k)) \geq f_i \cdot 2^{-k}$. Then $\frac{1}{2} \cdot r_i^{MP} \leq 2^{k_1} \leq 2 \cdot r_i^{MP}$ holds.*

Here, we define $r_i^{\text{LFL}} := 2^{k_1+1}$ as the radius of node p_i and $k_0 := k_1 + 1$. Because of Lemma 9.1, $r_i^{\text{MP}} \leq r_i^{\text{LFL}} \leq 4 r_i^{\text{MP}}$. Like the original radius, r_i^{LFL} can change over time. This can happen when a node moves into certain multiples (which will be defined exactly when the events are defined) of the radius of another node, since this is when the weight of the respective ball increases or decreases. Furthermore, $\frac{f_i}{d_i}$ is an upper bound for the original radius r_i^{MP} of node p_i, which is reached when no other node is in p_i's radius. With Lemma 9.1, this leads to an upper bound of $2^{\lfloor \log(\frac{f_i}{d_i}) \rfloor + 2} \leq 8 \frac{f_i}{d_i}$ for r_i^{LFL}. Since f_i and d_i are known to node p_i, it can compute this upper bound. Moreover, since we assume that all f_i and d_i are constant, the value of a radius is bounded from above by a constant. No lower bound on the radius is known to the nodes, since we assume that they do not know the total number of nodes n, and in accordance with the results in Subsection 6.4.1 on r_i^{MP} the lower bound of r_i^{LFL} is of order $\frac{1}{n}$. Note, that this is a difference to the kinetic data structure, where all knowledge is at hand.

9.2.1. The invariant

The main idea of the algorithm is that all nodes try to maintain at all times the following invariant, which is closely related to the invariant defined in Subsection 8.3.3 for the global scenario:

1. If $p_i \in G$, there is a facility $p_j \in F$ with $r_j^{\text{LFL}} \leq r_i^{\text{LFL}}$ and $D(p_i, p_j) \leq 4 \cdot r_i^{\text{LFL}}$
2. If $p_i \in F$, there is no other facility $p_j \in F$ with $r_j^{\text{LFL}} \leq r_i^{\text{LFL}}$ and $D(p_i, p_j) \leq 2 \cdot r_i^{\text{LFL}}$

As soon as a node discovers that its invariant is violated, it repairs it by changing its role. This can violate the invariant of other nodes. In Subsection 9.3.2.1 we show that the effect of role changes is bounded.

9.2.2. The algorithm

The algorithm works as follows. As soon as a node p_i turns active, it calculates its radius (see below). Usually a recalculation from scratch is not necessary, once it is calculated, because we assume that events are rare compared to the wake up pace. If the radius is not valid, and only one event occurred, the argumentation from Proposition 8.15 can be applied showing that only three possible values for r_i^{LFL} have to be checked. After updating the radius, the node p_i checks whether its invariant is fulfilled. If this is the case, it turns inactive again. Otherwise, its reaction depends on its role. If p_i is a facility, the invariant is violated because there is another facility with a smaller or equal radius within

Algorithm 8 MAINALGO

```
myradius ← CalculateRadius(me)
if myrole = open then
    if ∃ node i with ((i.role = open) and (i.radius ≤ myradius) and
    D(i.position, myposition) < 2 · myradius) then
        myrole ← closed
else
    if ∄ node i with ((i.role = open) and (i.radius ≤ myradius) and
    D(i.position, myposition) < 4 · myradius) then
        myrole ← open
```

distance $2 \cdot r_i^{\mathrm{LFL}}$ of p_i. So closing p_i repairs its invariant. If p_i is a client, its invariant can only be violated because there is no facility with a smaller or equal radius within distance $4 \cdot r_i^{\mathrm{LFL}}$ of p_i. Therefore, in this case opening p_i repairs its invariant. Algorithm 8 formally describes the algorithm.

To calculate its radius, a node p_i computes k_1 as defined in Lemma 9.1 and then returns $k_0 = k_1 + 1$. Accordingly, its radius is 2^{k_0}. To compute k_1, p_i first computes the largest possible value for k_1, k_1^{max}, which is $\lfloor \log(\frac{f_i}{d_i}) \rfloor + 1$, and sets k to k_1^{max}. Then it sorts all nodes within distance $2^{k_1^{max}}$ by their distances to p_i, and computes the weight of the ball with radius $2^{k_1^{max}}$ as the sum of all d_j of nodes p_j within this distance. Now it reduces k one by one until the inequality weight$(B(p_i, 2^k)) \geq f_i \cdot 2^{-k}$ (see Lemma 9.1) is not valid any longer. To compute the current weight, the old value is decreased by the d_j of all nodes p_j which are in the ball with radius 2^{k+1}, but not in the ball with radius 2^k, using the sorting of the nodes. Since weight$(B(p_i, 2^k))$ is monotonically increasing and $f_i \cdot 2^{-k}$ monotonically decreasing with k, the resulting k is the largest value for which the inequality is not kept and therefore $k+1$ the desired k_1. See Algorithm 9 for a formal description. Note, once again that in accordance with Proposition 8.15 upon an event the k of the radius can change at most by 1.

9.2.3. Events and initialization

To allow a better description of our algorithm and its analysis, we introduce the term *event*, which is different to the notion of event in Chapter 8. An event occurs each time a configuration might become unstable. This the case, when the radius of a node could change or an invariant could be violated because distances between nodes have changed. For a node p_i there are two reasons for an event to occur:

- The distance $D(p_i, p_j)$ between p_i and p_j changes in such a way that p_j enters or leaves the ball $B(p_i, \frac{1}{4} \cdot r_i^{\mathrm{LFL}})$ or the ball $B(p_i, \frac{1}{2} \cdot r_i^{\mathrm{LFL}})$. Here, the radius of p_i can change. For details confer to Figure 9.1

Algorithm 9 CALCULATERADIUS

$k_1^{max} \leftarrow \lfloor \log(\frac{f_i}{c_i}) \rfloor + 1$
$V_i \leftarrow$ all nodes in distance at most $2^{k_1^{max}+1}$ to me
sort V_i by the distance to me in decreasing order
weight $\leftarrow c_i$
for all nodes v_j in V_i **do**
  weight $\leftarrow$ weight + c_j
$k \leftarrow k_1^{max}$
while weight $\geq f_i \cdot 2^{-k}$ **do**
  for all nodes v_j in distance d, $2^{k-1} < d \leq 2^k$ **do**
    weight $\leftarrow$ weight $-c_j$
    $k \leftarrow k-1$
$k_1 \leftarrow k+1$
return $k_0 \leftarrow k_1 + 1$

- The distance $D(p_i, p_j)$ between p_i and p_j changes in such a way that p_j enters or leaves the ball $B(p_i, 2 \cdot r_i^{\mathrm{LFL}})$ or the ball $B(p_i, 4 \cdot r_i^{\mathrm{LFL}})$. This can violate p_i's invariant, leading to a role change of p_i. The first case might obviously violate the invariant as it is defined for facilities, while the second case might violate the invariant for clients.

