U ` S R “L S ”

D I I

XIX C

# Models and Algorithms for Online Server Routing

Vincenzo Bonifaci

U ` S R “L S ”

D I I

XIX C

Vincenzo Bonifaci

# Models and Algorithms for Online Server Routing

Thesis Committee

Prof. Giorgio Ausiello (Advisor)
Prof. Leen Stougie (Coadvisor)
Dr. Giovanni Rinaldi

Reviewers

Prof. Sven O. Krumke
Prof. Eric Torng

A ’ :

Vincenzo Bonifaci
Dipartimento di Informatica e Sistemistica
Università degli Studi di Roma “La Sapienza”
Via Salaria 113, I-00198 Roma, Italy
- : bonifaci@dis.uniroma1.it
: http://www.dis.uniroma1.it/~bonifaci/

# Acknowledgments

I wish to thank my advisor Giorgio Ausiello for being interested in working with me since the very beginning. Knocking at his door has definitely been a good idea. Among other things, Giorgio introduced me to many people in the Algorithms community, and most importantly to Leen Stougie, who soon became my coadvisor. I am extremely grateful to Leen for his role in this joint Ph.D. project. His enthusiasm and encouragement were of great help in establishing the results described in the thesis.

I thank Jan Karel Lenstra for accepting to act as a promotor in Eindhoven and for his detailed comments on my writing. I am also indebted to Sven Krumke, Eric Torng and Giovanni Rinaldi for reading a draft of the thesis and suggesting several improvements.

During these years I had a great time both at “La Sapienza”, where everything started, and at TU Eindhoven. I thank all the people in the Department of Computer and Systems Science in Rome and in the Combinatorial Optimization group in Eindhoven for creating a pleasant and lively working environment. Special thanks go to Luigi for all his support, and to Peter, for his help with all things Dutch. I also thank my other coauthors Alberto, Camil, Irene, Luca and Ugo: working with them was highly enjoyable and profitable.

Last but not least, a sentiment of gratitude goes to my parents Pietro and Francesca and to my sister Diana, who always supported me in every decision during these years. To them I dedicate this thesis.

# Contents

# Chapter 1

# Introduction

## 1.1 Optimization problems

In 1962, the worldwide company Procter & Gamble advertised a contest in which one had to find the shortest round trip route to visit all of 33 locations shown on a United States map. The company offered several thousands dollars for the solution. The participants of the contest might have recognized it or not, but they were facing an instance of an *optimization problem*. In an optimization problem there are data, solutions, and one clear objective. Mathematically, an optimization problem can be formalized in the following way.

**Definition 1.1.** An *instance* of an optimization problem is given by a pair $(F, c)$. The set $F$ is the set of *feasible solutions* of the instance. The function $c : F \to \mathbb{R}$ is the *cost* function. The problem is to find an *optimal solution*, that is an $s^* \in F$ such that $c(s^*) \leq c(s)$ for every $s \in F$. An *optimization problem* is then simply a set of instances. When the set $F$ is finite for every instance of the problem, the optimization problem is called a *combinatorial* optimization problem.

It is worth emphasizing that although the set of feasible solutions and the cost function together completely describe an instance of an optimization problem, they are usually not given explicitly. Instead, the instance is represented implicitly by more succinct data.

**Example 1.1.** The problem posed in the Procter & Gamble contest is an instance of the *Traveling Salesman Problem* (TSP). In an instance of the Traveling Salesman Problem we are given an integer $n > 0$ and the distance between every pair of $n$ cities in the form of an $n \times n$ matrix $[d_{ij}]$, where $d_{ij} \in \mathbb{Z}_+$. The set $F$ of feasible solutions is the set of cyclic permutations on $n$ objects. The cost $c(\varphi)$ associated to $\varphi \in F$ is $\sum_{i=1}^{n} d_{i\varphi(i)}$.

Notice that every combinatorial optimization problem admits a trivial solution method: simply enumerate all feasible solutions and keep track of the one with the least cost. However, in most interesting optimization problems the set of feasible

solutions is so large with respect to its description that such an approach is doomed to be impractical because it requires excessive amounts of time.

**Example 1.2.** In the TSP, a trivial enumeration of all $n$-city tours requires time proportional to at least $(n-1)!$, which is more than exponential in the number of cities.

We are interested in computing an optimal solution of an optimization problem in a reasonable amount of time. This requirement is formalized by saying that we want an algorithm whose running time is polynomial in the length of the *binary representation* of the input instance. Such an algorithm is called a *polynomial time algorithm*.

## 1.2 Approximation algorithms

Unfortunately, the optimization problems we are dealing with in this thesis, such as the Traveling Salesman Problem, are NP-hard. This implies that no polynomial time algorithm is known that solves any of these problems, and there is evidence that no such algorithm actually exists. The question "Does the TSP admit a polynomial time algorithm?" is even more important than it may appear at first sight, as it is equivalent to the P vs. NP question, which is central to complexity theory. After many unsuccessful attempts by researchers at answering this question, it has been declared one of the mathematical problems of the millennium [40].

While the P vs. NP question remains unsettled, we may want to relax our original requirements. In particular, we could consider ourselves satisfied if we could find, for a given NP-hard optimization problem, a polynomial time algorithm that finds solutions that are not necessarily optimal, but very close to optimal.

**Definition 1.2.** An algorithm for a combinatorial optimization problem $P$ is said to be an *$r$-approximation algorithm* if

- is a polynomial time algorithm;
- for every instance $x$ of $P$,

  $$(x) \leq r \cdot \quad (x)$$

  where $(x)$ is the cost of the solution found by $A$ on input $x$ and $(x)$ is the cost of any optimal solution to $x$.

A consequence of the definition is that if is an $r$-approximation algorithm, it is also an $r'$-approximation algorithm for any $r' > r$. Thus an interesting question, for a given algorithm , is to ask the value of the smallest $r$ such that is an $r$-approximation algorithm. This value is called the *approximation ratio* of .

Notice that the approximation ratio is a *worst case* measure, since the fact that is an $r$-approximation algorithm means that it finds $r$-approximate solutions on every instance.

The approximation ratio is useful because it stands as an objective benchmark for measuring the performance of algorithms. When comparing two algorithms for the same problem, the first may find better solutions on some subset of the instances and the other may find better solutions on another subset and thus it is not clear which one is better in general. Comparing them through their approximation ratios gives a reasonable way out of this dilemma. Moreover, and perhaps more importantly, designing and analyzing algorithms from the point of view of their approximation ratio is useful to obtain insights into the combinatorial structure of the problem being attacked.

The theory of approximation algorithms has developed very quickly in recent years and now we know that while some NP-hard problems do not admit approximation algorithms with any constant approximation ratio unless P = NP, others do and some even allow approximation ratio $r$ for any $r > 1$. This variety reflects the variety of all these combinatorial optimization problems, in contrast with their common NP-hardness.

## 1.3 Online optimization

In many applications, the input data for an optimization problem is not entirely available beforehand. It may very well be that the input is revealed piece by piece and a partial solution is needed for every partial input. That is, the solution process is required to make decisions before complete information is available. In this case we call the optimization problem an *online* optimization problem, as opposed to *offline* problems in which all data are already available at the beginning of the solution process.

There are two intuitive models of online computation, the *one-by-one* model and the *real-time* model. Mathematically, they can be unified in a single framework, that of *request-answer games*, but doing so weakens part of the intuition behind these models. In this introduction, we describe the two models separately. In both the one-by-one model and the real-time model the input is modeled as a sequence $\sigma = \sigma_1\sigma_2\cdots$ of *requests* which is revealed step by step to an online algorithm. In the one-by-one model, the online algorithm has to handle each request before seeing the next one; that is, while managing $\sigma_j$ the algorithm only knows about $\sigma_1\sigma_2\cdots\sigma_j$. Serving the request will have some influence on the overall cost of the solution, depending on the details of the problem. After the algorithm serves $\sigma_j$, the cost will be irrevocably influenced and only then the algorithm will discover $\sigma_{j+1}$.

**Example 1.3.** In an instance of the *paging* problem, an algorithm has to manage a fast memory of size $k$ (a *cache*) and a larger, slower memory of size $N > k$. Every request in the input sequence specifies which page of the slow memory has to be accessed. In order to access the page, it has to be copied in some block of the cache if the cache does not contain it already. Eventually, this will require the eviction of some old page in order to create more room in the cache. When a page is copied

from the slow memory to the cache, a *page fault* occurs. The cost incurred by the algorithm on an input sequence is the number of page faults that the algorithm generates. Thus, the goal is to design an eviction policy that minimizes the number of page faults.

