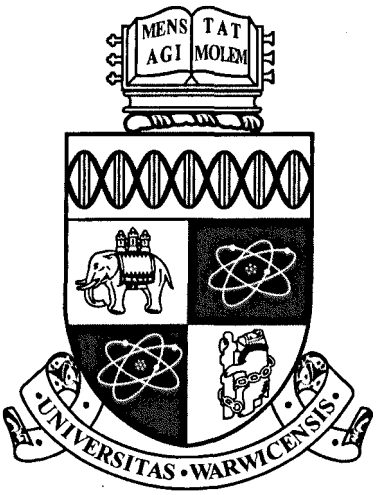

Approximation Algorithms for Geometric, Caching and Scheduling Problems

by

Anna Adamaszek

Thesis

Submitted to the University of Warwick

for the degree of

Doctor of Philosophy

Department of Computer Science

May 2012

THE UNIVERSITY OF
WARWICK

Contents

Acknowledgments

First of all, I would like to thank my supervisor Artur Czumaj for encouraging me to come to Warwick, and for his constant support during my studies.

I thank my collaborators, Michał Adamaszek, Matthias Englert, Andrzej Lingas, Alexandru Popa, Harald Räcke and Jakub Onufry Wojtaszczyk for interesting research discussions.

I am grateful for the financial support from the Centre for Discrete Mathematics and its Applications (DIMAP), EPSRC award EP/D063191/1. I received invaluable help from DIMAP Coordinators, Anna Guszcza and Yvonne Carty.

Special thanks to my office mates and friends from the university, in particular Konrad Dąbrowski, Jan Hladký, Nicholas Korpelainen, Sara Maloni, Diana Piguet, Chintan Shah and Michele Torielli.

Declarations

- Some of the results from Chapter 2 were obtained in collaboration with Artur Czumaj and Andrzej Lingas. A preliminary version has been presented at the 20th International Symposium on Algorithms and Computation (ISAAC 2009), and the full version [5] has been published in the International Journal of Foundations of Computer Science (IJFCS).

- The results from Chapter 3 were obtained in collaboration with Artur Czumaj, Andrzej Lingas and Jakub Onufry Wojtaszczyk, and presented at the 38th International Colloquium on Automata, Languages and Programming (ICALP 2011) [6].

- The results from Chapter 4 were obtained in collaboration with Artur Czumaj, Matthias Englert and Harald Räcke, and presented at the 43rd Annual ACM Symposium on Theory of Computing (STOC 2011) [3].

- The results from Chapter 5 were obtained in collaboration with Artur Czumaj, Matthias Englert and Harald Räcke, and presented at the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012) [4].

The thesis has not been submitted for a degree at another university.

Abstract

In this thesis we study approximation algorithms for optimization problems, which is one of the core areas of modern theoretical computer science. We focus on two areas of approximation. First we consider *geometric problems*, and we present approximation algorithms for the *capacitated location routing problem* and the *capacitated network design problem* in the Euclidean plane. Next, we investigate two well known caching and scheduling problems, the *generalized caching problem* and the *reordering buffer management problem.* We do this in an *online* setting, i.e. when instead of getting the whole input data at once, the data arrives in parts, during the execution of the algorithm.

In the capacitated location routing problem a fleet of vehicles with bounded capacity must serve a set of customers. The goal is to choose the depots for the vehicles from a set of possible locations, and fix the routes of the vehicles, to minimize the cost of opening the depots and the length of the routes. We present a quasi-polynomial time approximation scheme for the problem, and a polynomial time approximation scheme for some range of input parameters.

In the capacitated geometric network design problem we are given two sets of points in the plane, sources and sinks, where each source wants to send and each sink wants to receive a given amount of flow. The goal is to construct a minimum-length network with bounded edge capacity that allows to route the requested flow from sources to sinks. In addition to the sources and sinks, any other points in the plane can be used as vertices of the network. We present a quasi-polynomial time approximation scheme for the problem, and a polynomial time approximation scheme when the edge capacity is not too large.

The generalized caching problem is a classical problem in the area of online algorithms. We are given a set of pages, each page with an arbitrary size and fetching cost, and a cache of bounded size. At each time step a specific page is requested. If the page is not in the cache, it must be fetched into the cache, possibly evicting some other pages. The goal is to design an algorithm that specifies which pages to evict from the cache, minimizing the total cost incurred on the request sequence. We give a randomized online algorithm for the generalized caching problem which is asymptotically optimal, solving a long standing open problem.

The reordering buffer management problem is also a well known problem in the area of online algorithms. A stream of colored items arrives at a service station equipped with a reordering buffer of a given capacity. The cost for servicing the items depends on the processing order: servicing an item, when the previous item had a different color, incurs a context switching cost depending on the color of the current item. A scheduling strategy has to decide which item to service next, to minimize the cost of the output sequence. We show lower bounds on the competitive ratio of a deterministic and randomized online algorithm, and a deterministic online algorithm which nearly matches the lower bound.

Chapter 1

Introduction

1.1 Approximation Algorithms

Many practical optimization problems are NP-hard, and so it is unlikely that optimal solutions for them can be found in polynomial time. If we want to solve these problems in a reasonable amount of time, we have to give up optimality and aim for getting a solution with a value as close as possible to the value of an optimal solution. Algorithms which find efficiently sub-optimal solutions, and give a guarantee on the quality of the solution for any input instance, are called approximation algorithms.

Definition 1.1 ([86]). *An* α-approximation algorithm *for an optimization problem is a polynomial time algorithm which for all instances of the problem outputs a solution whose value is within a factor of* α *of the value of an optimal solution.*

The parameter α *is called the* approximation ratio *or* approximation factor *of the algorithm.*

We follow the convention that $\alpha > 1$ for minimization problems, and $\alpha < 1$ for maximization problems. The goal is to construct approximation algorithms with the approximation ratio α as close to 1 as possible.

Optimization problems differ a lot with respect to approximability. There are some problems which cannot be approximated at all, many others can be approximated within a factor which is a function of the input size n, e.g. $\alpha = n^{O(1)}$ or $\alpha = \log n$. Many problems admit an α-approximation algorithm for some constant α — the class of such problems is called APX. Some problems have a polynomial time approximation scheme, i.e. they can be approximated arbitrarily well.

Definition 1.2. *A* polynomial time approximation scheme *(PTAS) is an algorithm, which for any fixed* $\varepsilon > 0$*:*

- *for any problem instance finds a solution with value within a $(1+\varepsilon)$ factor of the value of an optimal solution, and*
- *has running time polynomial in the size of the problem instance.*

A polynomial time approximation scheme is essentially the best we can get for NP-hard optimization problems. For some problems we can get a similar approximation scheme, but with a quasi-polynomial running time, i.e. with a running time $n^{O((\log n)^c)}$ for input instances of size n, where c is a constant (possibly depending on the parameter ε).

Definition 1.3. *A* quasi-polynomial time approximation scheme *(QPTAS) is an algorithm, which for any fixed $\varepsilon > 0$:*

- *for any problem instance finds a solution with value within a $(1+\varepsilon)$ factor of the value of an optimal solution, and*
- *has running time quasi-polynomial in the size of the problem instance.*

Approximation algorithms for various problems have been studied since 1970s, see eg. [55; 84; 86] for an overview. Some of the techniques used in the design of approximation algorithms include combinatorial techniques, dynamic programming, linear and semidefinite programming, randomization.

In the thesis we study approximation algorithms for geometric problems. Optimization problems for which the input consists of points in a metric space have their geometric versions — the metric is then the constant-dimensional, or in the simplest case two-dimensional, Euclidean metric. Geometric versions of NP-hard problems usually remain NP-hard, but the structure of the Euclidean space often makes them simpler to approximate. For many geometric problems the existence of a polynomial time approximation scheme has been shown, although their metric versions are APX-hard, i.e. they do not admit a polynomial time approximation scheme unless $P = NP$.

The most known result of this kind has been shown independently by Arora [10] and Mitchell [69]. They designed polynomial time approximation schemes for the geometric version of the traveling salesman problem (TSP), as well as for the minimum Steiner tree problem (MST), k-TSP and k-MST. These results have been followed by polynomial time approximation schemes for geometric versions of many other problems, like the k-medians [12], minimum cost k-connected spanning subgraph for constant k [39], node-weighted geometric Steiner tree [77] and Euclidean Steiner forest [26]. They use the techniques introduced by Arora [10] and Mitchell

[69], but they also introduce many new ideas. For some geometric problems for which no polynomial time approximation scheme is known, the existence of a quasi-polynomial time approximation scheme has been shown — e.g. for the minimum latency problem [11], the minimum weight triangulation problem [76] and the capacitated vehicle routing problem [42].

We study geometric versions of some vehicle routing problems and of the capacitated network design problem.

1.1.1 Vehicle Routing Problems with Capacities

The *vehicle routing problem*, the problem of finding the optimal set of routes for a fleet of vehicles to serve a given set of customers, is one of the most important combinatorial optimization problems. It models the real-world problems of delivery or collection of goods, as well as more general transportation problems, such as waste collection, street cleaning, dial-a-ride systems or routing of maintenance units. There are many versions of the problem, which model different practical applications. In many applications the most important constraint is the capacity of the vehicles, and therefore one of the most important versions of the vehicle routing problem is the capacitated vehicle routing problem.

In the *capacitated vehicle routing problem* (CVRP) a fleet of vehicles with a limited capacity k located at a depot must serve a set of customers. The objective is to minimize the total distance traveled by the fleet.

The solution to the problem consists of a set of *tours* (cycles), where each tour corresponds to a route of a single vehicle. The length of a tour T is denoted by $|T|$. For a set $\mathcal{T}$ of tours, we define $|\mathcal{T}| = \sum_{T \in \mathcal{T}} |T|$.

The problem is defined formally as follows.

Definition 1.4. *In the capacitated vehicle routing problem we are given a set of points P, a point o and a distance function $\delta : (P \cup \{o\})^2 \to \mathbb{R}_{\geq 0}$. A solution consists of a set of tours $\mathcal{T}$ covering all the points from P, such that each tour visits the point o and at most k points from P.*

The goal is to find a solution which minimizes $|\mathcal{T}|$.

The point o is called the *origin* or *depot*. The set P represents the customers, and the tours — routes of the vehicles. An example of a solution to the CVRP problem can be seen in Figure 1.1. The problem is also known as a *k-tour cover problem.*

In the *multiple depot capacitated vehicle routing problem* (MDCVRP) we have multiple depots instead of a single one, and a customer can be served by a vehicle

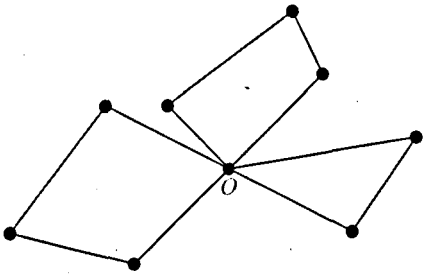

Figure 1.1: A solution to the CVRP problem for $k = 3$. The black dots represent the points from P, and the grey dot is the depot. The solution consists of three tours covering all the points from P, where each tour visits at most 3 points from P.

from an arbitrary depot.

Definition 1.5. *In the multiple depot capacitated vehicle routing problem we are given sets of points P and O, and a distance function $\delta : (P \cup O)^2 \to \mathbb{R}_{\geq 0}$. A solution consists of a set of tours $\mathcal{T}$ covering all the points from P, such that each tour visits a point from O and at most k points from P.*

The goal is to find a solution which minimizes $|\mathcal{T}|$.

We can consider a version of the problem called *MDCVRP with non-fixed return*, when the vehicles can finish their routes in a different depot from where they started. Then a solution consists of a set of paths, each starting and ending in O and visiting at most k points from P.

The CVRP problem is a special case of the MDCVRP problem, when the set O consists of a single point o.

In the *capacitated location routing problem* (CLRP) again we have to serve a set of customers with a fleet of vehicles of limited capacity k located at the depots, but first we have to choose a set of depots to be opened from a given set of locations O. We can open as many depots as we want, but opening each depot generates cost. The problem is defined formally as follows.

Definition 1.6. *In the capacitated location routing problem we are given sets of points P and O, a distance function $\delta : (P \cup O)^2 \to \mathbb{R}_{\geq 0}$ and a cost function $\phi : O \to \mathbb{R}_{\geq 0}$. A solution consists of a set of points $D \subseteq O$ and a set of tours $\mathcal{T}$ covering all the points from P, such that each tour visits a point from D and at most k points from P.*

The goal is to find a solution which minimizes $\sum_{d \in D} \phi(d) + |\mathcal{T}|$, i.e. the cost of opening the depots plus the total length of the tours.

As before, we can consider a version of the problem called *CLRP with non-fixed return*, when the vehicles can finish their routes in a different depot from where

they started.

The CLRP problem combines the MDCVRP problem with the *uncapacitated facility location problem* (UFL), where for a given set of customers and possible facility locations we have to decide on the set of facilities to be opened to minimize the total cost of opening the facilities and connecting each client to the closest open facility. The problem is defined formally as follows.

Definition 1.7. *In the uncapacitated facility location problem we are given sets of points P and O, a distance function $\delta : P \times O \to \mathbb{R}_{\geq 0}$ and a cost function $\phi : O \to \mathbb{R}_{\geq 0}$. The goal is to find a set of points $D \subseteq O$ that minimizes the cost $\sum_{d \in D} \phi(d) + \sum_{p \in P} \delta(p, D)$.*

The sets of points P and O represent the clients and the possible facility locations, and D the set of facilities to be opened.

The MDCVRP problem is a special case of the CLRP problem, when the cost of opening each depot is 0.

We consider the above problems in the *two-dimensional Euclidean* setting, where the input points correspond to points in the Euclidean plane $\mathbb{R}^2$, and the distances between them are the Euclidean distances.

1.1.2 Capacitated Network Design

The area of *network design*, problems of designing low cost networks satisfying some predefined constraints, plays a fundamental role in operational research and graph algorithms. We consider one important class of problems in this category, the *capacitated network design.* Here the goal is to find a multiset of edges in a given input network that will allow sending a predetermined amount of flow from a set of sources to a set of sinks subject to the capacity constraints on the edges.

While network design problems have been extensively studied in operational research and combinatorial optimization, the main focus has been on problems modeled by arbitrary graphs. However, many applications require to consider network design problems in the geometric setting.

In the *capacitated geometric network design problem* (CGND) we are given an integral edge capacity k and two sets of points on the Euclidean plane, sources and sinks, with an integral demand for each point. The demand of each source specifies the amount of flow that has to be shipped from the source, and the demand of each sink specifies the amount of flow that has to be shipped to the sink. The goal is to construct a minimum-cost network for which the requested flow from the sources to the sinks is feasible, and where each edge in the network has capacity k. The cost

of a network is defined as a sum of the lengths of the network edges. The vertices of the network are not constrained to the sets of sinks and sources — any point on the Euclidean plane can be used as a vertex. The flow is splittable and parallel edges are allowed.

An example of a solution to the CGND problem can be seen in Figure 1.2.

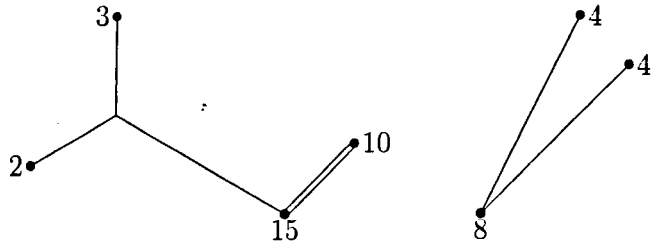

Figure 1.2: A solution to the CGND problem for the edge capacity $k = 5$. The sources and the sinks are denoted respectively by grey and black dots, and the demands are given next to the dots.

A special case of the CGND problem is a *single-sink capacitated geometric network design problem* (SCGND), where there is only one sink.

1.2 Online Algorithms

For many practical algorithmic problems we do not get the entire input sequence at once. Instead, new input data arrives in parts, and the decisions have to be made based on the data received so far. These problems are called *online problems*, and the algorithms for them are called *online algorithms*.

Among the most important classes of online problems are caching and scheduling problems. In a *caching problem* we have to maintain a two-level memory system consisting of a small fast memory and a large slow memory. Requested pages need to be fetched into the fast memory. The goal is to maintain the set of pages to stay in the fast memory, without knowing which pages will be requested in the future, to minimize the total fetching cost. In a *scheduling problem* a sequence of jobs must be scheduled on a set of machines, such that a given objective function is optimized. The jobs arrive one by one and must be scheduled, without any information about the jobs that will arrive later.

The performance of online algorithms is evaluated using *competitive analysis*. The idea is to compare the output of an online algorithm to the output of an *optimal offline algorithm*, which knows the whole input sequence in advance and finds an optimal solution for it. We consider deterministic and randomized online algorithms.

Definition 1.8. *An online algorithm is* α-competitive *if there is a constant* β *such that for all finite input sequences* I *the cost of the solution output by the algorithm is at most*

$$\alpha \cdot OPT(I) + \beta \ ,$$

where $OPT(I)$ *denotes the cost of a solution output by an optimal offline algorithm for* I.

The parameter α *is called the* competitive ratio *of the algorithm.*

In the above definition we assume that the algorithm is a deterministic online algorithm. For a randomized online algorithm instead of the cost of the solution we take the expected cost of the solution, where the expectation is taken over the random choices made by the algorithm.

One of the difficulties of the competitive analysis is that often the offline versions of the problems are NP-hard, which can make estimating the cost of the sequence output by an optimal offline algorithm difficult.

Online algorithms have been considered already in the 1970s, but the area has been extensively studied since the 1980s, when Sleator and Tarjan introduced the framework of competitive analysis [80]. An overview of the results in the area of online algorithms can be found e.g. in [25; 7; 27].

We study the reordering buffer management problem and the generalized caching problem in the online setting.

1.2.1 Reordering Buffer Management Problem

In the *reordering buffer management problem* a stream of colored items arrives at a service station and has to be processed. The cost for servicing the items depends heavily on the processing order: servicing an item with color c, when the most recently serviced item had color $c' \neq c$, incurs a context switching cost w_c.[1]

In order to reduce the total processing cost, the servicing station is equipped with a reordering buffer able to store k items. This buffer can be used to reorder the input sequence in a restricted fashion to construct an output sequence with a lower processing cost. At each point in time, the buffer contains the first k items of the input sequence that have not yet been processed. A scheduling strategy has to decide which item to service next. Upon its decision, the corresponding item is removed from the buffer and serviced, while the next item from the input sequence takes its place in the buffer.

[1] There exists a more general version of the problem, where the context switching cost for switching from an item with color c' to an item with color c depends on both c and c' (see Section 4.1.1).

This simple and versatile framework has many important applications in areas like production engineering, computer graphics, storage systems, and information retrieval, among others [16; 23; 51; 63; 74]. We give two examples.

In the paint shop of a car manufacturing plant, switching colors between two consecutive cars induces non-negligible cleaning and set-up costs. Therefore, paint shops are preceded by a reordering buffer (see [51]) to reorder the stream of incoming cars into a stream with a lower number of color changes. This setting is well modeled by the reordering buffer framework with *uniform costs*, i.e., $w_c = 1$ for all colors c.

In a 3D graphic rendering engine [63], a change in attributes between two consecutively rendered polygons slows down the GPU, as, for instance, the shader program needs to be replaced. A reordering buffer can be included between application and graphics hardware in order to reduce such state changes. This setting can be modeled by the reordering buffer framework with *non-uniform costs*. Non-uniform costs are required as the cost for a state change depends on the size of the program that has to be loaded.

We focus on the *online version* of the reordering buffer management problem, in which when the buffer becomes full, one has to decide which item to service next, without knowing the rest of the input sequence (see Figure 1.3). Instead, in the *offline version* the whole input sequence is known in advance, and the information about the colors of the items which will arrive later can be used to decide which items should be removed from the buffer.

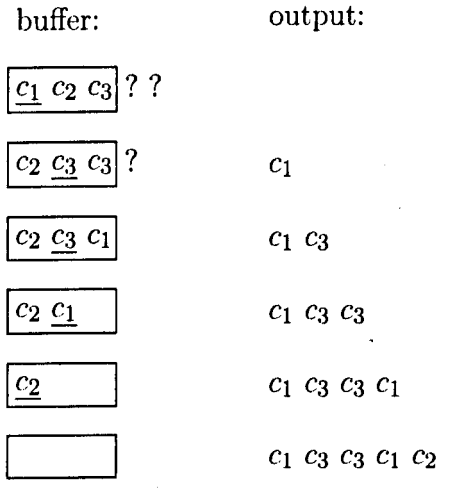

Figure 1.3: Solving an instance of the reordering buffer management problem. Here the size of the buffer $k = 3$. The item to be removed from the buffer is underlined. The cost of the generated output sequence is $2w_{c_1} + w_{c_2} + w_{c_3}$, and an optimal output sequence $c_2c_3c_3c_1c_1$ has cost $w_{c_1} + w_{c_2} + w_{c_3}$.

1.2.2 Generalized Caching Problem

In the basic two-level caching problem we are given a collection of n *pages* and a *cache* (a fast access memory). The cache has a limited capacity and can store up to k pages. At each time step a request to a specific page arrives and can be served directly if the corresponding page is in the cache; in that case no cost is incurred. If the requested page is not in the cache, a page fault occurs and, in order to serve the request, the page must be fetched into the cache, possibly evicting some other page, and a cost of one unit is incurred. The goal is to design an algorithm that specifies which page to evict in case of a fault such that the total cost incurred on the request sequence is minimized.

This classical problem can be naturally extended to the *generalized caching problem*, by allowing pages to have non-uniform fetching costs and to have non-uniform sizes. In the general model we are given a collection of n pages. Each page p is described by a fetching cost $c_p \geq 0$ and a size $w_p \geq 1$. The cache has limited capacity and can only store pages up to a total size of at most k. The framework of generalized caching has been motivated by applications in web caching and networking. The non-uniform sizes of the pages can correspond to the scenarios of caching web pages of different sizes, and the non-uniform costs of fetching a page can model scenarios in which the pages have different locations in a large network.

Various special cost models have been proposed in the literature. In the *bit model* [20; 58], each page p has $c_p = w_p$, and thus, for example, minimizing the fetching cost can correspond to minimizing the total traffic in the network. In the *fault model* [20; 58], for each page we have the fetching cost $c_p = 1$ and the size w_p may be arbitrary; in this case the fetching cost corresponds to the number of times a user has to wait for a page to be retrieved. In the *weighted caching* model [19; 58], for each page p we have the size $w_p = 1$ and the fetching cost c_p may be arbitrary; this models situations where some pages are more expensive to fetch than others because they may be on far away servers, or slower disks.

We consider the *online* version of the generalized caching problem. In this version as soon as a page which is not in the cache is requested, it has to be loaded into the cache, and while processing the request we have no information about the sequence of pages which will be requested later.

1.3 Thesis Organisation

In Chapter 2 we study two-dimensional Euclidean versions of the capacitated vehicle routing problem, the multiple depot capacitated vehicle routing problem and the

capacitated location routing problem. We first present a quasi-polynomial time approximation scheme for the capacitated location routing problem. Then we show a polynomial time approximation scheme for the capacitated vehicle routing problem for $k \leq 2^{\log^{o(1)} n}$, where n is the number of input points, and we extend it to work for the multiple depot capacitated vehicle routing problem and the capacitated location routing problem when $k, |O| \leq 2^{\log^{o(1)} n}$. Part of the chapter is based on joint work with Artur Czumaj and Andrzej Lingas [5].

In Chapter 3 we study the capacitated geometric network design problem and the single-sink capacitated geometric network design problem. We show that if the demands are polynomially bounded in the number of input points n, and the edge capacity k satisfies $k \leq 2^{O(\sqrt{\log n})}$, the single-sink capacitated geometric network design problem admits a polynomial time approximation scheme. If the capacity k is arbitrarily large, then we design a quasi-polynomial time approximation scheme for the capacitated geometric network design problem. Our results rely on a derivation of an upper bound on the number of vertices different from sources and sinks (the so called Steiner vertices) in an optimal network. The bound is polynomial in the total demand of the sources. The chapter is based on joint work with Artur Czumaj, Andrzej Lingas and Jakub Onufry Wojtaszczyk [6].

In Chapter 4 we study the reordering buffer management problem in an online setting. We present the first non-trivial lower bounds for this problem by showing that deterministic online algorithms have a competitive ratio of $\Omega(\sqrt{\log k / \log \log k})$ and randomized online algorithms have a competitive ratio of $\Omega(\log \log k)$. We complement this by presenting a deterministic online algorithm for the reordering buffer management problem that obtains a competitive ratio of $O(\sqrt{\log k})$, almost matching the lower bound. The chapter is based on joint work with Artur Czumaj, Matthias Englert and Harald Räcke [3].

In Chapter 5 we study the generalized caching problem in an online setting. We give a randomized $O(\log k)$-competitive online algorithm for the generalized caching problem, improving the previous bound of $O(\log^2 k)$ by Bansal, Buchbinder, and Naor [20]. This improved bound is tight and of the same order as the known bounds for the classic problem with uniform weights and sizes. The chapter is based on joint work with Artur Czumaj, Matthias Englert and Harald Räcke [4].

Chapter 2

Vehicle Routing Problems with Capacities in the Euclidean Plane

2.1 Introduction

In this chapter we study the capacitated vehicle routing problem, the multiple depot capacitated vehicle routing problem and the capacitated location routing problem. We consider the above problems in the *two-dimensional Euclidean* setting, where the input points correspond to points in the Euclidean plane $\mathbb{R}^2$, and the distances between them are the Euclidean distances.

2.1.1 Previous Results

Table 2.1 presents an overview of the results for the problems discussed in this chapter.

Capacitated vehicle routing problem. The CVRP problem is one of the most important special cases of a more general *vehicle routing problem*, introduced by Dantzig and Ramser [40] more than fifty years ago, and studied very extensively in the literature ever since (see e.g. [64; 82; 49] for an overview).

The CVRP problem contains the traveling salesman problem (TSP) as a special case, when $k = |P|$, and so it is NP-hard. When the capacity of each vehicle k is 2, the problem can be reduced to a minimum weight matching and can be solved in polynomial time. However, the problem is known to be NP-hard for all $k \geq 3$ [14; 15]. For this reason, the research has focused on heuristic algorithms and approximation algorithms. The most extensively studied variants of CVRP are the *metric* one, when the distance function is symmetric and satisfies the triangle inequality, and in particular the *two-dimensional Euclidean* one, when the points are

	metric	2-dimensional Euclidean
CVRP	2.5-approximation [52; 34] APX-hard for $k \geq 3$ [14; 15]	$(2+\varepsilon)$-approximation [52; 10] QPTAS [42] PTAS for $k = o(\log \log n)$ [52] PTAS for $k = O(\frac{\log n}{\log \log n})$ [15] PTAS for $k = \Theta(n)$ [15] **PTAS for $k \leq 2^{\log^{o(1)} n}$**
MDCVRP	4-approximation [65] APX-hard for $k \geq 3$ [14; 15]	QPTAS [41] PTAS for $k^3 \cdot \lvert O \rvert^2 = O(\frac{\log n}{\log \log n})$ [31] **PTAS for $k, \lvert O \rvert \leq 2^{\log^{o(1)} n}$**
MDCVRP with non-fixed return	2.5-approximation [65] APX-hard for $k \geq 3$ [14; 15]	**QPTAS** PTAS for $k^3 \cdot \lvert O \rvert^2 = O(\frac{\log n}{\log \log n})$ [31] **PTAS for $k, \lvert O \rvert \leq 2^{\log^{o(1)} n}$**
CLRP	4.38-approximation [53] APX-hard for $k \geq 3$ [14; 15]	$(4+\varepsilon)$-approximation [53] **QPTAS** **PTAS for $k, \lvert O \rvert \leq 2^{\log^{o(1)} n}$**
CLRP with non-fixed return	APX-hard for $k \geq 3$ [14; 15]	**QPTAS** **PTAS for $k, \lvert O \rvert \leq 2^{\log^{o(1)} n}$**

Table 2.1: An overview of the results for the vehicle routing problems considered in this chapter. The new results are presented using bold font.

placed in the plane and the distance is Euclidean.

The metric case of CVRP for any k is in APX, i.e. it admits a constant factor approximation. A simple *iterated tour partitioning heuristic*, introduced by Haimovich and Rinnooy Kan [52], has an approximation ratio $1+\alpha$, where α is an approximation ratio of the traveling salesman problem. That gives a 2.5-approximation for the metric case (using the 1.5-approximation for the metric TSP by Christofides [34]) and a $(2+\varepsilon)$-approximation for the constant-dimensional Euclidean case (using the $(1+\varepsilon)$-approximation for the Euclidean TSP by Arora [10] or Mitchell [69]). We describe the iterated tour partitioning heuristic in detail in Section 2.2. The general idea of the heuristic is to find a TSP tour visiting all points from the set $P \cup \{o\}$, cut it into paths visiting at most k points from P each and then connect the endpoints of the paths with the depot o to create tours.

For $k \geq 3$ the metric case of CVRP is APX-complete [14; 15], i.e. complete

for the class of optimization problems admitting constant factor approximation. In [14] Asano et al. show a reduction from the maximum bounded H-matching problem (i.e. the problem of determining the maximum number of vertex-disjoint copies of a fixed graph H in a given graph) which works for constant $k \geq 3$. For larger k, there is a reduction from TSP (see [15]).

The approximability status of the two-dimensional Euclidean CVRP problem, in particular, the problem of the existence of a PTAS, has not been completely settled yet.

One of the first studies of two-dimensional Euclidean CVRP has been due to Haimovich and Rinnooy Kan [52], who presented several heuristics for the metric and Euclidean CVRP, including a PTAS for the two-dimensional Euclidean CVRP with $k = o(\log \log n)$, [52, Section 6]. The idea of the algorithm is as follows. They divide the set of points from P into *inside* and *outside* points, depending on their distance from the origin, and they find solutions for the sets of inside and outside points independently. They show that the points can be divided into the two sets in such a way, that the number of outside points is at most $2^{O(k/\varepsilon)}$, for the inside points the iterated tour partitioning heuristic gives a solution with a near-optimal cost, and the increase in cost due to solving the problems for the outside and inside points separately is small. As the number of outside points is small, they find an optimal solution for them using an exponential-time algorithm. For the inside points they use the iterated tour partitioning heuristic. The algorithm finds a $(1+\varepsilon)$-approximate solution for the two-dimensional Euclidean CVRP problem in time $2^{2^{O(k/\varepsilon)}} + O(n \log n)$, where $n = |P \cup \{o\}|$, giving a PTAS for $k = o(\log \log n)$.

Asano et al. [15] substantially improved the above result by designing a PTAS for $k = O(\log n / \log \log n)$. As in [52], they divide the set of points P into inside and outside points, depending on their distance from the origin. They want to keep the same properties for the set of inside points, i.e. that the iterated tour partitioning heuristic finds for it a solution with a near-optimal cost. However, to decrease the number of outside points to $O(k^2/\varepsilon^2)$, they have to give up solving the outside and inside problems separately. Instead, they consider *outer tours*, which cover outside points together with some inside points. The remaining inside points are then covered using the iterated tour partitioning heuristic. They show that a nearly optimal solution using $O(k/\varepsilon^2)$ outer tours gives a $(1+\varepsilon)$-approximation to the CVRP problem. The algorithm finding such a solution is based on the Arora's scheme for geometric optimization problems [10]. The difference is that as the outer tours cover only $O(k^2/\varepsilon^2)$ points, the size of the grid in the randomized dissection depends only on k and ε, and not on n. As the cost of an optimal solution for

the set of inner points not included in the outer tours can be approximated well by knowing the sum of their distances from the origin, the dynamic programming as in Arora's scheme can be used to find a nearly optimal solution with $O(k/\varepsilon^2)$ outer tours. The running time is $(k/\varepsilon)^{O(k/\varepsilon^3)} + O(n \log n)$, which is polynomial for $k = O(\log n / \log \log n)$.

Asano et al. [15] also observed that Arora's [10] or Mitchell's [69] PTAS for the two-dimensional Euclidean TSP implies a PTAS for the corresponding CVRP problem for $k = \Theta(n)$.

There has not been any significant progress since the paper by Asano et al. [15] until recently, when Das and Mathieu [42] showed a quasi-polynomial time approximation scheme for the two-dimensional Euclidean CVRP for every k. Their algorithm combines the approach developed by Arora [10] for Euclidean TSP with some new ideas to deal with CVRP (in particular, how to handle a large number of possible values of the lengths of the subtours arising in the subproblems of the original CVRP), and gives a $(1+\varepsilon)$-approximation for the two-dimensional Euclidean CVRP in time $n^{\log^{O(1/\varepsilon)} n}$.

Multiple depot capacitated vehicle routing problem. Li and Simchi-Levi [65] consider the MDCVRP problem in the metric setting. They present constant factor approximation algorithms for both the MDCVRP problem and the MDCVRP problem with non-fixed return. They show that the MDCVRP problem with non-fixed return can be approximated by reducing it to the CVRP problem. The reduction works as follows. Let $I = (P, O, \delta)$ be an instance of the MDCVRP problem with non-fixed return. We modify I to create an instance I' of the CVRP problem. To do it, we merge the set of depots O into a single depot o, and set the distance between each point $p \in P$ and the depot o as the minimum distance between p and a depot. The distance function in the modified problem instance might not satisfy the triangle inequality. To fix this, we substitute the distance between each pair of points by the shortest path distance. An approximate solution S' for I' can be found e.g. using the iterated tour partitioning heuristic. Each edge in the solution S' corresponds to a path in I of the same length, where all the intermediate points on the path are the depots. We transform S' into a solution S for I by changing the edges of S' into paths in I and deleting the edges between the depots. The cost of S' is the same as the cost of S. As the optimal solutions for I and I' have the same cost, the above algorithm for the MDCVRP problem with non-fixed return has the same approximation ratio as the algorithm for the metric CVRP, i.e. 2.5. Notice that if we perform the above reduction for an Euclidean instance of the problem, the

instance of the CVRP problem obtained will be metric but not Euclidean, and the obtained approximation ratio of the Euclidean MDCVRP problem with non-fixed return is also 2.5.

Li and Simchi-Levi [65] present also a constant factor approximation algorithm for the MDCVRP problem. The algorithm first finds an approximate solution to the MDCVRP problem with non-fixed return for the same problem instance, using the algorithm described above. Then the solution is modified by transforming each path between a pair of depots into a tour, i.e. a path starting and ending in the same depot. It can be shown that the increase in cost caused by the modification is not too large, and the approximation ratio of the algorithm is 4.

As MDCVRP is an extension of the capacitated vehicle routing problem, its metric version is APX-complete.

In [31] Cardon et al. show that both the PTAS by Haimovich and Rinnooy Kan [52] and the PTAS by Asano et al. [15] for the two-dimensional Euclidean version of CVRP can be extended to work for the two-dimensional Euclidean MDCVRP problem and for the two-dimensional Euclidean MDCVRP problem with non-fixed return when the number of depots is small. The algorithm which is an extension of the result from [15] works as follows. The points from P are divided into outside and inside points depending on their distance to the nearest depot. The number of outside points is $O(k^2m^2/\varepsilon^2)$, where m is the number of depots. The inner points are grouped into m sets $P_1, \ldots, P_m$ according to their closest depot. The near-optimal solution consists of the outer tours covering outside points together with some inside points and found by a modification of the Arora's scheme (similarly as in [15]), and of independent approximate solutions for the inner points from each set P_i which were not covered by the outer tours. The algorithm finds a $(1+\varepsilon)$-approximate solution in time $(km/\varepsilon)^{O(k^3m^2/\varepsilon^2)}$, which is polynomial for small k and m, i.e. when $k^3 \cdot m^2 = O(\log n / \log\log n)$.

In [41] Das extends the QPTAS from [42] to work for the constant-dimensional Euclidean MDCVRP problem. The running time of the algorithm is $n^{\log^{O(1/\varepsilon)} n}$.

Capacitated location routing problem. The *location routing problem* is a generalization of the vehicle routing problem, where the costs of opening the depots (called also *facilities*) are considered together with the costs of the routes of the vehicles. Different versions of the problem have been widely studied for over 40 years and the overview of the results can be found e.g. in [71].

The capacitated location routing problem is an important special case of the general location routing problem. It contains as a special case the multiple

depot vehicle routing problem, when the costs of opening all depots are 0, and the uncapacitated facility location problem, when the vehicles have capacity $k = 1$.

The uncapacitated facility location problem has been extensively studied. Hochbaum [56] presented a greedy $O(\log n)$ approximation algorithm for the general (i.e. not restricted to metric distances) version of the problem. Shmoys, Tardos and Aardal [79] gave the first constant factor approximation algorithm for the metric version of the problem, with a constant 3.16. In a series of results [50; 36; 33; 59; 81; 67; 29] the constant has been improved and currently the best result is 1.488-approximation by Shi Li [66]. On the negative side, there is a lower bound of 1.463 on the approximation factor for the metric uncapacitated facility location problem (see [85], Section 4.4). The two-dimensional Euclidean uncapacitated facility location problem admits a polynomial time approximation scheme [12].

Harks et al. [53] present a constant factor approximation algorithm for the CLRP problem, which achieves 4.38 approximation ratio in the metric case and $(4 + \varepsilon)$ approximation ratio in the two-dimensional Euclidean case. They construct two lower bounds on the cost of an optimal solution. One is the cost of the optimal UFL solution for the same problem instance, with the costs of all the edges scaled down by a factor depending on k. The other is the cost of a minimum spanning tree of a graph on the set of the input points with weights representing the distances between the points, where the possible depot locations O are connected with edges of length 0, and the other edges going out from O have an increased cost, incorporating the cost of opening the depots. Then the solution is constructed from an approximate solution to the above UFL problem and from the MST of the above graph: the tree is split into subtrees, which are then transformed into tours and connected to the depots opened by the UFL solution or by the spanning tree. The algorithm is described in more detail in Section 2.2. The difference in the approximation ratio for the metric and two-dimensional Euclidean case comes from the accuracy with which we can approximate the UFL problem.

2.1.2 Overview of the New Results

In Section 2.3 we present a quasi-polynomial time approximation scheme for the two-dimensional Euclidean capacitated location routing problem and the two-dimensional Euclidean capacitated location routing problem with non-fixed return. The MD-CVRP problem is a special case of the CLRP problem, so that gives a QPTAS for the two-dimensional Euclidean MDCVRP problem and MDCVRP problem with non-fixed return. The algorithm is a modification of the QPTAS for the two-dimensional Euclidean capacitated vehicle routing problem by Das and Mathieu [42].

In Section 2.4 we first present a polynomial time approximation scheme for the two-dimensional Euclidean capacitated vehicle routing problem for $k \leq 2^{\log^{o(1)} n}$. The PTAS relies on a reduction of an instance of CVRP with a set of n points to an instance or a small number of independent instances of the problem with a small number of points. The first reduction takes any instance of CVRP on n points and reduces it to an instance of the problem with $(k/\varepsilon)^{O(1)} \cdot O(\log^2 n)$ points. Then we present a refinement, where the instance of CVRP is reduced to a small set of instances of CVRP, each with $(k/\varepsilon)^{O(1)}$ points. These results, when combined with the QPTAS due to Das and Mathieu [42], give the aforementioned PTAS. This part is based on joint work with Artur Czumaj and Andrzej Lingas [5].

At the end of Section 2.4 we present modifications needed to obtain a PTAS for the two-dimensional Euclidean capacitated location routing problem and the two-dimensional Euclidean capacitated location routing problem with non-fixed return for $k, |O| \leq 2^{\log^{o(1)} n}$. In this part we use the results from Section 2.3.

2.2 Bounds on the Cost of an Optimal Solution and Constant Factor Approximation Algorithms

In this Section we present some simple bounds on the cost of optimal solutions for the metric versions of the CVRP, MDCVRP and CLRP problems. We also present constant factor approximation algorithms for these problems.

Notation. For an instance I of the CVRP, MDCVRP or CLRP problem, $OPT(I)$ is the cost of an optimal solution for I. When the instance is clear from the context, we write OPT instead of $OPT(I)$. For a solution S for an instance of the CVRP, MDCVRP or CLRP problem, we define $C(S)$ to be the cost of S. For a set P of points, $TSP(P)$ is the length of the shortest tour through all the points from P. For a graph G, $MST(G)$ denotes the cost of a minimum spanning tree of G.

The distance between a pair of points p and q is denoted by $\delta(p, q)$. For the CVRP problem we set $r(p) = \delta(p, o)$, i.e. the distance between p and the origin o. For a point p and a set of points Q we denote by $\delta(p, Q)$ the distance between p and the set Q, i.e. $\delta(p, Q) = \min\{\delta(p, q) : q \in Q\}$. For two sets of points P and Q the distance between the sets $\delta(P, Q)$ is the minimum distance between the points from the sets, i.e. $\delta(P, Q) = \min\{\delta(p, q) : p \in P, q \in Q\}$.

We set: $n_P = |P|, n_O = |O|$, $n = n_P + n_O$ (in the case of MDCVRP and CLRP) and $n = n_P + 1$ for CVRP.

Capacitated vehicle routing problem. The following simple lower bound plays a very important role in the previous approaches to the CVRP problem, see [15, Proposition 2] and [52, Lemma 1].

Lemma 2.1. *For an instance $I = (P, o, \delta)$ of the metric CVRP problem*

$$OPT(I) \geq \frac{2}{k} \sum_{p \in P} r(p) \ .$$

Proof. Let T be a tour from an optimal solution for I, and $p \in P$ any point on T. We have $|T| \geq 2 \cdot r(p)$. As the tour visits at most k points from P, we get $|T| \geq \frac{2}{k} \cdot \sum_{p \in T} r(p)$. Summing up over all tours from the optimal solution gives the desired inequality. □

Here is another simple lower bound from [52, Lemma 1].

Lemma 2.2. *For an instance $I = (P, o, \delta)$ of the metric CVRP problem*

$$OPT(I) \geq TSP(P \cup \{o\}) \ .$$

Proof. A set of tours from an optimal solution for I can be transformed into a single tour by shortcutting (see Figure 2.1). □

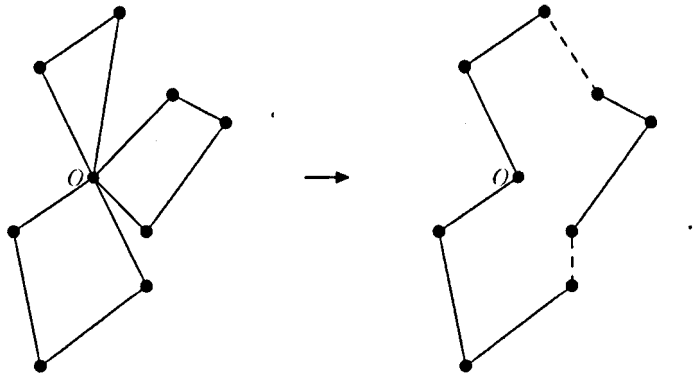

Figure 2.1: Shortcutting the solution for the CVRP problem to obtain a traveling salesman tour.

The *iterated tour partitioning heuristic* [52] can be implemented as follows. We find a near-optimal TSP tour T for the set of points $P \cup \{o\}$. Then for any $i \in \{1, \ldots, k\}$ we transform the tour T into a solution S_i to the CVRP problem in the following way. We split the tour into paths by removing every k-th edge of the tour, starting with the i-th edge. We then add the edges connecting the endpoints

of each path with the origin o (see Figure 2.2). We output the least expensive of the solutions S_i.

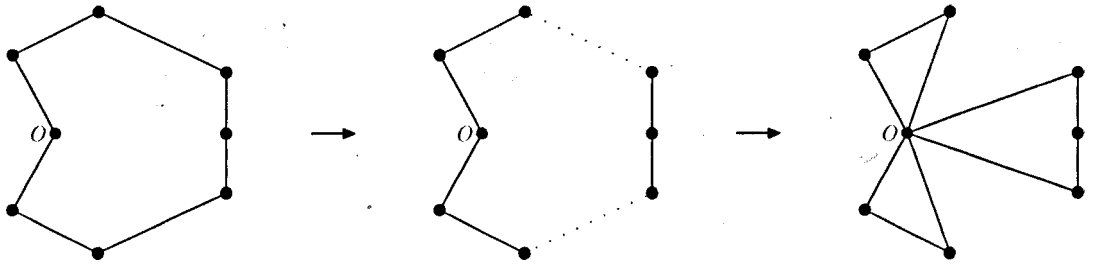

Figure 2.2: Transforming a TSP tour into a solution S_i to the CVRP problem in the iterated tour partitioning heuristic. Here $k = 3$ and $i = 3$.

Lemma 2.3. *The cost of the solution to the CVRP problem output by the iterated tour partitioning heuristic is at most*

$$\left(1 - \frac{1}{k}\right)|T| + \frac{2}{k}\sum_{p \in P} r(p) \ ,$$

where T is the TSP tour used by the heuristic.

The iterated tour partitioning heuristic is an $(1+\alpha)$-approximation algorithm for the CVRP problem, where α is the approximation ratio of the algorithm used for solving the TSP problem.

Proof. Each edge of the tour T is present in exactly $k-1$ of the k solutions S_i, as an ℓ-th edge of the tour is removed from the solution S_i if and only if $\ell \equiv i (\mathrm{mod}\ k)$ and $i \in \{1, \ldots, k\}$. An edge between a point $p \in P$ and the origin o is added in two of these solutions — when one of the edges incident with p on the tour T has been removed. Therefore

$$\sum_{i=1}^{k} |S_i| = (k-1)|T| + 2\sum_{p \in P} r(p) \ .$$

The least expensive of the solutions S_i has cost at most $(1 - \frac{1}{k})|T| + \frac{2}{k}\sum_{p \in P} r(p)$.

We take as T an α-approximate solution for the TSP problem on $P \cup \{o\}$. The cost of the obtained solution for the capacitated vehicle routing problem is at most $\alpha \cdot TSP(P \cup \{o\}) + \frac{2}{k}\sum_{p \in P} r(p)$. From Lemma 2.1 and Lemma 2.2 we get that it is at most $(1+\alpha) \cdot OPT$. □

Capacitated location routing problem. Similar lower bounds as for the capacitated vehicle routing problem can be shown for the metric version of the CLRP

problem.

We can show a lower bound on the cost of a solution for the CLRP problem based on the set of depots opened by the solution.

Lemma 2.4. *Let $S = (D, \mathcal{T})$ be a solution for an instance $I = (P, O, \delta, \phi)$ of the metric CLRP problem. Then*

$$\frac{2}{k} \sum_{p \in P} \delta(p, D) \leq |\mathcal{T}| \ .$$

Proof. Let $T \in \mathcal{T}$ and p be a point on T. As T visits some point from D, we have $|T| \geq 2\delta(p, D)$. A tour can visit at most k points from P, and therefore $|T| \geq \frac{2}{k} \sum_{p \in T} \delta(p, D)$. Summing up over all tours from $\mathcal{T}$ we get $|\mathcal{T}| \geq \frac{2}{k} \sum_{p \in P} \delta(p, D)$. □

The following two lower bounds on the cost of a CLRP solution have been shown in [53].