If one of these situations occurs, we say that p_j triggers an event at p_i, which is similar to the kinetic data structure. However, here the event is not part of the algorithm. A robot just notices that the invariant is not valid when it is its turn again, but it does not know whether this is the case due to an event, the change of a status of another robot, whether it is in the initialization phase or even whether some malfunctioning just happened in its local neighborhood. Note that similar to the kinetic data structure in Chapter 8 an event does not necessary require an action. However, if an action is required this is guaranteed to be detected by an event and therefore the number of events is a good upper bound on the number of times when a solution becomes unstable.

If p_i changes its role, invariants of neighboring nodes can be violated. A change of a radius can also affect other nodes: If the radius of a node p_i changes, it can become necessary for p_i to change its role as well. Moreover, if p_i increases its radius, there may be nodes which had a radius of r_i^{LFL} before and now have a smaller radius. If p_i is a facility, the invariants of those nodes can now be violated. The same can occur when p_i is a facility and decreases its radius: There can be nodes which had a smaller radius before and now have the same radius. Their invariants can now be violated, because they now take p_i into account. Note that if p_i is a client, changing its radius does not have any effects on the invariants of other nodes. Moreover, since the invariants only consider facilities and not clients, changing the radius of a facility can be viewed as two

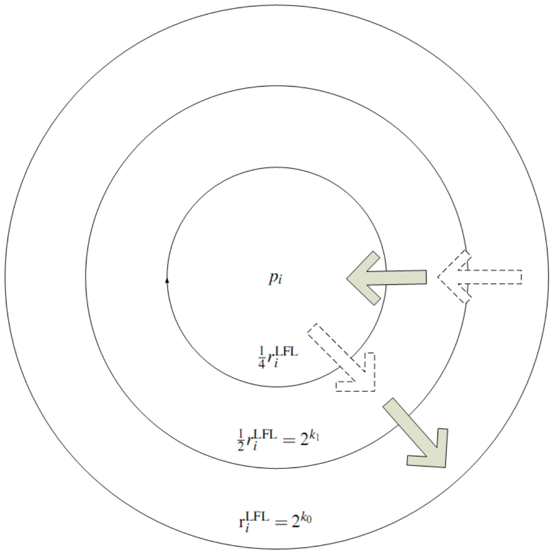


Figure 9.1.: Entering the ball $B(p_i, \frac{1}{4} \cdot r_i^{\mathrm{LFL}})$ or leaving the ball $B(p_i, \frac{1}{2} \cdot r_i^{\mathrm{LFL}})$ might change the value k_1 that defines the radius r_i^{LFL} by k_0. Hence the radius might change, which in turn might violate the invariant. For ease of description, we add the cases where the arrows are marked by dotted lines. This is not crucial, since the set of events is a superset of possible points of time, where a solution might become unstable.

simultaneous role changes at the same position: The original node closes, and a new *virtual* node with the same position and the new radius opens. We will explain this in detail in the respective proofs. Thus, from now on we only consider role changes when we talk about events.

Often, events only affect one node directly and possibly other nodes indirectly. Nevertheless, in some cases two nodes can be directly affected: if both involved nodes have the same radius, or one has the eightfold radius of the other. Even so, it can be shown that only one of these two nodes needs to change its radius or role. Thus, for the analysis we can assume that an event only occurs at one node.

For our analysis, we can distinguish the *initialization*, which is the time until the invariants are kept for the first time at all nodes, and the time after that. After the initialization, changes of the role or radius can only occur because of events. Note that, in contrast to the kinetic data structure, no explicit initialization is necessary. The nodes just perform their algorithm all the time. We just consider the initialization phase for the purpose of analysis.

9.3. Analysis of the algorithm

Starting with the obtained approximation factor of the algorithm, we show several properties such that it performs well under dynamics and finally give an overview over possible non-Euclidean applications. For instance, the same problems arise when choosing the right locations for services in a hardwired network with changing latencies on the edges.

Theorem 9.2. *When the invariant holds for each node, the resulting set of facilities yields a 17-approximation.*

Proof sketch. Each node p_i has a facility p_j with radius $r_j^{\mathrm{LFL}} \leq r_i^{\mathrm{LFL}}$ in $B(p_i, 4r_i^{\mathrm{LFL}})$. Lemma 9.1 in combination with our definition of a radius yields $r_i^{\mathrm{MP}} \leq r_i^{\mathrm{LFL}} \leq 4r_i^{\mathrm{MP}}$. Therefore $D(p_i, p_j) \leq 4 \cdot 4r_i^{\mathrm{MP}} = 16r_i^{\mathrm{MP}}$. The remaining of the analysis is analogous to the proof for the global algorithm. Details can be found below in Subsection 9.3.1. □

9.3.1. The approximation factor of the local algorithm

This section is very similar to the corresponding section for the global algorithm. Here we show that the local algorithm indeed keeps a 17-approximation in a stable solution. Also refer to Subsection 8.4.1 for several propositions and the definition of *charge*, which we use here.

Proposition 9.3. *If the solution is stable, for any point $p_i \in P$, there exists a point $p_j \in F$, such that $r_j^{LFL} \leq r_i^{LFL}$ and $D(p_i, p_j) \leq 16 \cdot r_i^{MP}$.*

Proof. For each point $p_i \in P$, there is a facility $p_j \in F$ with radius $r_j^{\mathrm{LFL}} \leq r_i^{\mathrm{LFL}}$ in distance $D(p_i, p_j) \cdot 4 \cdot r_i^{\mathrm{LFL}}$. Now, due to Lemma 9.1 and the definition of r_i^{LFL}, we have $D(p_i, p_j) \leq 4 \cdot 4 \cdot r_i^{\mathrm{MP}} \leq 16 \cdot r_i^{\mathrm{MP}}$. □

Proposition 9.4. *For any point $p_j \in P$ and an arbitrary set of facilities $X \subseteq P$,*

$$charge(p_j, F) \leq (16+1) \cdot charge(p_j, X)\,.$$

Proof. Let p_i be some point in X such that we have $D(p_j, p_i) = D(p_j, X)$. By Proposition 9.3, there exists a point $p_\ell \in F$ such that $r_\ell^{\mathrm{LFL}} \leq r_i^{\mathrm{LFL}}$ and $D(p_i, p_\ell) \leq 16 \cdot r_i^{\mathrm{MP}}$.