In the real-time model, each request has a *release date* at which it becomes available. The release date specifies the time at which the request becomes known to the algorithm; no request can be served before its release date. Release dates are nonnegative real numbers and the sequence $\sigma_1\sigma_2\cdots$ is ordered by nondecreasing release dates. Thus, the algorithm determines its behavior at any time $t$ as a function of $t$ and of the prefix of $\sigma$ consisting of all the requests with release date less than or equal to $t$. An important difference with the one-by-one model is that requests need not be served in the order they arrive: the algorithm can wait and defer the service of a particular request. However, waiting typically increases the overall cost incurred by the algorithm. Finally, and similarly to the one-by-one model, once the algorithm has taken some action, this cannot be revoked.

**Example 1.4.** In an instance of the *multiprocessor scheduling* problem, an algorithm has to process *jobs* on a set of *machines*. Every job $j$ has a release date $r_j$ and a processing time $t_j$. The algorithm can decide when and on which machine every job will start processing. Once started, a job cannot be interrupted and every machine can process only one job at a time. The time at which a job finishes processing is called the *completion time* of the job. A typical objective is to minimize the latest completion time of the jobs.

There are several methodologies for dealing with online optimization problems. In *stochastic optimization*, one assumes that the probability distribution of the input instances is known, and the goal is to develop algorithms that perform well in expectation with respect to that distribution. Thus, this type of analysis is focused on the average case behavior of the algorithms being considered. A different approach, and the one we will adopt in this thesis, is that of *competitive analysis*, which instead studies the worst case behavior of the online algorithms, by comparing their performance on every instance with that of an ideal optimal solver. The next sections introduce and discuss the concepts behind competitive analysis. Finally, *simulation* is a methodology that is often adopted in practice: the algorithms to be studied are implemented and executed on instances generated according to some appropriate model of the "real" instances of the problem. The results may lead to insights into both the worst case and the average case behavior of the algorithms.

Sometimes in the literature one also finds online optimization named as *dynamic optimization*. However, we prefer to keep different meanings for these two terms. In a dynamic optimization problem, data is also arriving step by step, but one does not need to commit in any way to the partial solution built so far. In a dynamic problem, computing an answer from scratch is always in principle a viable option, and the focus is actually on how to compute quickly a good new solution

given the last partial one. Instead, in an online optimization problem, one has to commit to the partial solution; it is not possible to undo any action, only new actions can be added. It is clear that in this case one cannot hope in general to obtain optimal solutions, even if there are no efficiency constraints on the online solver, because the lack of information about the future inputs cannot be compensated in any way.

Another frequently used expression is *real-time optimization*. A real-time problem is an online problem in which the bounds on producing every new piece of the solution are very tight. Thus, in a real-time problem, the efficiency of the online solver cannot be neglected. If we, as it is common, assume that efficiency is represented by polynomial running time, this gives one more reason why many interesting real-time problems cannot be solved optimally at all: because of their online aspect, and because often the associated offline problems are NP-hard, and thus polynomial time algorithms cannot solve them unless P = NP. Fortunately, it is still possible to resort to algorithms that have both a constant approximation ratio and a polynomial running time.

## 1.4 Competitive analysis

In online optimization, a consolidated framework for measuring the quality of online algorithms is *competitive analysis*.

**Definition 1.3.** An algorithm for an online optimization problem $P$ is said to be *c-competitive* if

- is an online algorithm;
- for any instance $\sigma$ of $P$,

$$(\sigma) \leq c \cdot \quad (\sigma)$$

where $(\sigma)$ is the cost incurred by on $\sigma$ and $(\sigma)$ is the cost of an optimal (offline) solution to $\sigma$.

Notice the analogy with the definition of an $r$-approximation algorithm (Definition 1.2). The only difference is that instead of requiring the algorithm to be polynomial time, we only allow algorithms that construct their solutions online. Again, we can define the *competitive ratio* of algorithm to be the smallest $c$ such that is a $c$-competitive algorithm.

The fact that competitive algorithms are not required to be polynomial time deserves further explanation. Indeed, as online problems often model real-time systems, it would seem appropriate that algorithms for such problems should be very efficient, and thus having a polynomial running time would be a minimum prerequisite. However, it is conceptually useful to separate the issue of dealing with limited time from that of dealing with limited information, which is really what online computation is about. Moreover, this separation is also justified by

the fact that many online algorithms, that use superpolynomial time subroutines to optimally solve hard offline subproblems, can be made polynomial time by replacing the optimal subroutines with approximation algorithms, while not affecting the competitive ratio much.

## 1.5 The competitive ratio as the value of a game

An interesting feature of competitive analysis is its relation to game theory. Indeed, the competitive ratio can be seen as the outcome of a game between an online *player* and a malicious *adversary*. We can think of the adversary as the process that generates the online sequence of requests, while the online player is the algorithm handling the requests. Consider now the following game: the online player chooses an online algorithm , while the adversary chooses a sequence $\sigma$. Then, the online player pays the adversary a monetary amount equal to $(\sigma)/$ $(\sigma)$. Then the value of the game for the adversary is precisely the competitive ratio. That is, if the competitive ratio is at least $c$, then the adversary can force the online player to pay at least $c$, and vice versa.

In the case of a deterministic online algorithm, the responses of the online player to the same inputs are fixed and one can assume that the adversary knows them. Thus, the adversary will simply try to construct a sequence that maximizes the ratio; such a sequence is sometimes called a *cruel* sequence. Cruel sequences can be used to prove *lower bounds* on the competitive ratio of an online algorithm. In the case of a randomized online algorithm, the assumption that the adversary can predict the behavior of the algorithm does not hold anymore and one has to be more careful in defining what the adversary knows about the online algorithm. We discuss this issue in the next section.

## 1.6 Online algorithms and randomization

So far, we have considered competitive analysis of deterministic algorithms. However, given the success of randomization in other fields of computing, it is reasonable to attempt to extend our definitions to *randomized algorithms*, that is, algorithms that choose some of their actions according to the value of some random variables. From the game-theoretical point of view, this can be shown to be equivalent to the assumption that the online player chooses his strategy randomly out of a set of deterministic strategies, implementing thus a so-called *mixed strategy* [30]. This motivates the following definition.

**Definition 1.4.** A *randomized online algorithm* is defined as a probability distribution over a set of deterministic online algorithms. The cost $(\sigma)$ of on input sequence $\sigma$ is thus a random variable.

In order to define the competitive ratio of a randomized algorithm, we should be careful about the precise adversary model used. Three such models are common

in the literature. The *oblivious adversary* chooses an input sequence based only on the description of the online algorithm. Thus, it cannot build the input sequence on the basis of the actual behavior of the online algorithm. The *adaptive offline adversary* can build the input sequence online and can base future requests on the actions of the online algorithm on previous requests. The *adaptive online adversary* can build the input sequence online like the adaptive offline adversary, but it must also generate its own solution online. In the case of a deterministic online algorithm, these distinctions vanish because the behavior of the algorithm is completely predictable, given the request sequence. Instead, when considering randomized online algorithms, adaptive adversaries are more powerful than the oblivious adversary; so powerful, indeed, that they almost negate all the advantages of randomization [22]. In this thesis we will only use the oblivious adversary model when dealing with randomized online algorithms.

**Definition 1.5.** A randomized online algorithm  distributed over a set $\{ \ _y\}$ of deterministic online algorithms for an online optimization problem $P$ is called *$c$-competitive against an oblivious adversary* if, for all instances $\sigma$ of $P$,

$$\mathbb{E}_y[ \ _y(\sigma)] \leq c \cdot \quad (\sigma).$$

As for the deterministic algorithms, we define the competitive ratio of  as the smallest $c$ such that  is $c$-competitive.

For randomized algorithms, it can be difficult to prove a lower bound on the cost of an arbitrary randomized algorithm on a specific instance. Fortunately, the use of *Yao's principle* [31, 87, 91] can be helpful in these cases. It will be enough for us to resort to the following form of the principle.

**Theorem 1.1** (Yao's principle)**.** *Let* $\{ \ _y : y \in \mathcal{Y}\}$ *denote the set of deterministic online algorithms for an online minimization problem. If $X$ is a distribution over input sequences* $\{\sigma_x : x \in \mathcal{X}\}$ *such that*

$$\inf_{y \in \mathcal{Y}} \mathbb{E}_X[ \ _y(\sigma_x)] \geq c \, \mathbb{E}_X[ \quad (\sigma_x)]$$

*for some real number $c \geq 1$, then $c$ is a lower bound on the competitive ratio of any randomized algorithm against an oblivious adversary.*

## 1.7 Beyond competitive analysis

One of the advantages of competitive analysis is that its adversarial nature allows one to prove unconditional lower bounds on the competitive ratio rather easily. The downside is that sometimes these lower bounds are too strong, in the sense that they do not reflect correctly the practical behavior of the algorithms being analyzed. For example, consider the paging problem. From the competitive analysis point of view, the two algorithms Least Recently Used ( ) and First In First Out ( ) have the same competitive ratio, that is $k$, the size of the cache. However, it has

long been known that is much better in practice than . This failure of competitive analysis in distinguishing the two algorithms is mainly due to the fact that the power of the offline adversary is too great. For this reason, refinements of and alternatives to competitive analysis have been proposed in the literature.