Lemma 2.5. *Let $I = (P, O, \delta, \phi)$ be an instance of the metric CLRP problem, and I' an instance of the UFL problem, with the same sets of points P and O, the same cost function ϕ and with the distance function δ' defined as: $\delta'(p, o) = \frac{2}{k}\delta(p, o)$ $\forall_{p \in P, o \in O}$.*

The cost of an optimal solution for I' is not greater than $OPT(I)$.

Proof. Let $S = (D, \mathcal{T})$ be an optimal solution to the instance I of the CLRP problem. Let S' be a solution for I', in which the set of opened facilities is D.

From Lemma 2.4 we obtain that the cost of the tours in S is not smaller than $\frac{2}{k} \sum_{p \in P} \delta(p, D) = \sum_{p \in P} \delta'(p, D)$, which is the cost of the edges in S'. The total cost of S is at least as large as the cost of S'. □

Lemma 2.6. *Let $I = (P, O, \delta, \phi)$ be an instance of the metric CLRP problem. Let G be a weighted complete graph on $P \cup O$, where the weights of the edges represent the distances δ between the pairs of points, with the weights within O set to 0, and the weights of the edges between each pair of points $o \in O$ and $p \in P$ increased by $\frac{1}{2}\phi(o)$.*

The cost of the minimum spanning tree for G is not greater than $OPT(I)$.

Proof. Let $S = (D, \mathcal{T})$ be an optimal solution for I. We will construct a spanning tree of G with a small cost. First we create an arbitrary spanning tree of the set O, which has cost 0. Then for each $d \in D$ we take the set of tours from $\mathcal{T}$ starting in d and transform them into a path Γ_d visiting the same set of vertices, by shortcutting the set of tours into one tour and then removing one of the edges of the tour incident with d (see Figure 2.3). The cost of the path Γ_d in G can be greater than the cost

(in I) of the corresponding tours from T by at most $\frac{1}{2}\phi(d)$, as the path contains one edge incident with the depot d.

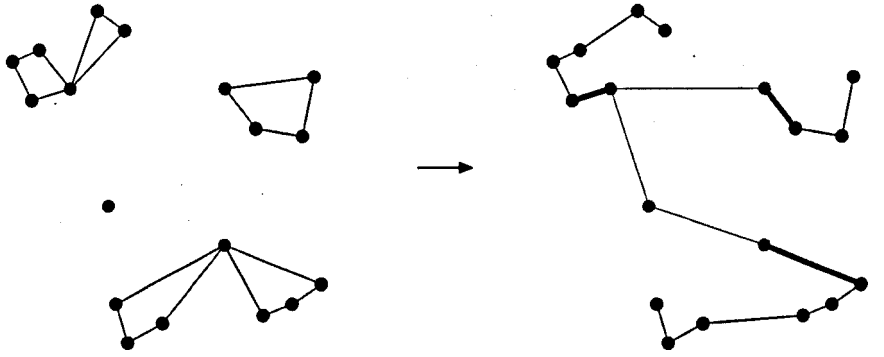

Figure 2.3: Transforming a solution for the CLRP problem (for $k = 3$) into a spanning tree of G (Lemma 2.6). The points from the sets P and O are denoted by black and grey dots respectively. The grey edges have cost 0 in G. The bold edges have cost increased by $\frac{1}{2}\phi(o)$. Notice that only 3 out of the 4 depots have been opened by the CLRP solution.

A collection of paths Γ_d for $d \in D$, together with the spanning tree of O, gives a spanning tree of G. The cost of the spanning tree is at most $\frac{1}{2}\sum_{d\in D}\phi(d) + \sum_{T\in\mathcal{T}}|T| \leq OPT(I)$. □

We now present the constant factor approximation algorithm for the metric CLRP problem by Harks et al. [53]. Let $I = (P, O, \delta, \phi)$ be an instance of the CLRP problem. Let I' be an instance of the UFL problem as in Lemma 2.5, i.e. $I' = (P, O, \delta', \phi)$, where the distance function δ' is defined as: $\forall_{p\in P, o\in O}\ \delta'(p, o) = \frac{2}{k}\delta(p, o)$. Let S' be a solution for I', and let $D_1 \subseteq O$ be the set of depots opened by S'.

Let G be a graph as in Lemma 2.6, i.e. a weighted complete graph on $P \cup O$, where the weights of the edges represent the distances δ between the pairs of points, with the weights within O set to 0, and the lengths of the edges between each pair of points $o \in O$ and $p \in P$ increased by $\frac{1}{2}\phi(o)$. We find a minimum spanning tree T of G and set D_2 to be the set of vertices from O which are incident in T with a vertex from P.

In a solution to the CLRP problem we open the depots $D = D_1 \cup D_2$. We create the tours in the following way. As T is a minimum spanning tree of G, after removing the edges within O and then the trees consisting only of single points $o \in O$, it becomes a forest covering $P \cup D_2$, where each connected component contains one point $d \in D_2$. We divide each tree into subtrees in such a way, that each subtree has at least $\frac{k}{2}$ and at most k points from P (except the subtree containing d, which

can be smaller). Some of the subtrees might share a vertex, but they do not share an edge. We transform each subtree into a tour visiting the nearest depot from D.

That gives a feasible solution to the instance of the CLRP problem with cost at most $2C(S') + 2|T|$ (or, if we want to be more precise, at most $\sum_{d \in D_1} \phi(d) + 2 \cdot \sum_{p \in P} \delta'(p, D_1) + 2|T|$). Lemma 2.5 and Lemma 2.6 give bounds on the values of $C(S')$ and $|T|$. Using a bifactor approximation algorithm to the UFL problem by Byrka and Aardal [29], which provides separate approximation ratios for the cost of opening the facilities and connecting the clients, to find the solution S' for UFL guarantees a 4.38 approximation ratio for the metric CLRP problem. Using a $(1+\varepsilon)$ approximation algorithm for the two-dimensional Euclidean UFL by Arora et al. [12] gives a $(4 + \varepsilon)$ approximation guarantee for the two-dimensional Euclidean CLRP problem.

Multiple depot capacitated vehicle routing problem. The results for the capacitated location routing problem can be simplified for MDCVRP, by setting $\phi(o) = 0$ for each $o \in O$. Then the optimal solution for the UFL problem opens the facilities in all possible locations. From Lemma 2.5 and Lemma 2.6 we instantly get the following results.

Lemma 2.7. *For an instance $I = (P, O, \delta)$ of the metric MDCVRP problem*

$$OPT(I) \geq \frac{2}{k} \sum_{p \in P} \delta(p, O) \ .$$

Lemma 2.8. *Let $I = (P, O, \delta)$ be an instance of the metric MDCVRP problem. Let G be a weighted complete graph on $P \cup O$, where the weights represent the distances δ between the points, with the distances within O set to 0.*

The cost of the minimum spanning tree for G is not greater than $OPT(I)$.

From the approximation algorithm for the CLRP problem we instantly get an approximation algorithm for the MDCVRP problem. The UFL problem is solved optimally by opening all possible facilities. We get the following result:

Lemma 2.9. *There is a 4-approximation algorithm for the metric MDCVRP problem. It outputs a solution with cost at most*

$$2 \cdot \frac{2}{k} \sum_{p \in P} \delta(p, O) + 2 \cdot MST(G) \ ,$$

where G is a weighted complete graph on $P \cup O$, in which the weights represent the distances δ between the points, with the distances within O set to 0.

2.3 A QPTAS for the Two-Dimensional Euclidean Capacitated Location Routing Problem

In this section we present a quasi-polynomial time approximation scheme for the two-dimensional Euclidean capacitated location routing problem. The algorithm is based on the QPTAS for the two-dimensional Euclidean capacitated vehicle routing problem by Das and Mathieu [42]. At the end of the analysis we describe the changes needed to obtain a QPTAS for the two-dimensional Euclidean CLRP problem with non-fixed return.

We want to prove the following result.

Theorem 2.10. *There is a randomized quasi-polynomial time approximation scheme for the two-dimensional Euclidean capacitated location routing problem. For any $\varepsilon > 0$ the algorithm finds a solution with expected cost at most $(1+\varepsilon)OPT$ in time $n^{\log^{O(1/\varepsilon)} n}$.*

In this part, by CLRP we denote the two-dimensional Euclidean CLRP problem, unless stated otherwise.

Note that we can assume $1/\varepsilon = O(\log n)$, as otherwise a simple exponential time algorithm finds an optimal solution in time $n^n = n^{\log^{O(1/\varepsilon)} n}$.

Idea of the algorithm. We use Arora's scheme for geometric optimization problems [10]. We first shift the input points onto a grid of polynomial size and perform a randomized dissection of the square containing the input points into smaller squares, until we obtain unit squares containing at most one input point each, or more than one point, but all of them in the same position. We then introduce portals on the boundaries of the squares and consider only solutions, in which the tours cross the boundaries of the squares only in portals, and in which a tour does not cross a boundary of a single square too many times. We show that for an appropriate choice of parameters, i.e. the size of the grid, the number of portals and the maximum number of crossings of the boundaries, we can consider only solutions satisfying the above properties, and the cost of an optimal solution can increase only by a factor $(1+O(\varepsilon))$. This part is described in detail in Section 2.3.2. We then want to find a nearly optimal solution satisfying the above conditions using dynamic programming.

Using Arora's scheme, in particular shifting the input points onto a grid of polynomial size, requires the cost of the optimal solution to be not too small with respect to the maximum distance between a pair of input points. This might not be the case for an instance of the CLRP problem, as an optimal solution might consist of independent parts situated arbitrarily far from each other. To avoid this problem,

we first partition the set of input points $P \cup O$ into subsets $P_i \cup O_i$, for which we solve the CLRP problem independently and which satisfy:

- the optimal solution for the original problem instance consists of independent solutions for the smaller problem instances,
- for each problem instance defined by $P_i \cup O_i$ the cost of an optimal solution is not too small compared to the maximum distance between any pair of input points.

Creating such a partition is a standard step in the approximation schemes for geometric optimization problems based on Arora's scheme, where the solution does not have to be connected (as for the Steiner forest problem [26]) or does not have to contain all the input points (as in the case of k-TSP and k-MST [10]). Our approach resembles the one by Borradaile et al. [26] and is described in detail in Section 2.3.1. To approximate the cost of an optimal solution, we use the constant factor approximation algorithm by Harks et al. [53], which is described in Section 2.2.

The dynamic program is very similar as for the capacitated vehicle routing problem [42]. As we want the algorithm to run in quasi-polynomial time, we cannot afford to remember the exact number of points from P visited by each tour inside a given dissection square. Instead, we round down the number of points from P visited by a tour to one of the logarithmic number of thresholds, keeping only a small number of tours with unrounded number of points. As a result of the dynamic program we get a solution with a cost not much greater than the cost of an optimal solution, but with tours which visit slightly more than k points from P. The dynamic program is described in detail in Section 2.3.3.

The last part of the algorithm, described in Section 2.3.4, deals with the excessive points from the tours. We remove the excessive points from the tours in such a way, that covering them with additional tours (using a constant factor approximation algorithm) does not generate a large cost compared to the cost of an optimal solution. That part resembles the idea from [42]. However, the analysis has to be modified, as the lower bounds on the cost of an optimal solution for the CVRP problem do not hold for the CLRP problem. Instead, we use the lower bounds introduced in Section 2.2. In the CLRP problem we have one additional difficulty — we have to deal with the opening costs for the depots. However, we show that we can restrict the set of depots used by the additional tours to the ones already opened by the solution output by the dynamic program, and still get a small additional cost compared to the cost of an optimal solution.

To simplify the analysis, we present a $(1+O(\varepsilon))$ approximation algorithm. To obtain a $(1+\varepsilon)$ approximation algorithm it is enough to divide ε by an appropriate constant.

2.3.1 Dividing into Subproblems

Lemma 2.11. *Let $I = (P, O, \delta, \phi)$ be an instance of the CLRP problem. In polynomial time we can partition the set of input points $P \cup O$ into sets $P_i \cup O_i$ such that every optimal solution consists of the disjoint solutions for the problem instances $I_i = (P_i, O_i, \delta|_{(P_i \cup O_i)^2}, \phi|_{O_i})$, and the cost of an optimal solution for any instance I_i is at least $\frac{\ell_i}{(4+\varepsilon)|P_i \cup O_i|}$, where ℓ_i is the side length of the smallest square containing all points from $P_i \cup O_i$.*

Proof. Let $\mathrm{ap}(I)$ be the cost of the solution found by the $(4+\varepsilon)$ approximation algorithm for the CLRP problem ([53], presented in Section 2.2) for the instance I. We have

$$OPT(I) \leq \mathrm{ap}(I) \leq (4+\varepsilon) \cdot OPT(I) \ .$$

If the distance between two input points is more than $\mathrm{ap}(I)$, in an optimal solution the points have to be in different connected components. We create a graph on the vertex set $P \cup O$ by connecting each pair of vertices with an edge if the distance between them is at most $\mathrm{ap}(I)$. We divide the set $P \cup O$ into subsets, each containing vertices from one connected component of the graph, and create a separate problem instance $I_i = (P_i, O_i, \delta|_{(P_i \cup O_i)^2}, \phi|_{O_i})$ for each subset $P_i \cup O_i$.

As long as the above operation returns more than one problem instance, we repeat it with each instance as the input. At some point the operation ends and we get a collection of problem instances $I_1, \ldots, I_j$, representing a partition of the set $P \cup O$ into subsets $P_i \cup O_i$. By the construction of the instances I_i we know that any optimal solution for I consists of disjoint solutions for each I_i. All the points from $P_i \cup O_i$ are contained in a square of length

$$\ell_i \leq |P_i \cup O_i| \cdot \mathrm{ap}(I_i) \leq |P_i \cup O_i| \cdot (4+\varepsilon) \cdot OPT(I_i) \ .$$

That gives the required lower bound on the cost of an optimal solution for each instance I_i. □

We divide the original problem instance according to Lemma 2.11. An example can be seen in Figure 2.4. We can now assume that all the input points are inside a square $\ell \times \ell$ and the cost of an optimal solution is at least $\frac{\ell}{(4+\varepsilon)n}$.

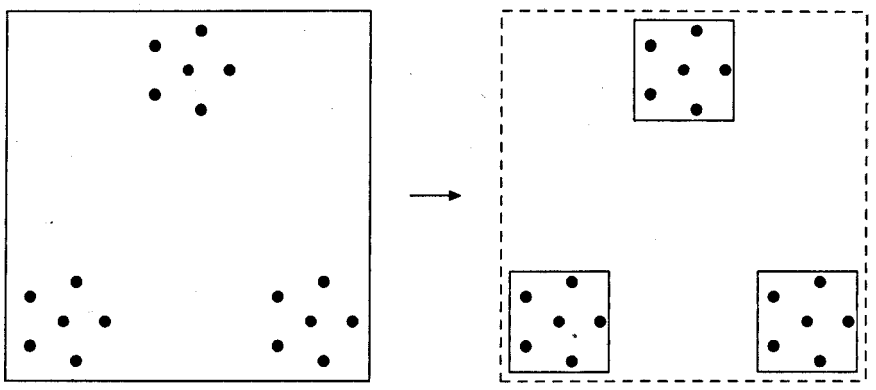

Figure 2.4: On the left is an instance of the CLRP problem, where the points from the sets P and O are denoted by black and grey dots respectively. On the right is its partition into independent subproblems, as described in Lemma 2.11.

2.3.2 Perturbation, Randomized Dissection and Portals

Perturbation. We show that we can modify the problem instance by moving all the input points onto a grid of size polynomial in n. To prove the following lemma we will use the result from Lemma 2.11.

Lemma 2.12. *Let $I = (P, O, \delta, \phi)$ be an instance of the CLRP problem, and S a solution for I. Moving each point from the set $P \cup O$ by a distance at most $\frac{\varepsilon \ell}{4(4+\varepsilon)n^2}$ changes the cost of S by at most $\varepsilon \cdot OPT(I)$.*

Proof. In a solution to the CLRP problem each point from P is visited by exactly one tour. Therefore the number of edges in S is at most $2n_P < 2n$. Moving each input point by a distance at most $\frac{\varepsilon \ell}{4(4+\varepsilon)n^2}$ changes the length of a single edge by at most $2 \cdot \frac{\varepsilon \ell}{4(4+\varepsilon)n^2}$, and so it changes the total length of the edges by at most $4n \cdot \frac{\varepsilon \ell}{4(4+\varepsilon)n^2} = \varepsilon \cdot \frac{\ell}{(4+\varepsilon)n} \leq \varepsilon \cdot OPT(I)$. □

Let L be the smallest power of 2 such that $\frac{L}{2} \geq \frac{4(4+\varepsilon)n^2}{\varepsilon}$. We divide the $\ell \times \ell$ square containing all the input points into a grid of $\frac{L}{2} \times \frac{L}{2}$ *unit squares* and move each input point to the nearest centre of a unit square (see Figure 2.5). Ties are resolved arbitrarily. Notice that multiple points can be moved into the same position.

From Lemma 2.12 we immediately get

Lemma 2.13. *Let $I = (P, O, \delta, \phi)$ be an instance of the CLRP problem, and S a solution for I. Moving each point from the set $P \cup O$ to the nearest centre of a unit square of the $\frac{L}{2} \times \frac{L}{2}$ grid changes the cost of S by at most $\varepsilon \cdot OPT(I)$.*

We can now assume that all the input points are in the centres of the grid squares of a $\frac{L}{2} \times \frac{L}{2}$ grid for $L = \Theta(\frac{n^2}{\varepsilon})$.

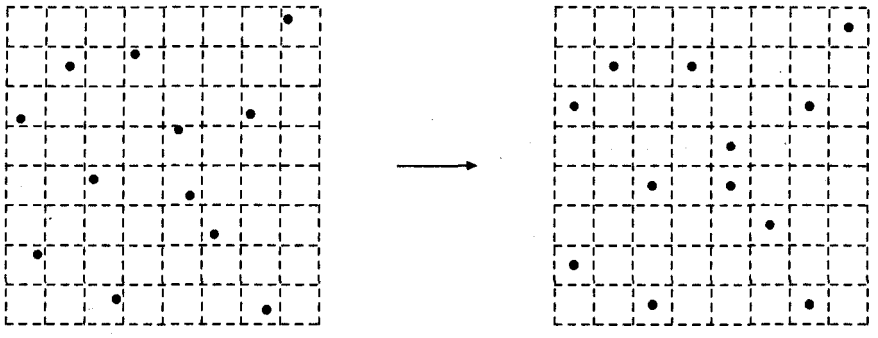

Figure 2.5: Perturbation performed on an instance of the CLRP problem. The points from the sets P and O are denoted by black and grey dots respectively. The dashed lines represent the grid lines.

Randomized dissection. We put the $\frac{L}{2} \times \frac{L}{2}$ grid inside a $L \times L$ grid with a random shift — the grid is shifted horizontally and vertically by a number of unit squares chosen independently and uniformly at random from the set $\{0, 1, \ldots, L-1\}$. We perform a *dissection* of the $L \times L$ grid — a recursive partitioning of a square into four *dissection squares*, performed until we obtain the unit squares (see Figure 2.6). Here we use the fact that L is a power of 2.

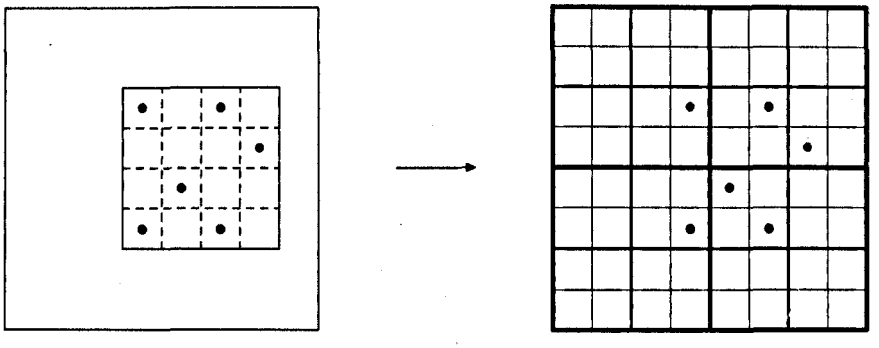

Figure 2.6: Randomized dissection — a 4×4 grid is shifted randomly inside a 8×8 grid; the grid is then recursively divided into unit squares. The width indicates the level of the grid lines. The points from the sets P and O are denoted by black and grey dots respectively.

Number of *levels* of the dissection is $\log L = O(\log \frac{n}{\epsilon}) = O(\log n)$. A dissection square has level i if it has been created during the i-th iteration of the dissection. The original $L \times L$ square has level 0, and the unit squares have level $\log L$. In a similar way we assign levels to the grid lines. A grid line has level i, if it is a boundary for level i dissection squares, but not for level $i-1$ dissection squares. The probability over the random shift in the randomized dissection that a particular grid line of the $\frac{L}{2} \times \frac{L}{2}$ square becomes a level i line for $i = 1, \ldots, \log L$ is $\frac{2^{i-1}}{L}$.

Portals and portal-respecting tours. Let m be the smallest power of 2 such that $m \geq \frac{2}{\epsilon} \cdot \log L$. On the boundary of each dissection square we put $m+1$ equidistant points called *portals*, in such a way that the corners of the dissection squares are covered by the portals. We choose m to be a power of 2 so that the portals of a level i dissection square are also portals of the level $i+1$ dissection squares contained in it.

A tour is called *portal-respecting* if it crosses the boundaries between dissection squares only in portals. A solution for the CLRP problem is called *portal-respecting* if it consists only of portal-respecting tours. We can transform any solution to a portal-respecting one as in [10], by creating *detours*. Instead of crossing a boundary of a square outside of the portals, the tour goes along the boundary of the square to the nearest portal (see Figure 2.7).

Notice that as the additional part of the tour goes along a boundary of a dissection square, it can cross the boundaries of other dissection squares (i.e. the squares which are contained in the original square), but only in the corners of the squares. Therefore by creating detours we do not introduce any crossings outside of the portals.

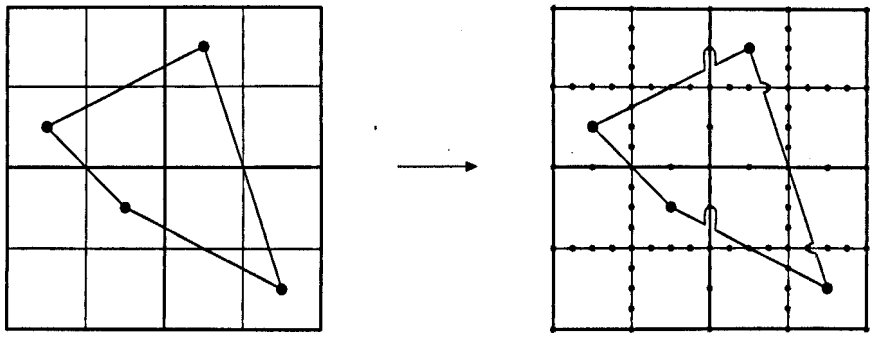

Figure 2.7: Making a tour portal-respecting. Portals are denoted by light-grey dots. Here $m = 4$.

Light tours. We call a tour *r-light* if it crosses each side of each dissection square at most r times. A solution is r-light if it consists only of r-light tours. We set $r = \frac{18}{\epsilon} + 5 = \Theta(\frac{1}{\epsilon})$.

We can modify a solution to be r-light using *patching*, which has been used by Karp in the analysis of his dissection heuristic [60], and is made explicit in [10]. We use the following lemma.

Lemma 2.14 (Patching lemma, [10]). *Let S be any line segment and T a tour that crosses S at least three times. By adding line segments on S of total length at most*

$6 \cdot |S|$, *where* $|S|$ *denotes the length of the segment* S, *we can change* T *into a tour* T' *that crosses* S *at most two times.*

Any point on S *belongs to at most* 6 *of the added segments.*

Proof. Let $s_1, \ldots, s_t$ (where $t \geq 3$) be the points where T crosses the line segment S. We substitute each s_i with two points, s_i^1 and s_i^2, one at each side of S. We cut T in the places where it crosses S (i.e. we remove the edges $s_i^1 s_i^2$), so that it falls apart into t paths connecting the points $\{s_1^1, s_1^2, \ldots, s_t^1, s_t^2\}$.

We create a graph G on the vertex set $\{s_1^1, s_1^2, \ldots, s_t^1, s_t^2\}$ in the following way. We add to G t edges representing the t paths obtained from T. For $i = 1, 2$ we add to G a TSP tour on $\{s_1^i, \ldots, s_t^i\}$ (e.g. a cycle $s_1^i - s_2^i - \ldots - s_t^i - s_1^i$). If t is even, we add to G the edges $s_{t-1}^1 s_{t-1}^2$ and $s_t^1 s_t^2$, and a minimum cost matching on $\{s_1^i, \ldots, s_{t-2}^i\}$ for $i = 1, 2$. If t is odd, we add to G the edge $s_t^1 s_t^2$, and a minimum cost matching on $\{s_1^i, \ldots, s_{t-1}^i\}$ for $i = 1, 2$.

Graph G is connected and 4-regular. Tour T' is an eulerian cycle on G, which crosses S in one or two places, depending on the parity of t. The total length of the added segments is at most $6|S|$ ($2|S|$ for each TSP tour and $|S|$ for each matching). Any point on S belongs to at most 6 of the added segments. □

The value $6 \cdot |S|$ in the statement of Lemma 2.14 is not optimal — it can be substituted with $3 \cdot |S|$, with a more complicated proof (see [10]).

Let T be a tour which we want to make r-light. First we modify T in such a way, that it does not cross two grid lines, a horizontal and a vertical one, at once at an intersection of a horizontal and a vertical line. For every such crossing we decide arbitrarily which of the lines is crossed first. Then, when we consider a level j line and level $i \geq j$ *segments* of the line, corresponding to the sides of the level i dissection squares, each crossing belongs to one segment. Also observe that if a tour crosses a segment in the same point at least three times, we can modify it, without introducing any new crossings, to reduce the number of crossing in that point to at most two.

Now we will present an algorithm from [10] that transforms a tour T into an r-light tour with a small expected increase in $|T|$. For each dissection level i from $\log L$ to 1 (we can ignore level 0, as we consider only tours that are contained in the bounding box) we perform the following operation. We consider all grid lines of level $j \leq i$, and all segments of these lines corresponding to the sides of level i dissection squares. For each such segment S we check the number of times the tour T crosses S outside of the corner portals. If the number of crossings is more than $r - 4$, we apply patching (Lemma 2.14) for the segment S.

Notice that adding new segments when applying patching at the i-th level of dissection can introduce new crossings, but only at level $\ell > i$ lines, in the corner portals of the dissection squares. It does not introduce any crossings which could have any influence on patching at the levels $j \leq i$. It also does not destroy the property of each level $\ell > i$ segment having at most $r - 4$ crossings outside the corner portals.

We then consider all level $\log L$ segments, and check the number of times the tour crosses the segment in each of the corner portals. Whenever the number of crossings in a portal is at least 3, we reduce it to at most 2. (We do not have to consider the level $j < \log L$ corner portals, as they are also level $\log L$ corner portals.) After this operation all the tours are r-light.

Light, portal-respecting solutions. Following [42], we introduce a new cost function C', that additionally to the cost C of the edges and of opening the facilities penalizes a solution by $\frac{L}{m2^i}$ for each crossing of a boundary of a level i dissection square. We need this technical detail in the proof of Lemma 2.23.

Definition 2.15. *Let $I = (P, O, \delta, \phi)$ be an instance of the CLRP problem, and $S = (D, \mathcal{T})$ a solution for it. We define the* extended cost of S *as*

$$C'(S) = \sum_{d \in D} \phi(d) + \sum_{T \in \mathcal{T}} |T| + \sum_{i=1}^{\log L} \frac{L}{m2^i} \pi(i) \ ,$$

where $\pi(i)$ denotes the number of times the tours from S cross the boundary of level i dissection squares.

Notice that the extended cost depends on the random shift in the dissection of the input square chosen at the beginning of the algorithm. We can show the following result.

Lemma 2.16. *Let S be a solution for an instance of the CLRP problem. Then*

$$E[C'(S)] \leq (1 + \varepsilon) C(S) \ .$$

Proof. We scale the distances between the points so that the unit squares have sides of length 1. Let T be any tour from S. T crosses at most $2|T|$ grid lines. A grid line has level i with probability $\frac{2^{i-1}}{L}$. If a grid line has level i, crossing it means crossing a boundary of level $\log L, \log L - 1, \ldots, i$ dissection squares. The expected increase

in cost of T is at most

$$2|T| \cdot \sum_{i=1}^{\log L} \frac{2^{i-1}}{L} \sum_{j=i}^{\log L} \frac{L}{m2^j} < 2|T| \cdot \sum_{i=1}^{\log L} \frac{2^{i-1}}{L} \cdot \frac{L}{m2^{i-1}} = \frac{2|T| \log L}{m} \leq \varepsilon |T| \ ,$$

as $m \geq \frac{2}{\varepsilon} \cdot \log L$. Summing up over all tours gives the desired result. □

Following Arora [10], we show that modifying a solution S to be r-light and portal-respecting generates, in expectation over the random shift in the randomized dissection, only a small cost. We modify the tours in the way described earlier, first by making them r-light, and then portal-respecting. We consider the extended cost function C'.

Lemma 2.17. *Let S be a solution for an instance of the CLRP problem, and S' an r-light solution created from it. Then*

$$E[C'(S')] \leq C'(S) + \varepsilon \cdot C(S) \ .$$

Proof. We will show that the expected increase in the extended cost of a single tour T from S is small. Let ℓ be a horizontal grid line. We fix the vertical shift in the randomized dissection, to fix the level i segments of ℓ for all the dissection levels i. We want to compute the expected increase in the extended cost of T due to patching the segments of ℓ. We scale the distances between the points so that the unit squares have sides of length 1.

For each dissection level i we set $c(\ell, i)$ to be the number of times that patching would be applied to the segments of ℓ at the i-th level of dissection, assuming that the level of ℓ is at most i. Let $t(\ell)$ be the number of times T crosses ℓ. As each patching operation decreases the number of crossings on a segment of ℓ by at least $r - 5$, we have $\sum_{i=1}^{\log L} c(\ell, i) \leq \frac{t(\ell)}{r-5}$.

We now upper bound the additional cost for crossing the boundaries of the dissection squares generated by the parts of the tour introduced during patching. Crossing a grid line of level j generates cost $\sum_{l=j}^{\log L} \frac{L}{m2^l} < \frac{L}{m2^{j-1}}$. Patching a level i segment generates additional crossings with level $j > i$ lines. A segment of level i crosses 2^{j-i-1} lines of level j for $i < j \leq \log L$. From Lemma 2.14 each line is crossed at most 6 times. The additional cost generated by one patching operation at level i is at most

$$\sum_{j=i+1}^{\log L} 2^{j-i-1} \cdot 6 \cdot \frac{L}{m2^{j-1}} = \sum_{j=i+1}^{\log L} \frac{6L}{m2^i} \leq \frac{6L \log L}{m2^i} \leq 3\varepsilon \frac{L}{2^i} \ ,$$

as $m \geq \frac{2}{\varepsilon} \cdot \log L$.

The length of a level i segment is $\frac{L}{2^i}$ and from Lemma 2.14 we get that patching a level i segment increases the length of a tour by at most $6 \cdot \frac{L}{2^i}$. Probability that a line has level at most i is $\frac{2^i}{L}$. The expected increase in the extended cost of T due to patching the segments of ℓ is at most

$$\sum_{i=1}^{\log L} \frac{2^i}{L} \cdot (6+3\varepsilon)\frac{L}{2^i} \cdot c(\ell,i) \leq 9 \sum_{i=1}^{\log L} c(\ell,i) \leq \frac{9}{r-5} t(\ell) \ .$$

We can perform the same computation for vertical grid lines. As the unit squares have sides of length 1, we have $\sum_{\ell-\text{grid line}} t(\ell) \leq 2|T|$, we get that the expected increase in the extended cost of S due to making a tour T r-light is at most

$$\frac{9}{r-5} \sum_{\ell-\text{grid line}} t(\ell) \leq \frac{18}{r-5} \cdot |T| = \varepsilon |T| \ ,$$

as $r = \frac{18}{\varepsilon} + 5$.

Summing up over all tours T from S gives the desired result. □

We now transform the r-light solution into a portal-respecting one by creating detours, as described earlier. The detours can create crossings in the corner portals of the dissection squares. In each corner we can then decrease the number of crossings to at most 2 without increasing the cost. The resulting solution is both r-light and portal-respecting.

Lemma 2.18. *Let S be a solution to the CLRP problem, and S' the r-light solution obtained from S using patching. The expected increase in the extended cost of S' while making it both r-light and portal-respecting is at most $\varepsilon \cdot C(S)$.*

Proof. All the additional crossings created while making the tours from S r-light are in the portals. The only crossings of S' that might be outside of the portals are therefore the original crossings of S which have not been removed during patching.

We scale the distances between the points so that the unit squares have sides of length 1. Let $c(\ell)$ be the number of times S crosses a grid line ℓ. If ℓ has level i, the distance between two consecutive portals on ℓ is $\frac{L}{2^i m}$, and making a detour for a single crossing increases the length of the solution by at most $\frac{L}{2^i m}$.

Making a detour at a level i line introduces new crossings, all in the corners of level $j > i + \log m$ dissection squares, as at the intersection of a level i line with a level j line for $i < j \leq i + \log m$ there is a portal. For $j > i + \log m$ the number of lines of level j between two adjacent portals of a level i line is $2^{j-i-\log m-1}$. The

cost charged for crossing them is

$$\sum_{j=i+\log m+1}^{\log L} 2^{j-i-\log m-1} \cdot \frac{L}{m2^{j-1}} = \sum_{j=i+\log m+1}^{\log L} \frac{L}{m^2 2^i} \leq \frac{\log L \cdot L}{m^2 2^i} \leq \frac{\varepsilon}{2} \cdot \frac{L}{m2^i} ,$$

as $m \geq \frac{2}{\varepsilon} \cdot \log L$.

As a line ℓ has level i with probability $\frac{2^{i-1}}{L}$, the expected increase in the extended cost of S' is at most

$$\sum_{\ell-\text{grid line}} \sum_{i=1}^{\log L} \frac{2^{i-1}}{L} \cdot c(\ell) \cdot (1+\frac{\varepsilon}{2}) \frac{L}{m2^i} \leq \frac{\log L}{m} \sum_{\ell-\text{grid line}} c(\ell) \leq \frac{2\log L}{m} C(S) \leq \varepsilon \cdot C(S) .$$

□

We can now prove the following result.

Theorem 2.19. *Let $I = (P, O, \delta, \phi)$ be an instance of the CLRP problem. Let S be an r-light, portal-respecting solution for I minimizing the extended cost function $C'(S)$. Then*

$$E[C'(S)] \leq (1 + 3\varepsilon) \cdot OPT(I) .$$

Proof. Let S_1 be an optimal solution for I. From Lemma 2.16

$$E[C'(S_1)] \leq (1 + \varepsilon) \cdot C(S_1) = (1 + \varepsilon) \cdot OPT(I) .$$

Let S_2 be an r-light solution obtained from S_1 using patching. From Lemma 2.17

$$E[C'(S_2)] \leq E[C'(S_1)] + \varepsilon \cdot C(S_1) \leq (1 + 2\varepsilon) \cdot OPT(I) .$$

Let S_3 be an r-light portal-respecting solution created from S_2 using detours. From Lemma 2.18

$$E[C'(S_3)] \leq E[C'(S_2)] + \varepsilon \cdot C(S_1) \leq (1 + 3\varepsilon) \cdot OPT(I) .$$

We instantly get the theorem. □

In the following part we consider only tours that are r-light and portal-respecting, and use the extended cost function C' instead of C.

2.3.3 Dynamic Program with a (Slightly) Infeasible Output

We want to find a (nearly) optimal r-light, portal-respecting solution using dynamic programming. To compute the cost we use the extended cost function C'. As we want the algorithm to run in quasi-polynomial time, we cannot remember the exact number of points on each tour in the dynamic programming tables. Instead, we define a set of thresholds, and at each level of dissection we round down the remembered number of points visited by each tour within a dissection square to the nearest threshold value, remembering the unrounded number of points only for a small number of tours.

The output generated by the dynamic program will not be a feasible solution — because of the rounding it can have tours visiting more than k points from the set P. However, the set of thresholds is chosen in such a way, that the number of excessive points in a tour cannot be large. The extended cost of a solution generated by the dynamic program is not greater than the cost of an optimal, with respect to the extended cost function C', feasible r-light, portal-respecting solution, and from Theorem 2.19 we get that the expected cost of such a solution is at most $(1+3\varepsilon)\cdot OPT$.

We define the set $\mathcal{X}$ of *thresholds* in $[1,k]$ in the following way.

$$\mathcal{X} = \left\{0, 1, 2, \ldots, \frac{1}{\varepsilon}, \frac{1}{\varepsilon}\left(1+\frac{\varepsilon}{\log n}\right), \ldots, \frac{1}{\varepsilon}\left(1+\frac{\varepsilon}{\log n}\right)^i, \ldots\right\}$$

The number of thresholds equals

$$|\mathcal{X}| = 1 + \frac{1}{\varepsilon} + \lfloor \log_{1+\varepsilon/\log n}(k\varepsilon) \rfloor = O\left(\frac{1+\log(k\varepsilon)\log n}{\varepsilon}\right) .$$

Later we will have to transform the solution output by the dynamic program into a feasible solution. That will require removing the excessive points from the tours, and covering them by newly created tours. The operation is described in detail in Section 2.3.4. As we want the cost of the additional tours to be small, we want to have either no excessive points in a dissection square, or have many of them at once. Therefore we only round the remembered number of points visited by the tours inside a given dissection square in large groups.

We set $\gamma = \lceil \frac{\log^4 n}{\varepsilon^4} \rceil$. In each dissection square we will keep the unrounded information about at most γ tours. Whenever the number of unrounded tours exceeds γ, we will round down the remembered number of points for all the tours to the closest value in $\mathcal{X}$.

Now we are ready to describe the dynamic program.

Interface of a dissection square. The interface of each dissection square consists of the information about all the tours visiting the square. Each tour can contain at most $2r$ paths (i.e. fragments of tours, starting and ending in the portals) from a given dissection square, as it crosses each side of the square at most r times.

In a dissection square there are at most γ unrounded tours. For each of them we remember the number of points from P visited by the tour, the pairs of portals where the tour enters and leaves the square, plus the information whether the tour visits an open depot within the square. The number of possibilities is $(k \cdot m^r)^{O(\gamma)} = n^{O(\log n/\varepsilon)^{O(1)}}$.

For a set of at most $2r$ pairs of portals and a threshold $x \in \mathcal{X}$ there can be some rounded tours entering and leaving the square in the given portals, visiting the given number of points from P (after rounding) and passing, or not passing, through an open depot. In each square we can therefore have $t = m^{O(r)} \cdot |\mathcal{X}| = \log^{O(1/\varepsilon)} n$ different *types* of rounded tours.

A solution to the CLRP problem consists of at most n tours. Each rounded tour can have one of the t types. The interface consists of the information about the unrounded tours, and the number of rounded tours of each type. The number of possible interfaces is $n^{O(\log n/\varepsilon)^{O(1)}} \cdot n^{\log^{O(1/\varepsilon)} n} = n^{\log^{O(1/\varepsilon)} n}$, which is quasi-polynomial in n.

Computing the solution: leaves. For each unit square and an interface we check if the interface is feasible for the given square and if so, compute the minimum cost of a solution satisfying it. All the points from $P \cup O$ that are inside the square are in the same position — in the centre of the square.

We check the feasibility of an interface in the following way. From the interface we reconstruct the number of tours of each type inside the dissection square. If some tour from the interface requires an open depot, we check if there is a point from O in the unit square. We then check if the number of points from P visited by the tours equals the number of points from P in the square (possibly after rounding; if the rounding took place, we check that at least γ tours have been rounded and that there are no unrounded tours left).

If the interface is feasible, we compute its cost. If some tour from the interface requires an open depot, we open the least expensive depot available. Otherwise, we do not open any depot inside the square. We then compute the cost of all tours, where the tours visiting at least one point from $P \cup O$ have one path passing through

the centre of the square, and all the other paths are intervals connecting the portals. At this stage we do not add the cost for crossing the boundaries of the dissection squares.

The total time of computing the feasibility and the costs of the interfaces of the unit squares is $n^{\log^{O(1/\varepsilon)} n}$, which is quasi-polynomial in n.

Computing the solution: non-leaves. For each dissection level $i < \log L$ and all dissection squares of level i we do the following operation. We take all combinations of feasible interfaces of the four level $i+1$ dissection squares contained in the level i square, and check which interfaces of the level i square can be obtained from them. For each interface we remember the cost of the cheapest solution yielding that interface.

From each of the level $i+1$ interfaces we get information about the tours visiting it — the number of rounded tours with each of the t types, and the information about the at most γ unrounded tours. A tour in a level i square is obtained from at most four parts, each from a different level $i+1$ square. The number of possibilities for obtaining a tour is therefore at most $(t+\gamma)^4 = \log^{O(1/\varepsilon)} n$.

As each tour can have at most $2r$ paths in a level $i+1$ dissection square, there are at most $r^{O(r)} = (\frac{1}{\varepsilon})^{O(1/\varepsilon)}$ possibilities of putting them together to obtain a level i tour. Such tour can consist of at most $2r$ paths, or be a complete tour within the level i square. We consider each possibility and check if the endpoints of the consecutive parts of a path or tour are in the same portal and whether all complete tours contain an open depot. We also check that all tours are r-light, and that the number of points visited by them does not exceed k.

As there are at most $(t+\gamma)^4 \cdot r^{O(r)} = \log^{O(1/\varepsilon)} n$ possible tour types at level i, they can generate at most $n^{\log^{O(1/\varepsilon)} n}$ interfaces of the level i square. We check which of these interfaces can be obtained, possibly after rounding all the tours.

The cost of a level i interface equals the sum of the costs of the level $i+1$ interfaces from which it has been created, plus the cost of crossing the boundaries of the level $i+1, i+2, \ldots, \log L$ dissection squares in the level $i+1$ portals.

For each dissection square the operation takes time $n^{\log^{O(1/\varepsilon)} n}$, which is quasi-polynomial in n.

Output of the dynamic program. The algorithm outputs the least expensive solution for the $L \times L$ square with an empty interface. The solution consists of r-light, portal-respecting tours covering all the points from P and containing open depots, but might be infeasible, as the tours can contain more than k points from P.

The running time of the algorithm is $n^{\log^{O(1/\varepsilon)} n}$, which is quasi-polynomial in n.

Lemma 2.20. *Let $I = (P, O, \delta, \phi)$ be an instance of the CLRP problem, and S a solution output for I by the dynamic program. Then*

$$E[C'(S)] \leq (1 + 3\varepsilon) \cdot OPT(I) \ .$$

Proof. Let S_{min} be an r-light, portal-respecting feasible solution for I minimizing the extended cost function C'. From Theorem 2.19 we get that $E[C'(S_{min})] \leq (1 + 3\varepsilon) \cdot OPT(I)$.

We will show that $C'(S) \leq C'(S_{min})$. For each dissection square there is an interface corresponding to the solution S_{min}. As S_{min} is a feasible r-light solution satisfying that interface, for each dissection square the dynamic program finds a (possibly slightly infeasible) solution for the given interface. The cost of the dynamic program solution is not greater than the cost of S_{min} on each dissection square. The dynamic program finds a solution for the $L \times L$ square with an empty interface with an extended cost not greater than $C'(S_{min})$. We get that $E[C'(S)] \leq E[C'(S_{min})] \leq (1 + 3\varepsilon) \cdot OPT(I)$. □

Lemma 2.21. *Each tour from a solution output by the dynamic program visits at most $(1 + O(\varepsilon))k$ points from P.*

Proof. At each of the $\log L = O(\log n)$ dissection levels the remembered number of points in a tour can be rounded down, but at most by a $\frac{\varepsilon}{\log n}$ fraction. The total number of points in the tour is therefore not greater than

$$k\left(1 + \frac{\varepsilon}{\log n}\right)^{O(\log n)} = ke^{O(\varepsilon)} = k(1 + O(\varepsilon)) \ .$$

□

2.3.4 Repairing the Solution

As a result of running the dynamic program from Section 2.3.3, we get a solution where each tour visits at most $(1 + O(\varepsilon))k$ points from P. We have to transform it into a feasible solution.

We do it by removing excessive points from the tours. We remove an appropriate number of points from the tours, for which the dynamic program rounded down the remembered number of points. Then, using a constant factor approximation algorithm, we construct additional tours covering the removed points.

We have to ensure that the expected cost of the additional tours is at most $O(\varepsilon) \cdot OPT$. To show that the removed points can be covered with tours of a small cost, we will use the property, that whenever the remembered number of points on a tour has been rounded down, it has been done for at least γ tours from a dissection square at once. Here we consider the cost C of the additional tours, and not the extended cost C'.

Choosing the points to be removed from the tours. Let S be a solution output by the dynamic program. For all unit squares, and then for all other dissection squares in the bottom-up fashion, we reconstruct the interfaces of the dissection squares for S. Whenever the remembered number of points visited by a tour T inside a dissection square Γ is rounded down by some integer x, we remove x points from the tour T in Γ. After performing this operation, the number of points on each tour is at most k, and at most an $O(\varepsilon)$ fraction of points have been removed from the tours.

We choose the x points to be removed from T in the dissection square Γ as follows. We choose a point $p \in P$ uniformly at random from all the points visited by T in Γ, and we drop x consecutive points from T that are within Γ, starting at p. If we encounter the end of T, we continue dropping points from the start of the tour. See Figure 2.8 for an example.

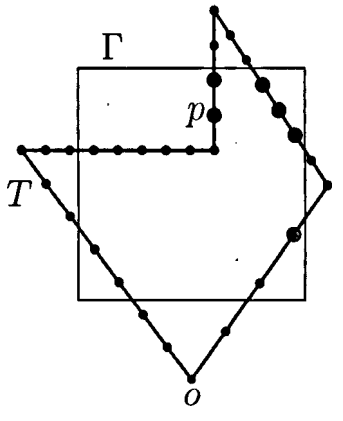

Figure 2.8: Removing 6 points from a tour T within a dissection square Γ. The point p has been chosen randomly from all points visited by T in Γ, and 6 consecutive points have been removed, starting from p. The removed points are represented by large grey dots.

Properties of the removed points. We can show that the following holds.

Lemma 2.22. *Let $I = (P, O, \delta, \phi)$ be an instance of the CLRP problem, and $S =$*

$(D, \mathcal{T})$ *the dynamic program solution for* I. *Let* R *be the set of excessive points removed from the tours* $\mathcal{T}$. *Then*

$$E\left[\frac{2}{k}\sum_{p\in R}\delta(p,D)\right] = O(\varepsilon)\cdot OPT(I) \ .$$

Proof. Let p be an arbitrary point from P. At each level of the dissection p is removed from its tour with probability at most $\frac{\varepsilon}{\log n}$. As the number of dissection levels is $O(\log n)$, the probability that $p \in R$ is $O(\varepsilon)$.