If $p_j \in B(p_\ell, r_\ell^{\mathrm{MP}})$, then $charge(p_j, F) \leq r_\ell^{\mathrm{MP}}$ by Proposition 8.10. The proposition follows since $r_\ell^{\mathrm{MP}} \leq r_\ell^{\mathrm{LFL}} \leq r_i^{\mathrm{LFL}} \leq 16 \cdot r_i^{\mathrm{MP}}$ and Proposition 8.9 implies $charge(p_j, X) \geq r_i^{\mathrm{MP}}$.

If $p_j \notin B(p_\ell, r_\ell^{\mathrm{MP}})$, then $charge(p_j, F) \leq D(p_j, p_\ell)$ by Proposition 8.11. Thus,

$$\begin{aligned} charge(p_j, F) &\leq D(p_j, p_i) + D(p_i, p_\ell) \\ &\leq D(p_j, p_i) + 16 \cdot r_i^{\mathrm{MP}}\,. \end{aligned}$$

Since the ratio of $D(p_j, p_i) + 16 \cdot r_i^{\mathrm{MP}}$ to the maximum of r_i^{MP} and $D(p_j, p_i)$ is at most $16+1$, the proposition now follows by Proposition 8.9. □

Due to Propositions 8.8 and 9.4, we have $cost(F) \leq 17 \cdot cost(X)$ for an arbitrary set of facilities $X \subseteq P$. Thus, the approximation factor is also true for an optimal set of facilities F_{OPT}, which completes the proof of Theorem 9.2.

9.3.2. Main results about the local algorithm

In this section we prove that our algorithm stabilizes fast, that there are only few changes in the set of facilities (depending on the motion pattern of course) and that the algorithm is indeed local in the sense that only nodes in a local neighborhood are affected by changes. We start by considering the efficiency of the local computations of the robots.

Theorem 9.5. *Each time a node turns active, its local computations require $\mathcal{O}(n \log n)$ time. The data it needs to transmit is bounded by $\mathcal{O}(\log\log n)$ bits.*

Proof. The time for computing the radius is dominated by sorting the nodes within the maximum radius, which takes $\mathcal{O}(n \log n)$ time. Checking whether the invariant is fulfilled takes linear time only. A node p_i needs to transmit three pieces of information: its role, taking 1 bit, its d_i, which is constant, and its radius. Since there are only $\mathcal{O}(\log n)$ possible radii, the current radius can be transmitted using $\mathcal{O}(\log\log n)$ bits. □

Note that the actual runtime to determine the radius is of order $n \log n$ only, when it needs to be calculated from scratch. Due to Proposition 8.15 it can be updated in three rounds after one event. Before we analyze some interesting bounds, we state one helpful lemma.

Lemma 9.6. *A node p_i opening itself can only violate invariants of nodes with a strictly larger radius than r_i^{LFL}.*

Proof. Node p_i only opens itself if there is no other facility with a radius less than or equal to r_i^{LFL} within distance $4 \cdot r_i^{\text{LFL}}$ of p_i, since otherwise the invariant of p_i is not violated. When p_i opens itself, this does not affect nodes with a radius smaller than r_i^{LFL} due to the construction of the invariant. Moreover, if a node p_l also has radius r_i^{LFL} and is a facility, it must be in distance more than $4 \cdot r_i^{\text{LFL}}$ from p_i and therefore its invariant cannot be violated by p_i. If p_l is closed, a new facility does not violate p_l's invariant. □

From the next theorem follows that the initialization is efficient.

Theorem 9.7. *After $\mathcal{O}(\log n)$ rounds without an event, the invariant holds at all nodes.*

Proof. For the remainder of the proof we assume that no events occur. Thus, no node changes its radius. Consider a set of nodes S_r with $p_i \in S_r \Leftrightarrow r_i^{\mathrm{LFL}} = r$ for a fixed radius r and a set $S_{<r}$ comprising all nodes with a radius smaller than r. Let t be the first round after which the invariant holds for all nodes with a radius smaller than r (i.e. those contained in $S_{<r}$). Since the invariants of the nodes belonging to $S_{<r}$ are not affected by nodes with a radius greater or equal r, starting from round $t+1$ none of these nodes will change its role. We say that these nodes are in a *stable* state, since their invariant can only be violated by the occurrence of an event. We now show that after round $t+2$ all nodes belonging to S_r have restored their invariant and are in a stable state. Consequently, after $2l$ rounds all invariants of nodes with one of the l smallest radii are in a stable state. Since $\mathcal{O}(\log n)$ is an upper bound on the number of different radii, the theorem follows.

Consider all nodes in S_r that are open directly before round $t+2$ begins. We claim that their invariants are not violated and that they will remain open (i.e. they are in a stable state). To see this, we analyze the nodes' behavior during round $t+1$. There are two reasons for a node to be open at the end of round $t+1$: It was either closed before its activation in round $t+1$ and its invariant was violated, or it was open and did not need to change its role. In the first case, we know due to Lemma 9.6 that the closed node's change of role did not violate the invariants of any node in S_r and $S_{<r}$. Lemma 9.6 also guarantees that this node will never close again. In the second case, the already open node did not change its role and in accordance with Lemma 9.6 its invariant will not be violated in the following rounds.

Having established that all open nodes in S_r are in a stable state at the end of round $t+1$, we now need to show that all closed nodes in S_r are in a stable state at the end of round $t+2$. Note, that it is possible for a closed node in S_r to not be in a stable state at the end of round $t+1$, since all facilities within distance $4r$ around it might have closed after its activation in round $t+1$. When a closed node becomes active during round $t+2$ and its invariant is violated, it will open, not violate any invariant of nodes in S_r and $S_{<r}$ and stay open for the remaining rounds (due to to Lemma 9.6). On the other hand, when the invariant of an active closed node during round $t+1$ is not violated, it will stay closed for the remaining rounds, since it will never lose its facility (open nodes from S_r and $S_{<r}$ do not close in round $t+2$ or after it). This means that all nodes in S_r are in a stable state at the end of round $t+2$. □

Corollary 9.8. *If no event occurs, the initialization takes $\mathcal{O}(\log n)$ rounds.*

We have proven that after $\mathcal{O}(\log n)$ rounds the nodes keep an $\mathcal{O}(1)$-approximation of the optimal solution until an event occurs. Before we analyze the effects of an event, we want to bound the number of events that can occur.