Perhaps the simplest way to weaken the all-powerful adversary is to strengthen the online algorithm by giving it more resources than the adversary. For example, in a scheduling problem the online algorithm could have more processors, while in the paging problem it could have a larger cache. This approach is called *resource augmentation* and it has been applied successfully in several cases [62, 81, 86].

Another method, introduced by Koutsoupias and Papadimitriou [65], is *comparative analysis*. In comparative analysis, one enlarges the scope of competitive analysis by distinguishing between different *information regimes*. Consider two classes $\mathcal{A}$ and $\mathcal{B}$ of algorithms. The *comparative ratio* $\mathcal{R}(\mathcal{A}, \mathcal{B})$ is defined as:

$$\mathcal{R}(\mathcal{A}, \mathcal{B}) = \max_{B \in \mathcal{B}} \min_{A \in \mathcal{A}} \max_{\sigma} \frac{A(\sigma)}{B(\sigma)}.$$

Not surprisingly, the clearest interpretation is again that of a game. The adversary chooses an algorithm $B \in \mathcal{B}$. As an answer to that, the online player picks an algorithm $A \in \mathcal{A}$. Then the adversary picks an input sequence $\sigma$, and the online player pays the adversary the ratio $A(\sigma)/B(\sigma)$. The amount that the adversary can hope to obtain is exactly $\mathcal{R}(\mathcal{A}, \mathcal{B})$. We obtain competitive analysis as a special case if $\mathcal{A}$ is the set of all online algorithms and $\mathcal{B}$ is the class of all algorithms, online and offline.

Another generalization of competitive analysis is the *diffuse adversary model*. While standard competitive analysis is completely worst case, in the diffuse adversary model one assumes that something is known about the input distribution. In particular, we assume that the actual input distribution $D$ belongs to a known class $\Delta$ of possible distributions. The competitive ratio is then defined as:

$$\mathcal{R}(\Delta) = \min_{A} \max_{D \in \Delta} \frac{\mathbb{E}_D[A(\sigma)]}{\mathbb{E}_D[\quad(\sigma)]}.$$

This means that the adversary chooses a distribution $D$ among those in $\Delta$ so that the expected value, with respect to $D$, of the competitive ratio is as large as possible. Thus, this generalization represents a bridge between classical competitive analysis (which is the particular case in which $\Delta$ is the class of all possible distributions) and stochastic optimization (which we obtain as a special case when $\Delta$ is a singleton).

## 1.8 Server routing problems

This thesis deals with the online versions of particular combinatorial optimization problems known as *server* (or *vehicle*) *routing problems*. In a server routing problem, a set of servers has to process a set of transportation requests in a metric space. In order to serve the requests, the servers incur some cost, the details depending

on the problem being considered. We have already seen a fundamental server routing problem, the Traveling Salesman Problem or TSP. In the TSP there is a single server, requests consist of isolated points that have to be visited, and the goal is to minimize the total length of a round trip route visiting all requested points. The TSP is a classical problem in combinatorial optimization and has been the testbed of many algorithmic techniques. The server routing problems we consider in this thesis can all be seen as online generalizations or variations of the standard TSP.

## 1.9 Outline of the thesis

In this section we briefly outline the contents of this thesis.

- In Chapter 2, after reviewing the basic complexity results for offline server routing problems, we introduce the online server routing framework and we survey the state of the art for online server routing. We show the basic proof techniques and we discuss several attempts in the literature to extend the basic competitive analysis setting. Part of the content of this chapter is joint work with L. Allulli, G. Ausiello, and L. Laura [5].

- In Chapter 3, we consider the *online asymmetric traveling salesman problem* from the point of view of competitive analysis. We prove that the homing version, where the server has to return to its starting point, admits a $(1+\phi)$-competitive algorithm, where $\phi$ is the golden ratio, and we show that this is best possible. We also consider the nomadic version (where returning to the starting point is not mandatory) and prove that it does not admit constant competitive algorithms. However, for the nomadic version we prove a competitive ratio as a function of the amount of asymmetry of the space. We also consider the competitiveness of zealous algorithms. We discuss the issue of polynomial time algorithms in this setting. The content of the chapter is joint work with G. Ausiello and L. Laura [12].

- In Chapter 4, we study the *online prize-collecting traveling salesman problem*. After discussing the approximation ratio of the offline version, we give a $7/3$-competitive algorithm. We also consider the special case of the halfline as the metric space, for which we prove lower and upper bounds of 1.89 and 2, respectively, on the competitive ratio of deterministic algorithms. The content of the chapter is joint work with G. Ausiello, L. Laura, A. Marchetti-Spaccamela and S. Leonardi [13, 14].

- In Chapter 5, we consider the *online nomadic traveling salesman and the online traveling repairman with k servers*. We give competitive algorithms whose competitive ratios match the ones for the single server variants. For the special case of the real line, we prove the existence of algorithms with competitive ratio $1 + O((\log k)/k)$, meaning that we can approach the optimal cost as $k$ grows. We also show that this phenomenon is limited to the

one dimensional case, since already in the Euclidean plane, we prove a lower bound of $4/3$ for the online nomadic TSP and of $5/4$ for the online TRP independently of the number of servers. Finally, we give resource augmentation results that are asymptotically best possible as the number of online servers grows beyond the number of offline servers. The content of the chapter is joint work with L. Stougie [29].

- In Chapter 6, in order to address the limits of competitive analysis, we introduce *a new model for online server routing based on adversarial queueing theory*. The model addresses the stability of online algorithms that are continuously operating. We call an online algorithm *stable* if there exists an upper bound on the number of unserved requests at any time that does not depend on the time the system has been running. We consider a number of natural algorithms in this model and we prove the existence of algorithms that are stable and such that the maximum flow time of a request also does not depend on the time the system has been running [28].

## 1.10 Related literature

Combinatorial optimization lies at the interface between operations research and computer science. A classical text by Papadimitriou and Steiglitz [79] combines the two approaches. We assume that the reader is familiar with basic notions from complexity theory; see Garey and Johnson [50] for a self-contained introduction focusing on NP-hardness. More modern references are available for the theory of approximation algorithms [15, 90]. A somewhat dated but still unique monograph on the traveling salesman and other server routing problems is the book by Lawler et al. [73].

Competitive analysis has its origins in the 1980s, when for the first time formal concepts for the analysis of online algorithms where introduced [86], although some online algorithms with guaranteed competitive ratio were already designed in the 1960s in the context of multiprocessor scheduling [54]. For an in-depth presentation of online computation, the reader can refer to the texts by Borodin and El-Yaniv [30] or by Fiat and Woeginger [47]. A useful reference for online vehicle routing is given by Krumke [66], together with a number of Ph.D. theses on the subject [43, 75, 82].

# Chapter 2

# Online server routing

## 2.1 Introduction

In this chapter, we survey the state of the art for online server routing. Since we do not want to disregard the running time of the algorithms we consider, we start with a brief review of basic complexity results for offline server routing problems (Section 2.2). Then we introduce the online server routing model, discuss the main results in the area and show by some examples the basic proof techniques (Section 2.3). We end by surveying extensions of the basic model in several directions (Section 2.4).

## 2.2 Offline server routing: complexity results

The Traveling Salesman Problem (TSP for short) is an archetypical problem in combinatorial optimization. This problem and its generalizations, *server* or *vehicle routing problems*, have been studied for more than fifty years [42], and several monographs are devoted to the subject [55, 61, 73, 83, 89]. In this section we discuss some basic complexity results for the TSP and some of its variants.

### 2.2.1 The Traveling Salesman Problem

The basic TSP was already defined in the Introduction (Example 1.1). Let us define it again here, in a slightly different but equivalent form.

**Definition 2.1.** An instance of the *Traveling Salesman Problem* is given by an integer $n > 0$ and the distance between every pair of $n$ cities in the form of a function $d : \{1, \ldots, n\}^2 \to \mathbb{Z}_+$. The set $F$ of feasible solutions is the set of cyclic permutations on $n$ objects. The cost $c(\varphi)$ associated to $\varphi \in F$ is

$$\sum_{i=1}^{n} d(\varphi^{(i-1)}(1), \varphi^{(i)}(1))$$

where $\varphi^{(i)}$ stands for the *i-th iterate* of $\varphi$, defined inductively by $\varphi^{(0)}(x) = x$ and $\varphi^{(i+1)}(x) = \varphi(\varphi^{(i)}(x))$.

The TSP is an NP-hard problem. To see why, consider the NP-hard *Hamiltonian Cycle* problem [50]. In the Hamiltonian Cycle problem, one is given an undirected graph, and the problem is that of deciding whether the graph has a Hamiltonian cycle, that is, a cycle spanning all the vertices. The following reduction from the Hamiltonian Cycle problem to the Traveling Salesman Problem shows that the latter is NP-hard.