As each tour in S visits at most $k(1+O(\varepsilon))$ points, from Lemma 2.4 we get that $C(S) \geq \frac{2}{(1+O(\varepsilon))k}\sum_{p\in P}\delta(p,D)$. On the other hand, from Lemma 2.20 we get that $E[C(S)] \leq (1+3\varepsilon)\cdot OPT(I)$. We get

$$E\left[\frac{2}{k}\sum_{p\in R}\delta(p,D)\right] = O(\varepsilon)\cdot E\left[\frac{2}{k}\sum_{p\in P}\delta(p,D)\right] \leq O(\varepsilon)\cdot E[C(S)] \leq O(\varepsilon)\cdot OPT(I) \ .$$

□

Lemma 2.23. *Let* $I = (P, O, \delta, \phi)$ *be an instance of the CLRP problem,* $S = (D, \mathcal{T})$ *the dynamic program solution for* I, *and* R *the set of excessive points removed from the tours* $\mathcal{T}$. *Let* G *be a weighted complete graph on* $R \cup D$, *where the weights represent the distances* δ *between the points, with the distances within* D *set to* 0. *Then*

$$E[MST(G)] = O(\varepsilon)\cdot OPT(I) \ .$$

Proof. We consider separately points from R removed from the tours when processing dissection squares of different levels. Let $R_i \subseteq R$ be the set of points removed at the i-th dissection level.

Let G_i be the subgraph of G induced by $R_i \cup D$. A spanning tree of G can be constructed from the spanning trees of all G_i. We will show that there is a spanning tree of G_i with expected cost $O(\frac{\varepsilon}{\log n})\cdot C'(S)$. As there are $O(\log n)$ dissection levels, and from Lemma 2.20 we get $E[C'(S)] \leq (1+3\varepsilon)\cdot OPT(I)$, the lemma follows.

We partition the set R_i into sets $R_i^1, R_i^2, \ldots, R_i^\ell$ of points from different level i dissection squares $\Gamma_i^1, \Gamma_i^2, \ldots, \Gamma_i^\ell$. Each set $R_i^j \subseteq R_i$ contains points from at least γ tours, as rounding is always performed for large groups of tours.

We construct a spanning tree of G_i in the following way. For each R_i^j we construct a spanning tree T_i^j of all the points R_i^j. Edges constructed in this part are called the *inner* edges. We then connect the trees T_i^j into a spanning tree of G_i with the *outer* edges. The inner and outer edges create a spanning tree of G_i.

The inner edges. We want to show that for each level i dissection square Γ_i^j we can construct a spanning tree of the set of points R_i^j with a small cost compared to the extended cost of the CLRP solution S within Γ_i^j, which we denote by $C'(S, \Gamma_i^j)$. The points in R_i^j have been removed from at least γ tours. The points removed from a single tour T within Γ_i^j have been chosen in such a way, that they were consecutive points on T within Γ_i^j. They were also at most a $\frac{\varepsilon}{\log n}$ fraction of all points from T in Γ_i^j.

We construct a spanning tree of R_i^j in two steps. First we connect each pair of points with an edge, if they were two consecutive points on some tour T. The expected length of these edges is at most $\frac{\varepsilon}{\log n} \cdot C'(S, \Gamma_i^j)$, as each edge from S which is inside Γ_j has been added with probability at most $\frac{\varepsilon}{\log n}$. The added edges form a collection of paths, i.e. a forest, on R_i^j (see Figure 2.9).

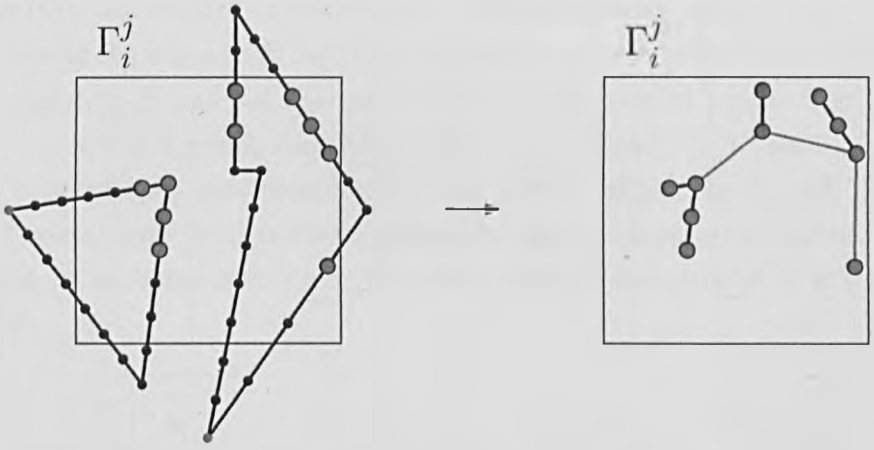

Figure 2.9: Constructing a spanning tree of R_i^j. The points from R_i^j are denoted by large grey dots. The consecutive points removed from the tours are connected into paths (black edges in the right figure). Then the connected components are joined into a spanning tree of R_i^j (grey edges).

In the second step we connect the forest into a spanning tree of R_i^j. Let α be the number of connected components. We have $\alpha \geq \gamma$, and we know that S consists of $\Omega(\alpha)$ paths within Γ_i^j. We choose one arbitrary point as a representative of each connected component and find a minimum spanning tree of these points. To upper bound the cost of the added edges we use the following lemma (see e.g. [52]).

Lemma 2.24 ([45]). *Let P be a set of n points lying inside a square of side length L. There is a path connecting the points from P of length $O(L \cdot \sqrt{n})$.*

As the side length of the square Γ_i^j is $\frac{L}{2^i}$, the cost of the edges added in the

second step is

$$O\left(\frac{L}{2^i}\cdot\sqrt{\alpha}\right)=O\left(\frac{m}{\sqrt{\alpha}}\right)\cdot\frac{\alpha L}{2^i m}=O\left(\frac{\varepsilon}{\log n}\right)\cdot\frac{\alpha L}{2^i m}\ ,$$

as $m=\Theta(\frac{\log n}{\varepsilon})$ and $\sqrt{\alpha}\geq\sqrt{\gamma}=\Theta(\frac{\log^2 n}{\varepsilon^2})$.

As for each crossing of a boundary of a level i dissection square the extended cost function C' charges $\frac{L}{2^i m}$, and the tours cross the boundary of the dissection square Γ_i^j $\Omega(\alpha)$ times, the total cost of the edges added to connect the trees into a spanning tree of R_i^j is $O(\frac{\varepsilon}{\log n})\cdot C'(S,\Gamma_i^j)$.

The expected cost of the inner edges at level i is $O(\frac{\varepsilon}{\log n})\cdot C'(S)$.

The outer edges. We construct a weighted multigraph M on the sets of points $\{R_i^1,R_i^2,\ldots,R_i^\ell\}\cup D$, where the weights of the edges represent the Euclidean distances between the sets of points with the weights within D set to 0.

We add the edges to M as follows. We consider each tour T from the dynamic program solution S and we connect with edges the sets of points from the sets $\{R_i^1,R_i^2,\ldots,R_i^\ell\}\cup D$, which have been visited consecutively by T (see Figure 2.10). We also connect each pair of points from D with γ edges. As in each dissection square Γ_i^j some points from at least γ different tours have been removed, the resulting multigraph M is γ-edge-connected. The total weight of the edges of M is not greater than $C(S)$.

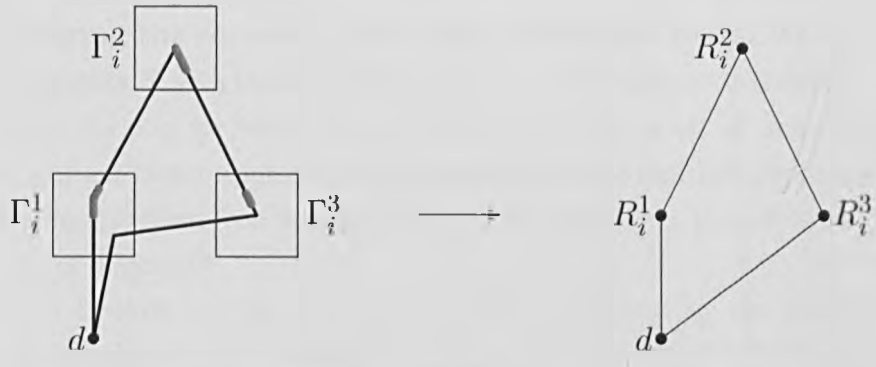

Figure 2.10: Constructing a multigraph M. The left figure shows a tour T; the fragments where points have been removed from T (i.e. the fragments where T visited points from the sets R_i^j) are denoted with a thick grey line. In the right figure are the edges of the multigraph constructed from T. Note that the weight of these edges is not greater than $|T|$.

We will show an upper bound on the cost of a minimum spanning tree of M. We will use the following theorem by Tutte [83] and Nash-Williams [72].

Theorem 2.25 ([83; 72]). *A multigraph $M = (V, E)$ contains m edge-disjoint spanning trees if and only if every partition of the vertex set V into sets $V_1, V_2, \ldots, V_\ell$ contains at least $m \cdot (\ell - 1)$ cross-edges.*

Lemma 2.26. *Let $\alpha \geq 2$ and let M be an α-edge-connected weighted multigraph with total weight $w(M)$. Then the weight of the minimum spanning tree of M is $O(\frac{w(M)}{\alpha})$.*

Proof. Let $M = (V, E)$, and let $V_1, V_2, \ldots, V_\ell$ an arbitrary partition of V. Each set V_i has at least α outgoing edges, so the number of cross-edges is at least $\frac{\alpha\ell}{2} \geq \lfloor \frac{\alpha}{2} \rfloor \ell$. From Theorem 2.25 M has $\lfloor \frac{\alpha}{2} \rfloor$ edge-disjoint spanning trees. One of these spanning trees has weight $O(\frac{w(M)}{\alpha})$. □

From Lemma 2.26 we get that the weight of the minimum spanning tree of M is $O(\frac{C(S)}{\gamma}) = O(\frac{\varepsilon}{\log n}) \cdot C(S)$.

From the minimum spanning tree T of M we generate the set of outer edges of G_i. For each pair of sets R_i^j and $R_i^{j'}$, if there is an edge in T connecting R_i^j and $R_i^{j'}$, we create an edge in G_i between the closest pair of points from R_i^j and $R_i^{j'}$. We do the same for edges between a set R_i^j and d, and the edges between d and d' for $d, d' \in D$. The set of outer edges has the same cost as the tree T, and the inner edges together with the outer edges create a spanning tree of G_i with expected cost $O(\frac{\varepsilon}{\log n}) \cdot C'(S)$. □

Covering the removed points with additional tours. We cover the set of removed points R with tours starting in the set of already opened depots D. That lets us avoid the cost for opening new facilities. To find a set of tours covering R, we use a constant factor approximation algorithm for the multiple depot capacitated vehicle routing problem (see Section 2.2), giving as the input the set of customers R and the set of depots D.

From Lemma 2.9 the cost of the solution S found by the constant factor approximation algorithm is at most $2 \cdot \frac{2}{k} \sum_{p \in R} \delta(p, D) + 2 \cdot MST(G)$, where G is a weighted complete graph on the set of vertices $R \cup D$, with weights representing the distances between the points, with the distances within D set to 0. From Lemma 2.22 and Lemma 2.23 the expected cost of S is $O(\varepsilon) \cdot OPT$.

We get the following result.

Lemma 2.27. *We can transform the solution returned by the dynamic program into a feasible solution for the CLRP problem with an additional expected cost $O(\varepsilon) \cdot OPT$.*

As the dynamic program runs in time $n^{\log^{O(1/\varepsilon)} n}$ and we can divide ε by an appropriate constant, Lemma 2.20 and Lemma 2.27 give Theorem 2.10.

2.3.5 Extension to the CLRP Problem with Non-Fixed Return

A solution for the CLRP problem is also a solution for the CLRP problem with non-fixed return. Moreover, the cost of an optimal solution for the CLRP problem with non-fixed return is at least half of the cost of an optimal solution for the CLRP problem. Therefore a constant factor approximation algorithm for the CLRP problem (see Section 2.2) gives constant approximation for the CLRP problem with non-fixed return.

The QPTAS for the CLRP problem with non-fixed return is very similar to the QPTAS for the CLRP problem. The dynamic program has to be slightly modified, as it has to allow paths which start in an opened depot inside a dissection square, and have only one endpoint in a portal. The modification is straightforward. The randomized dissection, as well as repairing the solution, are done in the same way as for the CLRP problem.

We get the following result

Theorem 2.28. *There is a randomized quasi-polynomial time approximation scheme for the two-dimensional Euclidean capacitated location routing problem with non-fixed return. For any $\varepsilon > 0$ the algorithm finds a solution with expected cost at most $(1+\varepsilon)OPT$ in time $n^{\log^{O(1/\varepsilon)} n}$.*

2.4 A PTAS for the Two-Dimensional Euclidean Capacitated Location Routing Problem for Moderately Large k and $|O|$

In this Section we first present an algorithm, which for any $\varepsilon > 0$ and $k \leq 2^{\log^{\mathfrak{c}(\varepsilon)} n}$, where $\mathfrak{c}(\varepsilon)$ is some constant depending on ε, gives a $(1+\varepsilon)$-approximate solution for the two-dimensional Euclidean capacitated vehicle routing problem in polynomial time. This yields a PTAS for $k \leq 2^{\log^{o(1)} n}$, and in particular for $k = \text{polylog}(n)$. The algorithm is a result of joint work with Artur Czumaj and Andrzej Lingas [5].

In Section 2.4.4 we extend the result to work for the two-dimensional Euclidean capacitated location routing problem, and therefore also for the multiple depot capacitated vehicle routing problem, where the number of possible depot locations $|O| \leq 2^{\log^{\mathfrak{c}(\varepsilon)} n}$. The same result holds for the capacitated location routing problem with non-fixed return.

In this section by CVRP we denote the two-dimensional Euclidean CVRP, unless stated otherwise. For simplicity of the presentation, we present $(1+O(\varepsilon))$-approximation algorithms; reduction to $(1+\varepsilon)$-approximation is straightforward.

2.4.1 Reducing the Number of Input Points

We present a reduction that takes as an input any instance of the capacitated vehicle routing problem on a set of n points in the Euclidean plane and reduces it to an instance of the problem with $(k/\varepsilon)^{O(1)} \cdot O(\log^2 n)$ points. Our construction uses a series of transformations that eliminate most of the input points and reduce the input problem instance to one significantly smaller.

Removing close points. Let L be the maximum distance from a point in P to the origin o, that is, $L = \max_{p \in P} r(p)$. Since $OPT \geq 2L$, we can ignore any point that is at a distance at most $\frac{L\varepsilon}{n}$ from the origin — we cover each such point p with a tour visiting only p and the origin o (see Figure 2.11), generating additional cost not greater than $n \cdot 2\frac{L\varepsilon}{n} \leq \varepsilon \cdot OPT$. Therefore, from now on, we assume that for each point $p \in P$ we have $r(p) \geq \frac{L\varepsilon}{n}$.

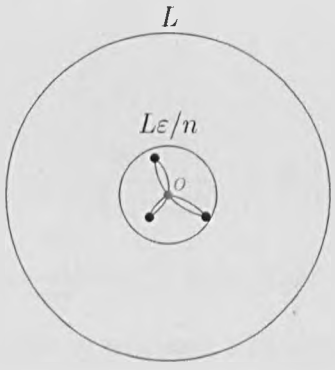

Figure 2.11: All points at a distance at most $\frac{L\varepsilon}{n}$ from the origin o (i.e. the points within the inner circle) are covered by tours visiting only one point each.

Circles, rays, and locations. Let us create *circles* around the origin, the i-th circle C_i with a radius

$$c_i = \frac{L\varepsilon}{n} \cdot \left(1 + \frac{\varepsilon}{k}\right)^i , \qquad \text{for } 0 \leq i \leq \left\lceil \log_{(1+\varepsilon/k)} \frac{n}{\varepsilon} \right\rceil .$$

Let us draw *rays* from the origin with the angle between any pair of neighboring rays equal to $\frac{2\pi}{s}$ (that is, partition the space into s sectors) with $s = \lceil \frac{2\pi k}{\varepsilon} \rceil$ (see Figure 2.12a).

Define a *location* to be any point on the plane that is the intersection of

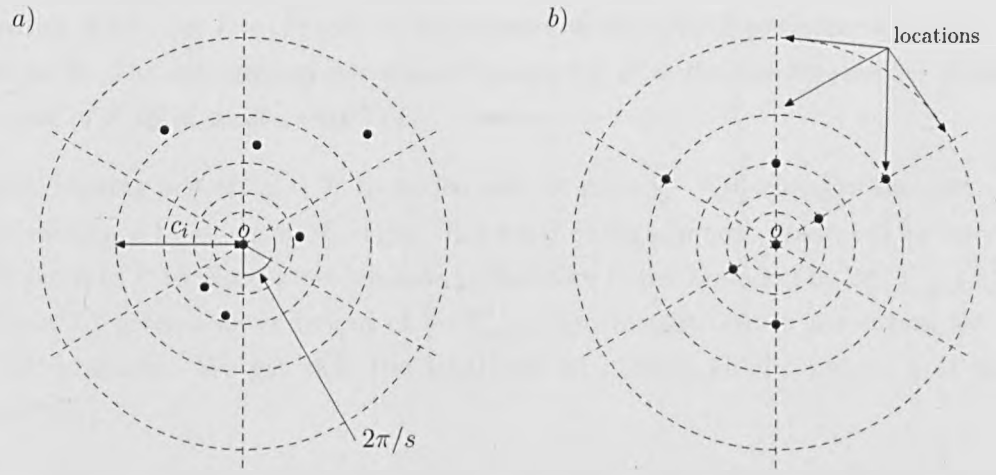

Figure 2.12: The structure of circles, rays, and locations. The point labeled o is the origin. Other fat dots represent the points from P. Figure a) presents the original position of the points from P. In Figure b) each point from P has been moved to its nearest location.

a circle and a ray. Since

$$\log_{(1+\varepsilon/k)} \frac{n}{\varepsilon} = \frac{\log \frac{n}{\varepsilon}}{\log(1+\varepsilon/k)} = \Theta\left(\frac{k}{\varepsilon} \cdot \log \frac{n}{\varepsilon}\right) ,$$

there are $\Theta\left(\frac{k}{\varepsilon} \log(n/\varepsilon)\right)$ circles and $\Theta\left(\frac{k}{\varepsilon}\right)$ rays. Therefore we obtain

Lemma 2.29. *The total number of locations t satisfies $t = \Theta(k^2\varepsilon^{-2} \log(n/\varepsilon))$.*

We can show an upper bound on a distance of any point $p \in P$ to its nearest location.

Lemma 2.30. *The distance between any $p \in P$ and a location closest to p is at most $\frac{\varepsilon}{k} \cdot r(p)$.*

Proof. Let p be a point in P. As the distance between p and the origin o is in the interval $[L\varepsilon/n, L]$, there is an index i such that p lies between the circles C_i and C_{i+1}. The distance between these circles equals $c_{i+1} - c_i = \frac{\varepsilon}{k} \cdot c_i$. The distance between two consecutive locations at the i-th circle is less than $2\pi c_i/s \leq \frac{\varepsilon}{k} \cdot c_i$. Therefore the distance between p and its nearest location is at most $\frac{\varepsilon}{k} \cdot c_i \leq \frac{\varepsilon}{k} \cdot r(p)$. □

Now, we modify P by moving each point $p \in P$ to its nearest location (see Figure 2.12b). The following Lemma gives an upper bound on the cost of the operation.

Lemma 2.31. *Let $I = (P, o, \delta)$ be an instance of the CVRP problem, and S a solution for it. The operation of moving each point $p \in P$ to the nearest location changes the cost of S by at most $\varepsilon \cdot OPT(I)$.*

Proof. Moving a point $p \in P$ by a distance at most $\frac{\varepsilon}{k} \cdot r(p)$ changes the cost of a tour visiting p by at most $2\frac{\varepsilon}{k} \cdot r(p)$. The total change in cost generated by moving each point in P to the nearest location is therefore upper bounded by $2\frac{\varepsilon}{k} \cdot \sum_{p \in P} r(p)$. Lemma 2.1 gives a lower bound of $\frac{2}{k} \cdot \sum_{p \in P} r(p)$ for the cost of a solution for the CVRP problem. We get that the total cost of moving all the points is at most $\varepsilon \cdot OPT(I)$. □

From a solution S' for the modified instance of the CVRP problem, i.e where all points have been moved to their nearest locations, we can easily get a solution S for the original version of the problem such that $C(S) \leq C(S') + \varepsilon \cdot OPT$. Therefore a $(1 + O(\varepsilon))$-approximate algorithm for the modified version yields a $(1 + O(\varepsilon))$-approximate algorithm for the original version. In the rest of this chapter we consider the modified version of the problem, i.e. we assume that all points from P are situated in locations.

Trivial and nontrivial tours. We say that a tour T *visits* a location ℓ if T contains at least one point from ℓ. If an edge of a tour T passes trough a location ℓ, but T does not contain any point from ℓ, then T does not visit ℓ.

We call a tour *trivial* if it visits only a single location in P; it is *nontrivial* otherwise.

Theorem 2.32. *For any instance $I = (P, o, \delta)$ of the CVRP problem there is an optimal solution with at most t nontrivial tours, where t is the number of locations.*

Proof. We say that a sequence of tours $T_1, T_2, \ldots, T_m$ $(m \geq 2)$ forms a *cycle* if there is a set of locations $\ell_1, \ell_2, \cdots, \ell_m, \ell_{m+1} = \ell_1$ such that each tour T_i visits locations ℓ_i and ℓ_{i+1}. Note that the origin o is not considered as a location.

To prove the theorem we need the following

Lemma 2.33. *For any instance $I = (P, o, \delta)$ of the CVRP problem there is an optimal solution with no cycles.*

Proof. Let $\mathcal{T}$ be an optimal solution for I which minimizes the sum over all nontrivial tours $T \in \mathcal{T}$ of the number of locations visited by T.

Let us suppose that $\mathcal{T}$ has a cycle, and let $T_1, T_2, \ldots, T_m$ be a cycle with a minimum number of tours m. Let $\ell_1, \ell_2, \ldots, \ell_m$ be the locations in which the

consecutive tours meet. From the minimality of m we get that both tours and locations are pairwise distinct.

Let $v(T, \ell)$ denote the number of points from a location ℓ visited by a tour T. Let $\alpha = \min_{i \in \{1,\ldots,m\}} v(T_i, \ell_i)$. Now we are ready to swap points between the tours: the i-th tour, instead of visiting $v(T_i, \ell_i)$ points in the location ℓ_i and $v(T_i, \ell_{i+1})$ points in the location ℓ_{i+1} will now visit $(v(T_i, \ell_i) - \alpha)$ points in ℓ_i and $(v(T_i, \ell_{i+1}) + \alpha)$ points in ℓ_{i+1}. Here ℓ_{m+1} denotes ℓ_1.

Observe that the modification does not change the number of points visited by each tour. It also does not increase the length of any tour. Therefore, we obtain another optimal solution, in which the sum over all nontrivial tours of the number of locations visited by the tour is smaller than in $\mathcal{T}$, as each tour T_i for which $v(T_i, \ell_i) = \alpha$ visits one location less than before. This contradicts the minimality of the above sum in $\mathcal{T}$.

Therefore the optimal solution $\mathcal{T}$ has no cycles. □

Consider an optimal solution without cycles. Note that the lack of 2-cycles means that no two tours visit the same pair of locations. To each nontrivial tour we can assign a pair of distinct locations visited by the tour. The chosen pairs are in one-to-one correspondence with the nontrivial tours and they induce an acyclic undirected graph on the locations.

Hence, we can have at most $t-1$ nontrivial tours in an acyclic solution, where t is the number of locations. Using Lemma 2.33 we get the theorem. □

Reduction to an instance of CVRP with $(k/\varepsilon)^{O(1)} \cdot O(\log^2 n)$ points. We are now ready to prove the following theorem.

Theorem 2.34. *Let $I = (P, o, \delta)$ be an instance of the CVRP problem. We can reduce I to an instance $I' = (P', o, \delta')$ of the CVRP problem, where $|P'| \leq t^2 k = (k/\varepsilon)^{O(1)} \cdot O(\log^2 n)$.*

Proof. Theorem 2.32 implies that there is an optimal solution S in which at most tk points are covered by nontrivial tours. In particular, if the number of points in a location ℓ is greater than tk, S covers some of them with trivial tours. We may assume, without loss of generality, that among all trivial tours visiting a given location there is at most one that visits less than k points. Moreover, if at least one point from some location is visited by a nontrivial tour, we can assume that all trivial tours visiting that location contain exactly k elements. Therefore, for each location ℓ containing x points, we only have to consider at most $\min\{x, x - k \cdot \lceil \frac{x-tk}{k} \rceil\} \leq tk$

points for nontrivial tours. After finding a $(1+\varepsilon)$-approximate solution for the reduced case, we add trivial tours covering all the remaining points. That gives us a $(1+\varepsilon)$−approximate solution for the original problem. □

2.4.2 PTAS for the Capacitated Vehicle Routing Problem with $k \leq 2^{\log^{o(1)} n}$

We use Theorem 2.34 to reduce any instance $I = (P, o, \delta)$ of the CVRP problem to an instance I' of the CVRP problem with $N \leq (k/\varepsilon)^{O(1)} \cdot O(\log^2 n)$ input points. For the instance I' we apply the quasi-polynomial time approximation scheme for the CVRP problem due to Das and Mathieu [42] (see also Section 2.3, as the CVRP problem is a special case of the CLRP problem). The algorithm returns a $(1+\varepsilon)$-approximate solution for I' in time $N^{\log^{O(1/\varepsilon)} N}$, and we transform it into a $(1+\varepsilon)$-approximate solution for I by adding trivial tours. The running time of the algorithm is polynomial for $k \leq 2^{\log^{\mathfrak{c}(\varepsilon)} n}$ for some constant $\mathfrak{c}(\varepsilon) > 0$ depending on ε. Hence, we have the following theorem.

Theorem 2.35. *There is a polynomial time approximation scheme for the two-dimensional Euclidean capacitated vehicle routing problem provided that* $k \leq 2^{\log^{o(1)} n}$.

2.4.3 Refinement: Reduction to $(k/\varepsilon)^{O(1)}$ Points

We have demonstrated that the problem of close approximation of the CVRP problem on an input set of n points in the Euclidean plane reduces to that for a set of points of size polynomial in k/ε and polylogarithmic in n, situated in the relevant locations. In this section, we shall eliminate the polylogarithmic dependency of n in the reduction. For small values of k this leads to a faster PTAS. Our approach resembles Baker's method [18] of closely approximating several hard problems on planar graphs and the shifting strategy by Hochbaum and Maas [57].

For a given $\varepsilon > 0$ we make a clustering of the circles (C_i) into *rings* of $\lceil \log_{1+\frac{\varepsilon}{k}}(6/\varepsilon) \rceil$ consecutive circles.

Our result relies on the following separation lemmas.

Lemma 2.36. *Let $I = (P, o, \delta)$ be an instance of the CVRP problem, where the points P are situated in locations. If we mark an arbitrary set of rings, any solution S for I can be transformed into a solution S' for the points in the unmarked rings such that*

1. *no tour in S' visits two points in P separated by a marked ring, and*
2. $C(S') \leq (1+\frac{\varepsilon}{2})C(S)$.

Proof. Let T denote a tour, and Γ a path obtained from it by removing the edges of T incident to o. Suppose that Γ crosses one of the marked rings (i.e. it visits points from P separated by the ring). Let C_i be the innermost circle of the ring. As a ring consists of $\lceil \log_{1+\frac{\varepsilon}{k}}(6/\varepsilon) \rceil$ consecutive circles, and the j-th ring has radius $c_i = \frac{L\varepsilon}{n} \cdot \left(1+\frac{\varepsilon}{k}\right)^i$, each minimal fragment of Γ crossing the aforementioned ring has length at least $\frac{4}{\varepsilon} \cdot c_i$. We split the path Γ along the circle C_i whenever Γ crosses the marked ring. Notice that the paths obtained from Γ do not cross the marked ring. We transform the paths into tours by connecting their endpoints with the origin o (see Figure 2.13). The total length of the additional edges is $2c_i x$, where x is the

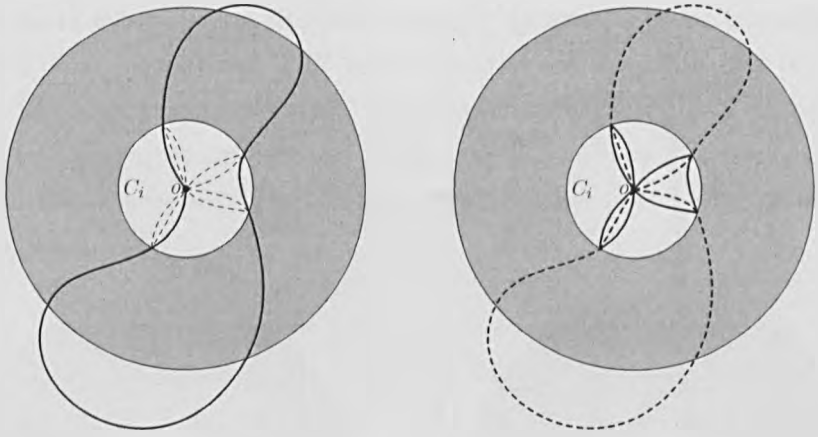

Figure 2.13: Splitting a tour T into five smaller tours. The grey area is the marked ring. In the left picture dashed lines represent the lines which will be added to the solution. The right picture shows the separate tours obtained from the original tour, before the short-cutting. The tours visiting the points in the area bounded by the marked ring are pictured as solid lines, and the tours visiting the points outside the marked ring are pictured with a dashed line.

number of times Γ crosses the marked ring. The cost of the added edges is therefore at most $\frac{\varepsilon}{2}$ of the total length of the aforementioned fragments of Γ.

We can iterate the elimination of the crossings of the smaller resulting tours but for their edges incident to o with more marked rings. Note that then other disjoint fragments of T will be charged with the increase of the length of the union of the resulting smaller tours. Finally, by applying short-cutting, we can drop the points in the marked rings from the resulting tours.

We conclude that we can transform S into a solution S' for the points in P in the unmarked rings such that no tour in S' crosses a marked ring and $C(S') \leq (1+\frac{\varepsilon}{2})C(S)$. □

Lemma 2.37. *Let $I = (P, o, \delta)$ be an instance of the CVRP problem, where the*

points P are situated in locations. There are integers $1 \leq b \leq a = O(\varepsilon^{-1})$ such that if we mark each $(b + j \cdot a)$-th ring then the points in the marked rings can be covered with tours visiting at most k points each of total length at most $\frac{\varepsilon}{2}OPT(I)$ produced by the iterated tour partitioning heuristic from [52] (see Section 2.2).

Proof. Let R_j denote the set of points from P lying in the j-th ring. Set a to $\lceil \frac{14}{\varepsilon} \rceil$. For each $b \in \{1, \ldots, a\}$, let P_b be the set of points in P in the marked rings for the given choice of b, i.e. $P_b = \sum_{j \equiv b \bmod a} R_j$. We will show that there is some $b \in \{1, \ldots, a\}$ such that the iterated tour partitioning heuristic using a 2-approximate TSP tour generates for the set of input points P_b a solution with cost at most $\frac{\varepsilon}{2}OPT(I)$.

We mark every second ring and we apply almost the same transformation as in Lemma 2.36 to an optimal TSP tour T on the set of points $P \cup \{o\}$, with the exception that after cutting a path Γ into several paths, instead of connecting each path by two rays to o, we connect the paths directly to obtain two tours, one visiting all points in the area bounded by the ring, and the other in the area outside the ring (see Figure 2.14).

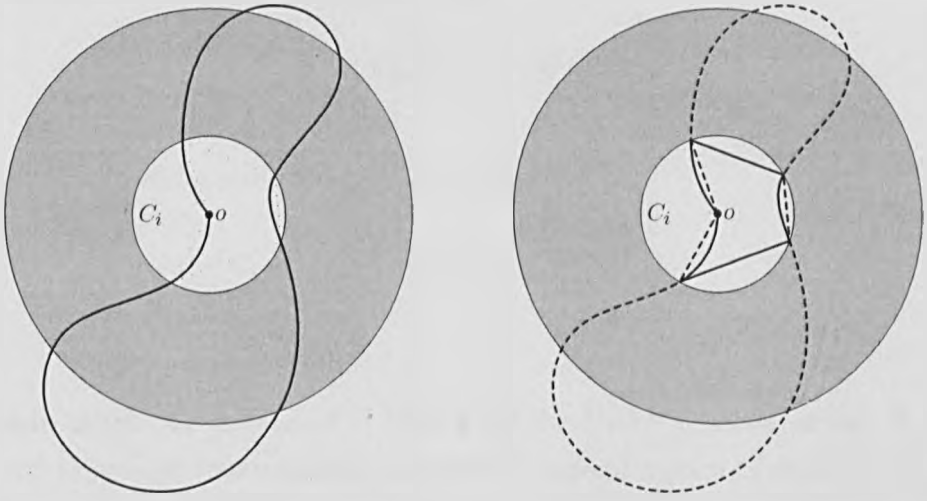

Figure 2.14: Splitting a TSP tour T into two tours, visiting all points from the areas separated by the marked ring. The grey area is the marked ring. In the left picture dotted lines represent the lines which will be added to the solution. The right picture shows two separate tours obtained from T — the tour visiting the points in the area bounded by the marked ring is pictured as a solid line, and the tour visiting the points outside the marked ring is pictured as a dashed line — before the short-cutting.

As a result, after short-cutting the tours, we get a collection of TSP tours, each covering a set of points R_j from one unmarked ring. The total length of these TSP tours is at most $(1 + \frac{\varepsilon}{2}) \cdot TSP(P)$. Assuming first that the unmarked rings are

the even ones, and then conversely, that the unmarked rings are the odd ones, and that $\varepsilon \leq 1$, we conclude that

$$\sum_j TSP(R_j) \leq 3 \cdot TSP(P) \ .$$

From Lemma 2.3 we get that the cost of the solution found by the iterated tour partitioning heuristic using a 2-approximate TSP tour for a set of points P_b satisfies

$$C_{it}(P_b) \leq \frac{2}{k} \sum_{p \in P_b} r(p) + 2 \cdot TSP(P_b) \ .$$

As $P_b = \sum_{j \equiv b \bmod a} R_j$, we get

$$\sum_{b \in \{1,\ldots,a\}} C_{it}(P_b) \leq \frac{2}{k} \sum_{p \in P} r(p) + 2 \sum_j TSP(R_j) \leq \frac{2}{k} \sum_{p \in P} r(p) + 6 \cdot TSP(P) \ .$$

Using Lemma 2.1 and Lemma 2.2 we obtain

$$\sum_{b \in \{1,\ldots,a\}} C_{it}(P_b) \leq 7 \cdot OPT(I) \ .$$

There must be some $b \in \{1, \ldots, a\}$ for which

$$C_{it}(P_b) \leq \frac{7}{a} \cdot OPT(I) \leq \frac{\varepsilon}{2} OPT(I) \ ,$$

as $a = \lceil \frac{14}{\varepsilon} \rceil$. □

Theorem 2.38. *An instance $I = (P, o, \delta)$ of the CVRP problem on the Euclidean plane can be reduced to a collection of $O(\log n)$ disjoint instances of the CVRP problem, each on $O(k/\varepsilon)^{O(1)}$ points and each having the maximum distance to the origin at most $(1/\varepsilon)^{O(1/\varepsilon)}$ larger than the minimum one, such that $(1+\varepsilon)$-approximate solutions to each of the latter problems yield a $(1+O(\varepsilon))$-approximation to the original CVRP problem. The reduction can be done in time $O(n \log n)$ for a fixed ε.*

Proof. We move the points to the locations and compute the sets R_j of input points lying in the rings. This all can be easily done in time $O(n \log n)$ by using standard data structures for point location [73].

Next, we compute the value a (i.e. the distance between marked rings) and for each $b \in \{1, \ldots, a\}$ compute a solution for the set P_b of points contained in the marked rings, using the iterated tour partitioning heuristic with a 2-approximate

TSP tour. Using the minimum spanning tree heuristic for TSP we can find a 2-approximation of the optimal TSP tour in time $O(n \log n)$. Given a TSP tour, the iterated tour partitioning heuristic can be implemented in time $O(k\frac{n}{k} + n)$ by repeatedly updating the previous partition and the solution to the next one in time $O(\frac{n}{k})$. Therefore all the a computations take $O(a \cdot n \log n) = O(n \log n)$ time. We fix b to that minimizing the cost of the aforementioned solution, which from Lemma 2.37 is at most $\frac{\varepsilon}{2} OPT(I)$.

Next, we compute approximate solutions for each maximal sequence of consecutive not marked rings. As the number of circles C_j is $\Theta(\frac{k}{\varepsilon} \log \frac{n}{\varepsilon})$, and a ring contains $\lceil \log_{1+\frac{\varepsilon}{k}}(6/\varepsilon) \rceil$ circles, we have $q = O(\log n)$ such sequences. For $i = 1, \ldots, q$, let P_i denote the set of points contained in the i-th sequence of rings. Note that the point sets P_i can be also easily computed in time $O(n \log n)$.

It follows from Lemma 2.36 that if we compute $(1+\varepsilon)$-approximate solution for each set P_i, the union of the coverings will have length at most $(1+O(\varepsilon))OPT(I)$.

Note that for a given i, the number of locations in P_i is $O(a \cdot \frac{k}{\varepsilon} \cdot \log_{(1+\frac{\varepsilon}{k})} \frac{1}{\varepsilon}) = O(k^2 \varepsilon^{-3} \log \frac{1}{\varepsilon})$. Hence, by the discussion in Section 2.4.1, we can decrease the number of points in each location to $O(k^3 \varepsilon^{-3} \log \frac{1}{\varepsilon})$. Thus, for each P_i we can reduce the problem to one with $O(k^5 \varepsilon^{-6} (\log \frac{1}{\varepsilon})^2) = (k/\varepsilon)^{O(1)}$ points.

Each P_i contains points from $O(\varepsilon^{-1})$ consecutive rings and for a point in a ring the maximum distance to the origin is at most $O(\varepsilon^{-1})$ times larger than the minimum one. Hence, for a point in P_i the maximum distance to the origin is at most $(1/\varepsilon)^{O(1/\varepsilon)}$ times larger than the minimum one.

The appropriate q sets of points can be computed in time $O(n \log n)$ and they specify the problems to which we approximately reduce the original CVRP problem. □

2.4.4 Extension for the Two-Dimensional Euclidean Capacitated Location Routing Problem with $|O| \leq 2^{\log^{o(1)} n}$

Let $m = |O|$ be the number of possible depots in an instance $I = (P, O, \delta, \phi)$ of the CLRP problem. We set $L = \max_{p \in P} \delta(p, O)$ and create a set of locations around each point from O in the same way as in the CVRP problem. As we have $OPT(I) \geq 2L$, moving each point to the nearest location changes the cost of a solution for the CLRP problem by at most $\varepsilon \cdot OPT(I)$. The number of locations $t' = (k/\varepsilon)^{O(1)} \cdot O(m \cdot \log n)$.

The analysis of trivial and nontrivial tours can be done in the same way as in the CVRP problem, and we can reduce the problem to one with $N' \leq t'^2 k = (k/\varepsilon)^{O(1)} \cdot O(m^2 \cdot \log^2 n)$ points.

For the reduced problem instance we use the QPTAS from Section 2.3 (The-

orem 2.10). The algorithm returns a $(1+\varepsilon)$-approximate solution for the reduced problem instance in time $N'^{\log^{O(1/\varepsilon)} N'}$, and we transform it into a $(1+\varepsilon)$-approximate solution for the original problem instance by adding trivial tours. The running time of the algorithm is polynomial for $k, m \leq 2^{\log^{\mathfrak{c}(\varepsilon)} n}$ for some constant $\mathfrak{c}(\varepsilon) > 0$ depending on ε. Hence, we have the following theorem.

Theorem 2.39. *There is a polynomial time approximation scheme for the two-dimensional Euclidean capacitated location routing problem provided that* $k, |O| \leq 2^{\log^{o(1)} n}$.

The same result holds for the capacitated location routing problem with non-fixed return. Here, instead of using the QPTAS from Theorem 2.10, we use the QPTAS from Theorem 2.28. We get the following result.

Theorem 2.40. *There is a polynomial time approximation scheme for the two-dimensional Euclidean capacitated location routing problem with non-fixed return provided that* $k, |O| \leq 2^{\log^{o(1)} n}$.

2.5 Open Problems

PTAS for CVRP for all values of k. The central open question left is whether there is a PTAS for the CVRP problem for all values of k. While we have enlarged the set of values of k for which a PTAS exists, we still do not know how to reach polynomial values for k, even $k = n^{0.001}$. In particular, a PTAS for CVRP for $k = \Theta(\sqrt{n})$ is elusive. For arbitrary values of k, the best currently known result is either a quasi-polynomial time approximation scheme by Das and Mathieu [42] that runs in time $n^{\log^{O(1/\varepsilon)} n}$, or the polynomial-time constant-factor approximation algorithm due to Haimovich and Rinnooy Kan [52]. We believe that the case $k = \Theta(\sqrt{n})$ is the hardcore of the difficulty in obtaining a PTAS for all values of k.

Following [52], let us observe that if we divide the range of k into a logarithmic number of intervals of the form $[\varepsilon^{-2i}, \varepsilon^{-2(i+1)})$, then for k in at most one of the intervals none of the inequalities $TSP(P) \leq \varepsilon \cdot \frac{2}{k} \sum_{p \in P} r(p)$, $\frac{2}{k} \sum_{p \in P} r(p) \leq \varepsilon \cdot TSP(P)$ hold. Note that if any of the inequalities holds then by plugging any PTAS for TSP in the iterated tour partitioning heuristic, we obtain an $(1+O(\varepsilon))$-approximation of CVRP. Thus, the aforementioned heuristic is in fact a PTAS for a substantial range of k depending on P: for every set of points P there is k_0 such that there is a polynomial-time $(1+O(\varepsilon))$-approximation algorithm for CVRP for every $k \leq \varepsilon k_0$ and for every $k > k_0/\varepsilon$. Despite this observation and despite recent progress in [15; 42], the problem of designing a PTAS for *all* k remains open.

PTAS for MDCVRP (CLRP) for larger number of depots (possible depot locations). The question is whether there is a polynomial time approximation scheme for the MDCVRP (CLRP) problem for all values of $|O|$.

PTAS for CVRP for random points. Another interesting question is finding a nearly optimal solution (or even estimating the cost of such a solution) when the customers are represented by independent and uniformly distributed points in the unit disc centered at the origin. The only interesting case is when $k = \Theta(\sqrt{n})$, as otherwise the iterated tour partitioning heuristic gives a good approximation with high probability:

- if $k = o(\sqrt{n})$, then the cost of the TSP is negligible compared to the radial cost $\frac{2}{k}\sum_{p\in P}\delta(p,o)$,
- if $k = \omega(\sqrt{n})$, then the radial cost $\frac{2}{k}\sum_{p\in P}\delta(p,o)$ is negligible compared to the cost of the TSP.

As the case $k = \Theta(\sqrt{n})$ seems to be the most difficult one to get a PTAS for, solving it for random points might be a step towards getting a PTAS for the CVRP problem for the whole range of k.

The CVRP problem where the customers are independent and uniformly distributed over the unit disc centered at the origin or over a unit square has been studied in [75; 24].

In [75] Rhee proves that the cost of the optimal solution for this model for a fixed integer k satisfies with high probability the inequality: $|OPT - \frac{2}{k}\sum_{p\in P}\delta(p,O) - \xi\sqrt{n}| \leq K(n\log n)^{1/3}$, where ξ and K are constants depending on k. That provides a good estimation of the cost of the solution for small values of k. However, K grows fast with k, and for $k = \Theta(\sqrt{n})$ the right handside is of the order $\omega(\sqrt{n})$, so the inequality does not give us any information in that case.

In [24] Bompadre et al. show that for random points the iterated tour partitioning heuristic has whp. approximation ratio $2 - c$ for some $c > 0$. They show it by providing a better lower bound on the cost of the optimal solution that holds with probability 1 as the number of customers goes to ∞. They also generalize the iterated tour partitioning heuristic for the case of multiple depots and show the same asymptotic result for MDCVRP, where the depots are fixed in advance, and the customers are independent and uniformly distributed over the unit square.

Distance constrained (capacitated) vehicle routing problem. Another interesting version of the vehicle routing problem is the distance constrained problem.

Here instead of (or together with) the capacity constraint we have a bound on the maximum length of each tour. It is not known if this problem admits a PTAS.

Other metrics. We can study the above problems in more general metrics, starting from the constant-dimensional Euclidean metric. For which metrics can we prove the existence of a PTAS (for some range of input parameters)?

Chapter 3

Capacitated Geometric Network Design

3.1 Introduction

In this chapter we study the capacitated geometric network design problem and the single-sink capacitated geometric network design problem.

Even the SCGND problem is NP-hard, as it includes as a special case the minimum Euclidean Steiner tree problem [48]. We can model the Steiner tree problem by the SCGND problem, by taking one of the input points as a sink, all other points as sources of unit demand, and set the edge capacity to $n-1$, where n is the number of input points.

In this chapter we focus on the design of approximation schemes for the CGND and SCGND problem.

3.1.1 Difficulties

Even in the case when there is only one sink, a near optimal capacitated geometric network is not necessarily a forest (see Figure 3.1 for an example). Similarly as in [78] we can show, that for the CGND problem a minimum cost of a network which is a forest is at most twice larger than the cost of an optimal solution. However, as we are interested in constructing approximation schemes, we cannot restrict the solution to be a forest.

Another major difficulty in deriving approximation schemes for the CGND and SCGND problem is caused by the *Steiner vertices*, i.e. the vertices of the network other than the sources and the sinks. Since Steiner vertices can be arbitrary points on the plane, one can consider all points in the vicinity of sources and sinks

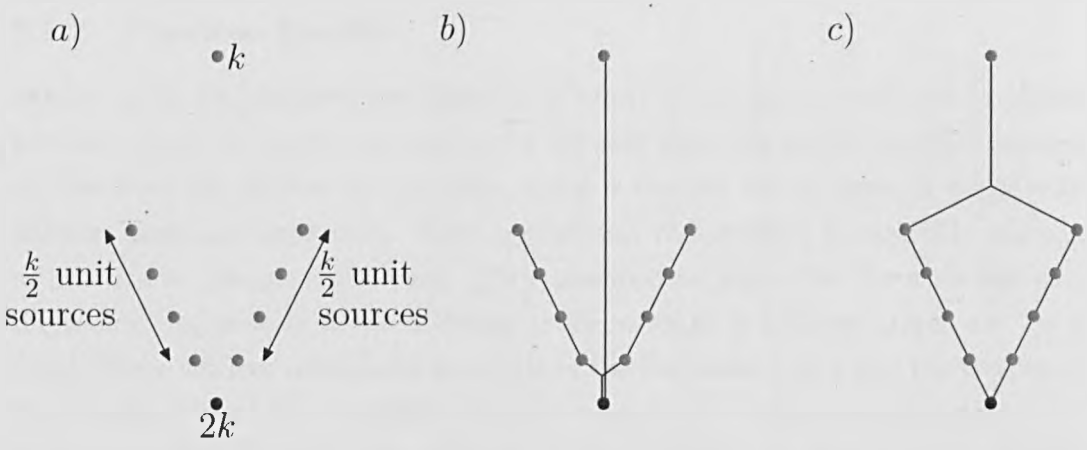

Figure 3.1: An instance of the SCGND problem (a), where the minimum cost of a network which is a forest (b) is significantly larger than the cost of an optimal solution (c). The sources and the sink are denoted respectively by grey and black dots. There is one source with demand k, k unit sources, and a single sink with demand $2k$.

as potential candidates for Steiner vertices. In the very special case of minimum Euclidean Steiner tree the number of Steiner vertices in an optimal solution can be easily upper bounded by $n - 2$, where n is the number of input points. If the network edges are required to be vertical or horizontal, then the Steiner vertices can be constrained to the quadratic number of intersections between vertical and horizontal straight-lines passing through the sources and sinks, the so called Hanan grid [38; 88]. However, in the general Euclidean case that we consider here, the problem of bounding the number of Steiner vertices in terms of the total demand of the sources has been open [38].