Theorem 9.9. *If the dynamics parameter in a time interval $[t_0, t_1]$ is upper bounded by x and all distances are in constant mode at time t_0, then the number of events in this time interval is $\mathcal{O}(x \log n)$.*

Proof. Consider a node p_i with radius r_i^{LFL} and another node p_j. If the distance between p_i and p_j remains the same (p_i and p_j are in constant mode), they cannot trigger events. If the distance decreases (they are in decreasing mode), p_j can trigger an event at p_i only when p_j enters a ball with center p_i and radius the $\frac{1}{4}$-, $\frac{1}{2}$-, two- or fourfold of one of the $\mathcal{O}(\log n)$ possible radii of p_i. Since p_j cannot leave a ball again as long as p_i and p_j stay in decreasing mode, p_j can trigger only $\mathcal{O}(\log n)$ events at p_i. Analogously, p_j can only trigger $\mathcal{O}(\log n)$ events at p_i when they are in increasing mode. Thus, for each change of mode, there are at most $\mathcal{O}(\log n)$ events resulting in $\mathcal{O}(x \log n)$ events in total. □

Note that the $\mathcal{O}(\log n)$ factor in Theorem 9.9 is necessary, because the current radius of a node p_i can change. Especially, when p_j enters the ball with radius $\frac{1}{4} r_i^{\mathrm{LFL}}$, r_i^{LFL} can decrease by one. Thus, when p_j enters the next smaller ball, this one again has radius $\frac{1}{4} r_i^{\mathrm{LFL}}$ and r_i^{LFL} decreases. Thus, by decreasing the distance between p_i and p_j, p_i's radius can change from the largest possible to the smallest one, resulting in $\mathcal{O}(\log n)$ events.

However, due to Theorem 9.7 and Theorem 9.9 we are now able to upper bound the number of rounds directly in dependency of the mode changes without stating the events explicitly.

Corollary 9.10. *If the dynamics parameter in a time interval $[t_0, t_1]$ is upper bounded by x and all distances are in constant mode at time t_0, then there are in total at most $\mathcal{O}(x \log^2 n)$ rounds where nodes change their status.*

Note that, if we use the same motion pattern that we used in the global scenario, which are the bounded degree polynomials, we get an x which is set to n^2 as mentioned before. The total processing time for the scenario is $\mathcal{O}(n^2 \log^2 n)$, which is of the same order up to poly-logarithmic terms as for the kinetic data structure.

9.3.2.1. Effects of events

Now we show that an event is handled efficiently. Especially, it affects only nodes in a constant distance from the event. We have already seen that it takes at most $\mathcal{O}(\log n)$ rounds until all invariants are kept again if no further event occurs in between. In the geometric setting that we consider under a slightly different round model, even in the worst case only a poly-logarithmic number of nodes can be affected by the event. Moreover, each affected node can change its role at most twice.

We state the central result that the algorithm is indeed local.

Theorem 9.11. *A node p_j can only be affected by an event if it is triggered at a node which is at most in distance $12 \cdot r_j^{LFL}$ from p_j.*

Proof. Let p_k be a node at which an event is triggered. Thus p_k (and possibly also a virtual node at the same position) changes its role. We prove that only nodes p_j are affected by a role change of p_k for which hold $D(p_k, p_j) \leq 12 \cdot r_j^{\mathrm{LFL}}$.

Let $e_i \cdot r_i^{\text{LFL}}$ be the maximal range around p_k in which nodes with radius at most $r_i^{\text{LFL}} = 2^i \cdot r_k^{\text{LFL}}$ are affected by the role change of p_k (note that r_i^{LFL} is not the radius of a specific node here). We show that $e_i \leq 12$, the theorem follows. We start with computing e_0. To do this, we first consider the case, where the role of p_k changes from open to closed. This does not affect nodes with radius less than p_k. Nodes with radius r_k^{LFL}, which are in distance at most $4 \cdot r_k^{\text{LFL}}$ from p_k, may now need to open themselves. Let p_l be such a node. Because of Lemma 9.6, no nodes with radius smaller than or equal to r_k^{LFL} close themselves because of p_l. Thus, $e_0 = 4$. If the role of p_k changes from closed to open, the same argument applies: No node with the same radius r_k^{LFL} can close itself due to p_k. Thus, in this case $e_0 = 0$.

We now prove that $e_i = \frac{e_{i-1}}{2} + 6$ for $i > 0$. This applies independent from the type of role change of p_k. By the definition of e_{i-1}, nodes with radius $2^{i-1} \cdot r_k^{\text{LFL}}$, which changed their role due to the role change of p_k, can be at most in distance e_{i-1} times their radius from p_k. Let p_j be a node with radius $r_j^{\text{LFL}} = 2^i \cdot r_k^{\text{LFL}} = r_i^{\text{LFL}}$, which changes its role due to a role change of a node p_l with a smaller radius (if there is a node with radius r_j^{LFL}, which changes its role, there must also exist such nodes p_j and p_l). We first consider the case that p_j opens itself. Then, p_l must have closed itself and be within distance $4 \cdot r_j^{\text{LFL}}$ of p_j. Moreover, because of Lemma 9.6, no facility with radius less than or equal to r_j^{LFL} can close itself due to p_j. Because of the triangle inequality, p_j is in distance at most $e_{i-1} \cdot r_l^{\text{LFL}} + 4 \cdot r_j^{\text{LFL}} \leq (\frac{e_{i-1}}{2} + 4) r_j^{\text{LFL}}$ of p_k. The second case is that p_j closes itself. Now, p_l must have opened itself and be in distance of at most $2 \cdot r_j^{\text{LFL}}$ of p_j. Here, the role change of p_j may affect nodes with the same radius: a node p_m with radius r_j^{LFL} may have to open itself, because no facility is left within a range of 4 times its radius. Thus, p_m can be at most in distance $4r_m^{\text{LFL}} = 4r_j^{\text{LFL}}$ of p_j and therefore $6r_j^{\text{LFL}}$ of p_l (triangle inequality). Again, there cannot exist a node with radius less than or equal to $r_m^{\text{LFL}} = r_j^{\text{LFL}}$ which needs to close itself because of the role change of p_m. Thus, no node with radius less than or equal to $r_j^{\text{LFL}} = 2^i \cdot r_k^{\text{LFL}}$ which changes its role due to the role change of p_k can be in distance of more than $e_{i-1} r_l^{\text{LFL}} + 6 r_j^{\text{LFL}} \leq (\frac{e_{i-1}}{2} + 6) r_j^{\text{LFL}}$ of p_k and because $r_j^{\text{LFL}} = r_i^{\text{LFL}}$ and by the definition of e_i, $e_i \leq \frac{e_{i-1}}{2} + 6$. This recurrence can be solved:

$$e_i \quad \leq \quad \frac{e_{i-1}}{2} + 6 = \frac{e_0}{2^i} + \sum_{k=0}^{i-1} \frac{6}{2^k} = \frac{e_0}{2^i} + 6 \cdot \frac{1 - (\frac{1}{2})^i}{1 - \frac{1}{2}} = 12 - \frac{12 - e_0}{2^i} \leq 12$$