**Theorem 2.1** ([84])**.** *The Traveling Salesman Problem is* NP*-hard.*

*Proof.* Let $G = (\{1, 2, \ldots, n\}, E)$ be an instance of the Hamiltonian Cycle problem. We define a TSP instance as follows: the number of cities is equal to $n$, the number of vertices of $G$, while the intercity distances are given by

$$d(i, j) = \begin{cases} 1 & \text{iff } \{i, j\} \in E \\ 2 & \text{otherwise.} \end{cases}$$

From the construction it follows that $G$ has a Hamiltonian Cycle if and only if the TSP instance has a solution of cost $n$. Thus, a polynomial time algorithm for the TSP implies a polynomial time algorithm for the NP-hard Hamiltonian Cycle problem. □

Indeed, the same proof technique can be used to prove a much more drastic fact: not only does the TSP have no polynomial time algorithms unless P=NP, it is also hard to approximate within any factor better than an exponential in $n$.

**Theorem 2.2** ([84])**.** *For any fixed $k \in \mathbb{Z}_+$, there is no $2^{n^k}$-approximation algorithm for the Traveling Salesman Problem unless* P=NP.

*Proof.* We can proceed as in Theorem 2.1, except that $d(i, j)$ is now $1 + n2^{n^k}$ when $\{i, j\} \notin E$. The reduction can be performed in time polynomial in $\log(1 + n2^{n^k})$, which is polynomial in $n$. □

### 2.2.2 The metric TSP

As we have seen in the previous section, we cannot hope to have good approximations to the general TSP unless P=NP. Fortunately, in many applications the input data satisfy some additional constraints that simplify the problem. Particularly of interest is the constraint that the intercity distances form a *metric*.

**Definition 2.2.** Given a set $\mathbb{M}$, a function $d : \mathbb{M}^2 \to \mathbb{R}_+$ is called a *metric* on $\mathbb{M}$ if:

1. $d(i, i) = 0$ for every $i \in \mathbb{M}$ *(definiteness)*;
2. $d(i, j) = d(j, i)$ for every $i, j \in \mathbb{M}$ *(symmetry)*;
3. $d(i, j) \leq d(i, k) + d(k, j)$ for every $i, j, k \in \mathbb{M}$ *(triangle inequality)*.

The set $\mathbb{M}$, equipped with $d$, is called a *metric space*.

The TSP with the metric constraint is called the *metric TSP*. The metric TSP is still an NP-hard problem (the instances constructed in the proof of Theorem 2.1 are metric). However, differently from the basic TSP, it admits constant approximation algorithms.

**Theorem 2.3.** *The metric TSP admits a 2-approximation algorithm.*

*Proof.* The algorithm constructs a minimum spanning tree $T$ on the input metric. The total weight of $T$ is at most the cost of the optimal solution to the metric TSP instance, since any Hamiltonian cycle contains a spanning tree.

Then, a Eulerian multigraph is obtained from $T$ by taking two copies of every edge of $T$. Finally, we take a Eulerian closed walk along this multigraph and we "shortcut" it by eliminating multiple occurrences of the same vertex. By the triangle inequality, doing so does not increase the cost of the walk. Thus, the final Hamiltonian cycle has cost at most twice that of the optimal solution. □

The best approximation algorithm known to date for the metric TSP is a heuristic by Christofides.

**Theorem 2.4** ([38])**.** *The metric TSP admits a* $3/2$*-approximation algorithm.*

It is quite striking that no progress has been made over this algorithm in thirty years, despite numerous attempts by researchers. It is thus natural to ask if $3/2$ is the best approximation ratio possible assuming P $\neq$ NP. Although an answer to this question is still missing, researchers were able to prove results that go in this direction.

**Theorem 2.5** ([80])**.** *There is no r-approximation algorithm for the metric TSP for* $r < 220/219$*, unless* P=NP.

In a variant of the TSP, called the Wandering Salesman Problem [61], the salesman is not required to return at the starting point at the end of the tour.

**Definition 2.3.** An instance of the *Wandering Salesman Problem* (WSP) is given by an integer $n > 0$ and the distance between every pair of $n$ cities in the form of a function $d : \{1, \ldots, n\}^2 \to \mathbb{Z}_+$. The set $F$ of feasible solutions is the set of cyclic permutations $\varphi$ over the set of $n$ cities. The cost $c(\varphi)$ associated to $\varphi \in F$ is

$$\sum_{i=1}^{n-1} d(\varphi^{(i-1)}(1), \varphi^{(i)}(1)).$$

Similarly to the TSP, we can define a metric variant of the WSP. It turns out that the approximation properties of the WSP are essentially the same as those of the TSP.

**Theorem 2.6** ([57])**.** *The metric WSP admits a* $3/2$*-approximation algorithm.*

Other special cases of the TSP have different approximability properties. An important case is that of $\mathbb{R}^p$ with the Euclidean metric, which was considered by Arora [9].

**Theorem 2.7** ([9])**.** *For any $r \in (1, \infty)$ and $p \in \mathbb{N}$, there is an $r$-approximation algorithm for the TSP in $\mathbb{R}^p$ with the Euclidean metric.*

### 2.2.3 The Traveling Repairman Problem

In the TSP, the objective function is quite partial to the salesman, since it asks to minimize the total length of the tour. Instead, we can consider objective functions that are more oriented to the customers in the cities. An important such function is the *total latency*, that is the objective of the so called *Traveling Repairman Problem*. The latency of a city is the distance traveled before first visiting the city.

**Definition 2.4.** An instance of the *Traveling Repairman Problem* (TRP) is given by an integer $n > 0$ and the distance between every pair of $n$ cities in the form of a function $d : \{1, \ldots, n\}^2 \to \mathbb{Z}_+$. The set $F$ of feasible solutions is the set of cyclic permutations on $n$ objects. The cost $c(\varphi)$ associated to $\varphi \in F$ is

$$\sum_{i=1}^{n} \sum_{j=1}^{i-1} d(\varphi^{(j-1)}(1), \varphi^{(j)}(1)).$$

Similarly to the TSP, the TRP is an NP-hard problem and it is very hard to approximate if one does not assume a metric distance function. Even with the metric assumption, a constant approximation algorithm was far harder to develop than for the TSP [26, 52]. The best result to date is due to Chaudhuri et al.

**Theorem 2.8** ([33])**.** *The metric TRP admits a* 3.59*-approximation algorithm.*

Finally, we mention that on the Euclidean line, the TRP is solvable in polynomial time by dynamic programming [1], while for fixed higher dimensions there is a quasipolynomial time approximation scheme [10].

## 2.3 Online server routing: competitive analysis

### 2.3.1 The online server routing framework

In a server routing problem, a server moves in a special kind of metric space (recall Definition 2.2). In particular, we generally assume that the metric space is *path-metric* and *continuous*. By path-metric we mean that the distance between any two points is equal to the length of the shortest path between them. A metric space $\mathbb{M}$ is continuous if, for all $x, y \in \mathbb{M}$ and $a \in [0, 1]$, there is a $z \in \mathbb{M}$ such that $d(x, z) = ad(x, y)$ and $d(z, y) = (1 - a)d(x, y)$. We call *general metric spaces* the metric spaces that are both path-metric and continuous. Finally, we assume that the metric space $\mathbb{M}$ has a distinguished point $o$ called the *origin*.

We will be interested in general metric spaces as well as some special cases. For example:

- the *real line*, i.e. $\mathbb{R}$ with the Euclidean distance $d(x, y) = |x-y|$ and the origin at 0;
- the nonnegative part of the real line, or simply the *halfline*, i.e. $\mathbb{R}_+$ with the Euclidean distance $d(x, y) = |x - y|$ and the origin at 0;
- the $p$-dimensional space $\mathbb{R}^p$ with the origin at 0 and the Euclidean distance

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2};$$

- the general metric space induced (in the natural way) by an undirected, connected edge-weighted graph.

In Chapter 3 we will also consider a generalization in which the distance function is not necessarily symmetric.

Being an online problem, a server routing problem has instances that consist of a sequence of requests $\sigma = \sigma_1\sigma_2\cdots$. In the basic framework, each request is a pair $\sigma_j = (r_j, x_j) \in \mathbb{R}_+ \times \mathbb{M}$. The parameter $r_j$ is the *release date* of request $\sigma_j$, that is, the time at which the request becomes known and available for processing, while $x_j$ represents the location of the request. In order to serve the request, the server will have to visit location $x_j$. In the basic model, there are no processing times, that is, once visited a request is instantaneously served. The *completion time* $C_j$ of a request $\sigma_j$ is the first time at which that request becomes served.