It has not even been settled if there is always an optimal network minimizing the cost, or if we can find solutions using more and more Steiner vertices, with the cost converging to a limit that cannot be achieved by any finite network.

An upper bound on the number of Steiner vertices in an optimal (or at least in a near optimal) solution is needed, if we want to use Arora's framework [10] for geometric optimization problems. The first step in this framework, perturbation, consists of creating a grid and shifting all input points and Steiner points to the grid corners. The grid size has to be chosen in such a way, that shifting all the points does not increase the total length of the edges by too much compared to the cost of the network. However, without a bound on the number of Steiner vertices, and therefore also without a bound on the number of edges of the network, we cannot give any upper bound on the increase of cost caused by shifting the points, and therefore we cannot perform perturbation.

3.1.2 Previous Results

Salman et al. [78] initiated the algorithmic study of the *buy-at-bulk network design problem*, where the goal is to construct a network that can send a specified amount of flow from the sources to the sinks, using a discrete set of types of edges with different costs and capacities. They argued that the problem is especially relevant in practice in the geometric case. They provided an algorithm for a version of a single-sink buy-at-bulk network design problem on an Euclidean graph, i.e. on a graph where vertices correspond to points in the Euclidean plane and the weights of the edges represent the Euclidean distance between the corresponding points, with an approximation ratio of $O(\log(D/k_1))$, where D is the total demand of the sources and k_1 is the smallest edge capacity. Besides allowing the use of many edge types, their model differs from ours in that it does not allow a flow to be split (i.e. the whole flow from each source has to use one path), and only permits a given finite set of points in the Euclidean plane to be used as vertices by the network.

Czumaj et al. [38] consider the *geometric* buy-at-bulk network design problem, where the only input is the locations of sources and sinks, their demands, and a set of types of edges, and any point on the Euclidean plane can be used as a vertex. They present a QPTAS for the *rectilinear* version of the geometric buy-at-bulk network design problem with polynomially bounded total demand, where the edges have to be horizontal or vertical. Their QPTAS relies on the observation that in the rectilinear case Steiner vertices can be constrained to lie on the *Hanan grid* of the input points, i.e. on the grid formed by the vertical and horizontal lines passing through the input points. That yields a polynomial upper bound on the number of Steiner vertices and the number of edges needed in an optimal solution to the problem, which allows using Arora's framework [10] for geometric optimization problems. A near optimal solution is found using dynamic programming, as in each dissection square it is enough to consider $O(\log n/\varepsilon)$ portals, and the amount of flow passing through each portal is not greater than the total demand of the sources.

Observe that the CGND problem can be considered as a special case of the geometric buy-at-bulk network design problem, in which only a single type of edge is available.

3.1.3 Overview of the New Results

In Section 3.2 we study structural properties of (near) optimal solutions for the instances of the CGND problem. The main result is an upper bound on the number of Steiner vertices in an optimal solution that is polynomial in the total demand of

the sources.

In Section 3.3 we present a quasi-polynomial time approximation scheme for the CGND problem with polynomially bounded demands of sources and sinks. The result is obtained by combining the upper bound on the number of Steiner vertices from Section 3.2 with Arora's framework [10] for geometric optimization problems.

In Section 3.4 we derive a polynomial time approximation scheme for the SCGND problem when the edge capacity is at most $2^{O(\sqrt{\log n})}$, and the demands of the sources and the sinks are polynomially bounded. The result relies on a geometric partition of the sources combined with a TSP-based heuristic and the QPTAS from Section 3.3 applied to an instance of the problem with a small number of points.

The results are based on joint work with Artur Czumaj, Andrzej Lingas and Jakub Onufry Wojtaszczyk [6].

3.1.4 Notation

We now give a formal definition of the problem.

Definition 3.1. *In the capacitated geometric network design problem we are given a set of* sources $\mathcal{S} = \{s_1, \ldots, s_{n_s}\}$ *and a set of* sinks $\mathcal{T} = \{t_1, \ldots, t_{n_t}\}$ *on the Euclidean plane, together with an integral* demand $\mathfrak{d}(s_i)$, $\mathfrak{d}(t_j)$ *for each source and sink, and an integral* edge capacity k.

The goal is to construct a minimum-cost network that can (simultaneously) route the demanded flow from the sources to the sinks, such that each edge in the network has capacity k. The network can use any points on the Euclidean plane as Steiner vertices, the flow is splittable and parallel edges are allowed.

The number of input points is denoted by n, i.e. $n = n_s + n_t$. We assume that $\sum_i \mathfrak{d}(s_i) = \sum_j \mathfrak{d}(t_j)$ and define $D = \sum_i \mathfrak{d}(s_i)$ to be the *total demand*. We can assume that no two input points are coincident, as otherwise we can merge them into a single source or sink.

A solution for an instance $I = (\mathcal{S}, \mathcal{T}, \mathfrak{d}, k)$ of the capacitated geometric network design problem is represented by a multigraph M embedded in $\mathbb{R}^2$. The vertex set of M consists of the sources $\mathcal{S}$ and the sinks $\mathcal{T}$, and potentially other vertices (called *Steiner vertices*). For technical reasons in Section 3.2 we consider directed multigraphs, where the direction of an edge represents the direction of the flow within the edge. The weights of the edges represent Euclidean distances between the vertices. The *cost* of a multigraph M, denoted by $C(M)$, is the sum of the lengths of all the edges of M.

A multigraph satisfying the above properties is a feasible solution for I if there is a flow in M from the sources to the sinks that is consistent with the directions of the edges, satisfies the demands $\mathfrak{d}$ of the sources and the sinks, and never exceeds the capacity k in an edge.

If M is a feasible solution for I then one can find such a flow in polynomial time using standard algorithms for the maximum-flow problem with multiple sources and multiple sinks. Furthermore, since the demands and the edge capacity are assumed to be integral, we can presume, without loss of generality, that the flow found is integral.

If the set of sinks is a singleton then the problem is termed the *single-sink capacitated geometric network design problem.*

3.2 Bounding the Number of Steiner Vertices in an Optimal Solution

In this section we show that for any instance of the CGND problem there is an optimal solution with the number of Steiner vertices upper bounded by a polynomial function of the total demand D.

For this purpose, we fix an instance I of the CGND problem and we consider a special class of multigraphs that are feasible solutions for I, which we call *minimizers.* We show that for any $\varepsilon > 0$ there is a minimizer that gives a $(1+\varepsilon)$-approximate solution for I. We then analyze geometric properties of the minimizers, and show that *each* minimizer has a special geometric structure, namely it can have only three types of Steiner vertices. We then define an operation of shifting a cycle in a minimizer. We show that this operation transforms a minimizer into another minimizer without increasing the cost, can be performed on a minimizer only a finite number of times, and when the operation cannot be performed any more, the resulting minimizer has a small number of Steiner vertices. We get that for any $\varepsilon > 0$ there *exists* a minimizer that gives a $(1+\varepsilon)$-approximate solution for I and has a small number of Steiner vertices.

We then take a set of multigraphs that are feasible solutions for I and satisfy some additional properties, and equip the set with a metric, such that the corresponding metric space is compact, and the cost is a continuous function in it. Then any sequence of multigraphs from that space has a subsequence converging to a multigraph, which is a feasible solution for I. We choose the multigraphs for the sequence as minimizers which give $(1+\varepsilon)$-approximate solutions for I, where the value of ε decreases towards 0, and each of the minimizers has a small number of Steiner

vertices. The multigraph obtained in the limit is an optimal solution for I (from the continuity of the cost function), and has a small number of Steiner vertices.

3.2.1 Minimizers

We consider multigraphs that are feasible solutions for the instance I of the CGND problem, together with associate integral flows certifying the feasibility. For a multigraph $M = (V, E)$ we define the *value* of M as

$$\text{val}(M) = \sum_{v \in V} (\deg(v) - 2) \ .$$

Among the solutions with the same cost, we prefer the one with the minimum value.

Without increasing the cost or the value we can modify the multigraph M and an associated integral flow f in M to satisfy the following properties:

- all Steiner vertices in M have degree at least 3,
- all Steiner vertices are contained in the smallest square containing all sources and sinks (this property can be satisfied by moving each Steiner vertex from outside the square to the closest point on the boundary of the square),
- if two vertices of M are coincident, then there is no edge between them (otherwise we merge them into one vertex),
- each edge of M is used by f,
- the amount of flow entering and leaving each vertex is at most D.

For technical reasons, and without loss of generality, in this section we consider only pairs (M, f) satisfying the above properties.

The degree of any vertex is upper-bounded by $2D$. Each Steiner vertex has degree at least 3, so it contributes at least 1 to $\text{val}(M)$. The only vertices that have negative contribution to $\text{val}(M)$ are non-Steiner vertices of degree 1, and there are at most $2D$ of them (as we can split each source and sink into unit sources and sinks), so the number of Steiner vertices in M is not greater than $\text{val}(M) + 2D$ and $|V| \leq \text{val}(M) + 4D$. Since $2|E| = \sum_{v \in V} \deg(v)$, we get that

$$|E| = \frac{1}{2}\text{val}(M) + |V| \leq \frac{3}{2}\text{val}(M) + 4D \ .$$

For any $\varepsilon > 0$ and integer $\mathfrak{v}$, let $\mathcal{M}^{\varepsilon}_{\mathfrak{v}}$ denote the set of all multigraphs that together with some flow give a $(1+\varepsilon)$-approximate solution for I and which have value at most $\mathfrak{v}$. We now show a technical lemma.

Lemma 3.2. *For any $\varepsilon > 0$ and $\mathfrak{v} \in \mathbb{N}$ we can equip the set $\mathcal{M}^{\varepsilon}_{\mathfrak{v}}$ with a metric δ such that the metric space $(\mathcal{M}^{\varepsilon}_{\mathfrak{v}}, \delta)$ is compact, and the network cost is a continuous function in $(\mathcal{M}^{\varepsilon}_{\mathfrak{v}}, \delta)$.*

Proof. We allow the following operations on multigraphs, and define a cost for them:

- moving a Steiner vertex by some distance Δ. The cost of the operation is Δ.
- merging a Steiner vertex v with some vertex w connected to v with at least one edge. We remove the edges that contract to the loops. The cost is the distance by which we move v to merge it with w.
- splitting a vertex v into two vertices v and w and connecting them with at least one edge. The edges that were incident with v can be arbitrarily divided between v and w. The cost is the distance by which we move w away from v.
- adding or deleting an edge between two non-Steiner vertices. The cost is the distance between the vertices. As two non-Steiner vertices cannot be coincident, the cost is greater than 0.

If a multigraph $M' \in \mathcal{M}^{\varepsilon}_{\mathfrak{v}}$ can be obtained from $M \in \mathcal{M}^{\varepsilon}_{\mathfrak{v}}$ by a sequence of the above operations, we denote the distance $\delta(M, M')$ as the infimum of the costs of such sequences. Notice that the intermediate multigraphs in the sequence do not have to belong to $\mathcal{M}^{\varepsilon}_{\mathfrak{v}}$.

The distance function δ satisfies the following properties.

- For each $M \in \mathcal{M}^{\varepsilon}_{\mathfrak{v}}$ we have $\delta(M, M) = 0$.
- For each $M, M' \in \mathcal{M}^{\varepsilon}_{\mathfrak{v}}$ we have $\delta(M, M') = \delta(M', M)$.

  Each operation modifying a multigraph is reversible and the dual operation has the same cost.
- The distance function δ satisfies the triangle inequality.

  For each $M, M', M'' \in \mathcal{M}^{\varepsilon}_{\mathfrak{v}}$ we have $\delta(M, M'') \leq \delta(M, M') + \delta(M', M'')$, as a sequence of operations transforming M into M' together with a sequence of operations transforming M' into M'' transforms M into M''.

- For each $M, M' \in \mathcal{M}^{\varepsilon}_{\mathfrak{v}}$ the distance $\delta(M, M')$ is well defined (i.e. it is not infinite).

  Let M_0 be a multigraph with the set of all non-Steiner vertices as a vertex set and an empty edge set. Every graph $M \in \mathcal{M}^{\varepsilon}_{\mathfrak{v}}$ can be transformed to M_0 using a sequence of the following operations: merging a Steiner vertex with any neighboring vertex and deleting an edge between two non-Steiner vertices. The number of these operations is upper bounded by $|V(M)| + |E(M)|$, and the cost of each operation is upper bounded by the maximum distance between two points in M. The distances $\delta(M, M_0)$ and $\delta(M', M_0)$ are finite and so $\delta(M, M')$ is finite.

- For $M, M' \in \mathcal{M}^{\varepsilon}_{\mathfrak{v}}$, $M \neq M'$, we have $\delta(M, M') > 0$.

  Let Δ be the smallest distance between two non-coincident vertices in any of the multigraphs M, M'. The sequences that transform M into M' and have cost smaller than Δ are the ones that only move points. That requires M and M' to be isomorphic. Since M and M' are finite, there can be only a finite number of isomorphisms between M and M', and transforming M into M' according to each of them has a cost greater than 0 (as $M \neq M'$). The infimum over them is also greater than 0.

That shows that $(\mathcal{M}^{\varepsilon}_{\mathfrak{v}}, \delta)$ is a metric space.

Next we show that the metric space $(\mathcal{M}^{\varepsilon}_{\mathfrak{v}}, \delta)$ is compact. To do it we need to show that every sequence of multigraphs $(M_i)_{i \in \mathbb{N}}$, for $M_i \in \mathcal{M}^{\varepsilon}_{\mathfrak{v}}$, has a convergent subsequence. As the number of vertices and edges of all the multigraphs from $\mathcal{M}^{\varepsilon}_{\mathfrak{v}}$ is upper bounded by a function of $\mathfrak{v}$, there is only a finite number of isomorphism classes in $(M_i)_{i \in \mathbb{N}}$, where we additionally require the isomorphism to be identity on the non-Steiner vertices. We choose from $(M_i)_{i \in \mathbb{N}}$ an infinite subsequence of isomorphic multigraphs and we remove the remaining multigraphs. We order the vertices of the graphs according to the isomorphism.

The vertices of all graphs from $\mathcal{M}^{\varepsilon}_{\mathfrak{v}}$ are inside some closed square Γ. With each multigraph M_i we associate a sequence s_i of points in Γ representing the positions of the vertices. The edges of the multigraphs are obtained from the isomorphism. A set of sequences of points, where the points belong to a compact set, the length of the sequence is fixed and the distance δ' between two sequences is the sum of the distances between the pairs of corresponding points, is compact. As the total flow in the graphs from $(M_i)_{i \in \mathbb{N}}$ is D, we get that $\delta(M_{i_1}, M_{i_2}) \leq D \cdot \delta'(s_{i_1}, s_{i_2})$, and the set of isomorphic graphs with the vertices in Γ is compact. Let M be the multigraph that is the limit of some subsequence (M_{σ_j}) from $(M_i)_{i \in \mathbb{N}}$ — the sequence of the

vertices of M is a limit of the sequence (s_{σ_j}) and the edges are obtained from the isomorphism. Some vertices connected by an edge might be coincident in M (if the sequences of points converge to the same point). In this case we merge them into one vertex.

Now it is enough to show that $M \in \mathcal{M}^{\varepsilon}_{\mathfrak{v}}$. The value of M is not greater than the value of any M_{σ_j}, so $\mathrm{val}(M) \leq \mathfrak{v}$. The same flow that is feasible for any multigraph M_{σ_j} is feasible for M. The cost of M_{σ_j} converges to the cost of M and each M_{σ_j} is a $(1+\varepsilon)$-approximate solution for the instance of CGND, so M is also a $(1+\varepsilon)$-approximate solution for the instance of CGND. We get that $M \in \mathcal{M}^{\varepsilon}_{\mathfrak{v}}$ and the space $(\mathcal{M}^{\varepsilon}_{\mathfrak{v}}, \delta)$ is compact.

We now show that the cost $C(M)$ is a continuous function in $(\mathcal{M}^{\varepsilon}_{\mathfrak{v}}, \delta)$. Let $M \in \mathcal{M}^{\varepsilon}_{\mathfrak{v}}$ and take arbitrary $\varepsilon_0 > 0$. We have to show that there is some $\Delta_0 > 0$ such that for any multigraph $M' \in \mathcal{M}^{\varepsilon}_{\mathfrak{v}}$ if $\delta(M, M') < \Delta_0$ then $|C(M) - C(M')| < \varepsilon_0$. Let Δ be the shortest distance between two non-coincident vertices in M. We set $\Delta_0 = \min\left(\Delta, \frac{\varepsilon_0}{3/2\mathfrak{v}+4D}\right)$. Let $M' \in \mathcal{M}^{\varepsilon}_{\mathfrak{v}}$ such that $\delta(M, M') < \Delta_0$. In a sequence of operations transforming the multigraph M into M' with cost smaller than Δ_0 no edge of M can be contracted or deleted. Each edge from M can change its length by less than Δ_0. Some new edges can appear in M', each of them of length smaller than Δ_0. The total number of edges in M' is upper bounded by $\frac{3}{2}\mathfrak{v} + 4D$, therefore $|C(M) - C(M')| < (\frac{3}{2}\mathfrak{v} + 4D) \cdot \Delta_0 \leq \varepsilon_0$. □

Definition 3.3. *A multigraph M is a* minimizer *if it is a feasible solution for the given instance I of the CGND problem and satisfies the conditions:*

- *there is no feasible solution M' s.t. $C(M') < C(M)$ and $val(M') \leq val(M)$,*
- *there is no feasible solution M' s.t. $C(M') = C(M)$ and $val(M') < val(M)$.*

Lemma 3.4. *For any $\varepsilon > 0$ and an instance I of the CGND problem there is a $(1+\varepsilon)$-approximate solution for I which is a minimizer.*

Proof. Consider any multigraph M which gives a $(1+\varepsilon)$-approximate solution for I. It has some finite value $\mathfrak{v}$. Let $(\mathcal{M}^{\varepsilon}_{\mathfrak{v}}, \delta)$ be the metric space from Lemma 3.2. The metric space $(\mathcal{M}^{\varepsilon}_{\mathfrak{v}}, \delta)$ is compact and the cost is a continuous function in it, therefore there exists a multigraph with a minimum cost in $\mathcal{M}^{\varepsilon}_{\mathfrak{v}}$. The multigraph with a minimum value amongst all multigraphs with the minimum cost in $\mathcal{M}^{\varepsilon}_{\mathfrak{v}}$ is the desired minimizer. □

In the remaining part of this section we will prove the following theorem.

Theorem 3.5. *For any $\varepsilon > 0$ and an instance I of the CGND problem there is a minimizer M which is a $(1+\varepsilon)$-approximate solution for I and for which $|V_S|+|V_C| \leq 32D^4+4D^2$, where V_S is the set of Steiner vertices of M and V_C is the set of points on the plane where the edges of M cross without a Steiner vertex.*

Theorem 3.5, together with Lemma 3.2, yields the main result of this section.

Theorem 3.6. *Any instance I of the CGND problem has an optimal solution M_{OPT} which satisfies the following properties. The number of Steiner vertices is at most $32D^4+4D^2$, there are no two coincident vertices, the degree of each vertex is at most $2D$ and the edges cross only in vertices.*

Proof. Let $\varepsilon_i = \frac{1}{i}$ for $i \in \mathbb{N}$. From Theorem 3.5 we know that for each ε_i there is a $(1+\varepsilon_i)$-approximate solution M_{ε_i}, for which the number of Steiner vertices plus the number of points where the edges of M_{ε_i} cross without creating vertices is upper bounded by $32D^4+4D^2$. We create a multigraph M'_{ε_i} by adding Steiner vertices in all points where the edges of M_{ε_i} cross. (The flow in the modified multigraph M'_{ε_i} is the same as in M_{ε_i} — in the introduced vertices the flow does not change the direction.) We also join any two coincident vertices of the multigraph into one vertex. We can assume that the degree of every vertex in M'_{ε_i} is upper bounded by $2D$, as we can enforce this condition by removing edges.

Let $\mathfrak{v} = (32D^4+4D^2+2D)\cdot 2D$. Let $(\mathcal{M}^1_{\mathfrak{v}}, \delta)$ be the metric space from Lemma 3.2. All the graphs M'_{ε_i} belong to that metric space. As $(\mathcal{M}^1_{\mathfrak{v}}, \delta)$ is compact, the sequence of multigraphs $\left(M'_{\varepsilon_i}\right)_{i\in\mathbb{N}}$ has a subsequence converging to some multigraph $M \in (\mathcal{M}^1_{\mathfrak{v}}, \delta)$. Network cost is a continuous function in $(\mathcal{M}^{\varepsilon_0}_{\mathfrak{v}}, \delta)$, and each M'_{ε_i} is an $(1+\varepsilon_i)$-approximate solution for I, so M is an optimal solution for I.

From the construction of the limit subgraph in the proof of Lemma 3.2 we know that the number of Steiner vertices in M is not greater than the maximum number of Steiner vertices in a graph from $\left(M'_{\varepsilon_i}\right)_{i\in\mathbb{N}}$, so it is upper bounded by $32D^4+4D^2$. The degree of each vertex is at most $2D$, as otherwise by removing some edge we could get a solution with a smaller cost, which contradicts the optimality of M. The edges of the graphs M'_{ε_i} cross only in vertices, and the same holds for M. □

3.2.2 Vertex Types of a Minimizer

We proceed by an analysis of a minimizer. We aim to prove that there can be only three types of Steiner vertices in a minimizer.

Definition 3.7. *Let $M = (V, E)$ be a multigraph with a flow f. A vertex $v \in V$ is called a* real vertex *if there is an integer $c \geq 0$ and a direction $\alpha \in [0, 2\pi)$ such that:*

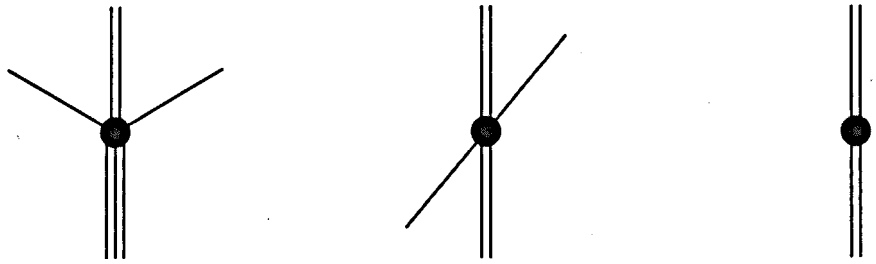

Figure 3.2: A real vertex, a crossing and an optional vertex (all for $c = 2$).

- *the edges incident with v are exactly: c edges with the direction α, $c+1$ edges with the direction $\alpha + \pi$, and two single edges with the directions $\alpha \pm \pi/3$ (see Figure 3.2),*
- *the group of $c+1$ edges is directed in one way (out of the vertex or into the vertex), and all the remaining edges are directed in the opposite way,*
- *the flow in either of the single edges is smaller than the flow in any of the $c+1$ edges.*

Definition 3.8. *Let $M = (V, E)$ be a multigraph with a flow f. A vertex $v \in V$ is called a* crossing *if there is an integer $c \geq 1$ and directions $\alpha, \beta \in [0, 2\pi)$ s.t. $\beta \neq \alpha, \alpha + \pi$ and:*

- *the edges incident with v are exactly: c edges with the direction α, c edges with the direction $\alpha + \pi$, and single edges with the directions β and $\beta + \pi$ (see Figure 3.2),*
- *all the c edges in one group are directed in one way, the directions of the two groups of edges are opposite, the directions of the two single edges are opposite,*
- *if $c \geq 2$ then the flow in any single edge is smaller than the flow in any of the non-single edges with the opposite direction.*

Definition 3.9. *Let $M = (V, E)$ be a multigraph with a flow f. A vertex $v \in V$ is called an* optional vertex *if there is an integer $c \geq 2$ and a direction $\alpha \in [0, 2\pi)$ such that:*

- *the edges incident with v are exactly: c edges with the direction α and c edges with the direction $\alpha + \pi$ (see Figure 3.2),*
- *all the c edges in one group are directed in one way, the directions of the two groups of edges are opposite.*

We specify some properties which the Steiner vertices of a minimizer must fulfill.

Fact 3.10. *Let $M = (V, E)$ be a minimizer, $v \in V$, and let $e_1, e_2 \in E$ be edges incident with v and directed in opposite ways. Then the angle between e_1 and e_2 is at least $2\pi/3$.*

Proof. Assume that the angle between e_1 and e_2 is smaller than $2\pi/3$. Let y and z be the endpoints of e_1 and e_2 other than v. Let f be the flow in the multigraph. We can assume that the amount of flow in the edges e_1 and e_2 is different — otherwise we can remove the edges e_1 and e_2 from the graph and replace them with an edge yz. Let M' be the modified graph. We have $C(M') < C(M)$ and $\mathrm{val}(M') < \mathrm{val}(M)$. The multigraph M' has a feasible flow, as the flow that was sent using the edges e_1 and e_2 in M can be sent using the added edge in M'. We get contradiction, as such a graph M' cannot exist if M is a minimizer.

Let x be the Torricelli point of the triangle vyz — the point that minimizes the total distance to v, y and z. If the angle at the vertex y (z) is greater or equal $2\pi/3$, then x is coincident with y (z). Otherwise it is inside the triangle vyz. In any case x is not coincident with v, so $|xv| + |xy| + |xz| < |vv| + |vy| + |vz|$.

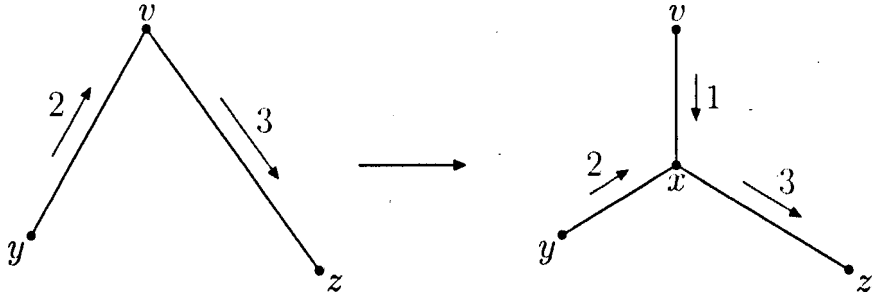

Figure 3.3: Reducing a graph when the flow in two edges with a small angle between them is directed in opposite ways. An example of a flow is given in grey.

Let M' be a multigraph created from M by introducing a new Steiner vertex x, removing edges e_1 and e_2 and adding edges xv, xy and xz (see Figure 3.3). The cost of M' is smaller that the cost of M, and the value is the same (the degree of v decreases by 1 and a new vertex with a degree 3 is added).

As the direction of the flow in e_1 and e_2 is opposite, the flow f can be modified to a feasible flow in M'. We set the flows on the edges xy and xz equal to the flow in e_1 and e_2 and use the additional edge vx to take care of the balance, which is no larger than k. We get contradiction, as such a graph M' cannot exist if M is a minimizer. □

Fact 3.11. *Let $M = (V, E)$ be a minimizer with a flow f. Let $v \in V$ and let $e_1, e_2 \in E$ be edges incident with v and directed in the same way. If the angle between e_1 and e_2 is smaller than $2\pi/3$, then $f(e_1) + f(e_2) > k$.*

Proof. If the angle between e_1 and e_2 is smaller than $2\pi/3$ and $f(e_1) + f(e_2) \leq k$, then the same reduction as in the proof of Fact 3.10 (Figure 3.3) gives a multigraph M' with the same value and a smaller cost than M. There is a feasible flow in M' — the amount of the balance flow on the added edge vx equals $f(e_1) + f(e_2) \leq k$. We get contradiction, as such a graph M' cannot exist if M is a minimizer. □

Fact 3.12. *Let $M = (V, E)$ be a minimizer with a flow f. Let $v \in V$ and let $e_1, e_2, e_3 \in E$ be edges incident with v contained in an open halfplane with v lying on its boundary. If e_1 and e_2 are directed in the same way and opposite to the direction of e_3, then $f(e_1) + f(e_2) - f(e_3) > k$ and, in particular, $f(e_3) < f(e_1), f(e_2)$.*

Proof. We can assume that the angle between the edges e_1 and e_3 is greater than or equal to the angle between e_2 and e_3. If the angle is the same, we can assume that the edge e_1 is at least as long as e_2. We call v_1, v_2, v_3 respectively the endpoints of e_1, e_2, e_3 different from v. We choose a point w in the following way: if the interval $[v_1, v_3]$ crosses the interval $[v, v_2]$, we set w to be the intersection point. Otherwise we set $w = v_2$.

Let M' be a multigraph obtained from M by removing the edges v_1v and v_3v and adding the edges v_1w and v_3w. If $w \neq v_2$ we also substitute the edge vv_2 with edges vw and wv_2 (see Figure 3.4).

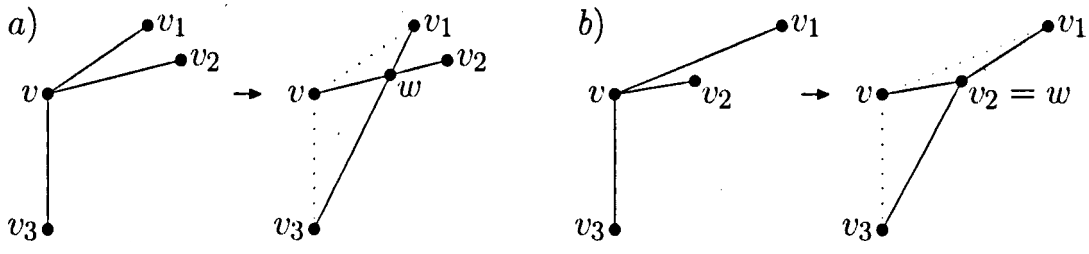

Figure 3.4: Reducing a multigraph in the proof of Fact 3.12.

The cost of M' is smaller than the cost of M. If $w \neq v_2$ then $|v_1w| + |v_3w| = |v_1v_3| < |v_1v| + |v_3v|$. If $w = v_2$ let z be the intersection point of the interval $[v, v_1]$ and the line containing v_2 and v_3. We have $|v_1w| + |v_3w| \leq |v_1z| + |zw| + |v_3w| = |v_1z| + |zv_3| < |v_1v| + |v_3v|$.

The value of M' is the same as the value of M. The degree of v drops by 2 and either we get a new Steiner vertex of degree 4 or the degree of v_2 increases by 2. In both cases the value remains unchanged.

If $f(e_1) + f(e_2) - f(e_3) \leq k$ then there is a feasible flow f' in M'. The flow on all the common edges of M and M' is the same as in f, there is a flow $f(e_1)$ on the added edge v_1w (the direction of the flow with respect to v_1 is the same as the direction of f on e_1), a flow $f(e_3)$ on the added edge v_3w (the direction of the flow with respect to v_3 is the same as the direction of f on v_3v), a flow $f(e_2)$ on wv_2 (if $w \neq v_2$; again the direction of the flow with respect to v_2 is the same as the direction of f on e_2) and a balancing flow $|f(e_1)+f(e_2)-f(e_3)|$ on the edge wv_2. That yields a solution with a smaller cost and the same value as M. We get contradiction, as such a solution cannot exist if M is a minimizer.

Therefore we must have $f(e_1) + f(e_2) - f(e_3) > k$ and $f(e_3) < f(e_1), f(e_2)$. □

Let $M = (V, E)$ be a minimizer. For a vertex v we call an edge e incident with v *incoming* (*outgoing*) if it is directed towards v (out of v).

Lemma 3.13. *Let $M = (V, E)$ be a minimizer. Let $v \in V$ be a Steiner vertex such that all outgoing (or incoming) edges have the same direction $\alpha \in [0, 2\pi)$. Then v is a real vertex or an optional vertex.*

Proof. Let f be the flow in the multigraph. Assume without losing generality that all outgoing edges have the same direction α. From Fact 3.10 there are no edges incident with v with directions in the interval $(\alpha - 2\pi/3, \alpha)$ or $(\alpha, \alpha + 2\pi/3)$.

Assume that v has only one outgoing edge e. The minimizer has no Steiner vertices of degree smaller than 3, so v has at least two incoming edges. The total flow entering v is the same as the total flow leaving v, so the amount of the flow entering v is not greater than k. From Fact 3.11 we get that the angle between any two incoming edges is at least $2\pi/3$. The only possibility is that there are exactly two incoming edges, with directions exactly $\alpha - 2\pi/3$ and $\alpha + 2\pi/3$. The flow in any of these edges is smaller than the flow in e. We get that v is a real vertex for $c = 0$.

Now assume that v has at least two outgoing edges. Let e_1 and e_2 be the two of them with the minimum amount of flow. Let e be any incoming edge with a direction different from α and $\alpha + \pi$. Then e, e_1 and e_2 are contained in an open halfplane with the borderline passing through v. From Fact 3.12 we get that $f(e) < f(e_1), f(e_2)$. Using Fact 3.12 again we get that for each of the intervals $(\alpha - \pi, \alpha)$ and $(\alpha, \alpha + \pi)$ there can be at most one incoming edge with a direction in the given interval. All the other edges have a direction $\alpha + \pi$.

As M is a minimizer, moving the Steiner vertex v by some distance Δ in the direction α or $\alpha + \pi$ does not decrease the cost. We have to consider the following cases.

1. The only edges incident with v have directions α and $\alpha + \pi$.
   The number of outgoing and incoming edges must be the same (and greater or equal 2) and v is an optional vertex.

2. There is exactly one incoming edge with a direction different from α and $\alpha + \pi$.
   This situation cannot happen. If the number of outgoing edges was at least as large as the number of incoming edges, moving v in the direction α would decrease the cost. Otherwise moving v in the direction $\alpha + \pi$ would decrease the cost.

3. There are exactly two incoming edges with directions different from $\alpha + \pi$ — one with a direction in $(\alpha - \pi, \alpha - 2\pi/3]$ and the other in $[\alpha + 2\pi/3, \alpha + \pi)$.
   Let c be the number of edges with a direction $\alpha + \pi$. The only possibility that moving v in any of the directions α and $\alpha + \pi$ does not increase the cost is when the number of edges with a direction α is $c + 1$. Then the single edges must have directions $\alpha - 2\pi/3$ and $\alpha + 2\pi/3$, because otherwise moving v in the direction $\alpha + \pi$ would decrease the cost. The vertex v is a real vertex.

□

Lemma 3.14. *Let $M = (V, E)$ be a minimizer. Let $v \in V$ be a Steiner vertex such that both incoming and outgoing edges have multiple directions. Then v is a crossing.*

Proof. Let f be the flow in the multigraph. Without losing generality we assume that the edge with a maximum flow is an outgoing edge. Let us denote it by e^1_{out}. Let $\alpha \in [0, 2\pi)$ be the direction of e^1_{out}. From Fact 3.12 we get that in each of the intervals $(\alpha - \pi, \alpha)$ and $(\alpha, \alpha + \pi)$ there can be at most one incoming edge.

Assume that there are no incoming edges with a direction $\alpha + \pi$. We have at most two incoming edges, each of them with a flow at most $f(e^1_{out})$. We must have exactly two incoming edges, as there are at least two different directions of incoming edges. Let us denote the two incoming edges by e^1_{in} and e^2_{in}. Let e^2_{out} be an outgoing edge different from e^1_{out}. Then one of the incoming edges, let's say e^1_{in}, is in the same halfplane with e^1_{out} and e^2_{out}. From Fact 3.12 we get that $f(e^1_{in}) < f(e^2_{out})$. We also know that $f(e^2_{in}) \leq f(e^1_{out})$, as the edge e^1_{out} has a maximum flow. We get that the total flow into v is smaller than the total flow out of v, which gives a contradiction.

We must have at least one incoming edge with a direction $\alpha + \pi$. Let us call it e^1_{in}. Assume that in one of the halfplanes $(\alpha - \pi, \alpha)$ or $(\alpha, \alpha + \pi)$ there is both an incoming edge e^2_{in} and an outgoing edge e^2_{out}. The edges e^1_{in}, e^2_{in} and e^2_{out} are in the same halfplane, so from Fact 3.12 we get that $f(e^2_{out}) < f(e^2_{in})$. The edges e^2_{in}, e^1_{out} and e^2_{out} are in the same halfplane, so from Fact 3.12 we get that $f(e^2_{in}) < f(e^2_{out})$.

That gives a contradiction. As we have both incoming and outgoing edges in more than one direction, we must have exactly one incoming edge in one of the halfplanes $(\alpha - \pi, \alpha)$ and $(\alpha, \alpha + \pi)$, and at least one outgoing edge in the other halfplane.

Assume that there are at least two outgoing edges e^2_{out} and e^3_{out} with directions different from α. The edges together with the incoming edge e^1_{in} (with a direction $\alpha + \pi$) are in one halfplane, so from Fact 3.12 we get that $f(e^1_{in}) < f(e^2_{out}), f(e^3_{out})$. Using Fact 3.12 again we get that there is only one incoming edge with the direction $\alpha + \pi$. There are exactly two incoming edges in total and we have $f(e^1_{in}) < f(e^2_{out})$ and $f(e^2_{in}) \leq f(e^1_{out})$. The total flow into v is smaller than the total flow out of v, which gives a contradiction.

Therefore we must have some outgoing edges in the direction α and one outgoing edge in a different direction, as well as some incoming edges with the direction $\alpha + \pi$ and one incoming edge with a different direction. One of the single edges is in the halfplane $(\alpha - \pi, \alpha)$ and the other in $(\alpha, \alpha + \pi)$.

As M is a minimizer, moving the Steiner vertex v by some distance Δ in the direction α or $\alpha + \pi$ does not decrease the cost. The only possible case is when the number of incoming edges equals the number of outgoing edges. Also the degree between the single outgoing and incoming edge must be π.

If the number of edges with a direction α (and also $\alpha + \pi$) is greater than one, then from Fact 3.12 we get that the flow in any single edge is smaller than the flow in any of the non-single edges with a flow directed in the opposite way. Vertex v is a crossing. □

From Lemma 3.13 and Lemma 3.14 we instantly get

Theorem 3.15. *Each Steiner vertex in a minimizer is a real vertex, a crossing or an optional vertex.*

3.2.3 Graph Analysis and Cycle Argument

From Theorem 3.15 we know that a Steiner vertex in a minimizer is a real vertex, a crossing or an optional vertex. We can easily remove optional vertices from a minimizer.

Lemma 3.16. *For any minimizer M there is a corresponding minimizer M' with the same cost, value, number of real vertices and crossings and with no optional vertices.*

Proof. We remove optional vertices one by one in the following way. Let v be any optional vertex and w a neighbor of v closest to v. We move v towards w and merge the two vertices together. As M is a minimizer, the vertices v and w were connected

by one edge (otherwise merging would decrease the value), and the merged vertex does not change its type (e.g. a real vertex remains a real vertex). The operation does not change the cost, value and the number of real vertices and crossings of the multigraph. □

In this section we prove Theorem 3.5 — for any instance I of the CGND problem we show the existence of a minimizer which gives a $(1+\varepsilon)$-approximate solution and has a small number of Steiner vertices. To do it, we introduce a procedure modifying minimizers to decrease the number of real vertices. We also show that if we cannot decrease the number of real vertices any further, the total number of Steiner vertices must be small.

Definition 3.17. *Let M be a minimizer without optional vertices. A cycle C in M is called a* Steiner cycle *if:*

- *C passes only through Steiner vertices,*
- *if C passes through a crossing vertex v, then C does not change the direction in v (i.e. the angle between two consecutive edges of C incident with v is π),*
- *if C passes through a real vertex v, then C either does not change the direction, or changes it by $\pi/3$ (i.e. the angle between two consecutive edges of C incident with v is either π or $2\pi/3$),*
- *C does not pass through an edge more than once.*

Lemma 3.18. *Let M be a minimizer without optional vertices, V_S the set of Steiner vertices of M and V_C the set of points on the plane where the edges of M cross without a Steiner vertex. If M has no Steiner cycle, then $|V_S| + |V_C| \leq 32D^4 + 4D^2$.*

Proof. Let M' be a multigraph obtained from M by performing the following operations. We remove all crossings and join pairs of edges that were incident with a crossing and had opposite directions arbitrarily into single edges (see Figure 3.5*a*). We also decrease the degree of every real vertex to 3 by pairing arbitrarily all the c edges with a direction α with any c edges with a direction $\alpha + \pi$ and merging them into single edges (see Figure 3.5*b*).

We then split each non-Steiner vertex v into $\deg(v)$ vertices of degree 1. As in the original multigraph there are at most $2D$ non-Steiner vertices and each of them has a degree at most $2D$, the number of vertices with a degree 1 after splitting is at most $4D^2$. A simple cycle in M' is a Steiner cycle in M. As there is no Steiner cycle in M, the resulting graph M' is a forest with at most $4D^2$ leaves and all inner

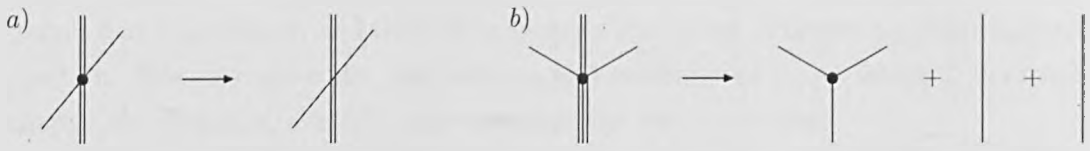

Figure 3.5: Removing a crossing and decreasing degree of a real vertex.

vertices of degree 3. Therefore the number of real vertices in M' (and also in M) is upper bounded by $4D^2$, and the number of all vertices in M' is upper bounded by $8D^2$. The number of edges in M' is also upper bounded by $8D^2$. Each of the at most $32D^4$ pairs of edges in M' can generate at most one crossing in M or one point in V_C. As each Steiner vertex in M is either a real vertex or a crossing, we get $|V_S| + |V_C| \leq 32D^4 + 4D^2$. □

We introduce an operation of *shifting* a Steiner cycle $\mathcal{C}$ by some distance $\Delta > 0$ in a minimizer M with no optional vertices. First we orient the edges of $\mathcal{C}$ in one of the two possible directions. We then shift each edge of the cycle by a distance Δ to the left according to the direction chosen. The real vertices where the cycle changes direction are moved to the intersection points of the shifted edges (see Figure 3.6). Notice that a Steiner cycle has no repeating edges, so we never have to shift an edge in two directions at once.

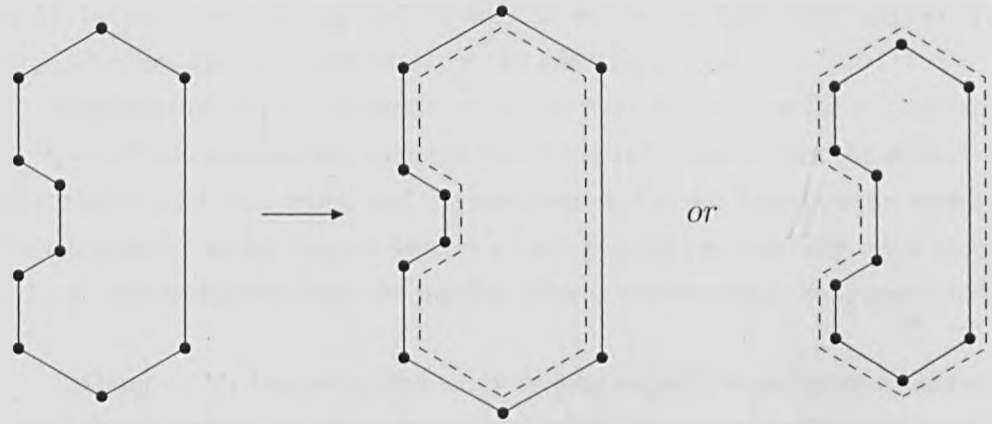

Figure 3.6: Shifting a Steiner cycle. Only the real vertices where the cycle changes direction have been pictured. The obtained cycle depends on the orientation of the original cycle.

Moving the real vertices as described above requires us to make some other modifications to the graph. We do not want to shift or change directions of edges not belonging to $\mathcal{C}$ — they can only become longer or shorter. To achieve this, in some cases instead of moving a real vertex we split the vertex into a real vertex of

degree 3 and a crossing, and then we only move the vertex of degree 3 to the required position. We also move the real vertices and crossings of $\mathcal{C}$, on which $\mathcal{C}$ does not change the direction, possibly after splitting the vertices in two.

The case analysis in Section 3.2.4 shows that shifting a Steiner cycle neither changes the cost and the value of the minimizer nor the number of the real vertices. The resulting graph has a feasible flow and is a minimizer.

When a vertex from $\mathcal{C}$ hits a vertex with which it is incident (i.e. after we move a vertex v from $\mathcal{C}$, it becomes coincident with some vertex w such that there is an edge connecting v and w), we merge the two vertices. In Section 3.2.5 we show that the shifting operation can be performed until one of the following happens: a vertex from $\mathcal{C}$ gets merged with a non-Steiner vertex or two real vertices get merged (e.g. when a cycle edge gets contracted to a single point). The resulting multigraph has the same cost and value and either a smaller number of real vertices, or the same number of real vertices and a larger sum of degrees of non-Steiner vertices.

We are now ready to prove Theorem 3.5.

Proof of Theorem 3.5. Fix an $\varepsilon > 0$. By Lemma 3.4, there is a $(1+\varepsilon)$-approximate solution which is a minimizer. Let M_0 be such a minimizer that also minimizes the number of real vertices — we cannot decrease the number of real vertices of M_0 without increasing the cost or the value of M_0. Let M_1 be the minimizer obtained from M_0 by removing the optional vertices, as in Lemma 3.16. The number of real vertices, the cost and the value of M_1 is the same as in M_0.

Suppose that M_1 has a Steiner cycle. We shift the cycle as far as possible. As a result, we obtain a minimizer with the same cost and value, where the shifted cycle is not a Steiner cycle any more, and the resulting multigraph has the same number of real vertices and a larger sum of degrees of non-Steiner vertices (any other result of shifting a cycle would decrease the number of real vertices, and that cannot happen for M_1).