Thus, each node can only be affected by events at nodes within 12 times its radius. Note that this also holds if two role changes occur at the same time at the same position. □

Since r_i^{LFL} is upper bounded by $8\frac{f_i}{d_i}$, nodes can only be in constant distance from events by which they are affected. This also implies that upon an change all information that is used by a node stems only form its neighborhood that is bounded above by a constant, although it might require $\mathcal{O}(\log n)$ rounds until it reaches its final decision. Remember that in Theorem 7.2 it was shown, that small changes in a local neighborhood lead to changes in a linear distance when considering exact FACILITY LOCATION. The same holds if the Mettu and Plaxton algorithm is not modified (besides not being applicable, because it is a central algorithm). Now we formulate for the only time in this chapter a theorem that holds in Euclidean spaces only. All other results hold in general metrics. Additionally we have a slightly more restricted round model. We show that the number of affected nodes upon an event is bounded from above.

Theorem 9.12. *In a Euclidean space with constant dimension and for an asynchronous round model where each node turns active exactly once per round in an arbitrary order, an event affects at most $\mathcal{O}(\log^2 n)$ nodes, if no further event occurs before all invariants hold again.*

Proof. For simplicity we consider the case of the Euclidean plane only. For higher dimensions the arguments are analogous. We know from Theorem 9.7 that after $\mathcal{O}(\log n)$ rounds all invariants hold again. It is left to show that in each round $\mathcal{O}(\log n)$ nodes change their role. Hence, we prove that for each of the $\mathcal{O}(\log n)$ possible radii at most a constant number of nodes change their role in each round. Consider one arbitrary round and a radius r_i^{LFL}. Let p_e be the node at which the event was triggered. If a node p_i has radius r_i^{LFL}, it has distance at most $12 \cdot r_i^{\mathrm{LFL}}$ to p_e, due to Theorem 9.11. Therefore all nodes with radius r_i^{LFL} which potentially change their role are in an area around p_e with radius $12 \cdot r_i^{\mathrm{LFL}}$ and therefore with an area $144\pi \cdot r_i^{\mathrm{LFL}} \cdot r_i^{\mathrm{LFL}}$. On the other hand, at any time all facilities with radius r_i^{LFL} have a minimal distance of $2 \cdot r_i^{\mathrm{LFL}}$ to each other leading to an area of $\pi \cdot r_i^{\mathrm{LFL}} \cdot r_i^{\mathrm{LFL}}$ around each facility, which does not intersect with according areas of other facilities. Hence, at any given point in time, there are at most 144 facilities with a radius of r_i^{LFL}. Because each node is active only once, only the constant number of facilities at the beginning of the round can close. Out of the same reason, nodes which open themselves stay open until the end of the round, and so at most 144 nodes can open in one round. Thus, no more than 288 nodes with radius r_i^{LFL} can change their role in one round.

For higher dimensions, we consider a d-dimensional ball of radius $12 \cdot r_i^{\mathrm{LFL}}$ and bound the number of balls with radius r_i^{LFL} that fit into the volume. □

Note that similar to the proof of Lemma 4.2 in the first part of the thesis, the only geometric argument that is used is the fact that for objects with an extend or volume only a bounded number of objects fits into a fixed sized neighborhood. Once again, the analysis can be applied to any doubling metric. Now, we consider general metrics and show that nodes do not flip their status arbitrary often upon an event.

Theorem 9.13. *For each event, each node changes its role at most twice.*

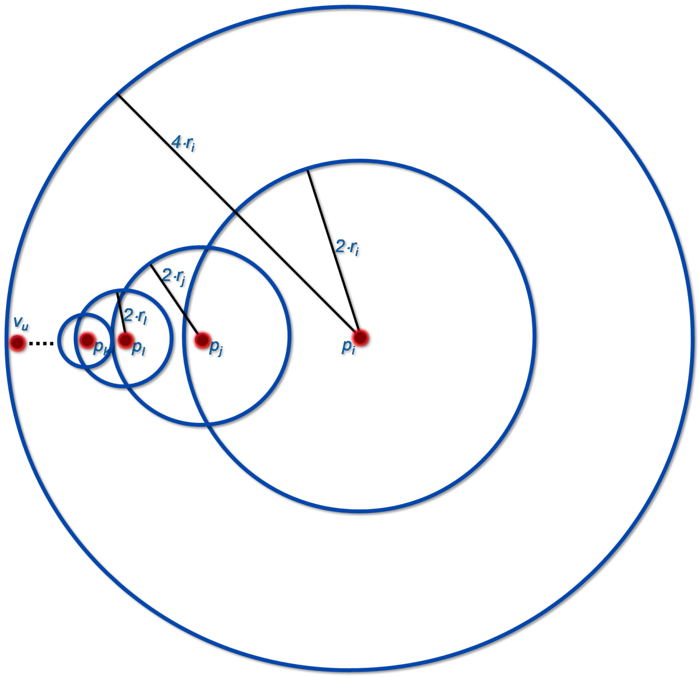


Figure 9.2.: For one event a node can only change its status twice.

Proof. We show that when a node has closed itself, it only opens itself again because of a later event. From this follows the theorem.

Let p_i be a node that closed itself because of a chain of node role changes that was initially triggered by an event e. Because of Lemma 9.6, the node p_j that forced p_i to close itself must have a smaller radius than p_i, and it must lie within distance $2r_i^{\mathrm{LFL}}$ of p_i. For p_i to open again, p_j must close. As long as no other event happens, this must be because another node p_l opens itself which is within distance $2r_j^{\mathrm{LFL}}$ of p_j and for which $r_l^{\mathrm{LFL}} < r_j^{\mathrm{LFL}}$ (Lemma 9.6). This continues until we have no smaller radii left. Let p_k be the last node to open itself in this chain. We now show that it is still within distance $4r_i^{\mathrm{LFL}}$ of p_i.

We know that p_j is in distance at most $2r_i^{\mathrm{LFL}}$ from p_i, $D(p_l, p_j) \leq 2r_j^{\mathrm{LFL}}$ and so on. This yields the following sum as an upper bound on the distance that p_k can have from

p_i:

$$\begin{aligned} & 2r_i^{\mathrm{LFL}} + 2r_j^{\mathrm{LFL}} + 2r_l^{\mathrm{LFL}} + \dots \\ = \; & 2r_i^{\mathrm{LFL}} + 2 \cdot \frac{1}{2} r_i^{\mathrm{LFL}} + 2 \cdot \frac{1}{4} r_i^{\mathrm{LFL}} + \dots \\ \leq \; & 2r_i^{\mathrm{LFL}} \sum_{s=0}^{\infty} (\frac{1}{2})^s \\ = \; & 2r_i^{\mathrm{LFL}} \frac{1}{1-\frac{1}{2}} \\ = \; & 4r_i^{\mathrm{LFL}} \end{aligned}$$

So as long as no new event occurs, there is always a facility with smaller radius within distance $4r_i^{\mathrm{LFL}}$ of p_i and thus p_i does not close again. Figure 9.2 illustrates this proof. □

Now, we have shown all crucial properties of our local algorithm. Before concluding our results in the next chapter, we present possible generalizations to other scenarios, which are not geometric domains.