In accordance to the real-time model of online problems, the sequence of requests is assumed to be ordered by nondecreasing release dates. Also, it is sometimes handy to denote by $\sigma^{\leq t}$ the prefix of $\sigma$ consisting of all requests released up to time $t$. Similarly we denote by $\sigma^{=t}$ and $\sigma^{\geq t}$ the requests released at time $t$ and the requests released since time $t$, respectively.

The online algorithm controls a server, located at the origin $o \in \mathbb{M}$ at time 0. We denote the position of the server at time $t$ by $s(t)$. By scaling distances appropriately, we can assume without loss of generality that the server can move at most at unit speed. Obviously, since the algorithm has to be online, it does not have information about the number of requests nor about the release date of the last request. Thus, at any time $t$, the algorithm must determine the behavior of the server as only a function of $t$ and $\sigma^{\leq t}$. Instead, an offline algorithm knows the entire sequence $\sigma$ already at time 0.

A solution to an online server routing problem is called a (feasible) *schedule*. Informally, a schedule for instance $\sigma$ is a sequence of moves of the server such that all requests in $\sigma$ are served. Notice that (1) the server must start in the origin at time 0; (2) no request can be served before its release date. In some variants

of online server routing problems, the schedule must be *closed*, that is, the server must also return to the origin after serving all requests.

Different server routing problem associate different costs to a schedule. Thus, we will introduce the various costs as we present the problems we are interested in.

### 2.3.2 The homing online TSP

In the online Traveling Salesman Problem, cities to be visited (requests) are revealed online while the salesman is traveling. The objective is to minimize the duration of a closed schedule serving all requests, that is, the time at which the server is back at the origin and every request has been served. Since the schedule is required to be closed, the problem is also called *homing* online TSP, to stress that the server has to return "home". The homing online TSP, as its name suggests, is probably the closest online analogue of the TSP. However, there are some interesting differences between the two problems. One is that in the offline TSP lengths and times have essentially the same role, while in the online TSP the cost is given not by the length of the tour but by the length of the schedule, that is, by the total time traveled by the server. If the server stays idle, this will result in an increased cost. Another difference is that while in the offline TSP a solution never needs to visit the same point twice, an online schedule might have to do so due to the release dates constraints.

The homing online TSP was first considered by Ausiello et al. [17]. They give a 2-competitive algorithm for general metric spaces and show that to be best possible. Although the 2-competitive algorithm is not polynomial time, the authors show how to derive from it a 3-competitive polynomial time algorithm, by applying Theorem 2.4. A better polynomial time algorithm, with a competitive ratio of $(7 + \sqrt{13})/4 \approx 2.65$, is derived by Ascheuer et al. [11].

Ausiello et al. also consider the homing online TSP on the real line, for which they devise a $7/4$-competitive polynomial time algorithm and prove a lower bound of $(9 + \sqrt{17})/8 \approx 1.64$. Later, the problem on the real line was closed by Lipmann [75], who gives a matching $(9+\sqrt{17})/8$-competitive algorithm. Finally, Blom et al. [25] consider the case of the halfline, for which they devise a simple, best possible $3/2$-competitive algorithm.

In order to expose the reader to the basic proof techniques, we would like to discuss a different 2-competitive algorithm for the homing online TSP. This algorithm, called SmartStart, was proposed and analyzed by Ascheuer et al. [11] in the more general context of dial-a-ride problems (see also Section 2.3.5). The analysis is particularly interesting because it shows how waiting can help in online server routing. We will also use the same technique in Chapter 3, where we will consider the online TSP in asymmetric spaces.

**A 2-competitive algorithm**

(Algorithm 1) depends on a real parameter $\alpha > 0$. From time to time, the algorithm calls a "work-or-sleep" subroutine that computes a (possibly approximate) shortest schedule $S$ for all unserved requests released so far, starting and ending at the origin. Let $\ell(S)$ be the length of schedule $S$. If $\ell(S) \leq \alpha t$, that is, the length $\ell(S)$ of the schedule is relatively short compared to the current time $t$, then the subroutine returns $(S, \text{work})$, otherwise it returns $(S, \text{sleep})$. Notice that the subroutine has to find a solution to a metric TSP instance. We assume that it uses a $\rho$-approximation algorithm, for some $\rho \in [1, 1+\alpha]$. When we are only interested in competitive analysis, the running time of the subroutine will not be relevant so that we can assume $\rho = 1$. However, when we are interested in a polynomial time algorithm we need to consider larger values of $\rho$ (for example, $3/2$ when using Christofides' heuristic).

The algorithm has three different states of operation. In the *idle* state, the server has served all known requests, is sitting at the origin and waiting for new requests to occur. In the *sleeping* state, the server is sitting at the origin and knows some unserved requests, but has decided, according to the work-or-sleep subroutine, to postpone their service. In the *working* state, the server is following the schedule last computed.

**Algorithm 1**

If the algorithm is idle at time $t$ and new requests arrive, it calls the work-or-sleep subroutine (that uses a $\rho$-approximation algorithm). If the result is $(S, \text{work})$, the algorithm enters the working state where it follows schedule $S$. Otherwise the algorithm enters the sleeping state with wake-up time $t'$, where $t' \geq t$ is the earliest time such that $\ell(S) \leq \alpha t'$ and $\ell(S)$ denotes the length of the just computed schedule $S$, i.e., $t' = \max\{t, \ell(S)/\alpha\}$.

In the sleeping state the algorithm simply does nothing until its wake-up time $t'$. At this time the algorithm consults again the work-or-sleep subroutine. If the result is $(S, \text{work})$, then the algorithm enters the working state and follows $S$. Otherwise the algorithm continues to sleep with new wake-up time $\ell(S)/\alpha$.

In the working state, that is, while the server is following a schedule, all new requests are (temporarily) ignored. As soon as the current schedule is completed the server either enters the idle-state (if there are no unserved requests) or it consults the work-or-sleep subroutine which determines the next state (sleeping or working).

**Theorem 2.9** ([11]). *is a c-competitive algorithm for the online TSP with*

$$c = \max\left\{1+\alpha, \rho\left(1+\frac{1}{\alpha}\right), \frac{1+\alpha}{2}+\rho\right\}.$$

*Proof.* Denote by $\sigma^{=r_m}$ the set of requests released at time $r_m$, where $r_m$ is the latest release date. We distinguish different cases depending on the state of

at time $r_m$.

1. The algorithm is idle.

   In this case the algorithm consults its work-or-sleep routine which computes an approximately shortest schedule $S$ for the requests in $\sigma^{=r_m}$. The server will start its work at time $t' = \max\{r_m, \ell(S)/\alpha\}$.

   If $t' = r_m$, it follows that $\ell(S) \leq \alpha r_m$ and the algorithm completes no later than time $(1+\alpha)r_m \leq (1+\alpha)$ $(\sigma)$. Otherwise $t' = \ell(S)/\alpha$ and it follows that $t' + \ell(S) = (1+\alpha)t'$. By the performance guarantee $\rho$ of the approximation algorithm employed in the work-or-sleep subroutine, we have $(\sigma) \geq \ell(S)/\rho = \alpha t'/\rho$. Thus, it follows that

   $$\begin{aligned} (\sigma) &= t' + \ell(S) \\ &\leq (1+\alpha)t' \leq (1+\alpha) \cdot \frac{\rho}{\alpha} \quad (\sigma) \\ &= \rho\left(1 + \frac{1}{\alpha}\right) \quad (\sigma). \end{aligned}$$

2. The algorithm is sleeping.

   Since the wake-up time of the server is no later than $\max\{r_m, \ell(S)/\alpha\}$, where $S$ is now a shortest schedule for all the requests in $\sigma$ not yet served by at time $r_m$, we can proceed as in the first case.

3. The algorithm is working.

   If after the completion of the current schedule the server enters the sleeping state, then by the same arguments as above we can ensure that the completion time of the server does not exceed $\rho(1 + \frac{1}{\alpha})$ $(\sigma)$.

   The remaining case is that the server starts its final schedule $S'$ immediately after having completed $S$. Let $t_S$ be the time when the server started $S$ and denote by $\sigma^{\geq t_S}$ the set of requests presented after the server started $S$ at time $t_S$. Notice that $\sigma^{\geq t_S}$ is exactly the set of requests that are served by in its last schedule $S'$:

   $$(\sigma) = t_S + \ell(S) + \ell(S'). \tag{2.1}$$

   Here, $\ell(S)$ and $\ell(S')$ denote the lengths of the schedules $S$ and $S'$, respectively. We have that

   $$\ell(S) \leq \alpha t_S \tag{2.2}$$

   since only starts a schedule at some time $t$ if its length is no larger than $\alpha t$. Let $\sigma_j \in \sigma^{\geq t_S}$ be the first request from $\sigma^{\geq t_S}$ served by . Since the optimal schedule must be closed, we conclude that

   $$(\sigma) \geq t_S + d(x_j, o). \tag{2.3}$$

On the other hand, since a feasible schedule for $\sigma^{\geq t_S}$ consists in moving to $x_j$ and then following the same route as the adversary, we have

$$\ell(S') \leq \rho(d(o, x_j) + \quad (\sigma) - t_S) \tag{2.4}$$

where we subtracted $t_S$ from the cost since we are computing a length, not a completion time, and the adversary will not serve $\sigma_j$ at a time earlier than $t_S$.