As long as M_1 has some Steiner cycle left, we shift it as far as possible. We can perform this operation only a finite number of times, as each time the sum of the degrees of the non-Steiner vertices increases, and in a minimizer it is upper bounded by $4D^2$. At some point there will be no Steiner cycle left. The graph obtained from M_1 will be a minimizer with no Steiner cycles, and from Lemma 3.18 we know that in such a minimizer $|V_S| + |V_C| \leq 32D^4 + D^2$. □

3.2.4 Detailed Analysis of Shifting a Steiner Cycle

We have to consider the following cases:

- Moving a real vertex v of degree 3.

  The operation is shown on Figure 3.7. The vertex v is moved to the desired place along the non-cycle edge of v.

Figure 3.7: Moving a real vertex v of degree 3. The bold edges represent the cycle edges. On the right hand-side the dashed edges and the white dots show the previous positions of the edges and vertices.

The non-cycle edge becomes longer or shorter and balances the change of the length of the cycle edges (see Figure 3.8). The cost and the value of the solution do not change. The same flow is feasible.

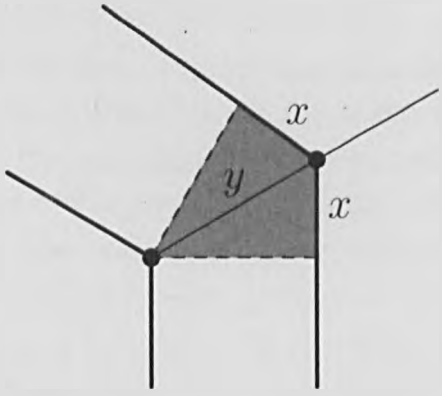

Figure 3.8: The change of the length of the non-cycle edges (y) balances the change of the length of the cycle edges ($2x$). The bold edges represent the cycle edges before and after the shifting operation. Dashed lines are orthogonal to the cycle edges. As the shaded triangles are right-angled triangles with the other angles $\pi/6$ and $\pi/3$, we get that $x/y = 1/2$.

- Moving a real vertex v of degree greater than 3 when the cycle goes along the two single edges.

  The new position of the real vertex is on one group of non-cycle edges incident with v. We move the vertex v along this group of edges into the required position. The operation is shown on Figure 3.9. Some of the non-cycle edges become longer and some shorter. The total cost does not change, as the changes in the lengths of the non-cycle edges balance the change of cost of the cycle

edges, as in the previous case. The value does not change and the same flow is feasible.

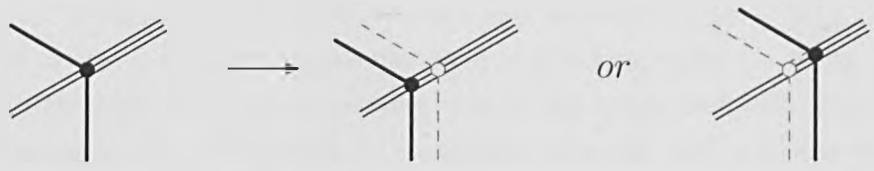

Figure 3.9: Moving a real vertex of degree greater than 3 when the cycle goes along the single edges.

- Moving a real vertex v of degree greater than 3 when the cycle goes along one single edge and one group of edges.

  This situation requires splitting the vertex v into two vertices connected with an edge: a real vertex v_r of degree 3 and a crossing v_c of degree $\deg(v) - 1$. We direct the edge $v_r v_c$ so that its direction with respect to v_r is the same as the direction of the single edges with respect to v.

  We have to consider two cases. The first case (the middle picture in Figure 3.10) is when the real vertex is moved towards the smaller angle created by the cycle edges. In this case the real vertex v_r is moved and the crossing v_c stays on the previous place of v. The vertex v_r is incident with the two cycle edges and with the edge $v_r v_c$. The vertex v_c is incident with all the non-cycle edges of v and with the edge $v_r v_c$. Notice that v_r lies on a line containing the non-cycle single edge of v, so v_c is a crossing (the directions of the edges incident with v_c satisfy the conditions of a crossing). As one of the cycle edges is incoming to v_r and the other one is outgoing, and the amount of flow in a single edge is smaller than in the one from a group of edges, the balance flow that needs to be sent through the edge $v_r v_c$ is at most k and its direction is consistent with the direction of the edge $v_r v_c$. There is a feasible flow in the modified part of the graph.

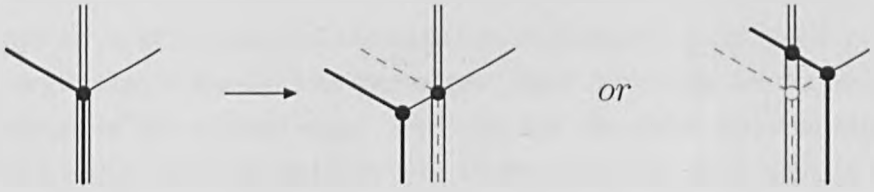

Figure 3.10: Moving a real vertex v of degree greater than 3 when the cycle goes along one single edge and one group of edges.

The second case (right picture in Figure 3.10) is when the real vertex is moved

towards the larger angle created by the cycle edges. In this case the real vertex v_r is moved towards its required position, and the vertex v_c is moved along the group of edges incident with v to the new intersection point with the cycle. The vertex v_r is incident with the cycle edge belonging to the group of edges, with the edge $v_r v_c$ (which becomes part of the cycle) and with the non-cycle single edge of v. The vertex v_c is incident with $v_r v_c$ and with the remaining edges that were incident with v. Vertex v_c is a crossing. The cycle now passes through both v_r and v_c. Similarly as before, the balance flow that needs to be sent between v_r and v_c is at most k and its direction is consistent with with the direction of the edge $v_r v_c$, so there is a feasible flow.

In both cases the cost of the solution stays the same (the changes in the lengths of the edges are the same as in the earlier cases considered). The value also stays the same, as $\deg(v_r) = 3$ and $\deg(v_c) = \deg(v) - 1$.

The other vertices we have to consider are the real vertices and crossings that belong to the shifted cycle $\mathcal{C}$, on which the cycle does not change the direction. We deal with these vertices in the following way.

- The single edges of a crossing are shifted.

  In this case the crossing is moved accordingly (see Figure 3.11). The cost, value and flow stay the same.

Figure 3.11: Moving a crossing.

- Two of the multiple edges of a crossing are shifted.

  A new crossing is created at the additional intersection point (see Figure 3.12). A single edge is divided into two edges. Their directions are the same as the direction of the original edge. The cost and the value stay the same. The new crossing vertex is incident with three edges that were incident with the previous crossing: two cycle edges and a single edge. The flow in the cycle edges is directed in the opposite ways. The flow in the single edge is smaller than the flow in the cycle edge where the flow is directed in the opposite way. We get that the amount of flow needed to be transferred between the two

crossings is at most k and its direction is consistent with the direction of the edge, so there is a feasible flow in the modified part of the graph.

Figure 3.12: Splitting a crossing into two crossings.

- Two edges of a real vertex with opposite directions are shifted (a cycle was going straight through the real vertex).

  The real vertex is not moved. A new crossing is created at the intersection point of the shifted cycle edges with a single edge incident with the real vertex (see Figure 3.13). A single edge is divided into two edges. Their directions are the same as the direction of the original edge. The cost and the value stay the same. The same flow is feasible as again the flow on a single edge of a real vertex is smaller than the flow on any other edge where the flow is directed in the opposite way. The amount of flow that needs to be transferred between the two vertices is at most k and its direction is consistent with the direction of the edge, so there is a feasible flow in the modified part of the graph.

Figure 3.13: Splitting a real vertex into a real vertex and a crossing.

3.2.5 Detailed Analysis of Shifting a Steiner Cycle — Stopping Conditions

In the analysis we use the following properties. If two vertices are coincident and connected by an edge, they can be merged into one vertex without increasing the cost or the value. All Steiner vertices in a minimizer have edges incident with them in at most four different directions. If as a result of merging two coincident Steiner vertices we get a vertex with edges leaving in at least five directions, the resulting

multigraph, as well as the original multigraph, is not a minimizer. If a Steiner vertex in a minimizer has two batches of edges (groups of at least two edges with the same direction), then the angle between the batches is π. If as a result of merging two coincident Steiner vertices we get a vertex with two non-opposite batches of edges, the resulting multigraph, as well as the original multigraph, is not a minimizer.

We assume that the original graph is a minimizer. We check which of the following situations can happen. Notice that the only vertices that are moved when a cycle is shifted are real vertices in which the cycle changes direction and crossings.

1. A crossing hits a crossing with which it is incident.

   We merge the two crossings into one vertex. The only case when the resulting vertex has only four incident directions is when the two crossing have edges incident with them with the same directions. As we cannot obtain two non-opposite batches of edges, the edge along which the crossing was moved was a single edge. This case is pictured in Figure 3.14. After hitting the intersection point we can continue shifting the cycle.

Figure 3.14: Moving a crossing over a crossing.

2. A crossing hits a real vertex with which it is incident.

   We merge the two vertices into one vertex. The only case when the resulting vertex has only four incident directions is when a group of edges incident with the real vertex (or one edge if the degree of the real vertex is 3) is parallel to the cycle edge. This case is pictured in Figure 3.15. After hitting the intersection point we can continue shifting the cycle.

3. A crossing hits a non-Steiner vertex with which it is incident.

   We merge the two vertices into one non-Steiner vertex. We do not shift the cycle any further, as it is not a Steiner cycle any more (the cycle crosses a non-Steiner vertex). The degree of the non-Steiner vertex increases, but it is still upper-bounded by $2D$, as the graph is a minimizer.

Figure 3.15: Moving a crossing over a real vertex.

4. A real vertex hits a crossing with which it is incident.

   We merge the two vertices into one vertex. The only case when the resulting vertex has only four incident directions is when a group of the edges incident with the crossing (or an edge if the degree of a crossing is 4) is parallel to one of the cycle edges (see Figure 3.16). In this case the merged vertices create a real vertex and the cycle can be shifted further.

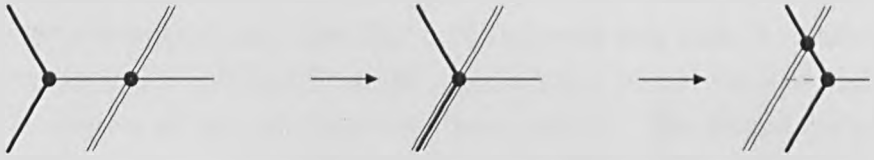

Figure 3.16: Moving a real vertex over a crossing.

5. A real vertex hits a real vertex with which it is incident.

   We merge the two vertices into one vertex. The only case when the resulting vertex has only four incident directions and there are no non-parallel batches of edges is shown in Figure 3.17. In this case the merged real vertices create a single crossing — the number of real vertices decreases. We do not shift the cycle any further, as it is not a Steiner cycle any more (it changes direction in a crossing).

Figure 3.17: Merging two real vertices into a crossing.

6. A real vertex hits a non-Steiner vertex with which it is incident.

   We merge the two vertices into one non-Steiner vertex. We do not shift the cycle any further, as it is not a Steiner cycle any more (the cycle crosses a

non-Steiner vertex). The number of real vertices decreases.

7. A cycle edge is contracted to a zero-length edge.

   We merge the two real vertices that are the endpoints of the edge into a single vertex. We get a crossing, similarly as in Figure 3.17. The number of real vertices decreases. We do not shift the cycle any further, as it is not a Steiner cycle any more (it changes direction in a crossing).

After performing the above operations there is still a feasible flow in the graph. Each operation consists of merging two incident vertices and then possibly splitting a real vertex or a crossing into two vertices, in the same way as considered in Section 3.2.4. The flow that was feasible before the operation is still feasible in the modified graph.

We shift the cycle as far as possible. The only possibilities that we cannot shift the cycle any further is when we obtain one of the cases 3, 5, 6 and 7. As a result we get a minimizer with the same cost and value and either a smaller number of real vertices (cases 5, 6 and 7) or the same number of real vertices and a larger sum of the degrees of the non-Steiner vertices (case 3). The shifted cycle is not a Steiner cycle any more — it either crosses a non-Steiner vertex or changes direction in a crossing.

3.3 A Quasi-Polynomial Time Approximation Scheme

In this Section we consider the capacitated geometric network design problem with total demand $D \leq n^{\mathfrak{c}}$ for some constant $\mathfrak{c} > 0$. We design a quasi-polynomial time approximation scheme that works for an arbitrary edge capacity k. We can assume that $k \leq D$.

In Section 3.3.4 we present a modified quasi-polynomial time approximation scheme for a version of the CGND problem when the sinks have unbounded demands.

From Theorem 3.6 we know that for any instance of the CGND problem there is an optimal solution $M_{OPT} = (V, E)$ that satisfies the following properties: the number of Steiner vertices is at most $32D^4 + 4D^2$, there are no two coincident vertices, the degree of each vertex is at most $2D$ and the edges cross only in vertices. We have $|V| \leq 38n^{4\mathfrak{c}}$, $|E| \leq 38n^{5\mathfrak{c}}$. These bounds make it possible to use Arora's framework [10] for geometric optimization problems.

3.3.1 Dividing into Subproblems

Lemma 3.19. *Let $I = (\mathcal{S}, \mathcal{T}, \mathfrak{d}, k)$ be an instance of the CGND problem. In polynomial time we can partition the set of input points $\mathcal{S} \cup \mathcal{T}$ into sets $\mathcal{S}_i \cup \mathcal{T}_i$ such that every optimal solution consists of the disjoint solutions for the problem instances $I_i = (\mathcal{S}_i, \mathcal{T}_i, \mathfrak{d}|_{\mathcal{S}_i \cup \mathcal{T}_i}, k)$, and the cost of an optimal solution for any instance I_i is at least $\frac{\ell_i}{k|\mathcal{S}_i \cup \mathcal{T}_i|}$, where ℓ_i is the side length of the smallest square containing all points from the set $\mathcal{S}_i \cup \mathcal{T}_i$.*

We first show the following result.

Lemma 3.20. *One can find in polynomial time a k-approximate solution to an instance of the CGND problem.*

Proof. We split each source and sink into a set of coincident sources or sinks with demand 1. As a result we get the set of at most D *unit* (i.e., with demand 1) sources and sinks. With each solution we can associate a matching between the unit sinks and sources, describing where the flow from a given source ends up. Each matching $\mathcal{M}$ has a corresponding cost $\mathsf{cost}(\mathcal{M})$, which is the sum of the distances between the matched pairs. As the edges have capacity k, a solution to which corresponds the matching $\mathcal{M}$ has cost at least $\frac{\mathsf{cost}(\mathcal{M})}{k}$.

Finding a minimum cost matching and connecting the source-sink pairs by direct edges gives a solution with a cost $\mathsf{cost}(\mathcal{M}) \leq k \cdot OPT$. □

Proof of Lemma 3.19. Let $\mathrm{ap}(I)$ be the cost of the k-approximate solution for I from Lemma 3.20. We have

$$OPT(I) \leq \mathrm{ap}(I) \leq k \cdot OPT(I) \ .$$

If the distance between two input points is more than $\mathrm{ap}(I)$, in an optimal solution the points have to be in different connected components. We create a graph on the vertex set $\mathcal{S} \cup \mathcal{T}$ by connecting each pair of vertices with an edge if the distance between them is at most $\mathrm{ap}(I)$. We divide the set $\mathcal{S} \cup \mathcal{T}$ into subsets, each containing vertices from one connected component of the graph, and create a separate problem instance $I_i = (\mathcal{S}_i, \mathcal{T}_i, \mathfrak{d}|_{\mathcal{S}_i \cup \mathcal{T}_i}, k)$ for each subset $\mathcal{S}_i \cup \mathcal{T}_i$.

As long as the resulting operation returns more than one problem instance, we repeat it with each instance as the input. At some point the operation ends and we get a collection of problem instances $I_1, \ldots, I_j$, representing a partition of the set $\mathcal{S} \cup \mathcal{T}$ into subsets $\mathcal{S}_i \cup \mathcal{T}_i$. By the construction of the instances I_i we know that any optimal solution for I consists of disjoint solutions for each I_i. All the points from

$\mathcal{S}_i \cup \mathcal{T}_i$ are contained in a square of length

$$\ell_i \leq |\mathcal{S}_i \cup \mathcal{T}_i| \cdot \mathrm{ap}(I_i) \leq |\mathcal{S}_i \cup \mathcal{T}_i| \cdot k \cdot OPT(I_i) \ .$$

That gives the required lower bound on the cost of an optimal solution for each instance I_i. □

We divide the original problem instance into a collection of independent instances, according to Lemma 3.19. We can now assume that all the input points are inside a square $\ell \times \ell$ and the cost of an optimal solution is at least $\frac{\ell}{nk}$.

3.3.2 Perturbation, Randomized Dissection and Portals

We divide the $\ell \times \ell$ square into *unit squares* using a uniform grid of size polynomial in n and D. We modify the problem by shifting all input points to the nearest gridpoint (i.e. intersection of grid lines), possibly merging coincident sinks and sources into a single sink or source or removing them if the balance flow is 0.

The multigraph M_{OPT} is an optimal solution for the instance of the CGND problem as in Theorem 3.6. Let M'_{OPT} be a multigraph obtained from M_{OPT} by moving each vertex to the nearest gridpoint and merging vertices that became coincident by this operation. Notice that the edges in M'_{OPT} cross only in Steiner vertices, similarly as the edges of M_{OPT}.

Lemma 3.21. *If the size of the grid is at least* $\frac{80}{\varepsilon} \cdot n^{6\mathfrak{c}+1} \times \frac{80}{\varepsilon} \cdot n^{6\mathfrak{c}+1}$, *we have*

$$C(M'_{OPT}) \leq (1+\varepsilon) \cdot C(M_{OPT}) \ .$$

Proof. For the multigraph $M_{OPT} = (V, E)$ we have $|E| < 40n^{5\mathfrak{c}}$. A vertex can be moved by a distance at most $\frac{\varepsilon\ell}{80n^{6\mathfrak{c}+1}}$, so the total change in the length of the edges is at most $2|E| \cdot \frac{\varepsilon\ell}{80n^{6\mathfrak{c}+1}} < \frac{\varepsilon\ell}{n^{\mathfrak{c}+1}} \leq \frac{\varepsilon\ell}{nk}$. On the other hand, $C(M_{OPT}) \geq \frac{\ell}{nk}$, and the total change of cost is at most $\varepsilon \cdot C(M_{OPT})$. □

We make a randomized dissection of the square — a recursive partitioning of the square into four equal squares, performed until we obtain the unit squares. The size of the grid is the smallest power of 2 which is at least $2 \cdot \frac{80}{\varepsilon} \cdot n^{6\mathfrak{c}+1}$. Also, instead of starting with the original $\ell \times \ell$ square, we start with a $2\ell \times 2\ell$ square with the original square shifted randomly inside. A more detailed description of the randomized dissection can be found in [10; 38], as well as in Section 2.3.2. The number of levels of the dissection is $\mathfrak{L} = O(\log(2 \cdot \frac{80}{\varepsilon} \cdot n^{6\mathfrak{c}+1})) = O(\mathfrak{c} \cdot \log \frac{n}{\varepsilon})$.

Let m be the smallest power of 2 such that $m \geq \frac{2\mathfrak{L}}{\varepsilon} = O(\frac{\mathfrak{c}}{\varepsilon} \log \frac{n}{\varepsilon})$. Similarly as in [10; 38] and in Section 2.3.2, on the boundary of each dissection square we put $m+1$ equidistant portals, in such a way that the corners of the dissection squares are covered by the portals.

We modify the solutions to the CGND problem in such a way, that the edges can cross the boundaries of the squares only in portals (i.e. we create additional Steiner vertices in portals). Notice that all the input points are in portals. We do not allow any Steiner vertices outside of portals, and we do not allow the edges to cross outside of the portals. In the remaining part of this section we will present a quasi-polynomial time algorithm that finds an optimal solution to the modified problem.

First we show the following lemmas.

Lemma 3.22. *An optimal solution to the modified problem has expected cost at most* $(1+\varepsilon)^2 \cdot C(M_{OPT})$.

Proof. Let us scale the distances such that the length of the unit square side is 1. Consider the multigraph M'_{OPT} and a multigraph M''_{OPT} obtained from M'_{OPT} by substituting all edges with portal-respecting paths. Notice that all the Steiner vertices of M''_{OPT} are in portals and the edges do not cross outside of the portals.

Let xy be an edge in M'_{OPT}. The number of times where the edge xy crosses a boundary of some dissection square is at most $2|xy|$. The expected increase in cost caused by crossing a single boundary is not greater than $\sum_{i=1}^{\mathfrak{L}} \frac{1}{2^i} \cdot \frac{2^i}{m} = \frac{\mathfrak{L}}{m}$. The edge $|xy|$ is modified into a path of the expected cost at most $|xy| + 2|xy| \cdot \frac{\mathfrak{L}}{m} \leq |xy|(1+\varepsilon)$ and the expected cost of the multigraph M''_{OPT} is at most $(1+\varepsilon) \cdot C(M'_{OPT}) \leq (1+\varepsilon)^2 \cdot C(M_{OPT})$. Therefore the expected cost of the optimal solution is at most $(1+\varepsilon)^2 \cdot C(M_{OPT})$. □

Lemma 3.23. *From an optimal solution to the modified problem we can get a solution for the original problem with an expected cost at most* $(1+\varepsilon)^3 \cdot C(M_{OPT})$.

Proof. Let M be an optimal solution to the modified problem. From Lemma 3.22 we know that the expected cost of M is at most $(1+\varepsilon)^2 \cdot C(M_{OPT})$. To obtain from M a solution to the original problem it is enough to put a Steiner vertex into the position of each sink and source after the perturbation and connect these points with the sinks and sources of the original problem by as many single edges as required by the demand of the sink or source.

We have to add at most $2n^{\mathfrak{c}}$ edges, each of length at most $\frac{\varepsilon \ell}{80n^{6\mathfrak{c}+1}}$, so the cost of the obtained solution is at most $C(M) + 2n^{\mathfrak{c}} \cdot \frac{\varepsilon \ell}{80n^{6\mathfrak{c}+1}} \leq C(M) + \varepsilon \cdot C(M_{OPT})$. The expected cost of the obtained solution is at most $(1+\varepsilon)^3 \cdot C(M_{OPT})$. □

3.3.3 Dynamic Programming

We want to find an optimal solution to the modified problem, i.e., where all sinks and sources are in portals, the Steiner vertices can be created only in portals and the edges do not cross outside of the portals. We can consider only solutions (multigraphs with a specified flow) where the flow through each portal goes in one direction only (into or out of the square) and the amount of it is at most D.

We find an optimal solution using dynamic programming. The interface of a square specifies the amount of flow and the direction of it (into or out of the square) for each portal. The number of interfaces of a square is $(1+2D)^{4m} = n^{O(c^2\varepsilon^{-1}\cdot\log(n/\varepsilon))}$, which is quasi-polynomial in n. For each square and for each interface we want to find the solution with the minimum cost satisfying the interface (if there is a feasible solution for a given interface). Since we have doubled the size of the original square enclosing the input points, we may assume that no input points are on the perimeter of this square. Hence, the solution to the problem is the minimum cost solution for the doubled original square with an empty interface.

We start with finding the minimum cost solutions to the interfaces of the unit squares.

Lemma 3.24. *The minimum cost solutions to the leaf-subproblems, i.e., to the interfaces of the unit squares, can be found in time* $n^{O(c^2\varepsilon^{-1}\log(n/\varepsilon))}$.

Proof. As we consider only solutions where all sinks, sources and Steiner vertices are in portals, and the edges do not cross outside of the portals, the edges inside the unit squares do not cross. That means that in each solution the edges go along some triangulation of the square, where the vertices of the triangulation are the portals. The number of triangulations for a given interface I is a Catalan number, it is smaller than $4^{4m} = (\frac{n}{\varepsilon})^{O(c\varepsilon^{-1})}$ and we can list all of them in time $4^{O(m)}$.

For each of the aforementioned triangulations Q, we generate all possible multigraphs whose multiedges are in one-to-one correspondence with the edges of Q and have multiplicity in the range $[0, \lceil D/k \rceil]$.

The number of such multigraphs corresponding to Q is $n^{O(cm)}$ and they can be generated in time $n^{O(cm)}$. Next, for each of the multigraphs we verify whether or not it is a feasible solution to the interface I and if so, compute its length.

To perform the test, for each multiedge of the multigraph, we define its capacity as the product of its multiplicity and k. Now, it is sufficient to run a maximum flow algorithm to see if there is an integral flow from the sources to the sinks equal to the total demand of the sources in the original multigraph. Thus, the test and computation of the total length take $O(m^3)$ time per multigraph.

Finally, we return the shortest among the feasible multigraphs for I as a solution to I. We conclude that we can find a solution to a given interface in time $n^{O(cm)}$.

Since the number of interfaces is also $n^{O(cm)}$, and $m = O(\frac{c}{\varepsilon} \log \frac{n}{\varepsilon})$, the lemma follows. □

Lemma 3.25. *The minimum cost solutions to the remaining subproblems, i.e., to the interfaces of the non-unit dissection squares can be found in time* $n^{O(c^2\varepsilon^{-1}\log(n/\varepsilon))}$.

Proof. For the non-unit squares computing the solutions for all interfaces is straightforward. Each such a square consists of four smaller squares, for which the solutions have been computed earlier. We consider all possible configurations of interfaces of the four squares, check whether they are consistent (i.e. in the Steiner points inside the larger square — the places where the portals of the smaller squares meet — the total flow must balance with the demands of the sinks or sources located there). We know the cost of the solutions for the smaller squares, so we can compute the cheapest solutions for all feasible interfaces of the larger square. For each square it is done in time $n^{O(c^2\varepsilon^{-1}\cdot\log(n/\varepsilon))}$ and the number of dissection squares is $\frac{1}{\varepsilon} \cdot n^{O(c)}$. □

By applying Lemmas 3.23 – 3.25 with ε divided by an appropriate constant, we obtain our main theorem in this section.

Theorem 3.26. *For any* $\varepsilon > 0$, *there is a randomized* $n^{O(c^2\varepsilon^{-1}\log(n/\varepsilon))}$*-time algorithm for capacitated geometric network design with a total of* n *sources and sinks and total demand upper bounded by* n^c, *which yields a solution whose expected cost is within* $(1+\varepsilon)$ *of the optimum.*

The QPTAS of Theorem 3.26 can be derandomized as the one in [10].

3.3.4 A QPTAS for Unlimited Demands of Sinks

The approximation scheme is very similar to the previous one. We know that there is an optimal solution satisfying the same properties as before (from Theorem 3.6). The only modifications in the approximation scheme are:

- We use a different k-approximate solution from the one in Lemma 3.20 — connecting each source to the closest sink.
- In the dynamic programming part, if a portal contains a sink, an arbitrary amount of flow going into the portal can be balanced with the sink. We also admit no flow coming to the sink (i.e. it can be an isolated vertex).

We get the following result

Theorem 3.27. *For any $\varepsilon > 0$, there is a randomized $n^{O(c^2\varepsilon^{-1}\log(n/\varepsilon))}$-time algorithm for capacitated geometric network design with unlimited demands of the sinks, with a total of n sources and sinks and total demand of the sources upper bounded by n^c, which yields a solution whose expected cost is within $(1+\varepsilon)$ of the optimum.*

3.4 A Polynomial Time Approximation Scheme for Single Sink

In this section we consider the single-sink capacitated geometric network design problem with a total demand $D \leq n^c$ for some constant c. Throughout this section we assume that the edge capacity $k \leq 2^{O(\sqrt{\log n})}$. We design a polynomial time approximation scheme for this problem.

The idea of the algorithm is as follows. We partition the set of sources $\mathcal{S}$ into two subsets, depending on their distance from the sink t. The set of outer sources has a total demand upper bounded by a polynomial function of k. Using the algorithm from Section 3.3 we construct a near-optimal network which can transfer the flow from the outer sources to some Steiner vertices closer to the sink. These Steiner vertices now become new sources. As the number of points and the total demand is much smaller than n, the running time is polynomial in n. We then solve the problem for the inner sources together with the new sources. We show that for such a set of points the cost of the minimum TSP tour is small compared to the cost of an optimal solution. That allows us to use a simple algorithm to find a near-optimal solution. Combining the two networks together gives a near-optimal solution to the SCGND problem.

Let L be the maximum distance between a source and the sink t, i.e. $L = \max_{s\in\mathcal{S}} \delta(s,t)$. Let OPT be an optimal solution for the SCGND problem, and TSP be the shortest traveling salesman tour for $\mathcal{S} \cup \{t\}$.

Similarly as for the CVRP problem in Section 2.2, we can show the following result.

Lemma 3.28. *There is a polynomial time algorithm that outputs a solution with cost at most*

$$(1+\varepsilon)\cdot C(TSP) + \frac{1}{k}\sum_{s\in\mathcal{S}} \delta(s,t)\cdot \mathfrak{d}(s) \ .$$

Furthermore, the following inequalities hold

$$\frac{1}{k}\sum_{s\in\mathcal{S}}\delta(s,t)\cdot\mathfrak{d}(s)\leq C(OPT)\leq C(TSP)+\frac{1}{k}\sum_{s\in\mathcal{S}}\delta(s,t)\cdot\mathfrak{d}(s)\ .$$

Proof. Sending a unit of flow from a source $s\in\mathcal{S}$ to the sink t requires the use of some edges of the network with total length at least $\delta(s,t)$. As each edge can be used by at most k units of flow, we obtain $C(OPT)\geq\frac{1}{k}\sum_{s\in\mathcal{S}}\delta(s,t)\cdot\mathfrak{d}(s)$.

Given a TSP tour T on $\mathcal{S}\cup\{t\}$, we can find a solution to the SCGND problem with cost upper bounded by $C(T)+\frac{1}{k}\sum_{s\in\mathcal{S}}\delta(s,t)\cdot\mathfrak{d}(s)$ as follows. We consider a point s on the tour T as $\mathfrak{d}(s)$ consecutive points, connected by edges of length 0. We divide the tour into consecutive paths, each containing k consecutive points (except of the last path which might be shorter). Then, for each path, except for the first and the last one (which are already connected to the sink), we connect to the sink the point that is closest to the sink. The cost of this network is at most $C(T)+\frac{1}{k}\sum_{s\in\mathcal{S}}\delta(s,t)\cdot\mathfrak{d}(s)$. There is a feasible flow in the network, as it consists of trees, each of them containing the sink t and at most k unit sources from $\mathcal{S}$. The trees can be used to transfer the flow from the sources to the sink. Taking an optimal TSP tour as T gives a bound $C(OPT)\leq C(TSP)+\frac{1}{k}\sum_{s\in\mathcal{S}}\delta(s,t)\cdot\mathfrak{d}(s)$.

If we take as T a $(1+\varepsilon)$-approximate solution to the TSP problem, which can be found in polynomial time [10; 69], we obtain the desired algorithm. □

If $C(TSP)<\frac{\varepsilon}{k}\sum_{s\in\mathcal{S}}\delta(s,t)\mathfrak{d}(s)$, Lemma 3.28 gives a polynomial time $(1+\varepsilon)$-approximation. Therefore, from now on we assume $C(TSP)\geq\frac{\varepsilon}{k}\sum_{s\in\mathcal{S}}\delta(s,t)\mathfrak{d}(s)$.

Working with this assumption, we use the following lemma to upper bound the sum of the distances from the sources to the sink.

Lemma 3.29. *If $C(TSP)\geq\frac{\varepsilon}{k}\sum_{s\in\mathcal{S}}\delta(s,t)\cdot\mathfrak{d}(s)$ then there is a constant β such that*

$$\sum_{s\in\mathcal{S}}\delta(s,t)\cdot\mathfrak{d}(s)\leq\left(\frac{\beta\cdot k}{\varepsilon}\right)^2\cdot L\ .$$

Proof. In the proof, we use the following lemma developed by Haimovich and Rinnooy Kan for the capacitated vehicle routing problem. The proof of the lemma can be found in [52].

Lemma 3.30 ([52], Theorem 3). *There is a constant β such that*

$$C(TSP)\leq\beta\cdot\sqrt{L\cdot\sum_{s\in\mathcal{S}}\delta(s,t)}\ .$$

By combining the assumption about the cost of the TSP tour with Lemma 3.30, we obtain the following chain of inequalities (with β being the same constant as in Lemma 3.30)

$$\varepsilon \cdot \frac{1}{k} \sum_{s \in \mathcal{S}} \delta(s,t) \cdot \mathfrak{d}(s) \leq C(TSP) \leq \beta \cdot \sqrt{L \cdot \sum_{s \in \mathcal{S}} \delta(s,t)} \leq \beta \cdot \sqrt{L \cdot \sum_{s \in \mathcal{S}} \delta(s,t) \cdot \mathfrak{d}(s)} \ ,$$

which yields

$$\sqrt{\sum_{s \in \mathcal{S}} d(s,t) \cdot \mathfrak{d}(s)} \leq \beta \cdot \frac{k}{\varepsilon} \cdot \sqrt{L} \ .$$

□

Consider a closed disk $\mathcal{Q}$ of radius αL with the center at the sink t, where we set $\alpha = \frac{\varepsilon^4}{\beta^4 \cdot k^2}$. Let $\mathcal{S}_{out}$ be the subset of points from $\mathcal{S}$ that are outside the disk $\mathcal{Q}$ (i.e., for which $\delta(s,t) > \alpha L$), and let $\mathcal{S}_{in} = \mathcal{S} \setminus \mathcal{S}_{out}$. By Lemma 3.29, we have $\sum_{s \in \mathcal{S}_{out}} \delta(s,t) \cdot \mathfrak{d}(s) \leq (\frac{\beta \cdot k}{\varepsilon})^2 \cdot L$, and hence $\sum_{s \in \mathcal{S}_{out}} \mathfrak{d}(s) \leq (\frac{\beta \cdot k}{\varepsilon})^2 \cdot \frac{1}{\alpha}$.

We partition the optimal network into two parts — OPT_{in} and OPT_{out}, which consist of the edges lying respectively inside and outside $\mathcal{Q}$. We add artificial Steiner vertices on the boundary of $\mathcal{Q}$ to divide the edges. Then $C(OPT) = C(OPT_{in}) + C(OPT_{out})$.

Lemma 3.31. *In time $k^{O(\varepsilon^{-1} \log k)}$ we can find a set $\mathcal{S}'_{out}$ of $\sum_{s \in \mathcal{S}_{out}} \mathfrak{d}(s)$ points on the boundary of the disk $\mathcal{Q}$ and a network of cost at most $C(OPT_{out}) + 3\varepsilon \cdot C(OPT)$ that allows sending all the flow from $\mathcal{S}_{out}$ to the unit sinks $\mathcal{S}'_{out}$.*

Proof. We know that $\sum_{s \in \mathcal{S}_{out}} \mathfrak{d}(s) \leq (\frac{\beta k}{\varepsilon})^2 \cdot \frac{1}{\alpha} = \frac{\beta^6 k^4}{\varepsilon^6}$. Let us create $m = \lceil \frac{\pi \beta^2 k^2}{\varepsilon^3} \rceil$ equidistant points, which we call portals, on the boundary of the disk $\mathcal{Q}$.

We know that there is a solution with cost $C(OPT_{out})$ which sends the flow from $\mathcal{S}_{out}$ to some unit sinks on the boundary (it is the network obtained from OPT by taking only the edges OPT_{out}). The distance between two portals is smaller than $\frac{2\pi\alpha L}{m} \leq \frac{2\varepsilon^7 L}{\beta^6 k^4}$ and the total flow is at most $\frac{\beta^6 k^4}{\varepsilon^6}$, so the total increase of cost generated by moving each sink to the nearest portal will be at most $\varepsilon \cdot L \leq \varepsilon \cdot C(OPT)$. There is a solution to the above problem with all the unit sinks in the portals and with a cost at most $C(OPT_{out}) + \varepsilon \cdot C(OPT)$.

Now we use the algorithm from Theorem 3.27 with at most $\frac{\beta^6 k^4}{\varepsilon^6}$ unit sources and m sinks (the portals) with unlimited demand. The running time is $k^{O(\varepsilon^{-1} \log k)}$ and we get a solution with a cost at most $(1+\varepsilon)(C(OPT_{out}) + \varepsilon \cdot C(OPT)) \leq C(OPT_{out}) + 3\varepsilon \cdot C(OPT)$. □

Lemma 3.32. *In polynomial time we can find a network which sends the flow from the points $\mathcal{S}_{in} \cup \mathcal{S}'_{out}$ to the sink t with cost at most $C(OPT_{in}) + 2\varepsilon \cdot C(OPT)$.*

Proof. Let us consider the network OPT. Using the edges from OPT_{in}, each source $s \in \mathcal{S}$ has a flow transported by a distance at least $\min\{\delta(s,t), \alpha L\}$. As each edge can be used by at most k units of flow, we get

$$C(OPT_{in}) \geq \frac{1}{k} \cdot \sum_{s \in \mathcal{S}} \min\{\delta(s,t), \alpha L\} \cdot \mathfrak{d}(s) = \frac{1}{k} \cdot \sum_{s \in \mathcal{S}_{in} \cup \mathcal{S}'_{out}} \delta(s,t) \cdot \mathfrak{d}(s) \ .$$

Let TSP' be an optimal traveling salesman tour for $\mathcal{S}_{in} \cup \mathcal{S}'_{out}$. By Lemma 3.30 and Lemma 3.29

$$\begin{aligned} C(TSP') &\leq \beta \cdot \sqrt{\alpha L \cdot \sum_{s \in \mathcal{S}_{in} \cup \mathcal{S}'_{out}} \delta(s,t)} \leq \beta \cdot \sqrt{\alpha L \cdot \sum_{s \in \mathcal{S}} \delta(s,t)} \leq \\ &\leq \frac{\beta^2 \cdot k}{\varepsilon} \cdot \sqrt{\alpha} \cdot L \leq \frac{\beta^2 \cdot k}{\varepsilon} \cdot \sqrt{\alpha} \cdot C(OPT) \ . \end{aligned}$$

Since $\alpha = \frac{1}{\beta^4} \cdot \frac{\varepsilon^4}{k^2}$, we have $C(TSP') \leq \varepsilon \cdot C(OPT)$.

Using the algorithm from Lemma 3.28 we get in polynomial time a solution with cost at most

$$(1+\varepsilon) \cdot C(TSP') + \frac{1}{k} \cdot \sum_{s \in \mathcal{S}_{in} \cup \mathcal{S}'_{out}} \delta(s,t) \cdot \mathfrak{d}(s) \leq (1+\varepsilon)\varepsilon \cdot C(OPT) + C(OPT_{in}) \ .$$

□

Combining the algorithms from Lemma 3.28, Lemma 3.31 and Lemma 3.32 yields a solution to the original SCGND problem with cost at most $(1+5\,\varepsilon) \cdot C(OPT)$ and running in time $k^{O(\varepsilon^{-1} \log k)} + n^{O(1)}$. By dividing ε by 5, we obtain our main result.

Theorem 3.33. *For any $\varepsilon > 0$, there is a deterministic algorithm for the SCGND problem with total demand polynomial in n and edge capacity k, which runs in time $k^{O(\varepsilon^{-1} \log k)} + n^{O(1)}$ and yields a solution whose cost is within $1 + \varepsilon$ of the optimum. For $k = 2^{O(\sqrt{\log n})}$, it runs in polynomial time.*

3.5 Open Problems

Geometric buy-at-bulk network design. A natural extension of the geometric network design problem is the *geometric buy-at-bulk network design problem*, where

instead of a single edge type of a given capacity k we are given multiple types of edges, where the cost of an edge depends on its length and type. The rectilinear version of that problem has been considered by Czumaj et al. [38], who present a QPTAS for it.

It is an interesting question of whether or not the upper bound on the number of Steiner vertices in an optimal solution to an instance of the CGND problem could be extended to include several types of edges. Such an extension would lead to corresponding extensions of our QPTAS and PTAS to work for the geometric buy-at-bulk network design problem.

Better upper bound on the number of Steiner vertices. Can we get an upper bound on the number of Steiner vertices needed by an optimal solution, which depends on the total number of sources and sinks, and not on the total demand of the sources?

PTAS for SCGND for all values of k. Another open problem is getting a polynomial time approximation scheme for the SCGND problem for all values of k.

PTAS for CGND. Can we extend the PTAS for the SCGND problem to work for the CGND problem, when instead of a single sink we are given an arbitrary number of sinks?

Other metrics. What results can we show for the capacitated geometric network design problem, when the input points are in a constant-dimensional Euclidean space instead of the Euclidean plane? What results can we show for other metrics?

Chapter 4

Reordering Buffer Management Problem

4.1 Introduction

In this chapter we study the reordering buffer management problem in an online setting.

4.1.1 Related Work

Table 4.1 presents an overview of the results for the reordering buffer management problem.

The reordering buffer management problem was introduced by Räcke et al. [74], who developed an $O(\log^2 k)$-competitive online algorithm for the version with uniform costs. Englert and Westermann [44] improved the competitive ratio to $O(\log k)$, and their algorithm is also able to handle non-uniform costs with the same bound. Their proof works in two steps. First, it is shown that an online algorithm with a buffer of size k is constant competitive w.r.t. an optimal offline algorithm with a buffer of size $\frac{k}{4}$. Then, it is shown that an optimal algorithm with a buffer of size $\frac{k}{4}$ only loses a logarithmic factor compared to an optimal algorithm with buffer of size k.

It was shown in [1] that with this proof technique it is not possible to derive online algorithms with a competitive ratio $o(\log k)$, by presenting an input sequence where the gap between an optimal algorithm with a buffer of size $\frac{k}{4}$ and an optimal algorithm with buffer size k is $\Omega(\log k)$. Nevertheless, Avigdor-Elgrabli and Rabani [16] were able to go beyond the logarithmic threshold by presenting an online algorithm with a competitive ratio $O(\frac{\log k}{\log \log k})$ using linear programming based

	online	offline
algorithms	$O(\log^2 k)$-competitive, uniform costs [74] $O(\log k)$-competitive [44] $O\left(\frac{\log k}{\log\log k}\right)$-competitive [16] **$O(\sqrt{\log k})$-competitive, deterministic**	$O(1)$-approximation, uniform costs [17]
lower bounds	**$\Omega\left(\sqrt{\frac{\log k}{\log\log k}}\right)$ for deterministic algorithms** **$\Omega(\log\log k)$ for randomized algorithms**	NP-hard for uniform costs [32; 13]

Table 4.1: An overview of the results for the reordering buffer management problem. The new results are presented using bold font.

techniques.

For the offline problem it was shown by Chan et al. [32] and independently by Asahiro, Kawahara, and Miyano [13] that the problem is NP-hard even for uniform costs. Recently Avigdor-Elgrabli and Rabani [17] gave a constant factor approximation algorithm for the offline problem with uniform costs.

There also exists a more general version of the problem. Instead of letting the context switching cost for switching from an item with color c' to an item with color c only depend on c, it is sometimes desirable to let it depend on c' and c. Khandekar and Pandit [61], and Gamzu and Segev [47] study the problem where the colors correspond to points in a line metric. Colors c' and c are integer points on the line and the cost for switching from c' to c is $|c' - c|$. This version of the problem is motivated by disc scheduling. Khandekar and Pandit [61] give a randomized $O(\log^2 n)$-competitive online algorithm for n uniformly spaced points on a line, and a constant factor offline approximation in quasi-polynomial time. Gamzu and Segev [47] improve the first result to $O(\log n)$. They also show a lower bound of 2.1547 on the competitive ratio of deterministic online algorithms on the line. This is the only non-trivial lower bound that exists for any variant (i.e. metric) of the problem.

Englert et al. [43] consider a more general version where colors correspond to

arbitrary points in a metric space (C, δ) and the cost for switching from color c' to color c is the distance $\delta(c', c)$ between the corresponding points in the metric space. They obtain a competitive ratio of $O(\log^2 k \log |C|)$.

Research has also been done on the maximization version of the problem, where the cost-measure is the number of color changes that the output sequence saved over the unordered input sequence. For this version there exist constant factor approximation algorithms due to Kohrt and Pruhs [62] and Bar-Yehuda and Laserson [22].

There is a whole area of scheduling that investigates *scheduling problems with setup times or costs*, which is related to the problem we consider in this chapter. In a scheduling problem involving setup times (costs) the goal is to schedule a set of tasks on a set of machines to optimize a given objective function, where together with the processing time (cost) of the tasks we have to consider the setup time (cost), which is incurred when a machine switches to a different task. There are a lot of papers studying many versions of this problem, for example the setup time can depend on or can be independent of the task processed previously on the machine, the tasks can be processed individually or in batches, and different objective functions can be considered. An overview of the results in the area can be found e.g. in [9].

4.1.2 Our Results

We improve the best known upper bound, as well as the best known lower bounds for the reordering buffer management problem.

In Section 4.2 we show how a change in the buffer size can affect the cost of an optimal solution for the reordering buffer management problem with uniform costs. Using these results we can obtain lower bounds on the competitive ratio of some online algorithms for the reordering buffer management problem which have been considered before, and for which no lower bounds were known.

In Section 4.3 we present first non-trivial lower bounds on the competitive ratio of online algorithms for the problem. We show that any deterministic online algorithm for the reordering buffer management problem has a competitive ratio of at least $\Omega\left(\sqrt{\frac{\log k}{\log \log k}}\right)$ even in the uniform case. For randomized algorithms we are able to construct a lower bound of $\Omega(\log \log k)$. Earlier no lower bounds were known, and it was quite conceivable that the existing *algorithms might actually have a much better competitive ratio than what was proven about them, possibly even a constant competitive ratio* [16].

In Section 4.4 we complement the lower bound for deterministic algorithms with a deterministic online algorithm whose competitive ratio nearly matches it. We

present a deterministic online algorithm that obtains a competitive ratio of $O(\sqrt{\log k})$ for the non-uniform case in which the ratio between the smallest and the largest weight of a color is polynomially bounded in k. This improves upon the result of Avigdor-Elgrabli and Rabani [16] who obtained a competitive ratio of $O(\frac{\log k}{\log \log k})$.

All previous results [16; 44; 74] for the reordering buffer management problem used very similar algorithms with only subtle differences between them. The differences between the results were mostly based on the analysis. In contrast, our new result relies on an important modification in the algorithm. In addition to techniques similar to those used in [16; 44; 74], our algorithm also relies on classifying colors according to the number of items of the color in the buffer. Then, the algorithm tries to evict items of a color class that currently occupy a large fraction of the buffer. This algorithmic ingredient plays a crucial role in reducing the competitive ratio to $O(\sqrt{\log k})$ in our analysis.

The results are based on joint work with Artur Czumaj, Matthias Englert and Harald Räcke [3].

4.2 Bounds for Modified Size of the Buffer

In this section we consider the reordering buffer management problem with uniform costs, i.e., where the *cost* of an output sequence equals the number of color-blocks in the sequence. A *color-block* (or a *block*) of a sequence is a maximal subsequence of consecutive elements with the same color. We show how a change in the size of the buffer can affect the cost of an optimal solution. We then explain how we can use these results to show lower bounds on the competitive ratio of some online algorithms.

By $\mathrm{OPT}_\ell(\sigma)$ we denote an optimal solution, and by $|\mathrm{OPT}_\ell(\sigma)|$ the cost of the respective solution, for an instance σ of the problem using a buffer of size ℓ. When the problem instance σ is known from the context, we write OPT_ℓ instead of $\mathrm{OPT}_\ell(\sigma)$.