9.3.3. The non-Euclidean case

While in the kinetic data structure framework we were restricted to Euclidean metrics by definition, we are not limited in such a way for our own local model.

Since the problem is not only defined for the Euclidean but for any metric, other applications are possible. A connected, weighted graph, where the edges represent hardwired connections between nodes and the weights the quality of the connections (i.e. latencies), can be used to represent the underlay for a computer network. This underlay induces a metric on the nodes by defining the distance function D as the shortest path (regarding the sum of weights on the path) in the underlay between two nodes. These latencies change over time and thus dynamics is introduced. As in the setting with moving robots, finding a clustering minimizing the costs can be an objective here, in order to provide a costly service at some of the nodes for instance.

This is also the reason why our results in this chapter are stated as general as possible and not for the Euclidean case only. Especially all theorems in this chapter hold except for Theorem 9.12 . However, it is often assumed that the shortest path metric induced by Internet latencies has constant doubling dimension. As mentioned before, Theorem 9.12 can be applied for any doubling metric.

There are a few aspects that have to be adopted in the non-Euclidean case. The round model can be applied unchanged. The same holds for the model of dynamics. Note, that our description of dynamics also fits the latency model above. The only real adjustment has to be made in the communication model, which can be done as follows.

A possible way of dealing with communication in the computer network scenario is to broadcast the information about the radius and role using the underlay. Since each node is only interested in the nodes within constant distance k, we can limit the messages' traveling distance by adding a label to each message telling how far it is supposed to be sent. In order for this approach to be reasonable, we need to assume that the rate at which the latencies are changing is low and that communication overhead produced by our algorithm is insignificant compared to the huge data streams sent through the network, whose impact on the network is modeled by the distance function D (i.e. the latencies between the nodes).

CHAPTER 10

Conclusion and open questions concerning internal assignment

In this part of the thesis we considered the FACILITY LOCATION problem under motion. We described and analyzed a global and a local algorithm and provided lower bounds, showing that our results are tight in many respects.

For the global scenario, we proposed a kinetic data structure that maintains a subset of the moving input points as facilities such that, at any point of time, the associated total cost is at most a constant factor larger than the current optimal cost. We showed that our kinetic data structure is compact, kinetic local, responsive, and efficient.

For the local scenario, we showed that after a logarithmic number of rounds the algorithm stabilizes in constant factor approximation. Therefore we are comparable with the runtime of state of the art algorithms while performing better concerning the locality. The viewing range is upper bounded by a constant and we showed that this is the best one can do when seeking a constant factor approximation. If the current solution becomes unstable due to motion, changes until a stable solution is reached again are bounded to a local neighborhood only and affects only a subset of nodes, whose cardinality is upper bounded by $\mathcal{O}(\log^2 n)$. Each of those nodes changes its role at most twice.

The complexity of our kinetic data structure depends poly-logarithmically on the ratio R as defined in Subsection 8.2.1.3, so that the compactness, kinetic locality, responsiveness, and efficiency are not fully poly-logarithmic, but only pseudo-poly-logarithmic. It would be nice future work to reduce this pseudo-poly-logarithmic term to a real poly-logarithmic term. Future work in the area of kinetic FACILITY LOCATION problems could include to consider an additional opening cost that arises at the moment when a point changes its status from client to facility. Here we point out that in our scenario the opening cost per event would be already bounded, because we open at most a logarithmic number of facilities per event. However, it is open, whether there are non-trivial lower bounds about this. This question remains open for the local model as well.

Furthermore, we presented a simple $\mathcal{O}(1)$-approximation algorithm for the local FACILITY LOCATION problem under worst case dynamics. We proved major key properties, such as upper bounds on the time until stabilization, the number of nodes affected by dynamics and - most important - that only a local neighborhood is affected by events. However, some questions remain open. For instance, we guarantee a constant-factor approximation only in stable configurations. We believe that this also holds before the algorithm stabilizes after an event.

An extension of our setting is the insertion and deletion of nodes. Furthermore, different from the kinetic data structure we assumed for our local algorithm that all f_i and d_i are constant. Although this makes sense in a robotic scenario (Section 9.1), it remains to show how our analysis changes when we allow those parameters to depend on n.

The number of events in our setting only depends on the dynamics parameter. However, it would be better if many changes of direction on a short time interval could not trigger many events. The approximation factor depends on the number of possible radii. Is it possible to improve the approximation factor by considering more possible values for the radii? Does a trade-off exist between the number of events, the approximation ratio and the locality of the neighborhood? This open question also applies to our kinetic data structure.

We need $\mathcal{O}(\log\log n)$ communication bits to guarantee a $\mathcal{O}(1)$-approximation. What kind of approximation can be obtained with a constant number of bits? Moreover, we do not know yet whether Theorem 9.12 (which states that only logarithmic many nodes are affected by an event if each node is invoked only once per round) also holds in an unrestricted round model. Finally, we believe that our algorithm is resilient against all kinds of transient failures: several events at the same time, nodes that wake up at the same time, nodes missing some events or nodes that temporarily display wrong information. Since we know that an initialization from any state stabilizes in $\mathcal{O}(\log n)$ rounds in a good approximation, this also holds after each transient failure recovers. However, a thorough analysis what influence such failures have on their neighborhood would be nice. The same holds for the case that events occur faster than the wake up pace of nodes for a limited period of time.

We assumed for our local algorithm that at any point of time, at most one node is awake. Obviously we can relax this restriction, such that it holds for any local neighborhood. However, it might be possible that far less symmetry-breaking is necessary, since we only run into troubles when two nodes in the same neighborhood are awake always at the same time and that have the same radius and both block each other permanently. A systematic analysis seems to be worthwhile.

Furthermore, other variants of the FACILITY LOCATION problem might be of interest, such as multi-commodity and robust FACILITY LOCATION under motion or even in a local scenario. In the latter, each node needs to connect not only to a single facility, but to a given integer number l of different facilities. In the former, there are different kinds of facilities and each node needs to be connected to one facility of each kind. Another possible variant to consider in our scenario is capacitated FACILITY LOCATION, where each facility can serve only a bounded number of clients. Additionally challenges rise since we have to deal with the problem of choosing the proper facility for each client.