Using (2.2), (2.3) and (2.4) in (2.1) and the assumption that $\rho \leq 1 + \alpha$, we obtain

$$\begin{array}{rcll}
(\sigma) & \leq & (1+\alpha)t_S + \ell(S') & \text{(by 2.2)} \\
& \leq & (1+\alpha-\rho)t_S + \rho d(o, x_j) + \rho \quad (\sigma) & \text{(by 2.4)} \\
& \leq & (1+\alpha) \quad (\sigma) + (2\rho - 1 - \alpha)d(o, x_j) & \text{(by 2.3)} \\
& \leq & (1+\alpha) \quad (\sigma) + \max\{(2\rho - 1 - \alpha), 0\}\frac{(\sigma)}{2} & \\
& \leq & \max\left\{\frac{1+\alpha}{2} + \rho, 1+\alpha\right\} \quad (\sigma) &
\end{array}$$

and the proof is complete.

□

**Corollary 2.10** ([11]). *is a 2-competitive algorithm for the homing online TSP when* $\alpha = \rho = 1$.

By using Christofides' algorithm, we can approximate the offline TSP in polynomial time with a ratio $\rho = 3/2$. For this value of $\rho$, the best value of $\alpha$ is $(\sqrt{13} - 1)/2 \approx 1.3$.

**Corollary 2.11** ([11]). *Let* $c = (7 + \sqrt{13})/4 \approx 2.65$. *Then is a c-competitive polynomial time algorithm for the homing online TSP when* $\alpha = (\sqrt{13} - 1)/2$ *and Christofides' algorithm is used in the work-or-sleep subroutine.*

### 2.3.3 The nomadic online TSP

As we have seen in Section 2.2.2, if we relax the condition that the server ends its schedule at the same point it departed from, we obtain the Wandering Salesman Problem. Its online version is called the *nomadic* online TSP and has been introduced together with the homing online TSP [17]. Ausiello et al. [17] give a 5/2-competitive algorithm for general metric spaces and a simple lower bound of 2. The same lower bound also holds for the real line, in which case the authors give a 7/3-competitive algorithm.

Lipmann [75] gives an improved algorithm for general spaces with competitive ratio $1 + \sqrt{2}$. This algorithm, as that of Ausiello et al., is not polynomial time, but it can be made so with only a constant increase in the competitive ratio, by combining it with the approximation algorithm of Theorem 2.6. Lipmann also

considers the problem on the real line, for which he gives a best possible, though fairly complicated, algorithm with competitive ratio 2.03 [76].

The case of the halfline received less attention, the only result being a 1.63 lower bound by Lipmann [75]. In this section we give a simple 2-competitive polynomial time algorithm for this variant. As the gaps in the results suggest, the nomadic online TSP is harder to analyze than its homing counterpart and no best possible algorithm is known, not even on special spaces such as the real line.

We will consider the nomadic TSP on asymmetric spaces in Chapter 3, as well as a multiserver variant in Chapter 5. In order to present the basic techniques that will be used there, we present here our 2-competitive algorithm for the halfline.

### A 2-competitive algorithm on the halfline

For the nomadic online TSP on the halfline, we give here a simple 2-competitive algorithm (Algorithm 2).

**Algorithm 2** -

The server follows the space-time line $s(t) = t/2$, that is, it departs from the origin at half-speed. The server always serves immediately every request it encounters along the way. On every new request released between the current position of the server and the origin, the server goes back at full speed to the unserved request that is closest to the origin, serving all requests it encounters on the way. Then, it proceeds at full speed away from the origin till encountering the space-time line $s(t) = t/2$. At that point, it remains on that space-time line until the next request, as before.

**Theorem 2.12.** - *is 2-competitive for nomadic online TSP on the halfline.*

*Proof.* If no request is ever released between the server and the origin, - is obviously 2-competitive. Otherwise, let $\sigma_j = (r, x)$ be the last request causing the server to back up, and let $x_*$ be the distance from the origin to the farthest request ever. We have

$$(\sigma) \geq \max\{r, x_*\}, \tag{2.5}$$

since no schedule can end before request $\sigma_j$ is released, and in every schedule the server should reach the farthest point requested. If $x_* \leq s(r)$, since $s(r) \leq r/2$, we have:

- $(\sigma) \leq r + s(r) + x_* \leq 2r \leq 2$ $(\sigma)$.

Otherwise, $x_* > s(r)$ and - $(\sigma) \leq r + r/2 - x + x_* - x$. We distinguish two cases.

1. $r \geq 2x_*$: then - $(\sigma) \leq 2r \leq 2$ $(\sigma)$.

2. $r < 2x_*$: in this case we need another lower bound on $(\sigma)$. If the adversary serves first request $\sigma_j$ and then the one in $x_*$, we have $(\sigma) \geq r + x_* - x$; otherwise, $(\sigma) \geq x_* + x_* - x$. In both cases,

$$(\sigma) \geq \min\{r, x_*\} + x_* - x. \tag{2.6}$$

Summing (2.5) to (2.6) we get $(\sigma) \geq x_* + (r - x)/2$. The competitive ratio is then

$$\begin{aligned} \frac{- \quad (\sigma)}{(\sigma)} &\leq \frac{r + r/2 + x_* - 2x}{r/2 + x_* - x/2} \\ &\leq \frac{r + r/2 + x_*}{r/2 + x_*} \\ &= 1 + \frac{r}{r/2 + x_*} \leq 2. \end{aligned}$$

□

### 2.3.4 The online TRP

The online version of the Traveling Repairman Problem has been considered by several authors. Feuerstein and Stougie [45] give a 9-competitive algorithm for the real line and a corresponding lower bound of $1 + \sqrt{2}$. This lower bound is still the best known, even for general spaces. Instead, the best known algorithm for general spaces was given by Krumke et al. [68] where a $(1 + \sqrt{2})^2 \approx 5.81$-competitive algorithm was devised for the more general dial-a-ride problem.

The large gap between the best known lower and upper bounds witnesses the fact that the online TRP is an analytically difficult problem. We will consider a multiserver variant of the online TRP in Chapter 5.

### 2.3.5 Other related problems

Following the initial body of results on online server routing problems, many other related problems have been considered in the literature.

In the *minimum total flow time* online routing problem, the objective function is $\sum_j (C_j - r_j)$, which is equivalent to the minimization of the average delay between the release date of a request and its completion time. This objective function is particularly relevant because it can represent average user dissatisfaction in a continuously running system. Unfortunately, there cannot exist constant competitive algorithms for this problem, even in very restricted settings [6]. This is reminiscent of similar results in the scheduling literature [20, 64, 74]. Analogous nonexistence results hold for the *minimum maximum flow time* problem [6]. Several attempts at studying these problems in weaker adversarial models have been considered (see Section 2.4).

Online *dial-a-ride problems* have also been considered by some authors. In a dial-a-ride problem, every request specifies both a source and a destination point in

the metric space. The object corresponding to the request must be transported by the server between the two points. The server has some capacity $C$, meaning that it can transport at most $C$ objects at a time, and preemption is not allowed: once an object has been picked up by the server, the server is not allowed to drop it at any other place than its destination. Dial-a-ride problems are generalizations of server routing problems, which are obtained as special cases when for every request its source and destination coincide. Some of the best known algorithms for server routing problems have been indeed devised in the context of dial-a-ride problems. Ascheuer et al. [11] give a best possible competitive algorithm for the homing dial-a-ride problem. For the nomadic dial-a-ride problem, Lipmann [75] gives a $(3 + \sqrt{5})/2 \approx 2.62$-competitive algorithm. The "latency" dial-a-ride problem, with cost $\sum_j C_j$, has been investigated by Feuerstein and Stougie [45], who give a lower bound of 3 on the competitive ratio of the problem and a 9-competitive algorithm for the real line. This result was generalized and improved by Krumke et al. [68], who devised a $(1+\sqrt{2})^2 \approx 5.81$-competitive algorithm for general metric spaces. Some results for the dial-a-ride problem with the objective of minimizing the maximum flow time are given by Krumke et al. [67].

Irani et al. [58] consider a server routing problem where requests have deadlines. The objective is to maximize the number of requests that are served by their deadline. The length of time between the arrival of a request and its deadline is a constant. The authors give both upper and lower bounds on the competitive ratio of the problem in terms of the diameter of the metric space. The same problem is also investigated by Krumke et al. [71], who give a constant competitive algorithm for the uniform metric space and prove that no deterministic algorithm can achieve a constant competitive ratio on the real line.