Theorem 4.1. *For any instance σ of the reordering buffer management problem with uniform costs and a parameter $\alpha = \alpha(k)$ such that $0 < \alpha < 1$ we have*

$$|OPT_{(1-\alpha)k}(\sigma)| \leq \left(3 + 2\frac{\alpha}{1-\alpha}\log((1-\alpha)k)\right) \cdot |OPT_k(\sigma)| \ .$$

Proof. It is enough to consider input sequences σ for which an optimal solution with a buffer of size k outputs all elements having the same color in a single block. Any input instance σ which does not satisfy this property can be modified in the following

way. The sets of elements that are output by the optimal solution as single blocks are recolored with unique colors. This operation does not change the cost of OPT_k, and it does not decrease the cost of $\mathrm{OPT}_{(1-\alpha)k}$.

Let σ be an input sequence satisfying the property above. Assuming that we know the optimal solution for the buffer of size k, we will design an algorithm ALG which uses buffer of size $(1-\alpha)k$ and has the desired cost.

Let m be the number of colors appearing in the input sequence σ. We order the colors based on the order in which they are output by the optimal solution with buffer of size k. Color c_i, where $i = 1, \ldots, m$, is the i-th color output in the optimal solution.

We define a *phase* of algorithm ALG as follows. If ALG has already output all elements with colors $c_1, \ldots, c_{i-1}$, but there are still some elements of color c_i which have not been output, the algorithm ALG is in the i-th phase.

The algorithm ALG works as follows. Let i be the index of the current phase of ALG. As long as there are some elements of color c_i in the buffer, the algorithm outputs them. At some point it might happen, that the whole buffer is filled with elements with colors in $\{c_{i+1}, c_{i+2}, \ldots, c_m\}$. The algorithm has to choose some color c_j (where $j > i$) to be output. Notice that if at least αk elements with colors in $\{c_{i+1}, c_{i+2}, \ldots, c_m\}$ have been output, the algorithm can finish outputting the elements with color c_i.

For any $j > i$ let s_j be the number of elements of color c_j currently in the buffer of ALG. Let the *potential* of color c_j be $p_j := s_j \cdot (j-i)$. The intuition behind the potential is that outputting the elements of color c_j would free s_j slots of the buffer for $(j-i)$ phases of the algorithm, so the total "gain" would be $s_j \cdot (j-i)$. The algorithm ALG always outputs a color with the largest potential p_j. Ties are resolved arbitrarily.

Lemma 4.2. *At each time when the buffer of ALG is full (i.e., when we are not running out of the input elements), there is a color with potential at least* $\frac{(1-\alpha)k}{\log((1-\alpha)k)}$.

Proof. Let $p = \max\{p_j : j > i\}$, where i is the index of the current phase of ALG. From the definition of the potential p_j we get that $s_{i+1} \leq p, s_{i+2} \leq \frac{p}{2}, s_{i+3} \leq \frac{p}{3}, \ldots, s_{i+p} \leq \frac{p}{p}$ and in the buffer there are no elements with colors c_ℓ for $\ell > i+p$. The total number of elements in the buffer is $(1-\alpha)k$, therefore we must have

$$(1-\alpha)k = \sum_{j=1}^{p} s_{i+j} \leq \sum_{j=1}^{p} \frac{p}{j} = p \cdot H_p \ .$$

We get that $p \geq \frac{(1-\alpha)k}{\log((1-\alpha)k)}$ and there is a color with potential p in the buffer. □

For each color c_ℓ we set $\phi(\ell)$ to be the number of elements with colors in $\{c_{\ell+1}, c_{\ell+2}, \ldots, c_m\}$ that are output by ALG before the last element of color c_ℓ. Let $\phi = \sum_{j=1}^{m} \phi(j)$. We update the values of $\phi(\ell)$ and ϕ while running the algorithm. When elements with color c_j are output during the i-th phase of ALG, for $j \neq i$, the value of ϕ increases by p_j, as the values of $\phi(i), \phi(i+1), \ldots, \phi(j-1)$ increase by s_j each. As we always choose a color with the largest potential, from Lemma 4.2 we get that the value of ϕ increases by at least $\frac{(1-\alpha)k}{\log((1-\alpha)k)}$.

When the value of $\phi(i)$ reaches αk, it means that at least αk elements with colors in $\{c_{i+1}, c_{i+2}, \ldots, c_m\}$ have been already output, and the algorithm can finish outputting all the elements with color c_i. The value of $\phi(i)$ does not increase any more. The number of times the value of ϕ is increased is at most

$$m + \frac{m \cdot \alpha k}{(1-\alpha)k/\log((1-\alpha)k)} = m\Big(1 + \frac{\alpha}{1-\alpha}\log((1-\alpha)k)\Big) \ .$$

Each time the value of ϕ is increased, the cost of the algorithm increases by at most 2 — first the algorithm outputs a block of elements of some color c_j, and then possibly a block of elements of the current color c_i. Additionally, for each i, at the beginning of the i-th phase the algorithm can output a block of elements with color c_i without increasing the value of ϕ.

The total number of color-blocks in a solution generated by ALG, and therefore also the cost of $\mathrm{OPT}_{(1-\alpha)k}$, is bounded from above by

$$m + 2m\Big(1 + \frac{\alpha}{1-\alpha}\log((1-\alpha)k)\Big) = \Big(3 + 2\frac{\alpha}{1-\alpha}\log((1-\alpha)k)\Big) \cdot |\mathrm{OPT}_k| \ ,$$

as $|\mathrm{OPT}_k| = m$. □

Corollary 4.3. *Let $\alpha = \frac{\mathfrak{c}}{\log k}$ for a constant $\mathfrak{c}$. We have*

$$|OPT_{(1-\alpha)k}(\sigma)| \leq 3(1+\mathfrak{c}) \cdot |OPT_k(\sigma)| \ .$$

Corollary 4.4. *Let $0 < \alpha < 1$ be a constant. There is a constant $\mathfrak{c} = \mathfrak{c}(\alpha)$ such that for any input sequence σ we have*

$$|OPT_{(1-\alpha)k}(\sigma)| \leq \mathfrak{c} \log k \cdot |OPT_k(\sigma)| \ .$$

We now show a lower bound which is of the same order as the upper bound from Theorem 4.1 if the parameter $\alpha = \alpha(k)$ satisfies $0 < \alpha < \mathfrak{c} < 1$ for some constant $\mathfrak{c}$.

Theorem 4.5. *For any parameter $\alpha = \alpha(k)$ such that $0 < \alpha < 1$ there are instances σ of the reordering buffer management problem with uniform costs such that*

$$|OPT_{(1-\alpha)k}(\sigma)| \geq \left(1 + \frac{1}{4}\alpha \cdot \log k\right) \cdot |OPT_k(\sigma)| \ .$$

Proof. We will construct an input sequence σ consisting of elements with colors $c_1, \ldots, c_m$, such that using a buffer of size k we can output all the elements of each color in one block (i.e., $|\mathrm{OPT}_k| = m$), starting with the color c_1 and then continuing until c_m, but any solution generated using a buffer of size $(1-\alpha)k$ has cost at least $(1 + \frac{1}{4}\alpha \cdot \log k) \cdot m$.

The number of elements of each color will be $k + \Theta(\frac{k}{\log k})$. For each color c_i the last block of color c_i in the input sequence consists of k elements, to force all algorithms using buffer of size at most k to output the remaining elements of color c_i at the time of getting the last block of this color from the input sequence. As in the previous theorem, for any algorithm processing the input sequence we define the i-th *phase* as the time when all the elements of colors $\{c_1, c_2, \ldots, c_{i-1}\}$ have been output, but not all the elements of the color c_i.

We construct the input sequence in such a way, that before the time the last block of color c_i appears, the input contains exactly $k-1$ elements with colors in $\{c_{i+1}, c_{i+2}, \ldots, c_{i+p}\}$[1] for a parameter $p = p(k)$ that will be fixed later, and no elements with colors c_{i+j} for $j > p$. For each $1 \leq j \leq p$ we want to have roughly $\frac{p}{j}$ elements of color c_{i+j} before the last block of color c_i.

We proceed as follows. Let p be the smallest integer such that $p \cdot H_p \geq k-1$. We can easily check that p satisfies $\frac{k}{\log k} < p < 2\frac{k}{\log k}$. As $\sum_{j=1}^{p} \frac{p}{j} = p \cdot H_p \geq k-1$, we can choose values $n_1, n_2, \ldots, n_p$ in such a way, that for each j we have: $n_j \leq \lceil \frac{p}{j} \rceil$, $n_j \geq n_{j+1}$ and $\sum_{j=1}^{p} n_j = k-1$. We construct the input sequence in such a way, that for each i and $i \leq j \leq p$ there are exactly n_j elements of color c_{i+j} before the last element of color c_i. As the number of elements with colors in $\{c_{i+1}, c_{i+2}, \ldots, c_m\}$ before the last element of color c_i is $k-1$, we have $|\mathrm{OPT}_k| = m$.

The input sequence σ, which satisfies the conditions above, is as follows:

$$\begin{aligned}&\left(c_1^{n_1}\ c_2^{n_1}\ c_3^{n_2}\ c_4^{n_3} \ldots c_{p+1}^{n_p}\right) c_1^k \left(c_3^{n_1-n_2}\ c_4^{n_2-n_3} \ldots c_{p+1}^{n_{p-1}-n_p}\ c_{p+2}^{n_p}\right) c_2^k \ldots \\ &\qquad \ldots c_{i-1}^k \left(c_{i+1}^{n_1-n_2}\ c_{i+2}^{n_2-n_3} \ldots c_{i+p-1}^{n_{p-1}-n_p} c_{i+p}^{n_p}\right) c_i^k \ldots\end{aligned}$$

Let ALG be an algorithm processing the input sequence σ, and using a buffer

[1]Except when we are approaching the end of the input sequence. However, as the number of colors m can be arbitrarily large, we can ignore the fact that we do not get any elements with colors c_j for $j > m$.

of size $(1-\alpha)k$. Let i be the index of the current phase of ALG, and let $i+1 \leq j \leq i+p$. As before, let s_j denote the number of elements of color c_j currently in the buffer of ALG, and let the potential of color c_j be $p_j := s_j \cdot (j-i)$. As the number of elements of color c_j that appear in the input sequence before the last block of color c_i is n_{j-i}, we have

$$p_j \leq n_{j-i} \cdot (j-i) \leq \left\lceil \frac{p}{j-i} \right\rceil \cdot (j-i) \leq 2p < 4\frac{k}{\log k} \ .$$

As before, let $\phi(i)$ be the number of elements with colors in $\{c_{i+1}, c_{i+2}, \ldots, c_m\}$ that are output by ALG before the last element of color c_i, and let $\phi = \sum_{i=1}^{m} \phi(i)$. When ALG switches to color c_j, the value of ϕ increases by $p_j < 4\frac{k}{\log k}$.

As ALG uses a buffer of size $(1-\alpha)k$, we must have $\phi(i) \geq \alpha k$ for each[2] phase i, and therefore $\phi \geq m \cdot \alpha k$. The number of color-blocks in a solution generated by ALG, and therefore also in an optimal solution $\mathrm{OPT}_{(1-\alpha)k}$, must be at least

$$m + \frac{\phi}{4k/\log k} \geq \left(1 + \frac{1}{4}\alpha \cdot \log k\right) \cdot |\mathrm{OPT}_k| \ .$$

□

The above theorem generalizes the result of Aboud [1], who shows that there are instances σ of the reordering buffer management problem for which $|\mathrm{OPT}_{k/4}(\sigma)| = \Omega(\log k) \cdot |\mathrm{OPT}_k(\sigma)|$. Aboud uses different sequences σ from the ones we use in the proof of Theorem 4.5, and his analysis is based on linear programming techniques.

Results from this section allow us to obtain lower bounds on the competitive ratio of many online algorithms for the reordering buffer management problem with uniform costs. We can try to show a lower bound for an algorithm ALG using a buffer of size k in the following way. We fix a parameter $\alpha = \omega(\frac{1}{\log k})$, and we take an input sequence σ, for which $|\mathrm{OPT}_{(1-\alpha)k}(\sigma)| = \Omega(1 + \alpha \cdot \log k) \cdot |\mathrm{OPT}_k(\sigma)|$. We then transform the input sequence σ into a sequence σ' by adding some *artificial blocks* with unique colors in such a way, that at each time the elements from the artificial blocks fill an α fraction of the buffer of ALG (if ALG is a randomized algorithm, we might need to proceed a bit more carefully). The size of the artificial blocks and the number of them has to be chosen depending on the algorithm ALG — the goal is to put them into the input sequence in such a way, that ALG does not output the artificial elements too fast, as then the number of blocks needed is small. We then get that the cost of ALG on σ' is at least $|\mathrm{OPT}_{(1-\alpha)k}(\sigma)| = \Omega(1+\alpha \cdot \log k) \cdot |\mathrm{OPT}_k(\sigma)|$. On

[2]We ignore the fact that the p last colors $c_{m-p+1}, \ldots, c_m$ have smaller values of $\phi(i)$, as no elements with larger colors arrive. As m can be arbitrarily large, it does not affect our analysis.

the other hand, the optimal algorithm for σ' can output the artificial blocks as soon as it gets them from the input, and we have that $|\mathrm{OPT}_k(\sigma')| \leq |\mathrm{OPT}_k(\sigma)| + 2N$, where N is the number of artificial blocks used. The factor 2 is needed here, as outputting an artificial block can force the algorithm to stop outputting the current block, which can be resumed only after the artificial block has been output. If we manage to get $N = o(1 + \alpha \cdot \log k) \cdot |\mathrm{OPT}_k(\sigma)|$, we get a lower bound on the competitive ratio of ALG.

Consider for example a simple randomized algorithm RAN, which works as follows. In each step it chooses uniformly at random an element from the buffer, and then it outputs all elements with the same color as the chosen element. We set $\alpha = \frac{1}{\sqrt{\log k}}$ and get an input sequence σ such that

$$|\mathrm{OPT}_{(1-\alpha)k}(\sigma)| = \Omega(\alpha \cdot \log k) \cdot |\mathrm{OPT}_k(\sigma)| = \Omega(\sqrt{\log k}) \cdot |\mathrm{OPT}_k(\sigma)| \ .$$

Each artificial block will have size αk. The expected time until an artificial block is removed by RAN is $\frac{1}{\alpha} = \sqrt{\log k}$ color changes of RAN. Let C be the expected cost of the algorithm RAN on the original input sequence σ, if RAN is using a buffer of size $(1-\alpha)k$ instead of k. We know that $C \geq |\mathrm{OPT}_{(1-\alpha)k}(\sigma)| = \Omega(\sqrt{\log k}) \cdot |\mathrm{OPT}_k(\sigma)|$. It is enough to use $N = \Theta(\frac{C}{\sqrt{\log k}})$ properly placed artificial blocks, and the expected cost of the algorithm RAN on σ' will be $\Omega(C)$. As we get $|\mathrm{OPT}_k(\sigma')| \leq |\mathrm{OPT}_k(\sigma)| + 2N = O(\frac{C}{\sqrt{\log k}})$, the competitive ratio of RAN is at least of the order of $\frac{C}{C/\sqrt{\log k}} = \sqrt{\log k}$.

A similar lower bound can be obtained for many versions of the algorithms considered in [44] and [16].

4.3 Lower Bounds

In this section we give lower bounds on the competitive ratio of online algorithms for the reordering buffer management problem with uniform costs. We do this by carefully constructing an input sequence σ for which any online algorithm ONL exhibits a large cost, while the optimal algorithm OPT can process σ with a significantly lower cost.

4.3.1 Preliminaries

We first describe the general scheme for constructing σ. For this we introduce parameters α, d and N_i, whose precise values will be fixed later. For simplicity of notation we assume that k is sufficiently large, and chosen in such a way that no

rounding issues occur.

The General Scheme for Constructing σ

The input sequence σ that we construct has the property that an optimal algorithm OPT$'$ with a buffer of size $(1+\alpha)k$ can process σ in such a way that the cost of the output sequence equals the total number of different colors contained in σ (i.e., all elements of a single color are output in one color-block). This means OPT$'$ is truly optimal for the sequence, and even increasing the size of the buffer further cannot reduce the cost for processing σ.

To specify the input sequence σ further, we will view the buffer of OPT$'$ as partitioned into d *classes* $C_1, \ldots, C_d$ (see Figure 4.1). Each class C_i can store $\frac{(1+\alpha)k}{d}$ elements (i.e., each class consists of a $\frac{1}{d}$ fraction of the buffer space of OPT$'$). We further partition the buffer of each class C_i into N_i *slots*, where each slot can store $s_i = \frac{(1+\alpha)k}{dN_i}$ elements (i.e., each slot consists of a $\frac{1}{N_i}$ fraction of the buffer space of the class C_i). The number of slots in a class will be decreasing with the index of the class, i.e., for $i < j$ we have $N_i > N_j$.

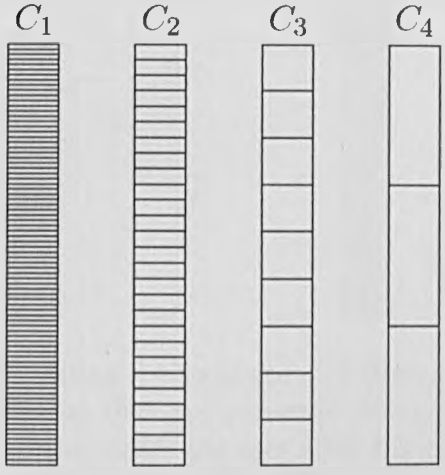

Figure 4.1: Partitioning a buffer space of size $(1+\alpha)k$ into d classes $C_1, \ldots, C_d$ (here $d = 4$). Each class C_i is further partitioned into N_i slots.

The main property of the input sequence σ will be as follows:

> Whenever OPT$'$ has to make a color change while processing σ, the buffer of OPT$'$ looks as follows. Each slot is completely filled with elements of the same color, and different slots contain elements of different colors.

This means, e.g., that if n_c denotes the number of elements with color c at a time right before a color change of OPT′, then $n_c = s_i$ for some $i \in \{1, \ldots, d\}$. Because of this property it makes sense to refer to the *color of a slot* as the color that every element in the slot has. Note that the color of a slot may change over time, but in the following proof the time will always be clear from the context.

We obtain the above property by constructing the sequence σ as follows: The initial $(1+\alpha)k$ elements of σ fill up the buffer of OPT′. They are chosen in such way that the invariant is satisfied right before OPT′ outputs its first element, i.e. amongst the first $(1+\alpha)k$ elements of σ there are exactly N_i colors with s_i elements for $i = 1, \ldots, d$. The exact order in which these first $(1+\alpha)k$ elements appear in σ is not important, but we assume that elements among them that share the same color appear consecutively.

Further elements in σ are chosen in *rounds* in the following way. For each round we choose a slot z_i, $i \in \{1, \ldots, d\}$ from every class (see Figure 4.2a).

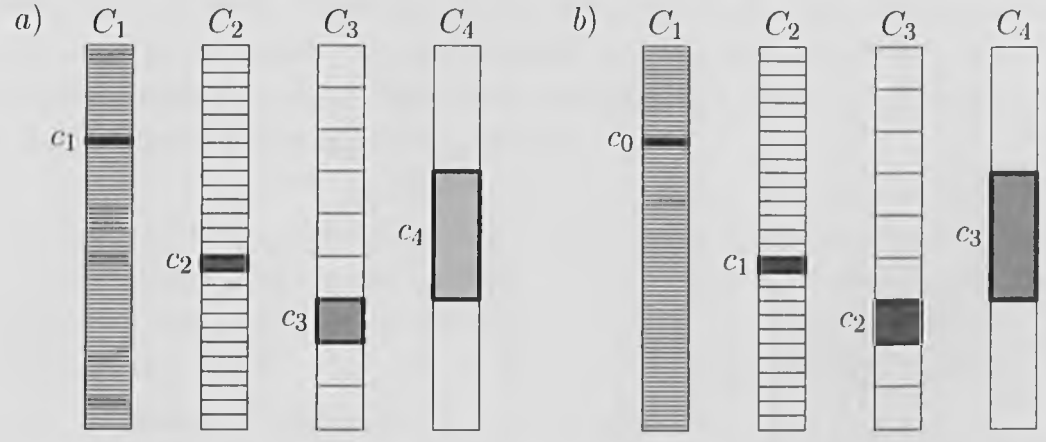

Figure 4.2: One round of creating the sequence σ. Figure *a*) shows the chosen slots, and the colors of the elements that are currently occupying the slots. Figure *b*) shows how the contents of the buffer changes after the round. The sequence has been extended as follows: $c_4^{k+1}\ c_3^{s_4-s_3}\ c_2^{s_3-s_2}\ c_1^{s_2-s_1}\ c_0^{s_1}$, where c_0 is the new color introduced in this round.

Then we first add $k+1$ elements of the color of z_d to σ, which will force any algorithm to switch to this color at this point. Then, for every i, starting from $d-1$ down to 1, we add $s_{i+1} - s_i$ elements of the color of slot z_i to σ. Finally, we add s_1 elements of a completely new color to σ. This finishes the round.

The algorithm OPT′ works as follows. In the beginning of a round it switches to the color of slot z_d. This frees up space s_d in the buffer, and, hence, OPT′ can hold all further elements appearing in the round without requiring any further color changes.

In order to maintain the main property, we do the following. For $i \leq d-1$, all s_i elements in slot z_i plus the $s_{i+1} - s_i$ elements with the same color arriving in the round are moved to slot z_{i+1} (and completely fill this slot).[3] We say that we *promote slot z_i in class C_i*. Finally, slot z_1 holds the s_1 elements of the new color (see Figure 4.2b).

This finishes the description of the construction of σ up to the selection of the d slots for each round. Note that regardless of how we choose these slots, it follows from the above discussion that the cost of OPT$'$ for processing σ is equal to the number of different colors in the sequence.

A Sketch of the Analysis

From the above description it is clear that the cost of OPT$'$ for processing the sequence σ is equal to the number of different colors in the sequence.

However, the online algorithm ONL and the optimal algorithm OPT only have a buffer of size k. Hence, at any time, these algorithms have already removed at least αk elements that are still held by OPT$'$. Suppose for the time being that these algorithms only remove whole slots (remember that we can simply view a slot as a set of elements that share the same color).

Clearly, if e.g. ONL removed all elements of a slot z_i, and this slot is promoted, then ONL will have an additional color change that OPT$'$ does not have. Now, if our aim is to maximize the ratio between the cost of ONL and the cost of OPT$'$, it turns into a caching problem, where in each round the adversary has to pick a slot from every class in such a way that she *hits* many slots that ONL has removed.

For making this idea work we need to show

- that it is more or less optimal for ONL to remove whole slots,
- that an adversary can always promote a large number of slots that have been removed by ONL, and
- that OPT can handle the resulting input sequence fairly well (as we are not interested in the ratio between the cost of OPT$'$ and ONL, but in the ratio between the cost of OPT and ONL).

[3] Observe that the notion of a slot has only been introduced for illustration. Since it is irrelevant where in the buffer something is stored, it is also possible to simply view a slot as the set of all elements of a particular color that are currently stored by OPT$'$.

The Caching Framework

In this section we make a formal connection of our problem to the caching-related problem hinted at in the above sketch.

We say that an algorithm ALG *cleared slot z before round r*, if right before the time ALG reads the first element of round r, there are no elements from the slot z in the buffer of ALG. This implies that ALG does not hold any elements of the color of slot z. Define the *cost* $\text{cost}^r_{\text{ALG}}$ of an algorithm ALG in round r as the number of slots promoted in round r that are cleared by ALG (i.e. $\text{cost}^r_{\text{ALG}}$ corresponds to the number of *cache misses* in round r).

A lower bound for ONL. The following lemma gives a lower bound on the cost of the online algorithm in terms of $\text{cost}^r_{\text{ONL}}$.

Lemma 4.6. *The total cost of ONL on the generated input sequence σ is at least*

$$\sum_{i=1}^{d} N_i + \sum_r \text{cost}^r_{ONL} \ .$$

Proof. First observe that we can compute the cost of an algorithm by increasing its cost by 1 whenever an element arrives whose color is different from the last color that was appended to the output sequence (called the *active output color*) and also different from all colors present in the buffer.

Initially, the first $(1+\alpha)k$ elements of the input sequence have $\sum_{i=1}^{d} N_i$ different colors, each of them contributing 1 to the cost of the algorithm.

Then, consider a slot z in class C_i for $i \leq d-1$ that is cleared by ONL and promoted in round r. Since ONL cleared z before round r, ONL does not store any elements of the color of slot z in its buffer at the beginning of the round. On the other hand, z is promoted, which means that elements of the color of z appear in σ in round r.

The first $k+1$ elements of round r belong to some color c of a slot in class C_d. After they arrived, the active output color of ONL is c. As the slot z was cleared by ONL before round r, at the time the first element of the color of z appears in σ in round r, the active output color of ONL has to be different from the color of z.

Therefore, each such slot z contributes 1 to the total cost of ONL and there are at least $\text{cost}^r_{\text{ONL}} - 1$ such slots, where the -1 accounts for the slot in class C_d. The cost of ONL is further increased by one in every round r, because of the single element of a completely new color appearing in σ.

By summing over all rounds and taking the first $(1+\alpha)k$ elements into account, it follows that the total cost of ONL is at least $\sum_{i=1}^{d} N_i + \sum_r \text{cost}^r_{\text{ONL}}$. □

An upper bound for OPT. In order to give an analogous upper bound on the cost of OPT, we will present specific offline algorithms and analyze their cost. In order to do this analysis in a round-based manner, we further restrict these offline algorithms in the following way. We require that right before a new round, the algorithm has less than $k - s_d$ elements stored in the buffer. To be more specific, we consider algorithms that process a round as follows.

1. Right before the first element of the round appears in σ, the number of elements stored in the buffer is reduced to less than $k - s_d$. This is done by selecting $\alpha N_i + 1$ slots from each class C_i and removing all of the elements in these slots from the buffer.

2. Let $z_1, \ldots, z_d$ denote the slots that are promoted during the round. The algorithm outputs every element of the color of slot z_d. This includes the first $k + 1$ elements arriving in the round and also elements of the same color that may be stored in the buffer.

3. Finally, the algorithm stores all other elements arriving during the round in the buffer. This is possible since the number of these elements is s_d.

In order to satisfy the buffer constraint for the first $(1 + \alpha)k$ elements of the input sequence, we assume that the algorithm immediately outputs elements from slots that get cleared from the buffer before the first round.

Lemma 4.7. *Given any offline algorithm OFF with the property above, the total cost of OPT on the generated input sequence σ is at most*

$$\sum_{i=1}^{d} N_i + \sum_r (\text{cost}^r_{OFF} + 1) \ .$$

Proof. The proof is similar to the proof of Lemma 4.6. Again, we observe that we can compute the cost of an algorithm by increasing its cost by 1 whenever an element arrives whose color is different from the last color that was appended to the output sequence and also different from all colors present in the buffer.

Initially, the first $(1 + \alpha)k$ elements of the input sequence have $\sum_{i=1}^{d} N_i$ different colors, each of them contributing 1 to the cost of the algorithm OFF. The cost of OFF also increases by 1 in each round due to the single element of a completely new color that appears in every round. The sum of all these costs is $\sum_{i=1}^{d} N_i + \sum_r 1$.

All further increases in the cost of OFF are caused by a sequence of elements arriving in some round r that have a color which is not currently present in the buffer of OFF but which is present in the buffer of OPT'. Such a sequence of elements

corresponds to the promotion of a slot z_i, where at the time of the promotion, and therefore also at the time the first element of the round is read from the input, OFF does not store any elements of the slot in the buffer.

In other words, each such increase in the cost of OFF is caused by a promotion of a cleared slot in some round r and therefore also contributes 1 to $\text{cost}^r_{\text{OFF}}$. Hence, $\sum_r \text{cost}^r_{\text{OFF}}$ is an upper bound on such increases to the cost of OFF.

Observing that the cost of OPT is upper bounded by the cost of OFF completes the proof. □

Choosing Parameters

For the remainder we fix the number of classes as $d := \frac{\log k}{2 \log \log k}$ and the size of a slot in class C_i as $s_i := \log^{i-1} k$. The parameter α will be chosen differently depending on whether we want to derive lower bounds for deterministic or for randomized online algorithms. If we deal with deterministic algorithms we choose $\alpha := \sqrt{\frac{\log \log k}{\log k}}$, otherwise we set $\alpha := \frac{(\log \log k)^2}{\log k}$.

An Important Lemma

We now prove a lemma that shows that in the beginning of a round the online algorithm has many *cleared slots* and that these lie in different classes (so that the adversary can choose many of them). This lemma forms the basis of our analysis.

We first require a technical claim that essentially states that for our specific choice of the values of s_i the online algorithm either has a slot cleared or has stored nearly all elements of the slot.

Claim 4.8. *For a round r and a slot z in a class C_i at least one of the following is true:*

(a) ONL cleared slot z before round r, or

(b) The color of slot z is equal to the color of the element that ONL appended to the output sequence right before reading the first element of round r,

(c) ONL holds at least $\log^{i-1} k - \log^{i-2} k$ of the $\log^{i-1} k$ elements OPT' stores in slot z, right before ONL reads the first element of round r.

Proof. Consider a slot z in a class C_i. There are only two possible reasons that z is not cleared (i.e., we are not in Case a), but ONL does not store all $\log^{i-1} k$ elements of z. Either z is the active output color, which means that ONL is in the process

of removing elements from z (Case b), or some elements of z have been previously removed by ONL.

In the latter case some elements have arrived after the removal, as otherwise z would be cleared. However, the last sequence of elements with the color of slot z was the sequence of length $\log^{i-1} k - \log^{i-2} k$. All these elements must still be in the buffer of ONL. This means we are in Case c. This proves the claim. □

Using the claim, we can now prove the desired bounds on the number of slots cleared by ONL.

Lemma 4.9. *Let ℓ_i be the number of slots from class C_i cleared by ONL before round r. The following holds:*

(a) $\sum_{i=1}^{d} \ell_i \log^{i-1} k \geq \frac{\alpha k}{2}$.

(b) At least $\frac{\alpha d}{4}$ of the values ℓ_i are not 0. In other words, at least $\frac{\alpha d}{4}$ different classes contain at least one cleared slot.

Proof. At the beginning of a round there must exist at least αk elements that ONL has already removed from its buffer while they are still held by OPT′.

Due to Claim 4.8, every slot of class C_i that is not cleared by ONL before round r and whose color is not the active output color of ONL, contains at least $\log^{i-1} k - \log^{i-2} k$ elements. Hence, the number of elements that are held by OPT′ but not by ONL is at most

$$\log^{d-1} k + \sum_{i=1}^{d} (N_i - \ell_i) \log^{i-2} k + \sum_{i=1}^{d} \ell_i \log^{i-1} k \ ,$$

where the first term accounts for the active output color of ONL and the second term accounts for the possible excess elements of OPT′ in slots that are not cleared. This, however, has to be at least αk. We get

$$\begin{aligned} \alpha k &\leq \log^{d-1} k + \sum_{i=1}^{d} (N_i - \ell_i) \log^{i-2} k + \sum_{i=1}^{d} \ell_i \log^{i-1} k \\ &\leq \frac{k}{\log k} + \frac{(1+\alpha)k}{\log k} + \sum_{i=1}^{d} \ell_i \log^{i-1} k \\ &\leq \frac{\alpha k}{2} + \sum_{i=1}^{d} \ell_i \log^{i-1} k \ , \end{aligned}$$

where the last step follows since $\frac{(2+\alpha)k}{\log k} \leq \frac{\alpha k}{2}$ for sufficiently large k. This implies the first claim.

For the second claim assume that less than $\frac{\alpha d}{4}$ of the values ℓ_i are greater than 0. Then we obtain

$$\sum_{i=1}^{d} \ell_i \log^{i-1} k < \frac{(1+\alpha)k}{d} \cdot \frac{\alpha d}{4} = \frac{\alpha(1+\alpha)k}{4} \leq \frac{\alpha k}{2} ,$$

which is a contradiction to the first claim. □

In the following sections we describe how to choose the slots to be promoted in a round in such a way that for ONL many cleared slots are promoted while for OPT this happens very rarely. This choice depends on whether we want to derive lower bounds for deterministic or for randomized algorithms.

4.3.2 Lower Bound for Deterministic Algorithms

In this section we present a lower bound of $\Omega\left(\sqrt{\frac{\log k}{\log\log k}}\right)$ on the competitive ratio of any deterministic online algorithm for the reordering buffer management problem. For this section, we define α to be $\sqrt{\frac{\log\log k}{\log k}}$.

As we consider deterministic algorithms, while constructing the input sequence σ we know what will be the exact contents of the buffer when the algorithm processes the input sequence up to the current round.

For every class C_i, we choose a slot for promotion as follows (see Figure 4.3):

> If in class C_i there exists a slot cleared by ONL, we choose an arbitrary such slot to be promoted. Otherwise, we promote the first slot of class C_i.

We present a randomized algorithm RND that processes σ with a buffer of size k and has small expected cost compared to ONL. As in the general outline for our offline algorithms in the previous section, the algorithm ensures that at the beginning of a round at least $\alpha N_i + 1$ slots from class C_i are cleared. More precisely, for each class C_i, RND chooses $\alpha N_i + 1$ slots uniformly at random from all but the first slot in the class. At the beginning of each round, RND removes all elements belonging to the selected slots from the buffer.

Lemma 4.10. *The expected cost of RND in round r is*

$$O\left(\sqrt{\frac{\log\log k}{\log k}}\right) \cdot cost^r_{ONL} - 1 .$$

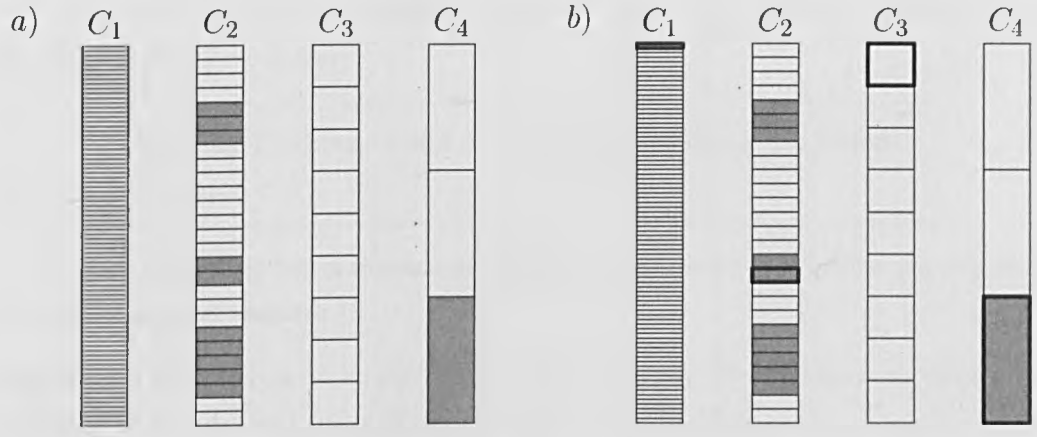

Figure 4.3: Constructing a bad input sequence for a deterministic algorithm. Figure *a*) shows the slots cleared by ONL (marked in grey). The slots chosen to be promoted are pictured in Figure *b*). As there are no cleared slots in classes C_1 and C_3, the first slots of these classes have been chosen.

Proof. Since RND never chooses to evict the first slot of a class, this slot is never cleared by RND. The probability that a specific other slot of a class is cleared is $\frac{\alpha N_i+1}{N_i-1} < 2\alpha$. Therefore, the expected cost of RND in round r is at most $2\alpha \operatorname{cost}^r_{\mathrm{ONL}}$. This is because, due to the way the slots are chosen for promotion, at most $\operatorname{cost}^r_{\mathrm{ONL}}$ slots are promoted that are not the first slots of a class.

From Lemma 4.9b we know that $\operatorname{cost}^r_{\mathrm{ONL}} \geq \frac{\alpha d}{4}$. Now since $\alpha = \frac{1}{\sqrt{2d}} = \sqrt{\frac{\log\log k}{\log k}}$, we get that the expected cost of RND is at most $2\alpha \cdot \operatorname{cost}^r_{\mathrm{ONL}} \leq 10\alpha \cdot \operatorname{cost}^r_{\mathrm{ONL}} - 1$, where the inequality holds since $8\alpha \operatorname{cost}^r_{\mathrm{ONL}} \geq 2\alpha^2 d = 1$. □

The lemma implies the following theorem.

Theorem 4.11. *Any deterministic algorithm for the reordering buffer management problem has a competitive ratio of* $\Omega\left(\sqrt{\frac{\log k}{\log\log k}}\right)$.

Proof. Clearly, the cost of OPT is at most the expected cost of RND which is, due to Lemma 4.7 and Lemma 4.10, at most $\sum_{i=1}^{d} N_i + O\left(\sqrt{\frac{\log\log k}{\log k}}\right) \sum_r \operatorname{cost}^r_{\mathrm{ONL}}$. Due to Lemma 4.6, the cost of ONL is at least $\sum_{i=1}^{d} N_i + \sum_r \operatorname{cost}^r_{\mathrm{ONL}}$. Therefore, the competitive ratio tends to $\Omega\left(\sqrt{\frac{\log k}{\log\log k}}\right)$ as the number of rounds tends to infinity. □

4.3.3 Lower Bound for Randomized Algorithms

In this section we provide a lower bound of $\Omega(\log\log k)$ on the competitive ratio of any randomized online algorithm for the reordering buffer management problem. For this section, we define α to be $\frac{(\log\log k)^2}{\log k}$.

For the analysis in this section, for every class C_i, we choose a slot for promotion in the following way:

> For class C_i choose a slot z in the class uniformly at random.
> Promote z.

We start by giving a bound on the expected cost of any online algorithm on the resulting input sequence.

Lemma 4.12. *For a randomized online algorithm ONL, for an input sequence consisting of R rounds, and for sufficiently large k it holds that*

$$\mathbf{E}\left[\sum_r \text{cost}^r_{ONL}\right] \geq R \cdot \frac{\log\log k}{8} \ .$$

Proof. Let ONL be an arbitrary online algorithm using a buffer of size k. We fix a round r and analyze $\mathbf{E}[\text{cost}^r_{\text{ONL}}]$. For a class C_i, let ℓ_i denote the number of slots from C_i cleared by ONL before round r. Note that the ℓ_i's are (dependent) random variables, but the following holds for any valid fixed choice of values.

According to Lemma 4.9a we have $\sum_{i=1}^{d} \ell_i \cdot \log^{i-1} k \geq \frac{\alpha k}{2}$. If during the round we promote one of the ℓ_i cleared slots in C_i, the value of $\text{cost}^r_{\text{ONL}}$ increases by one. This happens with probability

$$\frac{\ell_i}{N_i} = \ell_i \cdot \frac{\log^{i-1} k \cdot d}{k(1+\alpha)} \geq \ell_i \cdot \log^{i-1} k \cdot \frac{d}{2k} \ .$$

Summing this over all classes we obtain $\mathbf{E}[\text{cost}^r_{\text{ONL}}] \geq \frac{\alpha d}{4} = \frac{\log\log k}{8}$. Taking the sum over all R rounds completes the proof. □

Next we need to show that the expected optimal cost on the input sequence is significantly smaller.

Lemma 4.13. *There is an offline algorithm OFF using a buffer of size k such that for an input sequence consisting of R rounds, and for sufficiently large k,*

$$\mathbf{E}\left[\sum_r \text{cost}^r_{OFF}\right] \leq 2\alpha \sum_{i=1}^{d} N_i + O(R) \ .$$

Proof. We present an offline algorithm OFF using a buffer of size k that has the desired upper bound on the expected cost. The algorithm ensures that in the beginning of a round at least $\alpha N_i + 1$ slots are cleared from every class C_i. This means that the algorithm has more than $\alpha k + \log^{d-1} k$ elements removed that are held by OPT'.

For a class C_i, one slot is promoted in each round. Consider the sequence of slots from C_i that are promoted over all rounds. We partition this sequence into *phases* such that each phase contains $N_i - \alpha N_i - 1$ pairwise different slots. Then, in the beginning of each phase, i.e., in the beginning of the round in which the first promotion of the new phase takes place, OFF clears all slots that are not contained in the phase. Note that due to the definition of a phase, exactly $\alpha N_i + 1$ slots in class C_i are cleared. Also note that these slots remain cleared during the whole phase, since none of them is promoted.

Let $\text{cost}^r_{\text{OFF}}(i)$ denote the contribution of class C_i to $\text{cost}^r_{\text{OFF}}$, i.e., $\text{cost}^r_{\text{OFF}}(i)$ is 1 if a cleared slot in class C_i is promoted in round r, and 0 otherwise. Clearly $\sum_{i=1}^{d} \sum_r \text{cost}^r_{\text{OFF}}(i) = \sum_r \text{cost}^r_{\text{OFF}}$. In the following we analyze $\mathbf{E}[\sum_r \text{cost}^r_{\text{OFF}}(i)]$ for $i \in \{1, \ldots, d\}$.

Observe that the contribution of one phase to the value of $\sum_r \text{cost}^r_{\text{OFF}}(i)$ is at most $\alpha N_i + 1$. Let p_i be the total number of phases of class C_i, then we get $\mathbf{E}[\sum_r \text{cost}^r_{\text{OFF}}(i)] \leq \mathbf{E}[p_i](\alpha N_i + 1)$. Let X be the length of a single phase (except the last phase which may be incomplete and, therefore, shorter). Clearly,

$$\mathbf{E}[p_i] \leq 1 + \frac{1}{\mathbf{Pr}[X \geq N_i \ln(1/\alpha)/4]} \cdot \frac{4R}{N_i \ln(1/\alpha)} .$$

We first show the following:

Claim 4.14. $\mathbf{Pr}[X \geq N_i \ln(1/\alpha)/4] \geq \frac{1}{2}$.

Proof. The analysis of the random variable X is based on a straightforward coupon collector type argument.

Consider a phase and let X_j be the number of rounds between the promotion of the $(j-1)$-th distinct slot and the j-th distinct slot. Then $X = X_1 + X_2 + \cdots + X_{N_i - \alpha N_i} - 1$. The variable X_j has value ℓ with probability $\left(\frac{j-1}{N_i}\right)^{\ell-1} \cdot \frac{N_i - j + 1}{N_i}$ for any integer $\ell \geq 1$. We have

$$\mathbf{E}[X_j] = \frac{N_i}{N_i - j + 1}, \quad \mathbf{E}[X_j^2] = \frac{(N_i + j - 1)N_i}{(N_i - j + 1)^2} \quad , \text{ and}$$

$$\mathbf{Var}[X_j] = \mathbf{E}[X_j^2] - \mathbf{E}[X_j]^2 = \frac{(j-1)N_i}{(N_i - j + 1)^2} .$$

We get

$$\begin{aligned}\mathbf{E}[X] &= \frac{N_i}{N_i} + \frac{N_i}{N_i - 1} + \cdots + \frac{N_i}{\alpha N_i + 1} - 1 \\ &= N_i(H_{N_i} - H_{\alpha N_i}) - 1 \\ &\geq N_i \cdot \ln N_i - N_i \cdot (\ln(\alpha N_i) + 1) - 1 \\ &= N_i \ln(1/\alpha) - (N_i + 1) \geq N_i \ln(1/\alpha)/2 \ ,\end{aligned}$$

where the first inequality uses the fact that $\ln a < H_a \leq \ln a + 1$ for $a \geq 1$, and the second inequality holds for sufficiently small α (i.e., for sufficiently large k).

From Chebyshev's inequality we get

$$\begin{aligned}&\mathbf{Pr}\left[X \leq N_i \ln(1/\alpha)/4\right] \\ &\qquad \leq \mathbf{Pr}\left[|X - \mathbf{E}[X]| \geq N_i \ln(1/\alpha)/4\right] \\ &\qquad \leq \frac{16}{N_i^2 \ln^2(1/\alpha)} \cdot \mathbf{Var}[X] \\ &\qquad = \frac{16}{N_i^2 \ln^2(1/\alpha)} \cdot \sum_{j=1}^{(1-\alpha)N_i} \mathbf{Var}[X_j] \\ &\qquad \leq \frac{16}{\ln^2(1/\alpha)} \cdot \sum_{j=1}^{(1-\alpha)N_i} \frac{1}{(N_i - j + 1)^2} \\ &\qquad \leq \frac{16}{\ln^2(1/\alpha)} \cdot \sum_{j=1}^{\infty} \frac{1}{j^2} \leq \frac{32}{\ln^2(1/\alpha)} \leq \frac{1}{2} \ ,\end{aligned}$$

where the third step follows since the X_j's are independent and the last step follows for sufficiently small α. □

The lemma follows since

$$\begin{aligned}\mathbf{E}\left[\textstyle\sum_r \mathrm{cost}^r_{\mathrm{OFF}}\right] &= \sum_{i=1}^{d} \mathbf{E}\left[\textstyle\sum_r \mathrm{cost}^r_{\mathrm{OFF}}(i)\right] \leq \sum_{i=1}^{d} \mathbf{E}[p_i](\alpha N_i + 1) \\ &\leq \sum_{i=1}^{d} 2\,\mathbf{E}[p_i]\alpha N_i \leq \sum_{i=1}^{d} \left(2\alpha N_i + \frac{16R\alpha}{\ln(1/\alpha)}\right) \\ &\leq 2\alpha \sum_{i=1}^{d} N_i + \frac{16dR\alpha}{\ln(1/\alpha)} = 2\alpha \sum_{i=1}^{d} N_i + O(R) \ .\end{aligned}$$

Here, the second inequality uses the fact that $\alpha N_i \geq 1$, which follows from the integrality of αN_i. □

Theorem 4.15. *Any online algorithm for the reordering buffer management problem has competitive ratio at least* $\Omega(\log\log k)$.

Proof. Combining Lemma 4.12 with Lemma 4.6 shows that the expected cost of an online algorithm ONL on the input sequence consisting of R rounds is at least $\sum_{i=1}^{d} N_i + R \cdot \frac{\log\log k}{8}$. Combining Lemma 4.13 with Lemma 4.7 shows that the expected cost of OPT on the input sequence consisting of R rounds is at most $(2\alpha+1)\sum_{i=1}^{d} N_i + O(R)$. Therefore, the competitive ratio is at least

$$\frac{\sum_{i=1}^{d} N_i + R \cdot \log\log k/8}{(2\alpha+1)\sum_{i=1}^{d} N_i + O(R)}.$$

Letting R tend to infinity gives the theorem. □

4.4 The Deterministic Upper Bound

In this section we present a deterministic, $O(\sqrt{\log k})$-competitive online algorithm for the reordering buffer management problem. The cost for switching to a color c can be described by a weight w_c for this color. We assume that for all colors c it holds that $1 \leq w_c \leq W$, where W is polynomially bounded in k.