Bibliography

[AAE03] P. K. Agarwal, L. Arge, and J. Erickson. Indexing moving points. *Journal of Computer and System Sciences*, 66(1):207 – 243, 2003. Special Issue on PODS 2000.

[ABdB+99] P. K. Agarwal, J. Basch, M. de Berg, L. J. Guibas, and J. Hershberger. Lower bounds for kinetic planar subdivisions. In *SCG '99: Proceedings of the fifteenth annual symposium on Computational geometry*, pages 247–254, New York, NY, USA, 1999. ACM.

[AdBPS09] M. A. Abam, M. de Berg, S.-H. Poon, and B. Speckmann. Kinetic collision detection for convex fat objects. *Algorithmica*, 53(4):457–473, 2009.

[AdBS07] M. A. Abam, M. de Berg, and B. Speckmann. Kinetic kd-trees and longest-side kd-trees. In *SCG '07: Proceedings of the twenty-third annual symposium on Computational geometry*, pages 364–372, New York, NY, USA, 2007. ACM.

[AEG98] P. K. Agarwal, J. Erickson, and L. J. Guibas. Kinetic binary space partitions for intersecting segments and disjoint triangles. In *SODA '98: Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pages 107–116, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics.

[AGG02] P. K. Agarwal, J. Gao, and L. J. Guibas. Kinetic medians and kd-trees. In *ESA '02: Proceedings of the 10th Annual European Symposium on Algorithms*, pages 5–16, London, UK, 2002. Springer-Verlag.

[AGHV97] P. K. Agarwal, L. J. Guibas, J. Hershberger, and E. Veach. Maintaining the extent of a moving point set. In *WADS '97: Proceedings of the 5th International Workshop on Algorithms and Data Structures*, pages 31–44, London, UK, 1997. Springer-Verlag.

[AGMV00] P. K. Agarwal, L. J. Guibas, T. M. Murali, and J.Scott Vitter. Cylindrical static and kinetic binary space partitions. *Comput. Geom. Theory Appl.*, 16(2):103–127, 2000.

[AHPV04] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Approximating extent measures of points. *J. ACM*, 51(4):606–635, 2004.

[Aro98] S. Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *J. ACM*, 45(5):753–782, 1998.

[ARR98] S. Arora, P. Raghavan, and S. Rao. Approximation schemes for euclidean k-medians and related problems. In *Proc. of the thirtieth annual ACM symposium on Theory of computing (STOC)*, pages 106–113, 1998.

[AV91] B. Awerbuch and G. Varghese. Distributed program checking: a paradigm for building self-stabilizing distributed protocols (extended abstract). In *SFCS '91: Proc. of the 32nd annual symposium on Foundations of computer science*, pages 258–267. IEEE Computer Society, 1991.

[Awe85] B. Awerbuch. Complexity of network synchronization. *J. ACM*, 32(4):804–823, 1985.

[BBKS00] S. Bespamyatnikh, B. Bhattacharya, D. Kirkpatrick, and M. Segal. Mobile facility location (extended abstract). In *Proc. of the 4th international workshop on Discrete algorithms and methods for mobile computing and communications (DIALM)*, pages 46–53, 2000.

[BCIS05] M. Badoiu, A. Czumaj, P. Indyk, and C. Sohler. Facility location in sublinear time. In *Proc. of the 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 866–877, 2005.

[BDKP09] O. Bonorden, B. Degener, B. Kempkes, and P. Pietrzyk. Complexity and approximation of a geometric local robot assignment problem. In *Algosensors*, 2009.

[BEG$^+$04] J. Basch, J. Erickson, L. J. Guibas, J. Hershberger, and L. Zhang. Kinetic collision detection between two simple polygons. *Computational Geometry*, 27(3):211 – 235, 2004.

[BGH97] J. Basch, L. J. Guibas, and J. Hershberger. Data structures for mobile data. In *SODA '97: Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, pages 747–756, Philadelphia, PA, USA, 1997. Society for Industrial and Applied Mathematics.

[BGZ97] J. Basch, L. J. Guibas, and L. Zhang. Proximity problems on moving points. In *SCG '97: Proceedings of the thirteenth annual symposium on Computational geometry*, pages 344–351, New York, NY, USA, 1997. ACM.

[BK99] P. Berman and M. Karpinski. On some tighter inapproximability results (extended abstract). In *26th International Colloquium on Automata, Languages and Programming*, pages 200–209, 1999.

[BMSS05] W. Burgard, M. Moors, C. Stachniss, and F.E. Schneider. Coordinated multi-robot exploration. *Robotics, IEEE Transactions on*, 21(3):376–386, 2005.

[CFS07] A. Czumaj, G. Frahling, and C. Sohler. Efficient kinetic data structures for maxcut. In *Canadian Conference on Computational Geometry (CCCG)*, pages 157–160, 2007.

[CG99] M. Charikar and S. Guha. Improved combinatorial algorithms for the facility location and k-median problems. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:378, 1999.

[CLRS01] T. H. Cormen, Ch. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. The MIT Press, 2001.

[CV86] R. Cole and U. Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Inf. Control*, 70(1):32–53, 1986.

[DGL08a] B. Degener, J. Gehweiler, and C. Lammersen. Kinetic facility location. *Algorithmica*, 2008.

[DGL08b] B. Degener, J. Gehweiler, and C. Lammersen. The kinetic facility location problem. In *Proceedings of the 24th European Workshop on Computational Geometry*, pages 251–254, 2008.

[DGL08c] B. Degener, J. Gehweiler, and C. Lammersen. The kinetic facility location problem. In *Proceedings of the 11th Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 378–389, 2008.

[DKP10] B. Degener, B. Kempkes, and P. Pietrzyk. A local, distributed constant-factor approximation algorithm for the dynamic facility location problem. In *24th IEEE International Parallel and Distributed Processing Symposium*, 2010. to appear.

[FR07] C. Frank and K. Römer. Distributed facility location algorithms for flexible configuration of wireless sensor networks. In *Distributed Computing in Sensor Systems*, pages 124–141, 2007.

[GGH+01] J. Gao, L. Guibas, J. Hershberger, L. Zhang, and A. Zhu. Discrete mobile centers. In *Proc. of the seventeenth annual symposium on Computational geometry (SOCG)*, pages 188–196, 2001.

[GGN04] J. Gao, L. J. Guibas, and A. Nguyen. Deformable spanners and applications. In *SCG '04: Proceedings of the twentieth annual symposium on Computational geometry*, pages 190–199, New York, NY, USA, 2004. ACM.