In Chapter 3 we study an online version of the *asymmetric TSP with triangle inequality*, that is, we drop the symmetry condition on the metric space. We give a best possible competitive algorithm for the homing version, and we show that there cannot be constant competitive algorithms for the nomadic case. Indeed, we show that, in a precise sense, in the nomadic case the competitive ratio has to be a function of the amount of asymmetry of the space.

In Chapter 4 we consider the *online Prize-Collecting TSP*. In the Prize-Collecting TSP, individual requests are not required to be served, but each request has a weight and every feasible schedule has to serve enough requests to collect at least a certain total weight, called the *quota*. The quota is specified before requests start being released. Moreover, every request has an associated value called its *penalty*, and the cost of a schedule is given by its length plus the sum of the penalties of the requests not in the schedule. For this problem we give a $7/3$-competitive algorithm. For the special case when the metric space is the real halfline we obtain an improved competitive ratio of 2, which compares with a lower bound of 1.89. A special case of the online Prize-Collecting TSP called *online Quota TSP* has already been considered in the literature by Ausiello et al. [16]. The Quota TSP is defined as the Prize-Collecting TSP, except that there are no penalties. Ausiello et al. give a simple 2-competitive algorithm for the online Quota TSP and prove it to

be best possible among deterministic algorithms. They also consider the problem on the halfline, and give a best possible $3/2$-competitive algorithm for this case.

## 2.4 Alternative online models

As we have seen in the Introduction of this thesis, competitive analysis is sometimes criticized for leading to estimates that are too pessimistic. This critique applies also in the context of online server routing, and in particular to problems such as the minimum total/maximum flow time, for which no constant competitive algorithms are at all possible. For this reason, many alternative models have been suggested in the literature in the hope of restricting the adversary enough so that the differences in quality between various online algorithms for such problems become evident. These efforts have been partially successful on the individual problems considered, but still a more general, unified approach has not emerged.

### 2.4.1 Resource augmentation

In the online algorithms literature, a method of analysis that is often used to partially mitigate the power of the adversary is resource augmentation (see also Section 1.7). Although resource augmentation has been used for other online problems following the real-time model, such as scheduling, no application of resource augmentation specific for server routing problems was known, prior to this thesis. We give some resource augmentation results for the nomadic online TSP and the online TRP in Chapter 5. We notice that resource augmentation in terms of number or speed of the servers is not enough when dealing with the total or maximum flow time objective functions.

### 2.4.2 Fair adversary

One of the first alternative adversarial models proposed in the context of online server routing has been the *fair adversary* model. The observation is that for some algorithms, the worst case is obtained when the adversary moves away from previously released requests without giving any information to the online algorithm. A fair adversary, instead, always keeps its server within the convex hull of the requests released so far. The fair adversary model has been introduced by Blom et al. [25], who consider the homing online TSP on the nonnegative part of the real line. They devise an online algorithm with a competitive ratio of $(1 + \sqrt{17})/4 \approx 1.28$ against a fair adversary, and also show that this ratio is best possible. Lipmann [75] also has several lower bounds for routing problems with fair adversaries.

Unfortunately, already on the real line, the fair adversary is not weak enough to allow competitive algorithms for cost functions such as the maximum flow time [70]. However, when considering the same objective function in a uniform metric space, the simple first-come first-serve algorithm which always serves an oldest

unserved request next is 2-competitive and best possible against a fair adversary [67].

### 2.4.3 Non-abusive adversary

The fair adversary is still too powerful in the sense that it can move to a point where it knows that a request will pop up without revealing any information to the online server before reaching that point. The *non-abusive adversary*, introduced by Krumke et al. [70], is a natural restriction of the adversary on the real line. A non-abusive adversary may only move in a direction if there are yet unserved requests on that side. Krumke et al. [70] give an 8-competitive algorithm for the maximum flow time server routing problem on the real line against a non-abusive adversary.

### 2.4.4 Reasonable load

A completely different approach to avoid pathological worst case input sequences is based on the notion of *reasonable load*, proposed by Hauptmeier et al. [56]: informally, they define a set of requests, that come up in a sufficiently large period, to be reasonable if they can be served by an optimal algorithm in a time period of the same length. This corresponds to a stability condition, similar to those used in queueing theory, implying that the system is not overloaded by requests even when it is continuously operating. The authors analyze in this model the online dial-a-ride problems for the minimization of maximum and total flow time. Under the reasonable load condition, they are able to distinguish between the quality of two classical algorithms for which pure competitive analysis gives the same negative result.

Following similar motivations, in Chapter 6 we propose a model for continuously operating server routing systems that is based on *adversarial queueing theory* [32]. We assume that requests are introduced in the system by an adversary with some bounded rate. This allows us to study questions such as whether an algorithm is *stable*, that is, it maintains a bounded number of unserved requests at any time, or whether it is such that the maximum flow time of every request is also bounded (that is, independent of the age of the system).

### 2.4.5 Algorithms with lookahead

*Lookahead* is the capability of an online algorithm of seeing some requests in advance, and as such it can be seen as a relaxation of the online model. Classic online problems have been studied in the presence of lookahead, such as paging [2] and list update [3]. Different models of lookahead have been proposed, but usually the online algorithms are allowed to see a certain fixed number of requests in advance. For online problems following the real-time model, such as online server routing problems, a more natural notion is that of *time lookahead*: an online algorithm endowed with time lookahead $\Delta$ foresees, at any time $t$, all the requests that will

be released up to time $t + \Delta$, no matter their number. Allulli et al. [6] consider server routing problems in metric spaces with limited diameter $\delta$, and give results where the competitive ratio is a function of $\Delta/\delta$. Jaillet and Wagner [59] instead give competitiveness results where the competitive ratio is a function of the ratio between the amount of time lookahead $\Delta$ and some characteristic quantities of the input instance, related to its optimal cost.

### 2.4.6 Randomized algorithms

Generally speaking, allowing randomization can be seen as a way to improve the power of an online algorithm. Indeed, for some classical online problems, such as paging, it is known that randomized algorithms obtain strictly better competitive ratios than their deterministic counterparts [46].

For online server routing problems, it is unclear whether randomization really helps online algorithms. Krumke et al. [68, 69] present a $(2 + \sqrt{2})/\ln(1 + \sqrt{2}) \approx 3.87$-competitive randomized algorithm for the online TRP against an oblivious adversary. This algorithm has a better performance than the best known deterministic algorithm. However, its competitive ratio is still well above the lower bound of $1 + \sqrt{2}$ for deterministic algorithms. Thus, it could be the case that a better, deterministic algorithm is discovered. For other problems, the best known algorithms are deterministic. However, this is probably mostly due to the fact that the literature has focused mainly on deterministic algorithms.

### 2.4.7 Zealous algorithms

A distinctive feature of online problems in the real-time model is that the server can not only serve the requests in an arbitrary order, but it can also be idle, waiting. Although at first thought this may not sound as a good idea, there are indeed situations in which some waiting is beneficial. In particular, by waiting, the server could avoid the need to do part of its work two or more times. If this extra work costs more than waiting, then by waiting the algorithm is actually saving something.

The phenomenon we have just described is not uncommon in server routing problems. In order to study the matter more quantitatively, Blom et al. [25] introduced the notion of *zealous algorithms* for online server routing problems. Informally, a zealous algorithm should never sit and wait when it could serve unserved requests. Also, a zealous server should move towards work that has to be done directly, without any detours.

**Definition 2.5.** An algorithm for an online server routing problem is called *zealous* if it satisfies the following conditions:

1. If there are still unserved requests, the direction of the server changes only if a new request becomes known, or if the server is either in the origin or it has just served a request.

2. At any time when there are unserved requests, the server either moves towards an unserved request or the origin at maximum (that is, unit) speed.

3. If there are unserved requests, the server cannot wait at the origin.

By comparing the competitive ratio of zealous and non-zealous algorithms for the same problem, we can measure the importance of waiting. For example, Blom et al. show that no zealous online algorithm for the online TSP on the real line can have competitive ratio less than $7/4$. Thus, the $7/4$-competitive algorithm presented by Ausiello et al. [17], which is in fact a zealous algorithm, is best possible within its class. Instead, a non-zealous algorithm for the same problem attains the better ratio $(9 + \sqrt{17})/8 \approx 1.64$ [75]. Other comparisons between zealous and non-zealous algorithms for online server routing problems are given by Lipmann [75].

### 2.4.8 Restricted information

A model of online dial-a-ride problems in which the online algorithm, instead of the adversary, has more restrictions than usual is the *restricted information model* considered by Lipmann et al. [77]. In this model, the online algorithm becomes aware of the destination of a ride only when the ride begins (such as when the server picks up the customer from the source point, as it actually happens, for example, with many radio-taxi services). They show that lower and upper bounds become considerably worse in this model, concluding that, in many real-world scenarios, it is worthwhile to invest on the information system in order to gather all the information as soon as a request is presented.