4.4.1 The Algorithm

Without loss of generality we can assume that an algorithm for the reordering buffer management problem works according to the following general scheme. In each step the algorithm has an *active output color*, which is equal to the color of the last element that was appended to the output sequence. If there is at least one element with this active color in the buffer, the earliest among these elements is removed, appended to the output sequence, and the next element from the input sequence takes its place in the buffer. Otherwise, if there are no more elements of the active output color in the buffer, the algorithm performs a color change and chooses a new color (among the colors present in the buffer) to output next.

Note that the algorithm only has to make a decision if a color change is performed. Therefore, we describe our algorithm LCC (Largest Color Class) by specifying how the new output color is chosen when a color change is required. But first, we introduce some further notation. The *i-th step* of an algorithm is the step in which the algorithm appends the i-th element to the output sequence. The *buffer content at step i* for an algorithm ALG is the set of elements in ALG's buffer right before the i-th element is moved to the output. For the analysis we will assume that

the buffer always contains k elements. This may not be true at the end of the input sequence as the algorithm runs out of elements to fill the buffer. However, this part of the sequence does not influence our asymptotic bounds.

Let for a given color c at a given time t, $\phi_c^t = \frac{w_c}{n_c^t}$ denote the *cost effectiveness* of color c at time t, where n_c^t denotes the number of elements of color c that are in the buffer of LCC at time t. In the following we drop the superscript, as the time step t will be clear from the context.

For each time step, we partition the colors into classes according to their cost effectiveness. For $i \in \{-\lceil \log k \rceil, \ldots, \lceil \log W \rceil\}$, the class C_i consists of colors with cost effectiveness between 2^i and 2^{i+1}. Let $d = O(\log k)$ denote the number of different classes.

The general idea behind the algorithm is that it aims to remove colors from classes that occupy a large fraction of the space in the buffer. To this end the algorithm selects the class that currently occupies the largest space in the buffer (i.e., it contains at least $\frac{k}{d}$ elements) and *marks* all colors in this class for eviction (Line 6 in Algorithm 4.1). Whenever a color change is required, one of these marked colors is chosen as the new output color, and unmarked. If there are no marked colors left, the new class that occupies the largest space is selected and the process is repeated.

This algorithmic idea is combined with a mechanism that penalizes colors for using up space in the buffer at the time a color change occurs. This is similar to techniques used e.g. in [74; 44; 16], and ensures that colors (in particular colors with a low weight) do not stay in the buffer for too long, thereby blocking valuable resources.

To realize all this, our algorithm LCC maintains a counter P and additional penalty counters P_c for every color c. LCC also maintains a flag for every color that indicates if the color is marked. Whenever a color is not in the buffer, its penalty counter is zero. In particular, in the beginning of the algorithm all penalty counters, including the counter P, are zero. The formal description of our algorithm for selecting a new output color is given as Algorithm 4.1.

Before the algorithm selects a marked color c_m as the new output color, it assigns a value of w_{c_m} to the penalty counter P (Line 8). In a *post-processing phase* (after outputting all elements of color c_m) this penalty is distributed to penalty counters of individual colors, as follows. The penalty counter P is continuously decreased at rate 1, while the penalty counters of colors in the buffer are increased at rate $\frac{n_c}{k}$ where n_c denotes the number of elements of color c that are in the buffer (Line 1 and Line 2). Note that we assume that the buffer is full, hence, the rate of

Algorithm 4.1 Largest Color Class (LCC)

Output: a new output color

// let n_c denote the number of elements with color c in the buffer

1: $\forall$color $c : t_c \leftarrow \frac{w_c - P_c}{n_c/k}$; $t \leftarrow \min(\{t_c \mid \text{color } c\} \cup \{P\})$;

2: $P \leftarrow P - t$; $\forall$color $c : P_c \leftarrow P_c + \frac{n_c}{k} \cdot t$

// the above ensures that t is small enough s.t. $P \geq 0$ and $P_c \leq w_c$ for all c

3: **if** $P = 0$ **then**

4: **if** no marked color exists **then**

5: *// let $C_{\max}$ be the class that occupies the largest space in the buffer*

6: mark all colors in $C_{\max}$

7: **end if**

// let c_m denote an arbitrary marked color

8: $P \leftarrow w_{c_m}$

9: $P_{c_m} \leftarrow 0$

10: unmark color c_m

11: **return** color c_m as the new output color

12: **else**

13: $c_a \leftarrow \arg\min_c t_c$ *// pick color c_a such that $P_{c_a} = w_{c_a}$*

14: $P_{c_a} \leftarrow 0$

15: unmark color c_a if it was marked

16: **return** color c_a as the new output color

17: **end if**

decrease of the P-counter equals the total rate of increase of P_c-counters.

When a counter P_c reaches w_c the penalty distribution is interrupted; the P_c-counter is reset to 0; and the corresponding color c returned as the new output color (Lines 13–16). The penalty distribution resumes when all elements with color c have been removed and the next color change takes place. The penalty distribution and the post-processing phase ends once the P-counter reaches 0.

We note that the algorithm can be significantly simplified for the uniform cost model, i.e. when all colors c have weight $w_c = 1$.

4.4.2 The Analysis

Let for a reordering algorithm ALG and an input sequence σ, ALG(σ) denote the output sequence generated by ALG on input σ. A *color-block* of an output sequence is a maximal subsequence of consecutive elements with the same color. The *cost* of a

color-block of color c is equal to the weight w_c of c. The cost $|\mathrm{ALG}(\sigma)|$ of algorithm ALG on input σ is defined as the sum of the costs taken over all color-blocks in the output sequence $\mathrm{ALG}(\sigma)$.

For a color-block b we use $s_{\mathrm{start}}(b)$ and $s_{\mathrm{end}}(b)$ to denote the start index of b and end index of b, respectively, in the output sequence. This is the same as the time step when the first and the last element of b is appended to the output sequence.

A Few Simple Cases

In this section we first identify different types of color-blocks for which we can fairly easily derive a bound on their respective contribution to the cost $|\mathrm{LCC}(\sigma)|$ of our online algorithm. Later we will introduce a technique that enables us to handle the remaining color-blocks.

We call a color-block of LCC that is not generated in a post-processing phase a *normal color-block* (these are the color-blocks produced when the algorithm switches to the respective color in Line 11). Other color-blocks are called *forced color-blocks* (the ones caused by Line 16). The following lemma shows that we can focus our analysis on normal color-blocks.

Lemma 4.16. *The sum of the costs of forced color-blocks is at most the sum of the costs of normal color-blocks.*

Proof. The total cost for forced color-blocks does not exceed the total penalty that is distributed to colors during the post-processing phase of normal color-blocks. The penalty that is distributed during the post-processing phase of a (normal) color-block b with color c is equal to w_c, i.e., the cost for b. Summing over all normal color-blocks gives the lemma. □

In the following we use OPT to denote an optimal offline algorithm. We say that an element is *online-exclusive in step i*, if in this step the element is in LCC's buffer but has already been removed from OPT's buffer. Similarly we call an element *opt-exclusive in step i*, if it is in OPT's buffer but not in LCC's buffer at this time. Note that by this definition in every step the number of online-exclusive elements equals the number of opt-exclusive elements, since the size of LCC's and OPT's buffer is the same.

We extend the above definition to colors. We say that a *color c is online-exclusive* in step i, if there exists an element of color c that is online-exclusive. Finally, we say that a *color-block b is online-exclusive* if its color is online-exclusive in step $s_{\mathrm{start}}(b)$. The following lemma derives a bound on the cost of online-exclusive color-blocks.

Lemma 4.17. *The cost of LCC for online-exclusive color-blocks is at most* $|OPT(\sigma)|$.

Proof. Let b denote an online-exclusive color-block in $\mathrm{LCC}(\sigma)$, let e denote its first element, and let c be the color of b. Let b' denote the color-block of color c that precedes b. In case b is the first color-block of color c we define $s_{\mathrm{end}}(b') = -1$ for the following argument. Note that element e is not yet in the buffer at step $s_{\mathrm{end}}(b') + 1$ as in this case it would be appended to the output sequence in step $s_{\mathrm{end}}(b') + 1$.

Let b_{opt} denote the color-block of OPT that contains the element e. Clearly, this block ends after step $s_{\mathrm{end}}(b') + 1$ as e only arrives after this step. Since b is online-exclusive, its first element (i.e. e) is removed from the buffer of OPT before step $s_{\mathrm{start}}(b)$.

Altogether, we have shown that there exists a color-block b_{opt} in $\mathrm{OPT}(\sigma)$ which has color c and ends in the interval $\big(s_{\mathrm{end}}(b') + 1, s_{\mathrm{end}}(b)\big)$. We match the online-exclusive block b to b_{opt}. In this way we can match every online-exclusive block to a unique block in $\mathrm{OPT}(\sigma)$ with the same color. This gives the lemma. □

Another class of color-blocks for which we can easily derive a bound on the contribution to the cost $|\mathrm{LCC}(\sigma)|$ is given by so-called *opt-far color-blocks* defined as follows. A normal color-block b from the sequence $\mathrm{LCC}(\sigma)$ is called *opt-far*, if during its post-processing phase the number of online-exclusive elements never drops below $\frac{k}{\sqrt{\log k}}$. This means that throughout the whole post-processing phase for b the buffers of LCC and OPT are fairly different. The following lemma derives an upper bound on the cost of opt-far blocks in an output sequence generated by LCC.

Lemma 4.18. *The cost of LCC for opt-far color-blocks is* $O(\sqrt{\log k}) \cdot |OPT(\sigma)|$.

Proof. Fix an opt-far color-block b, and let c denote the color of b. During the post-processing phase for b the number of online-exclusive elements is always at least $\frac{k}{\sqrt{\log k}}$. Therefore, at least a $\frac{1}{\sqrt{\log k}}$ fraction of the penalty distributed during the post-processing phase goes to online-exclusive colors. The total cost for online-exclusive color-blocks is at least as large as the penalty that these colors receive, since the penalty of a color c cannot increase beyond its cost w_c.

Hence, the total penalty distributed during the post processing phases of opt-far color-blocks is at most $\sqrt{\log k}$ times the cost for online-exclusive color-blocks. This in turn is at most as large as $|\mathrm{OPT}(\sigma)|$ due to Lemma 4.17. The lemma follows by observing that the total penalty distributed during post-processing phases of opt-far color-blocks is equal to the cost of these blocks. □

The Potential

A crucial ingredient for the proof of the upper bound on the competitive ratio of the LCC algorithm is the way we handle *normal* color-blocks that are neither *online-exclusive* nor *opt-far*. For this we introduce the notion of *potential*. The idea is that, on the one hand, the total potential is bounded by some function in terms of the optimal cost $|\mathrm{OPT}(\sigma)|$ (see Claim 4.19a). On the other hand, we will show that normal color-blocks that are neither opt-exclusive nor opt-far generate a large potential. This allows us to derive a bound on the contribution of these color-blocks to the cost $|\mathrm{LCC}(\sigma)|$.

The definition of potential is based on the differences in the buffer between LCC and OPT. In the following we use τ_j to denote the start index of the j-th color-block of OPT.

For an element e_τ that is appended to the output sequence $\mathrm{LCC}(\sigma)$ at time τ we define for $\tau_j > \tau$

$$\varphi(\tau, \tau_j) = \begin{cases} 0 & \text{if OPT processed } e_\tau \text{ before step } \tau_j, \\ 1 & \text{otherwise.} \end{cases}$$

$\varphi(\tau, \tau_j)$ simply measures whether the element e_τ occupies a slot in OPT's buffer at time τ_j. We say that element e_τ generates *potential* $w_{c_j} \cdot \varphi(\tau, \tau_j)$ for time step τ_j, where c_j denotes the color of the j-th color-block in $\mathrm{OPT}(\sigma)$.

For technical reasons we also introduce a *capped potential* as follows. We define

$$\hat{\varphi}(\tau, \tau_j) = \begin{cases} 0 & \text{if OPT processed } e_\tau \text{ before step } \tau_j \text{ or at least } k/\sqrt{\log k} \\ & \text{elements } e_{\tau'} \text{ with } \tau' < \tau \text{ have } \varphi(\tau', \tau_j) = 1, \\ 1 & \text{otherwise.} \end{cases}$$

$\hat{\varphi}(\tau, \tau_j)$ measures whether the element e_τ is one of the first $\frac{k}{\sqrt{\log k}}$ elements to occupy a slot in OPT's buffer at time τ_j, where elements are ordered according to their appearance in $\mathrm{LCC}(\sigma)$. We say that element e_τ generates *capped potential* $w_{c_j} \cdot \hat{\varphi}(\tau, \tau_j)$ for time step τ_j, where c_j denotes the color that is processed by OPT at time τ_j.

We use $\hat{\varphi}(\tau) := \sum_{j:\tau_j > \tau} w_{c_j} \hat{\varphi}(\tau, \tau_j)$ to denote the total capped potential generated by the element at position τ in $\mathrm{LCC}(\sigma)$. We define the *total capped potential* $\hat{\varphi}$ by $\hat{\varphi} := \sum_\tau \hat{\varphi}(\tau)$.

Claim 4.19. *The capped potential fulfills the following properties:*

(a) $\hat{\varphi} \leq \frac{k}{\sqrt{\log k}} \cdot |OPT(\sigma)|$.

(b) *Let* $\tau < t < \tau_j$, *and assume that the number of online-exclusive elements in step* t *is at most* $\frac{k}{\sqrt{\log k}}$. *Then* $\hat{\varphi}(\tau, \tau_j) = \varphi(\tau, \tau_j)$, *and, hence, the* capped potential $w_{c_j} \cdot \hat{\varphi}(\tau, \tau_j)$ *generated by* e_τ *for* τ_j *is equal to the potential. In other words the contribution of* e_τ *is not capped.*

Proof. The first statement follows from the fact that the capped potential generated for a time step τ_j cannot exceed $\frac{k}{\sqrt{\log k}} \cdot w_{c_j}$, where w_{c_j} is the cost of OPT in the step. This holds because of the cap. Since the potential is generated for time steps τ_j that correspond to color changes by OPT, the statement follows.

Now, assume for contradiction that the second statement does not hold. Since obviously $\hat{\varphi}(\tau, \tau_j) \leq \varphi(\tau, \tau_j)$, it must hold that $\hat{\varphi}(\tau, \tau_j) = 0$ and $\varphi(\tau, \tau_j) = 1$. This means that element e_τ occupies a place in OPT's buffer at time τ_j, but there are at least $\frac{k}{\sqrt{\log k}}$ elements $e_{\tau'}$, $\tau' < \tau < t$, that also occupy a place in OPT's buffer at time τ_j, and therefore e_τ's contribution is *capped.* But all these elements are opt-exclusive at time t. Since at any time step the number of opt-exclusive elements must be equal to the number of online-exclusive elements, we can conclude that in step t there are more than $\frac{k}{\sqrt{\log k}}$ online-exclusive elements. This is a contradiction. □

The Main Theorem

Theorem 4.20. *LCC is a deterministic online algorithm with competitive ratio* $O(\sqrt{\log k})$.

Proof. The algorithm LCC marks all colors in a class, and then selects an arbitrary marked color whenever it has to do a normal color change. When no marked colors are left, it again marks all colors in some class and continues.

We call the time between two marking operations, or after the last marking operation, a *phase.* Fix some phase P and let C denote the set of colors that get marked in the beginning of the phase. Let for $c \in C$, s_c denote the number of elements of color c in LCC's buffer at the time of the marking operation that starts P. Further, let ϕ denote the lower bound on the cost effectiveness of colors in C, i.e., $\phi \leq \frac{w_c}{s_c} \leq 2\phi$ holds for all colors $c \in C$. We call a color change *normal* (*forced*) if it starts a normal (forced) color-block in the output sequence LCC(σ). In LCC(σ) the phase consists of a consecutive subsequence of elements, starting with an element of a color in C and ending with the last element of a color-block from the post-processing phase of the last normal color change of the phase.

Let cost(P) denote the cost incurred by LCC during the phase. This cost consists of color changes to colors in C (either normal or forced), and of color changes

to colors not in C (these are forced). Let $\text{ncost}(P)$ and $\text{fcost}(P)$ denote the cost incurred by LCC during the phase for normal and forced color changes, respectively. Further, let $\text{ncost} := \sum_{\text{phase } P} \text{ncost}(P)$ denote the total *normal cost* summed over all phases. In the light of Lemma 4.16 it is sufficient to relate ncost to the optimal cost $|\text{OPT}(\sigma)|$. In order to do this we distinguish the following cases:

Case 1 The normal cost $\text{ncost}(P)$ is at most $\frac{9}{10} \cdot \text{fcost}(P)$. Let $\text{ncost}_{\text{small}}$ denote the normal cost summed over all phases P that fulfill this condition and let $\text{ncost}_{\text{large}}$ denote the normal cost summed over other phases (i.e., $\text{ncost}_{\text{large}} = \text{ncost} - \text{ncost}_{\text{small}}$). Then

$$\text{ncost}_{\text{small}} \leq \frac{9}{10} \sum_{P} \text{fcost}(P) \leq \frac{9}{10} \text{ncost} = \frac{9}{10} (\text{ncost}_{\text{small}} + \text{ncost}_{\text{large}}) ,$$

where the second inequality follows from Lemma 4.16. This gives $\text{ncost}_{\text{small}} \leq 10\,\text{ncost}_{\text{large}}$. In the following cases we show that $\text{ncost}_{\text{large}} = O(\sqrt{\log k}) \cdot |\text{OPT}(\sigma)|$. Then we have that the normal cost $\text{ncost}_{\text{small}}$ generated by phases that fulfill the condition for Case 1 is $O(\sqrt{\log k}) \cdot |\text{OPT}(\sigma)|$.

Case 2 The cost of OPT during the phase is at least $\frac{1}{4} \cdot \text{ncost}(P)$. The total normal cost generated by phases that fulfill this condition is $O(|\text{OPT}(\sigma)|)$.

Case 3 The cost of online-exclusive color-blocks generated during the phase is at least $\frac{1}{4} \cdot \text{ncost}(P)$. Then we can amortize the normal cost of the phase against the cost of online-exclusive color-blocks, which in turn can be amortized against the cost of OPT by Lemma 4.17. This gives that the total normal cost generated by phases that fulfill the conditions for this case is $O(|\text{OPT}(\sigma)|)$.

Case 4 The cost of opt-far color-blocks generated during the phase is at least $\frac{1}{4} \cdot \text{ncost}(P)$. Then we can amortize the cost of the phase against the cost of opt-far color-blocks, which in turn can be amortized against the cost of OPT by Lemma 4.18.

Hence, the total cost for phases that fulfill the conditions for this case is $O(\sqrt{\log k}) \cdot |\text{OPT}(\sigma)|$.

Case 5 In the following we assume that none of the above cases occurs. This means there must exist a subset $C' \subset C$ of colors marked in the phase P such that for each color $c \in C'$ its first color-block in the phase is

Property a not online-exclusive,

Property b not opt-far, and

Property c not forced.

Further, we have that

Property d elements of colors in C' are not appended to the output sequence by OPT during the phase;

Property e and $\text{cost}(C') \geq \frac{1}{10}\,\text{cost}(C)$,

where we use $\text{cost}(S) := \sum_{c \in S} w_c$ for a set S of colors.

To see this we generate C' as follows. First take all colors from C (colors initially marked in the phase) and remove colors among them for which the first color change is forced (this ensures Property c). The cost of the remaining set of colors is exactly $\text{ncost}(P)$. Then remove colors for which the first block of the phase is online-exclusive or opt-far, and colors that are requested by OPT during the phase. Since we are not in Case 2, Case 3 or Case 4, this step can only remove colors with a total cost of $\frac{3}{4}\,\text{ncost}(P)$. After this Properties a, b and d hold. This gives the set C'.

Property e can be seen as follows. By the construction $\text{cost}(C') \geq \frac{1}{4}\,\text{ncost}(P)$. From the fact that Case 1 does not hold we get that $\text{ncost}(P) \geq \frac{9}{10}\,\text{fcost}(P)$, and, hence, $2\,\text{ncost}(P) \geq \frac{9}{10}\,\text{cost}(P) \geq \frac{9}{10}\,\text{cost}(C)$. This gives $\text{cost}(C') \geq \frac{1}{10}\,\text{cost}(C)$.

Let S denote the set of elements with colors in C' that are in LCC's buffer in the beginning of the phase. We will show that these elements generate a large potential after the end of the phase. From this it follows that we can amortize the cost of the phase against $|\text{OPT}(\sigma)|$ because of the following argument.

Assume that for some value Z the elements in S generate a potential of at least $Z \cdot \text{cost}(C')$ after time t, where t is the index of the last time step of the phase. Observe that, according to Property b above, the (first) color-blocks of colors in C' that are generated during the phase are not opt-far. This means that during the post-processing phase of each of these blocks, the number of online-exclusive items falls below $\frac{k}{\sqrt{\log k}}$ at some point. This means that we can apply Claim 4.19b to all elements in S, which gives that elements in S also generate at least $Z \cdot \text{cost}(C') \geq \frac{Z}{10} \cdot \text{cost}(C)$ *capped potential* after time t, as their contribution to the potential is not capped.

Claim 4.19a tells us that the total capped potential is at most $\frac{k}{\sqrt{\log k}} \cdot |\text{OPT}(\sigma)|$. Therefore, the total normal cost generated by phases that fulfill the conditions for Case 5 is $O(\frac{k}{Z\sqrt{\log k}}) \cdot |\text{OPT}(\sigma)|$.

By showing that elements in S generate at least $Z \cdot \text{cost}(C') = \Omega(\frac{k}{\log k}) \cdot \text{cost}(C')$ potential we get that the cost of the phases satisfying Case 5 is $O(\sqrt{\log k}) \cdot |\text{OPT}(\sigma)|$.

For completing the analysis of Case 5 it remains to show the above bound on the potential generated by the elements of S. For this, we first show that the

cardinality of the set S is large. We have

$$|S| = \sum_{c \in C'} s_c \geq \sum_{c \in C'} \frac{w_c}{2\phi} \geq \frac{1}{20\phi} \sum_{c \in C} w_c \geq \frac{1}{20} \sum_{c \in C} s_c \geq \frac{k}{20d} = \Omega\Big(\frac{k}{\log k}\Big) \ ,$$

where the first and third inequality follows since $\phi \leq \frac{w_c}{s_c} \leq 2\phi$; the second inequality holds since $\sum_{c \in C'} w_c = \text{cost}(C') \geq \frac{1}{10}\text{cost}(C) \geq \frac{1}{10}\sum_{c \in C} w_c$. The last inequality follows since the algorithm LCC selects a class that occupies the largest space in the buffer, and, hence, occupies at least space $\frac{k}{d}$, where d denotes the number of classes.

Claim 4.21. *Let S denote a set of elements that are opt-exclusive at time t, and let s_c denote the number of elements of color c in S. Assume that there is a value ϕ such that $\phi \leq \frac{w_c}{s_c} \leq 2\phi$ holds for all colors with elements in S. Then the contribution to the potential by elements from S generated after time t is at least $\frac{1}{8}|S| \cdot \text{cost}(S)$.*

Proof. Let $c_1, \ldots, c_\ell$ denote the colors of the elements in S, ordered according to the times $\tau_1 < \cdots < \tau_\ell$ at which the first element of a color is evicted by OPT. Let i denote the smallest number such that $\sum_{j=1}^{i} w_{c_j} \geq \frac{1}{2}\text{cost}(S)$.

We show that the number of elements with colors $c_i, \ldots, c_\ell$ is large. For any j we have $\phi \leq \frac{w_{c_j}}{s_{c_j}} \leq 2\phi$, and, hence, $\text{cost}(S) \geq \phi|S|$. Therefore

$$\sum_{j=i}^{\ell} s_{c_j} \geq \frac{1}{2\phi} \sum_{j=i}^{\ell} w_{c_j} \geq \frac{1}{4\phi}\text{cost}(S) \geq \frac{1}{4}|S| \ .$$

Each element e of a color in $\{c_i, \ldots c_\ell\}$ generates potential w_{c_j} at time τ_j for $1 \leq j \leq i$. The contribution of e to the potential generated after time t is therefore at least $\sum_{j=1}^{i} w_{c_j} \geq \frac{1}{2}\text{cost}(S)$. As the number of elements with color in $\{c_i, \ldots c_\ell\}$ is at least $\frac{1}{4}|S|$, the potential generated by them after time t is at least $\frac{1}{8}|S| \cdot \text{cost}(S)$. □

Applying the claim with t being the last step of the phase, gives that the elements from S generate potential $\Omega(\frac{k}{\log k}) \cdot \text{cost}(C')$ after the end of the phase. This finishes the analysis of Case 5.

The above cases show that the contribution of all phases to the cost of LCC is $O(\sqrt{\log k}) \cdot |\text{OPT}(\sigma)|$. This gives the theorem. □

4.5 Open Problems

An interesting open problem is improving the upper or lower bound on the competitive ratio of a randomized online algorithm for the reordering buffer management problem. In particular — is there an algorithm with competitive ratio $O(\log \log k)$?

Chapter 5

Generalized Caching Problem

5.1 Introduction

In this chapter we study the generalized caching problem in an online setting.

5.1.1 Related Work

Table 5.1 presents an overview of the results for the online caching problem in the randomized setting.

The study of the caching problem with uniform sizes and costs (*the paging problem*) in the online setting has been initiated by Sleator and Tarjan [80] in their work that introduced the framework of competitive analysis. They show that well known paging rules like LRU (Least Recently Used) or FIFO (First In First Out) are k-competitive, and that this is the best competitive ratio that any deterministic algorithm can achieve.

Fiat et al. [46] extend the study to the randomized setting and design a randomized $2H_k$-competitive algorithm, where H_k is the k-th Harmonic number. They also prove that no randomized online paging algorithm can be better than H_k-competitive. Subsequently, McGeoch and Sleator [68], and Achlioptas et al. [2] design randomized online algorithms that achieve this competitive ratio.

Weighted caching, where pages have uniform sizes but can have arbitrary cost, has been studied extensively because of its relation to the k-server problem. The results for the k-server problem on trees due to Chrobak et al. [35] yield a tight deterministic k-competitive algorithm for weighted caching. The randomized complexity of weighted caching has been resolved only recently, when Bansal et al. [19] designed a randomized $O(\log k)$-competitive algorithm.

The caching problem with non-uniform page sizes seems to be much harder.

	uniform sizes	arbitrary sizes
uniform costs	$2H_k$-competitive [46] H_k-competitive [68; 2] lower bound H_k [46]	$O(\log^2 k)$-competitive [58] $O(\log k)$-competitive [20]
arbitrary costs	$O(\log k)$-competitive [19]	$O(\log^2 k)$-competitive [20] **$O(\log k)$-competitive**

Table 5.1: An overview of the results for the online caching problem in the randomized setting. The new result is presented using bold font.

Already the offline version is NP-hard [58], and there was a sequence of results [58; 8; 37] that lead to the work of Bar-Noy et al. [21] which gives a 4-approximation for the offline problem.

For the online version, the first results consider special cases of the problem. Irani [58] shows that for the bit model and for the fault model in the deterministic case LRU is $(k+1)$-competitive. Cao and Irani [30], and Young [87], extend this result to the generalized caching problem.

In the randomized setting, Irani [58] gives an $O(\log^2 k)$-competitive algorithm for both bit and fault models, but for the generalized caching problem no $o(n)$-competitive ratio was known until the recent work of Bansal et al. [20]. They show how to obtain a competitive ratio of $O(\log^2 k)$ for the general model, and also a competitive ratio of $O(\log k)$ for the bit model and the fault model.

Since it has been known that no randomized online $o(\log k)$-competitive algorithm for generalized caching exists, the central open problem in this area was whether it is possible to design a randomized $O(\log k)$-competitive algorithm.

5.1.2 Result and Techniques

We present a randomized $O(\log k)$-competitive online algorithm for the generalized caching problem, improving the previous bound of $O(\log^2 k)$ by Bansal et al. [20]. This improved bound unifies all earlier results for special cases of the caching problem. It is asymptotically optimal as already for the problem with uniform page sizes and uniform fetching costs there is a lower bound of $\Omega(\log k)$ on the competitive ratio of any randomized online algorithm [46].

Our approach is similar to the approach used by Bansal et al. in their results

on weighted caching [19] and generalized caching [20]. In both these papers the authors first formulate a packing/covering linear program that forms a relaxation of the problem. They can solve this linear program in an *online* manner by using the online primal-dual framework for packing and covering problems introduced by Buchbinder and Naor [28]. However, using the framework as a black-box only guarantees an $O(\log n)$-factor between the cost of the solution obtained online and the cost of an optimal solution. They obtain an $O(\log k)$-guarantee by tweaking the framework for the special problem using ideas from the primal-dual scheme. Note that this $O(\log k)$-factor is optimal, i.e., in general one needs to lose a factor of $\Omega(\log k)$ when solving the LP online.

In the second step, they give a randomized rounding algorithm to transform the fractional solution into a sequence of integral cache states. Unfortunately, this step is quite involved. In [20] they give three different rounding algorithms for the general model and the more restrictive bit and fault models, where in particular the rounding for the fault model is quite complicated.

We use the same LP as Bansal et al. [20] and also the same algorithm for obtaining an online fractional solution. Our contribution is a more efficient and also simpler method for rounding the fractional solution online. We first give a rounding algorithm for monotone cost models (where $w_p \geq w_{p'}$ implies $c_p \geq c_{p'}$) and then extend it to work for the general model. Our rounding algorithm only loses a constant factor and, hence, we obtain an $O(\log k)$-competitive algorithm for generalized caching.

The chapter is based on joint work with Artur Czumaj, Matthias Englert and Harald Räcke [4].

5.2 The Linear Program

This section describes an LP for the generalized caching problem. It also shows how to generate good variable assignments which are used in the rounding algorithm of the next section. Although there are some minor notational differences, this largely follows the work of Bansal, Buchbinder, and Naor [20].

We assume that cost is incurred when a page is evicted from the cache, not when it is loaded into the cache. This means we will not pay anything for the pages remaining in the cache at the end. We may assume that the last request is always to a page of size k and zero cost. This does not change the cost of any algorithm in the original cost model. However, it does ensure that the cost in our alternate cost model matches the cost in the original model.

We begin by describing an integer program IP for the generalized caching problem. The IP has variables $x_p(i)$ indicating if page p has been evicted from the cache after the page has been requested for the i-th time. If $x_p(i) = 1$, page p was evicted after the i-th request to page p and has to be loaded back into the cache when the page is requested for the $(i+1)$-st time. The total cost is then $\sum_p \sum_i c_p x_p(i)$.

Let $B(t)$ denote the set of pages that have been requested at least once until and including time t and let $r(p,t)$ denote the number of requests to page p until and including time t. In a time step t in which page p_t is requested, the total size of pages other than p_t in the cache can be at most $k - w_{p_t}$. Thus, we require

$$\sum_{p \in B(t) \setminus \{p_t\}} w_p(1 - x_p(r(p,t))) \leq k - w_{p_t} \ .$$

Rewriting the constraint gives

$$\sum_{p \in B(t) \setminus \{p_t\}} w_p x_p(r(p,t)) \geq \sum_{p \in B(t)} w_p - k \ .$$

To shorten the notation, we define the total weight of a set of pages S as $W(S) := \sum_{p \in S} w_p$. Altogether, this results in the following IP formulation for the generalized caching problem.

$$\begin{aligned} \min \quad & \textstyle\sum_p \sum_i c_p \, x_p(i) \\ \text{s.t. } \forall_t : \quad & \textstyle\sum_{p \in B(t) \setminus \{p_t\}} w_p \, x_p(r(p,t)) \geq W(B(t)) - k \\ \forall_{p,i} : \quad & x_p(i) \in \{0,1\} \end{aligned} \tag{IP 1}$$

To decrease the integrality gap of our final LP relaxation, we add additional, redundant constraints to the IP.

$$\begin{aligned} \min \quad & \textstyle\sum_p \sum_i c_p \, x_p(i) \\ \text{s.t. } \forall_t \forall_{S \subseteq B(t) : W(S) > k} : \quad & \textstyle\sum_{p \in S \setminus \{p_t\}} w_p \, x_p(r(p,t)) \geq W(S) - k \\ \forall_{p,i} : \quad & x_p(i) \in \{0,1\} \end{aligned} \tag{IP 2}$$

Unfortunately, even the relaxation of this IP formulation can have an arbitrarily large integrality gap. However, in an integral solution any $w_p > W(S) - k$ cannot give any additional advantage over $w_p = W(S) - k$ for a constraint involving set S. Therefore, it is possible to further strengthen the constraints without affecting an integral solution. For this, we define $\tilde{w}_p^S := \min\{W(S) - k, w_p\}$. Relaxing the

Procedure 5.1 fix-set(S, t, x, y)

Input: Current time step t, current variable assignments x and y, a minimal set $S \subseteq B(t)$.

Output: New assignments for x and y. Return if S becomes non-minimal or constraint t, S in primal-LP is satisfied.

1: **while** $\sum_{p \in S \setminus \{p_t\}} \tilde{w}_p^S \cdot x_p(r(p,t)) < W(S) - k$ **do** // *constraint for* t, S *violated*
2: infinitesimally increase $y_S(t)$
3: **for** each $p \in S$ **do**
4: $v := \sum_{\tau : r(p,\tau) = r(p,t), p \neq p_\tau} \sum_{S \subseteq B(\tau) : p \in S, W(S) > k} \tilde{w}_p^S \, y_S(\tau) - c_p$
// *v is a violation of the dual constraint for* $x_p(r(p,t))$
5: ***if*** $v \geq 0$ ***then*** $x_p(r(p,t)) := \frac{1}{k} \exp\left(\frac{v}{c_p}\right)$
6: ***if*** $x_p(r(p,t)) = 1$ ***then return*** // *S is not minimal any more*
7: **end for**
8: **end while**
9: **return** // *the primal constraint for step t and set S is fulfilled*

integrality constraint, we obtain an LP. As shown in Observation 2.1 of Bansal et al. [20], we can assume without loss of generality that $x_p(i) \leq 1$. This results in the final LP formulation.

$$\begin{aligned} & \min && \textstyle\sum_p \sum_i c_p \, x_p(i) \\ \text{s.t. } \forall_t \forall_{S \subseteq B(t) : W(S) > k} : & && \textstyle\sum_{p \in S \setminus \{p_t\}} \tilde{w}_p^S \, x_p(r(p,t)) \geq W(S) - k \qquad \text{(primal-LP)} \\ & \forall_{p,i} : && x_p(i) \geq 0 \end{aligned}$$

The dual of primal-LP is

$$\begin{aligned} & \max && \textstyle\sum_t \sum_{S \subseteq B(t) : W(S) > k} (W(S) - k) \, y_S(t) \\ & \text{s.t. } \forall_{p,i} : && \textstyle\sum_{t : r(p,t) = i, p \neq p_t} \sum_{S \subseteq B(t) : p \in S, W(S) > k} \tilde{w}_p^S \, y_S(t) \leq c_p \qquad \text{(dual-LP)} \\ \forall_t \forall_{S \subseteq B(t) : W(S) > k} : & && y_S(t) \geq 0 \ . \end{aligned}$$

Procedure 5.1 will be called by our online rounding algorithm to generate assignments for the LP variables. Note that, as the procedure will not be called for all violated constraints, the variable assignments will not necessarily result in a feasible solution to primal-LP but will have properties which are sufficient to guarantee that our rounding procedure produces a feasible solution. We assume that all primal and dual variables are initially zero.

For a time step t, we say a set of pages S is *minimal* if, for every $p \in S$, $x_p(r(p,t)) < 1$. We note that by Observation 2.1 of Bansal et al. [20], whenever there is a violated constraint t, S in primal-LP, there is also a violated constraint $t, S' \subseteq S$ for a minimal set S'. The idea behind Procedure 5.1 is that it is called with a minimal set S. The procedure then increases the primal (and dual) variables of the current solution in such a way that one of two things happen: either the set S is not minimal any more because the value of $x_p(r(p,t))$ reaches 1 for some page $p \in S$ or the constraint t, S is not violated any more. At the same time, the following theorem guarantees that the primal variables are not increased too much, that is, that the final cost is still bounded by $O(\log k)$ times the cost of an optimal solution. Its proof follows exactly the proof of Theorem 3.1 from Bansal et al. [20].

Theorem 5.1. *Let $x_p(i)$ be the final variable assignments generated by successive calls to Procedure 5.1. The total cost $\sum_p \sum_i c_p x_p(i)$ is at most $O(\log k)$ times the cost of an optimal solution to the caching problem.*

5.3 The Online Algorithm

The online algorithm for the generalized caching problem works as follows. It computes primal and dual assignments x and y for LPs primal-LP and dual-LP, respectively, by repeatedly finding violated primal constraints and passing the constraint together with the current primal and dual solution to Procedure 5.1. Procedure 5.2 gives the outline of a single step of the online algorithm.

Procedure 5.2 online-step(t)

1: $x(p_t, r(p_t,t)) := 0$ // *put page p_t into the cache*
   // *some constraints may be violated*
2: $S := \{p \in B(t) \mid \gamma \cdot x(p, r(p,t)) < 1\}$
3: **while** constraint for S is violated **do**
4: fix-set(S, t, x, y) // *change the current solution*
5: adjust distribution μ to mirror new x
6: $S := \{p \in B(t) \mid \gamma \cdot x(p, r(p,t)) < 1\}$ // *recompute S*
7: **end while**
   // *the constraints are fulfilled*

Note that the primal "solution" may not be feasible. It may only fulfill a subset of the constraints, which, however, will be sufficient for our rounding procedure.

In addition to the fractional solutions x and y, the online algorithm maintains a probability distribution over valid cache states. Specifically, μ will be the uniform

distribution over k^2 subsets — each subset specifying a set of pages that are currently evicted from the cache. Some of the k^2 subsets may be identical. A randomized algorithm then chooses a random number r from $[1, \ldots, k^2]$ and behaves according to the r-th subset, i.e., whenever the r-th subset changes it performs the corresponding operations.

We will design the distribution μ in such a way that it closely mirrors the primal fractional solution x. In Section 5.3.1 we will show that each set in the support of μ is a complement of a valid cache state, i.e., the size constraints are fulfilled and the currently requested page is contained in the cache. In Section 5.3.2 we will show the way of updating μ in such way that a change in the fractional solution of the LP that increases the fractional cost by ϵ is accompanied by a change in the distribution μ with (expected) cost at most $O(\epsilon)$.

The rounding algorithm loses only a constant factor, which gives us a $O(\log k)$-competitive algorithm for generalized caching.

5.3.1 Ensuring that Cache States are Valid

We will set up some constraints for the sets in the support of μ, which guarantee that the sets describe complements of valid cache states. In order to define these constraints we introduce the following notation.

Let t denote the current time step and and set $x_p := x(p, r(p,t))$. Let $\gamma \geq 2$ denote a scaling factor to be chosen later, and define $z_p := \min\{\gamma x_p, 1\}$. The variable z_p is a scaling of the primal fractional solution x_p. We also introduce a rounded version of the scaling: we define $\bar{z}_p := \lfloor k \cdot z_p \rfloor / k$, which is simply the value of z_p rounded down to the nearest multiple of $1/k$. Note that due to the way the LP-solution is generated, $z_p > 0$ implies that $z_p \geq \gamma/k$. Therefore, rounding down can only change the value of z_p by a small factor. More precisely, we have $\bar{z}_p \geq (1 - 1/\gamma) \cdot z_p$.

We use S to denote the set of pages p that are fractional in the scaled solution, i.e., have $z_p < 1$ (or equivalently $\bar{z}_p < 1$). We divide these pages into *size classes* as follows. The class S_i contains pages whose size falls into the range $[2^i, 2^{i+1})$. See Figure 5.1 for an illustration.

We construct a set $L \subseteq S$ of "large pages" by selecting pages from S in decreasing order of size (ties broken according to page-id) until either the values of $\bar{z}$ for the selected pages add up to at least 1, or all pages in S have been selected. We use w_ℓ to denote the size of the smallest page in L, and i_ℓ to denote its class-id. Note that this construction guarantees that either $1 \leq \sum_{p \in L} \bar{z}_p < 2$ or $L = S$. The following claim shows that the second possibility only occurs when the weight of S

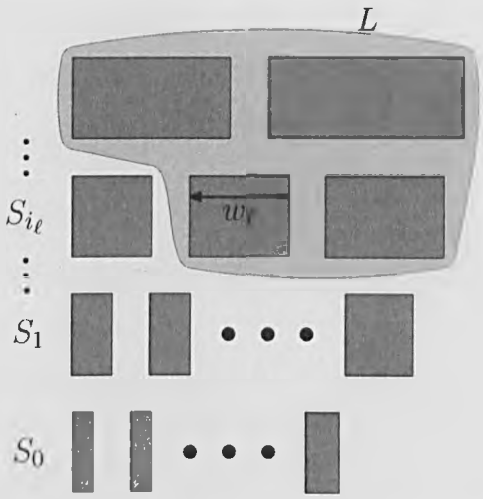

Figure 5.1: The size classes S_i and the set of large pages L obtained from the set S.

is small compared to the size of the cache k or while the online algorithm is serving a request (for example when the online algorithm iterates through the while-loop of Procedure 5.2).

Claim 5.2. *After a step of the online algorithm, we either have* $1 \leq \sum_{p \in L} \bar{z}_p < 2$ *or* $W(S) \leq k$.

Proof. If $W(S) \leq k$ there is nothing to prove. Otherwise, we have to show that we do not run out of pages during the construction of the set L. Observe that after the while-loop of Procedure 5.2 finishes, the linear program enforces the following condition for the subset S:

$$\sum_{p \in S} \min\{W(S) - k, w_p\} \cdot x_p \geq W(S) - k \ .$$

In particular, this means that $\sum_{p \in S} x_p \geq 1$ and hence $\sum_{p \in S} \bar{z}_p \geq (1 - 1/\gamma)\gamma \geq 1$, as $\gamma \geq 2$. Since the values of $\bar{z}_p$ for the pages p in S sum up to at least 1, we will not run out of pages when constructing L. □

Let D denote a subset of pages that are evicted from the cache. With a slight abuse of notation we also use D to denote the characteristic function of the set, i.e., for a page p we write $D(p) = 1$ if p belongs to D and $D(p) = 0$ if it does not. We are interested whether at time t the set D describes a complement of a valid cache state.

Definition 5.3. *We say that a subset D of pages* γ-mirrors *the fractional solution x if:*

1. $\forall p \in B(t) : \bar{z}_p = 0$ *implies* $D(p) = 0$ *(i.e., p is in the cache).*

2. $\forall p \in B(t) : \bar{z}_p = 1$ *implies* $D(p) = 1$ *(i.e., p is evicted from the cache).*

3. *For each class* S_i: $\lfloor \sum_{p \in S_i} \bar{z}_p \rfloor \leq \sum_{p \in S_i} D(p)$.

4. $\lfloor \sum_{p \in L} \bar{z}_p \rfloor \leq \sum_{p \in L} D(p)$.

Here $\bar{z}$ *is the solution obtained after scaling* x *by* γ *and rounding down to multiples of* $1/k$.

We refer to the constraints in the first two properties as *integrality constraints*, to the constraints in the third property as *class constraints*, and the constraint in the fourth property is called the *large pages constraint.*

Lemma 5.4. *A subset of pages that* γ*-mirrors the fractional solution* x *to the linear program, describes a complement of a valid cache state for* $\gamma \geq 16$.

Proof. Let D denote a set that mirrors the fractional solution x. In order to show that D is a complement of a valid cache state we need to show that the page p_t which is accessed at time t is not contained in D, and that the size of all pages which are not in D sums up to at most k.

Observe that the fractional solution is obtained by applying Procedure 5.1. Therefore, at time t the variable $x_{p_t} = x(p_t, r(p_t, t))$ has value 0. (It is set to 0 when Procedure 5.2 is called for time t, and it is not increased by Procedure 5.1 until time $t+1$.) Hence, we have $\bar{z}_{p_t} = 0$ and Property 1 in Definition 5.3 guarantees that p_t is not in D.

It remains to show that the size of all pages which are not in D sums up to at most k. This means we have to show

$$\sum_{p \in B(t) \setminus \{p_t\}} w_p D(p) \geq W(B(t)) - k \ . \tag{5.1}$$

Because of the integrality constraints we have

$$\sum_{p \in B(t) \setminus \{p_t\}} w_p D(p) = \sum_{p \in B(t) \setminus S} w_p D(p) + \sum_{p \in S \setminus \{p_t\}} w_p D(p) =$$
$$= \sum_{p \in B(t) \setminus S} w_p + \sum_{p \in S} w_p D(p) = W(B(t)) - W(S) + \sum_{p \in S} w_p D(p) \ .$$

In order to obtain Equation 5.1 it suffices to show that

$$\sum_{p \in S} w_p D(p) \geq W(S) - k \ .$$

For the case that $W(S) \leq k$ this is immediate, since the left hand side is always non-negative. Therefore in the following we can assume that $W(S) > k$, and, hence, $1 \leq \sum_{p \in L} \bar{z}_p < 2$ due to Claim 5.2.

If $2^{i_\ell} \geq W(S) - k$, then

$$\sum_{p \in S} w_p D(p) \geq \sum_{p \in L} w_p D(p) \geq 2^{i_\ell} \sum_{p \in L} D(p) \geq 2^{i_\ell} \geq W(S) - k \ ,$$

where the third inequality follows from the large pages constraint, and the fact that $\sum_{p \in L} \bar{z}_p \geq 1$.

In the remainder of the proof we can assume $2^{i_\ell} < W(S) - k$. We have

$$\begin{aligned}
\sum_{p \in S} w_p D(p) &\geq \sum_{i \leq i_\ell} \sum_{p \in S_i} w_p D(p) \\
&\geq \sum_{i \leq i_\ell} 2^i \cdot \sum_{p \in S_i} D(p) \\
&\geq \sum_{i \leq i_\ell} 2^i \cdot \left(\sum_{p \in S_i} \bar{z}_p - 1 \right) \\
&= \frac{1}{2} \sum_{i \leq i_\ell} \sum_{p \in S_i} 2^{i+1} \bar{z}_p - \sum_{i \leq i_\ell} 2^i \\
&\geq \frac{1}{2} \sum_{i \leq i_\ell} \sum_{p \in S_i} w_p \bar{z}_p - 2^{i_\ell + 1} \\
&\geq \frac{\gamma}{4} \sum_{i \leq i_\ell} \sum_{p \in S_i} \tilde{w}_p^S x_p - 2(W(S) - k) \ .
\end{aligned} \tag{5.2}$$

Here the second inequality follows since $w_p \geq 2^i$ for $p \in S_i$; the third inequality follows from the class constraints; the fourth inequality holds since $w_p \leq 2^{i+1}$ for $p \in S_i$. The last inequality uses the fact that $\bar{z}_p \geq (1 - 1/\gamma)\gamma x_p \geq \gamma/2 \cdot x_p$ for every $p \in S$, and that $w_p \geq \tilde{w}_p^S$.