[GHSZ01] L. Guibas, J. Hershberger, S. Suri, and L. Zhang. Kinetic connectivity for unit disks. *Discrete & Computational Geometry*, 25(4):591–610, 2001.

[GK98] S. Guha and S. Khuller. Greedy strikes back: improved facility location algorithms. In *Proc. of the ninth annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 649–657, 1998.

[GLS06] J. Gehweiler, C. Lammersen, and C. Sohler. A distributed O(1)-approximation algorithm for the uniform facility location problem. In *Proc. of 18th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 237–243, 2006.

[GM04] B. P. Gerkey and M. J. Mataric. A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems. *The International Journal of Robotics Research*, 23(9):939–954, 2004.

[GPS87] A. Goldberg, S. Plotkin, and G. Shannon. Parallel symmetry-breaking in sparse graphs. In *STOC '87: Proc. of the nineteenth annual ACM symposium on Theory of computing*, pages 315–324. ACM, 1987.

[Gui98] L. J. Guibas. Kinetic data structures: a state of the art report. In *WAFR '98: Proceedings of the third workshop on the algorithmic foundations of robotics on Robotics : the algorithmic perspective*, pages 191–209, Natick, MA, USA, 1998. A. K. Peters, Ltd.

[Her03] J. Hershberger. Smooth kinetic maintenance of clusters. In *SCG '03: Proceedings of the nineteenth annual symposium on Computational geometry*, pages 48–57, New York, NY, USA, 2003. ACM.

[Her04] J. Hershberger. Kinetic collision detection with fast flight plan changes. *Information Processing Letters*, 92(6):287 – 291, 2004.

[HP04] S. Har-Peled. Clustering motion. *Discrete Comput. Geom.*, 31(4):545–565, 2004.

[HS01] J. Hershberger and S. Suri. Simplified kinetic connectivity for rectangles and hypercubes. In *SODA '01: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 158–167, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

[Ind04] P. Indyk. Algorithms for dynamic geometric problems over data streams. In *STOC '04: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 373–380, New York, NY, USA, 2004. ACM.

[JMS02] K. Jain, M. Mahdian, and A. Saberi. A new greedy approach for facility location problems. In *STOC '02: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 731–740, New York, NY, USA, 2002. ACM.

[JV01] K. Jain and V. V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *J. ACM*, 48(2):274–296, 2001.

[Kar72] R. M. Karp. Reducibility Among Combinatorial Problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. New York: Plenum, 1972.

[KMW05] F. Kuhn, T. Moscibroda, and R. Wattenhofer. On the locality of bounded growth. In *PODC '05: Proce. of the twenty-fourth annual ACM symposium on Principles of distributed computing*, pages 60–68. ACM, 2005.

[KR07] S. G. Kolliopoulos and S. Rao. A nearly linear-time approximation scheme for the euclidean k-median problem. *SIAM Journal on Computing*, 37(3):757–782, 2007.

[KSS00] D. Kirkpatrick, J. Snoeyink, and B. Speckmann. Kinetic collision detection for simple polygons. In *SCG '00: Proceedings of the sixteenth annual symposium on Computational geometry*, pages 322–330, New York, NY, USA, 2000. ACM.

[KSW05] D. Krivitski, A. Schuster, and R. Wolff. A local facility location algorithm for sensor networks. In *Distributed Computing in Sensor Systems*, pages 368–375, 2005.

[LBK$^+$05] M. G. Lagoudakis, M. Berhault, S. Koenig, P. Keskinocak, and A. J. Kleywegt. Simple auctions with performance guarantees for multi-robot task allocation. In *Multi-Robot Systems. From Swarms to Intelligent Automata Volume III*, pages 27–38. Springer Netherlands, 2005.

[Lin92] N. Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21(1):193–201, 1992.

[LSW09] C. Lenzen, J. Suomela, and R. Wattenhofer. Local algorithms: self-stabilization on speed. In *11th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, 2009.

[LZ05] L. Lin and Z. Zheng. Combinatorial bids based multi-robot task allocation method. In *Procc of the 2005 IEEE International Conference on Robotics and Automation (IRCA)*, pages 1145–1150, 2005.

[MGB+03] F. Mondada, A. Guignard, M. Bonani, D. Bär, M. Lauria, and D. Floreano. Swarm-bot: From concept to implementation. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robot and Systems (IROS)*, pages 1626–1631, 2003.

[MP00] R.R. Mettu and C.G. Plaxton. The online median problem. In *Proc. of the 41st Annual Symposium on Foundations of Computer Science*, pages 339–348, 2000.

[MW05] T. Moscibroda and R. Wattenhofer. Facility location: distributed approximation. In *Proc. of the twenty-fourth annual ACM symposium on Principles of distributed computing (PODC)*, pages 108–117, 2005.

[MYZ02] M. Mahdian, Y. Ye, and J. Zhang. Improved approximation algorithms for metric facility location problems. In *APPROX '02: Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 229–242, London, UK, 2002. Springer-Verlag.

[OMS01] E.H. Ostergaard, M.J. Mataric, and G.S. Sukhatme. Distributed multi-robot task allocation for emergency handling. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, Proc.*, volume 2, pages 821–826, 2001.

[PP09a] S. Pandit and S. V. Pemmaraju. Finding facilities fast. In *ICDCN*, volume 5408 of *Lecture Notes in Computer Science*, pages 11–24. Springer, 2009.

[PP09b] S. Pandit and S. V. Pemmaraju. Return of the primal-dual: distributed metric facility location. In *PODC*, pages 180–189, 2009.

[PT03] M. Pal and E. Tardos. Group strategy proof mechanisms via primal-dual algorithms. In *Proc. of the 44th Annual IEEE Symposium on Foundations of Computer Science*, pages 584–593, 2003.

[STA97] D. B. Shmoys, É. Tardos, and K. Aardal. Approximation algorithms for facility location problems (extended abstract). In *Proc. of the twenty-ninth annual ACM symposium on Theory of computing (STOC)*, pages 265–274, 1997.

[STK03] S. Sakai, M. Togasaki, and K.Yamazaki. A note on greedy algorithms for the maximum weighted independent set problem. *Discrete Applied Mathematics*, 126(2-3):313 – 322, 2003.

[SW08] J. Schneider and R. Wattenhofer. A log-star distributed maximal independent set algorithm for growth-bounded graphs. In *Proc. of the 27th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, 2008.

[TP07] F. Tang and L.E. Parker. A complete methodology for generating multi-robot task solutions using ASyMTRe-D and market-based task allocation. In *IEEE International Conference on Robotics and Automation*, pages 3351–3358, 2007.

[WS08] G. Wittenburg and J. Schiller. A survey of current directions in service placement in mobile ad-hoc networks. In *Sixth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 548–553, 2008.