Using the fact that $\bar{z}_p \geq \gamma/2 \cdot x_p$ we get

$$\frac{\gamma}{4} \sum_{p \in L \setminus S_{i_\ell}} \tilde{w}_p^S x_p \leq \frac{1}{2} \sum_{p \in L} \tilde{w}_p^S \bar{z}_p \leq \frac{1}{2} (W(S) - k) \sum_{p \in L} \bar{z}_p \leq W(S) - k \ ,$$

where the last inequality uses the fact that $\sum_{p \in L} \bar{z}_p \leq 2$. Adding the inequality $0 \geq \frac{\gamma}{4} \sum_{p \in L \setminus S_{i_\ell}} \tilde{w}_p^S x_p - (W(S) - k)$ to Equation 5.2 gives

$$\sum_{p \in S} w_p D(p) \geq \frac{\gamma}{4} \sum_{p \in S} \tilde{w}_p^S x_p - 3(W(S) - k) \geq (\gamma/4 - 3)(W(S) - k) \geq W(S) - k \ ,$$

for $\gamma \geq 16$. Here the second inequality holds because after serving a request the online algorithm guarantees that the constraint $\sum_{p \in S} \tilde{w}_p^S x_p \geq W(S) - k$ is fulfilled for the current set S. □

5.3.2 Updating the Distribution Online

We will show how to update online the distribution μ over subsets of pages in such a way, that we can relate the update cost to the cost of our linear programming solution x. We show that in each step the subsets in the support of μ *mirror* the current linear programming solution. Then Lemma 5.4 guarantees that we have a distribution over complements of valid cache states.

However, directly ensuring all properties in Definition 5.3 leads to a very complicated algorithm. Therefore, we partition this step into two parts. We first show how to maintain a distribution μ_1 over subsets D that fulfill the first three properties in Definition 5.3 (i.e., the integrality constraints and the class constraints). Then we show how to maintain a distribution μ_2 over subsets that fulfill the first and the last property.

From these two distributions we obtain the distribution μ as follows. We sample a subset D_1 from the first distribution and a subset D_2 from the second distribution, and compute $D = D_1 \cup D_2$ (or $D = \max\{D_1, D_2\}$ if D is viewed as the characteristic function of the set).

Clearly, if both sets D_1 and D_2 fulfill Property 1 from Definition 5.3, then the union fulfills Property 1. Furthermore, if one of D_1, D_2 fulfills one of the properties 2, 3, or 4, then the corresponding property is fulfilled by the union as these properties only specify a lower bound on the characteristic function D.

We will construct μ_1 and μ_2 to be uniform distributions over k subsets. Then the combined distribution μ is a uniform distribution over k^2 subsets, where some of the subsets may be identical.

In the following we assume that the values of $\bar{z}_p$ change in single steps by $1/k$. This is actually not true. Consider for example Line 1 of Procedure 5.2 where, after the time step t is increased, the variable $x(p_t, r(p_t, t))$ is set to 0. As x_{p_t} is a shorthand for $x(p_t, r(p_t, t))$, the value of x_{p_t}, and therefore also the value of $\bar{z}_{p_t}$, is set to 0. However, the drop in the value of $\bar{z}_{p_t}$ larger than $1/k$ can be simulated by several consecutive changes by a value of $1/k$.

Maintaining Distribution μ_1

In order to be able to maintain the distribution μ_1 at a small cost we strengthen the conditions that the sets D in the support of μ_1 have to fulfill. For each size class S_i we introduce *cost classes* $C_i^0, C_i^1, \ldots$, where $C_i^s = \{p \in S_i : c_p \geq 2^s\}$ (see Figure 5.2). Note that we have $S_i = C_i^0$.

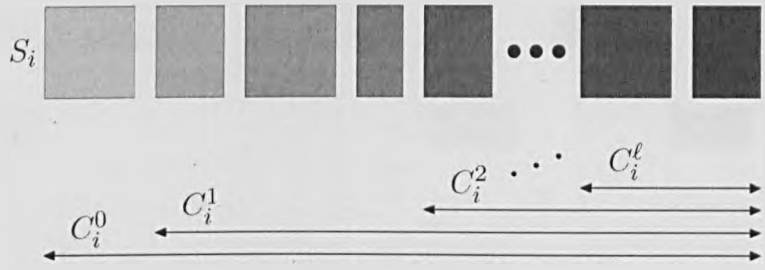

Figure 5.2: Cost classes.

For the subsets D in the support of μ_1 we require

A. For each subset D, for each size class S_i, and for all cost classes C_i^s

$$\left\lfloor \sum_{p \in C_i^s} \bar{z}_p \right\rfloor \leq \sum_{p \in C_i^s} D(p) \leq \left\lceil \sum_{p \in C_i^s} \bar{z}_p \right\rceil .$$

B. For each page p

$$\sum_D D(p) \cdot \mu_1(D) = \bar{z}_p .$$

Note that the second constraint above ensures that the integrality constraints are fulfilled, and the first set of constraints ensures that the class constraints are fulfilled. An example of a distribution that satisfies the constraints A and B is in Figure 5.3.

Increasing $\bar{z}_p$. Suppose that for some page p the value of $\bar{z}_p$ increases by $1/k$ (see Figure 5.4 for an example). Assume that $p \in S_i$ and $c_p \in [2^r, 2^{r+1})$, i.e., $p \in C_i^0, \ldots, C_i^r$. As we have to satisfy the property $\sum_D D(p)\mu_1(D) = \bar{z}_p$, we have to add the page p to a set D^* in the distribution μ_1, which currently does not contain p (i.e., we have to set $D^*(p) = 1$ for this set). We choose this set arbitrarily.

However, after this step some of the constraints

$$\left\lfloor \sum_{p \in C_i^s} \bar{z}_p \right\rfloor \leq \sum_{p \in C_i^s} D(p) \leq \left\lceil \sum_{p \in C_i^s} \bar{z}_p \right\rceil$$

corresponding to the cost classes C_i^s for $s \leq r$ may become violated. We repair the violated constraints step by step from $s = r$ to 0. We do that by moving the pages

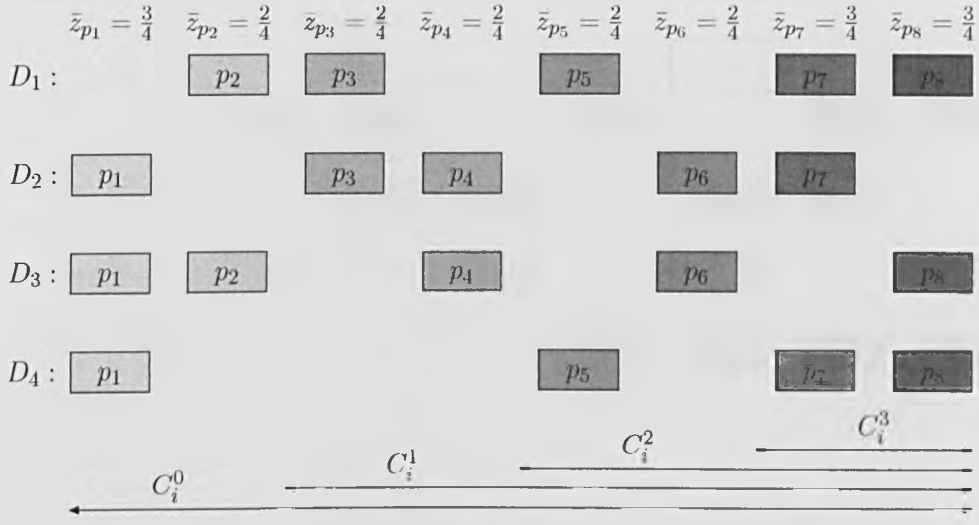

Figure 5.3: An example of distribution μ_1 for the cache of size $k = 4$ and the set of pages $S_i = \{p_1, p_2, \ldots, p_8\}$. The values $\bar{z}_{p_i}$ are given at the top of the figure. μ_1 is a uniform distribution over 4 sets: $D_1 = \{p_2, p_3, p_5, p_7, p_8\}$, ..., $D_4 = \{p_1, p_5, p_7, p_8\}$. Constraints A and B are satisfied.

between the sets D in such a way, that while repairing the constraints for the cost class C_i^s we keep the following invariant: all but one of the sets D from the support of μ have the same number of pages from the set C_i^s, as they had before we increased the value of $\bar{z}_p$. The remaining set, which we denote by D^+, has one additional page. At the beginning $D^+ = D^*$.

Notice that $\sum_{p \in C_i^s} \bar{z}_p = \sum_D \sum_{p \in C_i^s} D(p) \cdot \mu_1(D)$ is equal to the average number of pages from C_i^s that a set in the support of μ_1 has. If the number of pages from C_i^s in the sets D in the support of μ_1 differs by at most one, each set has either $\lfloor \sum_{p \in C_i^s} \bar{z}_p \rfloor$ or $\lceil \sum_{p \in C_i^s} \bar{z}_p \rceil$ pages from C_i^s, and all the constraints for C_i^s are satisfied.

Fix s and assume that the constraints hold for all $s' > s$, but some are violated for s. Let $a := \lceil \sum_{p \in C_i^s} \bar{z}_p - \frac{1}{k} \rceil$, i.e., before increasing the value of $\bar{z}_p$ each set D contained at most a, and at least $a - 1$, pages from C_i^s. As the only set that has now a different number of pages from C_i^s is D^+, and some constraints for C_i^s are violated, it must be the case that D^+ has now $a + 1$ pages from C_i^s, and some set D' with positive support in μ_1 has $a - 1$ pages from C_i^s. The constraints for C_i^{s+1} are satisfied, so the number of pages in class C_i^{s+1} differs by at most 1 between D^+ and D'. Hence, there must exist a page in $C_i^s \setminus C_i^{s+1}$ that is in D^+ but not in D'. We move this page to D', which incurs an expected cost of at most $2^{s+1}/k$. Now all the sets in the support of μ_1 have either $a - 1$ or a pages from C_i^s, and so all the constraints for C_i^s are satisfied. As we did not modify any pages from C_i^{s+1}, the

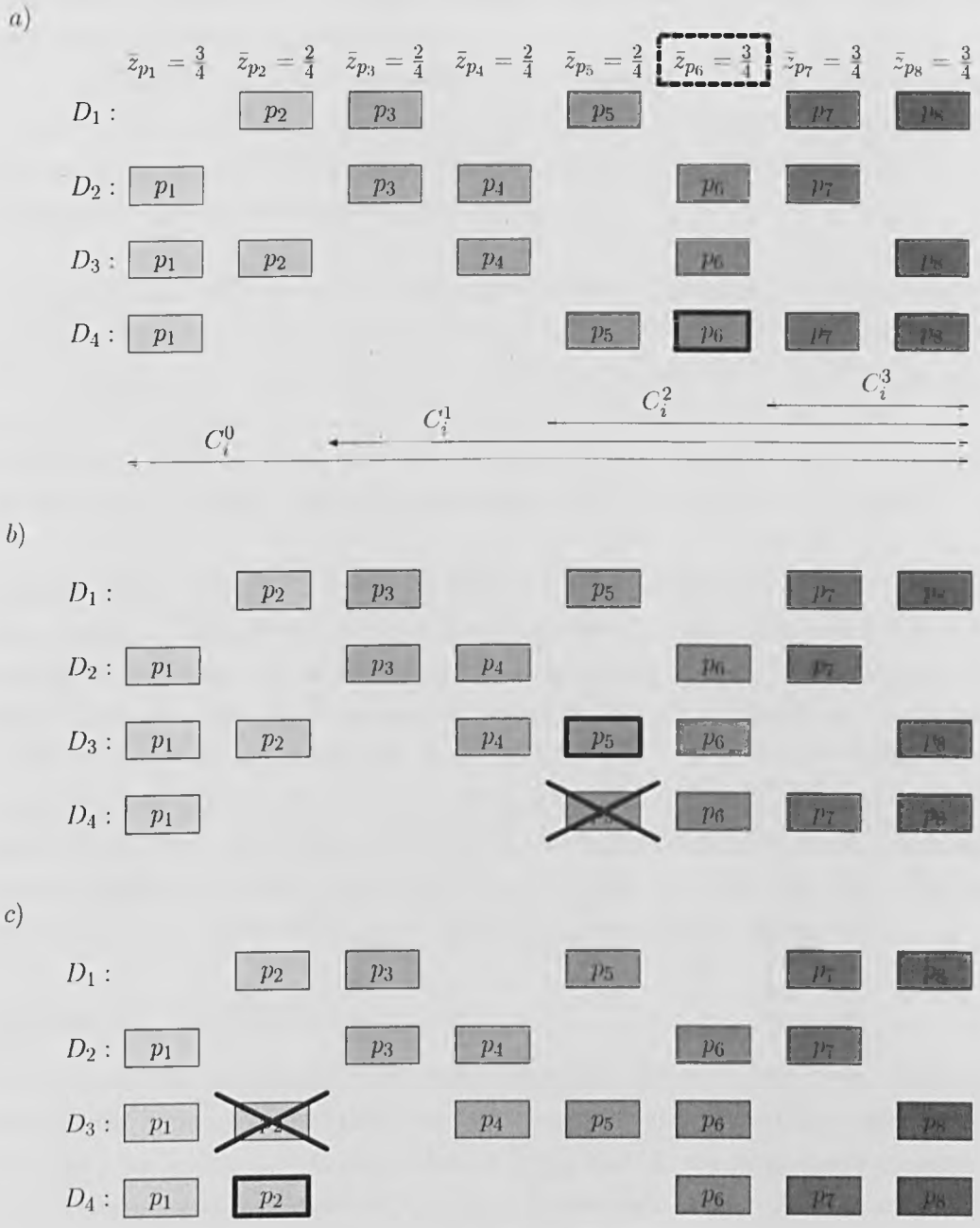

Figure 5.4: In the setting as in Figure 5.3 the value of $\bar{z}_{p_6}$ increases by $1/k = 1/4$. *a*) To satisfy Constraint B, we add page p_6 to the set D_4. After this modification Constraint A is not satisfied for the cost class $C_i^2 = \{p_5, p_6, p_7, p_8\}$ and the set D_4. *b*) To satisfy Constraint A for the cost class C_i^2 we move the page p_5 from D_4 to D_3. Now Constraint A is satisfied for the cost classes C_i^3, C_i^2 and C_i^1, but it is violated for C_i^0 and the sets D_3, D_4. *c*) To satisfy Constraint A for the cost class C_i^0 we move the page p_2 from D_3 to D_4. Now all the constraints are satisfied.

constraints for values $s' > s$ remain satisfied. Now the set D' has one additional page, and it becomes the new set D^+.

Performing the above procedure incurs expected cost of $2^{s+1}/k$ for s from r to 0. In total we have expected cost $O(2^r/k)$. The increase of $\bar{z}_p$ increases the LP-cost by at least $2^r/k$. Therefore, the cost in maintaining the distribution μ_1 can be amortized against the increase in LP-cost.

Decreasing $\bar{z}_p$. When for some page p the value of $\bar{z}_p$ decreases, we have to delete the page p from a set D in the support of μ_1 that currently contains p. The analysis for this case is completely analogous to the case of an increase in $\bar{z}_p$. The resulting cost of $O(2^r/k)$, where $c_p \in [2^r, 2^{r+1})$, can be amortized against the cost of LP — at a loss of a constant factor we can amortize $O(2^r/k)$ against the cost of LP when the value of $\bar{z}_p$ increases, and the same amount when the value of $\bar{z}_p$ decreases.

Change of the set S. The class constraints depend on the set S that is dynamically changing. Therefore we have to check whether the constraints are fulfilled if a page enters or leaves the set S. When a page p with $c_p \in [2^r, 2^{r+1})$ increases its $\bar{z}_p$ value to 1 we first add it to the only set in the support of μ_1 that does not contain it. This induces an expected cost of at most $2^{r+1}/k$. Then we fix Constraint A, as described in the procedure for increasing a $\bar{z}_p$ value. This also induces expected cost $O(2^r/k)$. After that we remove the page from the set S. Constraint A will still be valid because for every cost class C_i^s that contains p and for every set D in the support of μ_1 the values of $\sum_{p \in C_i^s} \bar{z}_p$ and $\sum_{p \in C_i^s} D(p)$ change by exactly 1.

Maintaining Distribution μ_2

We will show how to maintain a distribution μ_2 over subsets of pages, such that each set D in the support of μ_2 fulfills the large pages constraint and does not contain any page p for which $\bar{z}_p = 0$. Note that as $\sum_{p \in L} \bar{z}_p < 2$, the large pages constraint can be reformulated as follows: if $\sum_{p \in L} \bar{z}_p \geq 1$ then each subset D in the support of μ_2 contains at least one page from the set of large pages L.

In the following we introduce an alternative way of thinking about this problem. A variable $\bar{z}_p$ can be in one of $k+1$ different states $\{0, 1/k, 2/k, \ldots, 1-1/k, 1\}$. We view the $k-1$ non-integral states as *points*. We say that the *ℓ-th point for page p appears* (or *becomes active*) if the value of $\bar{z}_p$ changes from $(\ell-1)/k$ to ℓ/k. Points can disappear for two reasons. Suppose the ℓ-th point for page p is active. We say that it *disappears* (or *becomes inactive*) if either the value of $\bar{z}_p$ decreases from ℓk to $(\ell-1)/k$, or when the value of $\bar{z}_p$ reaches 1.

Note that if for a page p we have $\bar{z}_p = 1$, the page p is not in the set S, and it only enters S once the value of $\bar{z}_p$ is decreased to 0 again. The appearance of a point for page p corresponds to a cost of c_p/k of the LP-solution. At a loss of a factor of 2 we can also amortize c_p/k against the cost of the LP-solution when a point for page p disappears.

Observation 5.5. *The set of pages with active points is the set of pages in S with the value $\bar{z}_p \neq 0$.*

The above observation says that if we guarantee that a set D in the support of μ_2 can contain only those pages from S which have an active point, we guarantee one of our constraints — the set D does not contain any page p for which $\bar{z}_p = 0$.

We assign priorities to the active points, according to the size of the corresponding page, where points corresponding to larger pages have higher priorities. Ties are broken first according to page-ids, and then to the point-numbers. Let at any time step the set Q denote the set of the k active points with highest priority, or all active points if there are less than k (see Figure 5.5). The following observation follows directly from the definition of Q and L, as we used the same tie-breaking mechanisms for both constructions.

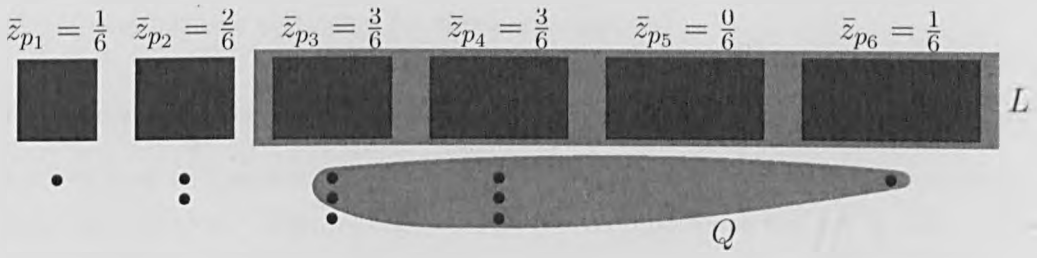

Figure 5.5: Each page $p \in S$ with $\bar{z}_p = i/k$ has i corresponding points (here $k = 6$). The set of k points with highest priority (Q) and the set of large pages (L) are marked in grey.

Observation 5.6. *For any time step, the set of pages p in L that have a value of $\bar{z}_p \neq 0$ is exactly the set of pages that have at least one point in Q.*

We assign to active points labels from the set $\{1, \ldots, k\}$, with the meaning that if a point q has label ℓ, then the ℓ-th set in the support of μ_2 contains the page corresponding to q. At each point in time the ℓ-th set consists of pages for which one of the corresponding points has label ℓ. In general we will allow a point to have several labels. Note that this definition of the sets in the support of μ_2 directly ensures that a page that has $\bar{z}_p = 0$ is not contained in any set in the support of μ_2, because a page with this property does not have any active points.

Adding a label to a point q increases the expected cost of the online algorithm by at most $c_{p(q)}/k$, where $p(q)$ is the page corresponding to the point q. Deleting a label is for free, and in particular if a point disappears (meaning its labels also disappear), the online algorithm has no direct cost while we can still amortize c_p/k against the LP-cost.

The following observation forms the basis for our method of maintaining the distribution μ_2.

Observation 5.7. *If the points in Q have different labels, then all sets in the support of the distribution μ_2 fulfill the large pages constraint.*

This means that we only have to show that there is a labeling scheme that on the one hand has a small re-labeling cost, i.e., the cost for re-labeling can be related to the cost of the LP-solution, and that on the other hand guarantees that at any point in time no two points from Q have the same label. We first show that a very simple scheme exists if the cost function is monotone in the page size, i.e., $w_p \leq w_{p'}$ implies $c_p \leq c_{p'}$ for any two pages p, p'. Note that the bit model and the fault model that have been analyzed by Bansal et al. [20] have monotone cost functions. Therefore, the following section gives an alternative proof for an $O(\log k)$-competitive algorithm for these cost models.

Maintaining μ_2 for Monotone Cost

We show how to maintain a labeling of the set Q such that all labels assigned to points are different. Assume that currently every point in the set Q has a single unique label.

Appearance of a point q. Suppose that a new point q arrives. If q does not belong to the k points with the highest priority, it will not be added to Q and we do not have to do anything.

If the set Q currently contains strictly less than k points, then the new point will be contained in the new set Q, but at least one of the k labels has not been used before and we can label q with it. In the new set Q all points have different labels. The online algorithm paid a cost of $c_{p(q)}/k$, where $p(q)$ denotes the page corresponding to the point q.

If Q already contains k pages, then upon appearance of q, a point q' with lower priority is removed from Q and q is added. We can assign the label of q' to the new point q, and then all points in the new set Q have different labels. Again the online algorithm pays a cost of $c_{p(q)}/k$.

In all cases the online algorithm pays at most $c_{p(q)}/k$ whereas the LP-cost is $c_{p(q)}/k$.

Disappearance of a point q. Now, suppose that a point q in the current set Q is deleted. This means that a point q' with a lower priority than q may be added to the set Q (if there are at least k points in total). We give q' the same label that q had. This incurs a cost of $c_{p(q')}/k \leq c_{p(q)}/k$, where the inequality holds due to the monotonicity of the cost function. Since we can amortize $c_{p(q)}/k$ against the cost of the LP-solution we are competitive.

Maintaining μ_2 for General Cost

We want to assign labels to points in Q in such a way that we are guaranteed to see k different labels if the set Q contains at least k points. In the last section we did this by always assigning different labels to points in Q. For the case of general cost functions we proceed differently.

Let Q_i denote the set of k active points with the highest priority that correspond to pages with cost at least 2^i. In case there are less than k such points, Q_i contains all active points corresponding to pages with cost at least 2^i (see Figure 5.6).

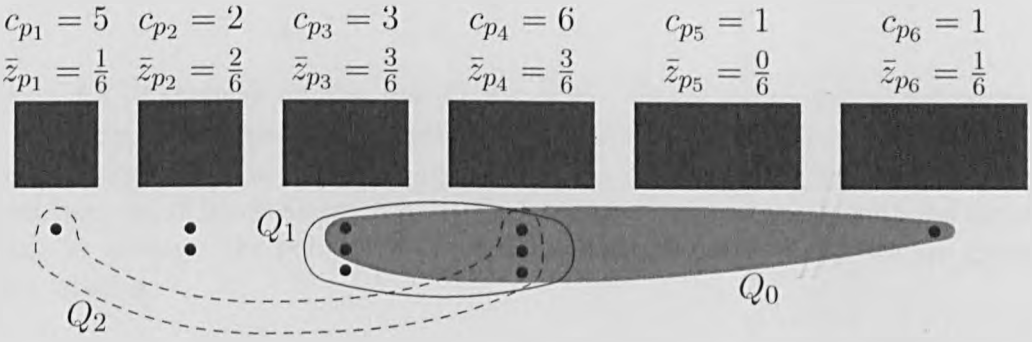

Figure 5.6: Each page $p \in S$ with $\bar{z}_p = i/k$ has i corresponding points (here $k = 6$). The sets Q_i of points with the highest priority that correspond to the pages with cost at least 2^i have been marked. The sets Q_0 and Q_1 have k points each, and the set Q_2 has less than k points.

Essentially our goal is to have a labeling scheme with small re-labeling cost that guarantees that each set Q_i sees at least $|Q_i|$ different labels. Since $Q = Q_0$, this gives the desired result. However, for the case of general cost, it will not be sufficient any more to assign unique labels to points, but we will sometimes be assigning several different labels to the same point. At first glance, this may make a re-labeling step very expensive in case a point with a lot of labels disappears.

To avoid this problem we say that a set Q_i has to commit to a unique label for every point q contained in it, where the chosen label is from the set of all labels assigned to q (see Figure 5.7). The constraint for Q_i is that it commits to different labels for all points contained in it. If a point currently has labels ℓ and ℓ', then a set Q_i may either commit to ℓ or ℓ', but furthermore during an update operation it may switch the label it commits to for free, i.e., no cost is charged to the online algorithm. Recall that if a point corresponding to a page p has several labels then all sets D corresponding to these labels contain the page p; therefore committing to a different label is for free as no change has to be made for any set D from the support of μ_2. The label to which a set Q_i commits for a point $q \in Q_i$ is denoted by $Q_i(q)$.

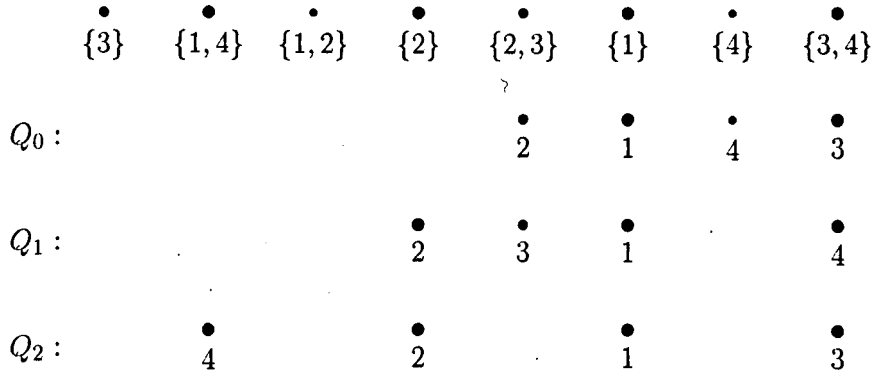

Figure 5.7: Labelings for the sets Q_i ($k = 4$). The 8 active points are ordered increasingly with respect to the priority. The size of the dots corresponds do the cost of 1, 2 or 4 of the respective pages — larger dots represent larger costs. Each point has a set of labels assigned to it. Each set Q_i contains 4 points with the highest priorities amongst the points with cost at least 2^i. For each set Q_i we are given a valid labeling.

Appearance of a point q. Suppose that a point q_0 corresponding to a page p with $c_p \in [2^r, 2^{r+1})$ appears. We assign an arbitrary label ℓ_0 to this point and, as ℓ_0 is the only label of q_0, we set $Q_s(q_0) = \ell_0$ for all subsets Q_s that contain q_0. The sets Q_s where $s > r$ are not affected by the appearance of q_0, and their labelings remain valid. We only have to fix the labelings for the sets Q_s where $s \leq r$.

Assume that labelings for all sets $Q_{s'}$ where $s' > s$ are valid, but the labeling for Q_s is violated. We want to fix it, paying only $O(2^s/k)$. We call a label ℓ a *duplicate label* for Q_s if there exist two points in Q_s for which Q_s commits to ℓ. We call the corresponding points *duplicate points*. We call a label ℓ *free* for Q_s if currently there is no point in Q_s for which Q_s commits to ℓ. When we start processing Q_s

there exists exactly one duplicate label, namely the label ℓ_0 that we assigned to q_0, and for which we have $Q_s(q_0) = \ell_0$.

Since the total number of labels is k and there are at most k points in Q_s, there must exist a free label ℓ_{free}. We could fix the condition for Q_s by assigning the label ℓ_{free} to one of the duplicate points q, and setting $Q_s(q) = \ell_{\text{free}}$. However, this would create a cost that depends on the cost of the page corresponding to the chosen point q. This may be too large, as our aim is to only pay $O(2^s/k)$ for fixing the condition for set Q_s. Therefore, we will successively switch the labels that Q_s commits to for the duplicate points, until the cost of one of the duplicate points q is in $[2^s, 2^{s+1})$. During this process we will maintain the invariant that there are at most two duplicate points for Q_s. Hence, in the end we can assign the free label ℓ_{free} for a duplicate point q with cost at most 2^{s+1}, set $Q_s(q) = \ell_{\text{free}}$, and obtain a valid labeling for Q_s.

The process of switching the labels for the set Q_s is as follows. Suppose that currently ℓ denotes the duplicate label and that the two duplicate points both correspond to pages with cost at least 2^{s+1}. This means that both points are in the set Q_{s+1}. As the labeling for Q_{s+1} is valid, we know that Q_{s+1} commits to different labels for these points. One of these labels must differ from ℓ. Let q' denote the duplicate point for which Q_{s+1} commits to a label $\ell' \neq \ell$. We set $Q_s(q') = \ell'$. Now, ℓ' may be the new duplicate label for the set Q_s.

The above process can be iterated. With each iteration the number of points in the intersection of Q_s and Q_{s+1} for which both sets commit to the same label increases by one. Hence, after at most k iterations we either end up with a set Q_s that has no duplicate points, or one of the duplicate points corresponds to a page with cost smaller than 2^{s+1}.

An example of fixing the labeling after adding a new point can be seen in Figure 5.8. As we only pay cost $2^{s+1}/k$ for fixing the labeling of Q_s, the total payment summed over all sets Q_s with $s \leq r$ is $O(2^r/k)$, which can be amortized to the cost of LP.

Disappearance of a point q. Now, suppose that a point q corresponding to a page p with $c_p \in [2^r, 2^{r+1})$ is deleted. Then a new point may enter the sets Q_s for which $s \leq r$. The only case for which this does not happen is when Q_s already contains all active points corresponding to pages with cost at least 2^s. For each Q_s we commit to an arbitrary label for this point (recall this doesn't induce any cost, as any point, when it becomes active, gets a label). Now, for each Q_s we have the same situation as in the case when a new point appears. The set either fulfills its

Figure 5.8: In the setting as in Figure 5.7 a new point (marked in grey) arrives. The point is assigned label 4. The changes in the sets Q_i, and the label changes made while fixing the labelings are marked in grey.

condition or has exactly two duplicate points. As before we can fix the condition for set Q_s at cost $O(2^s/k)$, and the total cost of $O(2^r/k)$ can be amortized to the cost of LP.

5.4 Open Problems

For the caching problem with uniform sizes and costs there are randomized algorithms with competitive ratio matching the lower bound H_k. For the generalized caching problem the algorithm presented here is only asymptotically tight, i.e. it has competitive ratio $O(\log k)$, while the lower bound on the competitive ratio is H_k, as in the uniform case.

Is there a better lower bound on the competitive ratio for the generalized caching problem? Can we design an algorithm matching the lower bound, as in the uniform case?

Bibliography

[1] Amjad Aboud. Correlation Clustering with Penalties and Approximating the Reordering Buffer Management Problem. Master's thesis, Computer Science Department, The Technion — Israel Institute of Technology, 2008.

[2] Dimitris Achlioptas, Marek Chrobak, John Noga. Competitive Analysis of Randomized Paging Algorithms. *Theoretical Computer Science*, 234(1-2):203–218, 2000.

[3] Anna Adamaszek, Artur Czumaj, Matthias Englert, Harald Räcke. Almost Tight Bounds for Reordering Buffer Management. *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC)*, pp. 607-616, 2011.

[4] Anna Adamaszek, Artur Czumaj, Matthias Englert, Harald Räcke. An $O(\log k)$-competitive Algorithm for Generalized Caching. *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 1681-1689, 2012.

[5] Anna Adamaszek, Artur Czumaj, Andrzej Lingas. PTAS for k-Tour Cover Problem on the Plane for Moderately Large Values of k. *International Journal of Foundations of Computer Science (IJFCS)*, 21(6):893-904, 2010.

[6] Anna Adamaszek, Artur Czumaj, Andrzej Lingas, Jakub Onufry Wojtaszczyk. Approximation Schemes for Capacitated Geometric Network Design. *Proceedings of the 38th International Colloquium on Automata, Languages and Programming (ICALP)*, pp. 25-36, 2011.

[7] Susanne Albers. Online Algorithms: a Survey. *Mathematical Programming*, 97(1-2):3-26, 2003.

[8] Susanne Albers, Sanjeev Arora, Sanjeev Khanna. Page Replacement for Gen-

eral Caching Problems. *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 31-40, 1999.

[9] Ali Allahverdi, C.T. Ng, T.C. Edwin Cheng, Mikhail Y. Kovalyov. A Survey of Scheduling Problems with Setup Times or Costs. European Journal of Operational Research, 187(3):985-1032, 2008.

[10] Sanjeev Arora. Polynomial Time Approximation Schemes for Euclidean Traveling Salesman and Other Geometric Problems. *Journal of the ACM*, 45(5):753-782, 1998.

[11] Sanjeev Arora and George Karakostas. Approximation Schemes for Minimum Latency Problems. *Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC)*, pp.688-693, 1999.

[12] Sanjeev Arora, Prabhakar Raghavan, Satish Rao. Approximation Schemes for Euclidean k-Medians and Related Problems. *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC)*, pp.106-113, 1998.

[13] Yuichi Asahiro, Kenichi Kawahara, Eiji Miyano. NP-Hardness of the Sorting Buffer Problem on the Uniform Metric. Unpublished manuscript, 2010.

[14] Tetsuo Asano, Naoki Katoh, Hisao Tamaki, Takeshi Tokuyama. Covering Points in the Plane by k-tours: a Polynomial Time Approximation Scheme for Fixed k. IBM Tokyo Research Laboratory Research Report RT0162, 1996.

[15] Tetsuo Asano, Naoki Katoh, Hisao Tamaki, Takeshi Tokuyama. Covering Points in the Plane by k-tours: Towards a Polynomial Time Approximation Scheme for General k. *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 275-283, 1997.

[16] Noa Avigdor-Elgrabli and Yuval Rabani. An Improved Competitive Algorithm for Reordering Buffer Management. *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 13-21, 2010.

[17] Noa Avigdor-Elgrabli and Yuval Rabani. A Constant Factor Approximation Algorithm for Reordering Buffer Management. arXiv:1202.4504.

[18] Brenda S. Baker. Approximation Algorithms for NP-complete Problems on Planar Graphs. *Journal of the ACM*, 41(1):153-180, 1994.

[19] Nikhil Bansal, Niv Buchbinder, Joseph Naor. A Primal-Dual Randomized Algorithm for Weighted Paging. *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 507-517, 2007.

[20] Nikhil Bansal, Niv Buchbinder, Joseph Naor. Randomized Competitive Algorithms for Generalized Caching. *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 235-244, 2008.

[21] Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph Naor, Baruch Schieber. A Unified Approach to Approximating Resource Allocation and Scheduling. *Journal of the ACM*, 48(5):1069-1090, 2001.

[22] Reuven Bar-Yehuda and Jonathan Laserson. Exploiting Locality: Approximating Sorting Buffers. *Proceedings of the 3rd Workshop on Approximation and Online Algorithms (WAOA)*, pp. 69-81, 2005.

[23] Dan Blandford and Guy Blelloch. Index Compression through Document Reordering. *Proceedings of the Data Compression Conference (DCC)*, pp. 342-351, 2002.

[24] Agustín Bompadre, Moshe Dror, James B. Orlin. Probabilistic Analysis of Unit-Demand Vehicle Routeing Problems. *Journal of Applied Probability*, 44(1):259-278, 2007.

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

[26] Glencora Borradaile, Philip N. Klein, Claire Mathieu. A Polynomial-Time Approximation Scheme for Euclidean Steiner Forest. *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 115-124, 2008.

[27] Niv Buchbinder and Joseph Naor. The Design of Competitive Online Algorithms via a Primal-Dual Approach. *Foundations and Trends in Theoretical Computer Science*, 3(2-3):93-263, 2009.

[28] Niv Buchbinder and Joseph Naor. Online Primal-Dual Algorithms for Covering and Packing Problems. *Proceedings of the 13th European Symposium on Algorithms (ESA)*, pp. 689-701, 2005.

[29] Jarosław Byrka and Karen Aardal. An Optimal Bifactor Approximation Algorithm for the Metric Uncapacitated Facility Location Problem. *SIAM Journal on Computing*, 39(6):2212-2231, 2010.

[30] Pei Cao and Sandy Irani. Cost-Aware WWW Proxy Caching Algorithms. *USENIX Symposium on Internet Technologies and Systems*, pp. 193-206, 1997.

[31] Stefan Cardon, Sander Dommers, Ceyhun Eksin, René Sitters, André Stougie and Leen Stougie. A PTAS for the Multiple Depot Vehicle Routing Problem. SPOR Report No. 2008-03, Eindhoven University of Technology, 2008.

[32] Ho-Leung Chan, Nicole Megow, René Sitters, Rob van Stee. A Note on Sorting Buffers Offline. *Theoretical Computer Science*, 2012, DOI http://dx.doi.org/10.1016/j.tcs.2011.12.077.

[33] Moses Charikar and Sudipto Guha. Improved Combinatorial Algorithms for Facility Location Problems. *SIAM Journal on Computing*, 34(4):803-824, 2005.

[34] Nicos Christofides. Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem. Technical Report, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, 1976.

[35] Marek Chrobak, Howard J. Karloff, T. H. Payne, Sundar Vishwanathan. New Results on Server Problems. *SIAM Journal on Discrete Mathematics*, 4(2):172-181, 1991.

[36] Fabián A. Chudak, David B. Shmoys. Improved Approximation Algorithms for the Uncapacitated Facility Location Problem. *SIAM Journal on Computing*, 33(1):1-25, 2003.

[37] Edith Cohen and Haim Kaplan. LP-based Analysis of Greedy-dual-size. *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 879-880, 1999.

[38] Artur Czumaj, Jurek Czyzowicz, Leszek Gąsieniec, Jesper Jansson, Andrzej Lingas and Pawel Zylinski. Approximation Algorithms for Buy-at-Bulk Geometric Network Design. *Proceedings of the 11th Workshop on Algorithms and Data Structures (WADS)*, pp. 168-180, 2009.

[39] Artur Czumaj and Andrzej Lingas. On Approximability of the Minimum-Cost k-Connected Spanning Subgraph Problem. *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 281-290, 1999.

[40] G. B. Dantzig and R. H. Ramser. The Truck Dispatching Problem. *Management Science*, 6(1):80-91, 1959.

[41] Aparna Das. Approximation Schemes for Euclidean Vehicle Routing Problems. Ph.D. thesis, Brown University, 2011.

[42] Aparna Das and Claire Mathieu. A Quasi-Polynomial Time Approximation Scheme for Euclidean Capacitated Vehicle Routing. *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 390-403, 2010.

[43] Matthias Englert, Harald Räcke, Matthias Westermann. Reordering Buffers for General Metric Spaces. *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 556-564, 2007.

[44] Matthias Englert and Matthias Westermann. Reordering Buffer Management for Non-Uniform Cost Models. *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, pp. 627-638, 2005.

[45] L. Few. The Shortest Path and the Shortest Road Through n Points. *Mathematika*, 2:141-144, 1955.

[46] Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel D. Sleator, Neal E. Young. Competitive Paging Algorithms. *Journal of Algorithms*, 12(4):685-699, 1991.

[47] Iftah Gamzu and Danny Segev. Improved Online Algorithms for the Sorting Buffer Problem. *Proceedings of the 24th Symposium on Theoretical Aspects of Computer Science (STACS)*, pp. 658-669, 2007.

[48] M. R. Garey and D. S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-completeness.* W.H. Freeman and Company, New York, 1979.

[49] Bruce L. Golden, Subramanian Raghavan, Edward A. Wasil. *The Vehicle Routing Problem: Latest Advances and New Challenges.* Volume 43 of Operations Research/Computer Science Interfaces Series, Springer, 2008.

[50] Sudipto Guha and Samir Khuller. Greedy Strikes Back: Improved Facility Location Algorithms. *Journal of Algorithms*, 31(1):228-248, 1999.

[51] Kai Gutenschwager, Sven Spiekermann, Stefan Voß. A Sequential Ordering Problem in Automotive Paint Shops. *International Journal of Production Research*, 42(9):1865-1878, 2004.

[52] M. Haimovich and A.H.G. Rinnooy Kan. Bounds and Heuristics for Capacitated Routing Problems. *Mathematics of Operation Research*, 10(4):527-542, 1985.

[53] Tobias Harks, Felix G. König, Jannik Matuschke. Approximation Algorithms for Capacitated Location Routing. T. U. Berlin, Preprint 010-2010.

[54] R. Hassin, R. Ravi, and F.S. Salman. Approximation Algorithms for a Capacitated Network Design Problem. *Algorithmica*, 38: 417-431, 2004.

[55] Dorit S. Hochbaum (ed.). *Approximation Algorithms for NP-hard Problems.* PWS Publishing, Boston, 1997.

[56] Dorit S. Hochbaum. Heuristics for the Fixed Cost Median Problem. *Mathematical programming*, 22:148-162, 1982.

[57] Dorit S. Hochbaum and Wolfgang Maass. Approximation Schemes for Covering and Packing Problems in Image Processing and VLSI. *Journal of the ACM*, 32(1):130-136, 1985.

[58] Sandy Irani. Page Replacement with Multi-Size Pages and Applications to Web Caching. *Algorithmica*, 33(3):384-409, 2002.

[59] Kamal Jain, Mohammad Mahdian, Evangelos Markakis, Amin Saberi, Vijay V. Vazirani. Greedy Facility Location Algorithms Analyzed Using Dual Fitting with Factor-Revealing LP. *Journal of the ACM*, 50(6):795-824, 2003.

[60] Richard M. Karp. Probabilistic Analysis of Partitioning Algorithms for the Traveling-Salesman Problem in the Plane. *Mathematics of Operations Research*, 2:209-224, 1977.

[61] Rohit Khandekar and Vinayaka Pandit. Online and Offline Algorithms for the Sorting Buffers Problem on the Line Metric. *Journal of Discrete Algorithms*, 8(1):24-35, 2010.

[62] Jens S. Kohrt and Kirk Pruhs. A Constant Factor Approximation Algorithm for Sorting Buffers. *Proceedings of the 6th Latin American Symposium on Theoretical Informatics (LATIN)*, pp. 193-202, 2004.

[63] Jens Krokowski, Harald Räcke, Christian Sohler, Matthias Westermann. Reducing State Changes with a Pipeline Buffer. *Proceedings of the 9th International Fall Workshop Vision, Modeling, and Visualization (VMV)*, pp. 217-224, 2004.

[64] G. Laporte. The Vehicle Routing Problem: An Overview of Exact and Approximate Algorithms. *European Journal of Operational Research*, 59(3):345-358, 1992.

[65] Chung-Lun Li, David Simchi-Levi. Worst-Case Analysis of Heuristics for Multidepot Capacitated Vehicle Routing Problems. *ORSA Journal on Computing*, 2(1):64-73, 1990.

[66] Shi Li. A 1.488 Approximation Algorithm for the Uncapacitated Facility Location Problem. *Proceedings of the 38th International Colloquium on Automata, Languages and Programming (ICALP)*, part II, pp. 77-88, 2011.

[67] Mohammad Mahdian, Yinyu Ye, Jiawei Zhang. Approximation Algorithms for Metric Facility Location Problems. *SIAM Journal on Computing*, 36(2):411-432, 2006.

[68] Lyle A. McGeoch and Daniel D. Sleator. A Strongly Competitive Randomized Paging Algorithm. *Algorithmica*, 6(6): 816–825, 1991.

[69] J. S. B. Mitchell. Guillotine Subdivisions Approximate Polygonal Subdivisions: A Simple Polynomial-Time Approximation Scheme for Geometric TSP, k-MST, and Related Problems. *SIAM Journal on Computing*, 28(4):1298-1309, 1999.

[70] E. Morsy and H. Nagamochi. Approximation to the Minimum Cost Edge Installation Problem. *Proceedings of the 18th International Symposium on Algorithms and Computation (ISAAC)*, pp. 292-303, 2007.

[71] Gábor Nagy and Saïd Salhi. Location-Routing: Issues, Models and Methods. *European Journal of Operational Research*, 177(2):649-672, 2007.

[72] C. St.J. A. Nash-Williams. Edge-Disjoint Spanning Trees of Finite Graphs. *Journal of the London Mathematical Society*, s1-36(1):445-450, 1961.

[73] Franco P. Preparata and Michael Ian Shamos. *Computational Geometry – an Introduction.* Springer Verlag, New York, NY, 1985.

[74] Harald Räcke, Christian Sohler, Matthias Westermann. Online Scheduling for Sorting Buffers. *Proceedings of the 10th European Symposium on Algorithms (ESA)*, pp. 820-832, 2002.

[75] Wansoo T. Rhee. Probabilistic Analysis of a Capacitated Vehicle Routing Problem-II. *The Annals of Applied Probability*, 4(3):741-764, 1994.

[76] Jan Remy and Angelika Steger. A Quasi-Polynomial Time Approximation Scheme for Minimum Weight Triangulation. *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 316-325, 2006.

[77] Jan Remy and Angelika Steger. Approximation Schemes for Node-Weighted Geometric Steiner Tree Problems. *Algorithmica*, 55(1):240-267, 2009.

[78] F. S. Salman, J. Cheriyan, R. Ravi, and S. Subramanian. Approximating the Single-Sink Link-Installation Problem in Network Design. *SIAM Journal on Optimization*, 11(3):595-610, 2000.

[79] David B. Shmoys, Éva Tardos, Karen Aardal. Approximation Algorithms for Facility Location Problems (Extended Abstract). *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 265-274, 1997.

[80] Daniel D. Sleator and Robert E. Tarjan. Amortized Efficiency of List Update and Paging Rules. *Communications of the ACM*, 28(2):202-208, 1985.

[81] Maxim Sviridenko. An Improved Approximation Algorithm for the Metric Uncapacitated Facility Location Problem. *Proceedings of the 9th Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pp. 240-257, 2002.

[82] Paolo Toth and Daniele Vigo. *The Vehicle Routing Problem.* SIAM, Philadelphia, 2001.

[83] W. T. Tutte. On the Problem of Decomposing a Graph into n Connected Factors. *Journal of the London Mathematical Society*, s1-36(1):221-230, 1961.

[84] Vijay V. Vazirani. *Approximation Algorithms.* Springer, 2001.

[85] Jens Vygen. Approximation Algorithms Facility Location Problems (Lecture Notes), 2005.

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

[87] Neal E. Young. On-Line File Caching. *Algorithmica*, 33(3):371-383, 2002.

[88] Martin Zachariasen. A Catalog of Hanan Grid Problems. *Networks*, 38(2):76-83, 2001.

Bibliography

Fluid mechanics measurements 2nd ed. 1996 R.J.Goldstein. pub. Taylor & Francis

Mechanics of Fluids 3rd.ed. 1992 I.H.Shames. pub.McGraw-Hill

Measurements in fluid mechanics 1st ed. 2005 A.Tavoularis. pub.Cambridge University Press.

Boundary Layer Theory 8th ed. 2000 H.Schlichting, K.Gersten. pub.Springer.

Elementary fluid dynamics 1990 D.J.Acheson. pub.Oxford University Press.

Fluid Mechanics Demystified 2009 M.C. Potter. pub. McGraw-Hill