# Chapter 3

# The online asymmetric traveling salesman problem

## 3.1 Introduction

In the classical traveling salesman problem, a set of cities has to be visited in a single tour with the objective of minimizing the total length of the tour. As we saw in Chapter 2, this, together with its dozens of variations, is one of the most studied problems in combinatorial optimization. In the asymmetric version of the problem, the distance from one point to another in a given space can be different from the inverse distance. This variation, known as the Asymmetric Traveling Salesman Problem (ATSP) arises in many applications; for example, one can think of a delivery vehicle traveling through one-way streets in a city, or of gasoline costs when traveling through mountain roads.

Here we are interested in the online version of the ATSP, named online ATSP. In the online TSP and ATSP, the places to be visited in the space are requested over time and a server (the salesman or vehicle) has to decide in what order to serve them, without knowing the entire sequence of requests beforehand. The objective is to minimize the completion time of the server. To analyze our algorithms, we use the established framework of *competitive analysis* (see Chapter 1). This work is the first to address the online ATSP from the point of view of competitive analysis. Previous work, both theoretical and experimental, has focused on the offline version [39, 49, 63].

Our results are summarized in Table 3.1, where they are also compared with the known results for the symmetric case. As we will see, the asymmetric TSP is substantially harder than the normal TSP also when considered from an online point of view; in other words, the online ATSP is not a trivial extension of the online TSP. In fact, as Table 3.1 shows, most bounds on the competitive ratio are strictly higher than the corresponding bounds for the online TSP, and in particular in the nomadic case there cannot be online algorithms with a constant competitive ratio.

Although the algorithmic techniques we adopt in the asymmetric case come essentially from the symmetric case, they require some adjustment in order to attain useful competitive ratios. On the other hand, it is worth noting that the lower bound techniques are quite different from the previously known ones and we hope they can be of some use in future work.

We should also mention that we present our algorithms in simplified versions that compute optimal traveling salesman tours or paths. Thus, they will not run in polynomial time unless P=NP. However, if one is interested in polynomial running time, it is possible to compute approximately optimal tours instead, the competitive ratio degrading by a factor that is essentially (at most) the approximation ratio of the subroutine being used. For example, as a consequence of our results, a constant approximation algorithm for the ATSP would automatically imply a constant competitive polynomial time algorithm for the online ATSP. We further discuss this issue in Section 3.6.

The rest of this chapter is organized as follows. After giving a brief review of the approximation results for the offline problem in Section 3.2, we give the necessary definitions and the discussion of the online model in Section 3.3. In Section 3.4, we study the homing case of the problem, in which the server is required to finish its tour at the same place where it started; for this problem we give a $\frac{3+\sqrt{5}}{2} \approx 2.62$-competitive algorithm and show that this is best possible. In Section 3.5, we address the nomadic version, also known as the wandering traveling salesman problem (see also Section 2.2.2), in which the server is not required to finish its tour at the origin. For this case we show that in general an online algorithm with a competitive ratio independent of the space cannot exist; indeed, we show that the competitive ratio has to be a function of the amount of asymmetry of the space (in a precise sense introduced in Section 3.3). In Section 3.6 we explain how our algorithms can be combined with polynomial time approximation algorithms in order to obtain polynomial time online algorithms. In the last section, we give our conclusions and discuss some open problems.

## 3.2 The offline ATSP

The ATSP has been well studied from the point of view of approximation algorithms. However, if the condition is that every city or place has to be visited *exactly* once, the general problem does not admit any $\rho$-approximation in polynomial time, unless P=NP or $\rho$ is a superpolynomial function of the number of cities (recall Theorem 2.2).

Instead, when every city or place given in the input has to be visited *at least* once or, equivalently, the distance function satisfies the triangular inequality (which we will tacitly assume for the rest of the chapter), approximation algorithms exist having an approximation ratio of $O(\log n)$ [49, 63].

**Theorem 3.1** ([49])**.** *There is a* $\log_2 n$*-approximation algorithm for the* ATSP *on* $n$ *cities.*

| Problem | Lower Bound | Upper Bound | References |
|---|---|---|---|
| Homing TSP | 2 | 2 | [11, 17] |
| Homing ATSP | $(3+\sqrt{5})/2$ | $(3+\sqrt{5})/2$ | Th. 3.3, 3.4 |
| Homing TSP (zealous) | 2 | 2 | [17] |
| Homing ATSP (zealous) | 3 | 3 | Th. 3.5, 3.6 |
| Nomadic TSP | 2.03 | $1+\sqrt{2}$ | [75] |
| Nomadic ATSP | $\sqrt{\psi}$ | $1+\sqrt{\psi+1}$ | Th. 3.9, 3.10 |
| Nomadic TSP (zealous) | 2.05 | 2.5 | [17, 75] |
| Nomadic ATSP (zealous) | $(\psi+1)/2$ | $\psi+2$ | Th. 3.11, 3.10 |

Table 3.1: The competitive ratio of symmetric and asymmetric routing problems. Refer to Section 3.3 for the definition of $\psi$.

For the sake of completeness, we give a short proof of this simple but elegant result.

*Proof.* The algorithm is based on solving multiple instances of the *cycle cover* problem. In the cycle cover problem, the input is the same as in the ATSP: a complete weighted directed graph on $n$ nodes. A solution is simply a cycle cover, that is, a collection of node-disjoint directed cycles, such that every city appears in some cycle, and every cycle has length at least two. The cost of a cycle cover is the sum of the weights of all its cycles. In the cycle cover problem, the objective is to minimize the weight of a cycle cover. Thus, the ATSP is just the cycle cover problem with the additional constraint that there cannot be more than one cycle. Despite this similarity, the cycle cover problem can be solved in polynomial time, as it can be easily reduced to the well-known assignment problem [72], while the ATSP is NP-hard.

The algorithm of Frieze et al. simply solves the cycle cover problem repeatedly and "patches" together the cycles thus obtained until a single cycle remains. More precisely, at stage $i$ the algorithm solves the cycle cover problem on the remaining nodes $S_i$, obtaining a cycle cover $CC_i$. The cost of $CC_i$ is at most the cost of the optimal ATSP tour, since the restriction of this tour to $S_i$ is a feasible cycle cover. Then, $S_{i+1}$ is constructed by selecting one arbitrary node per cycle in $CC_i$. Since at each stage every cycle has length at least two, $|S_{i+1}| \leq |S_i|/2$. Thus, there are at most $\log_2 n$ stages, and the total cost of the cycle covers found is at most $(\log_2 n) \cdot$ . Finally, the cycle covers are merged together to form an Eulerian directed multigraph $M$ on the initial set of $n$ nodes. From this multigraph, an Eulerian tour is obtained and then "shortcut" into a single $n$-node tour that, by the triangle inequality, has at most the same cost as $M$. □

Using a more sophisticated approach, Kaplan et al. [63] obtain an approxima-

tion algorithm with ratio $0.842 \log_2 n$, which is currently the best known result.

On the negative side, it is known that the problem does not admit polynomial time approximation schemes if P $\neq$ NP.

**Theorem 3.2** ([80])**.** *There is no r-approximation algorithm for the ATSP with triangle inequality for* $r < 117/116$*, unless* P=NP.

The question of the existence of an algorithm with a constant approximation ratio for the asymmetric case is still open after more than two decades.

## 3.3 The online model

An input for the online ATSP consists of a space $\mathbb{M}$ from the class $\mathcal{M}$ defined below, a distinguished point $o \in \mathbb{M}$, called the origin, and a sequence of requests $\sigma_i = (r_i, x_i)$ where $x_i$ is a point of $\mathbb{M}$ and $r_i \in \mathbb{R}_+$ is the time when the request is presented. The sequence is ordered so that $i < j$ implies $r_i \leq r_j$.

The server is located at the origin $o$ at time 0 and the distances are scaled so that, without loss of generality, the server can move at most at unit speed.

We will consider two versions of the problem. In the *nomadic* version, the server can end its schedule anywhere in the space; the objective is just to minimize the makespan, that is, the time required to serve all presented requests. In the *homing* version, the objective is to minimize the time required to serve all presented requests and return to the origin.

An *online* algorithm for the online ATSP has to determine the behavior of the server at a certain moment $t$ as a function only of the requests $(r_i, x_i)$ such that $r_i \leq t$. Thus, an online algorithm does not have knowledge about the number of requests or about the time when the last request is released. We will use $S$ to denote a schedule over a subset of the requests; in this case, $\ell(S)$ will be the length of that schedule.

Finally, we would like to clarify the conditions that the space $\mathbb{M}$ should satisfy. Usually, in the context of the online TSP, continuous path-metric spaces are considered (as defined in Section 2.3.1). However, here the main issue is precisely asymmetry, so we have to drop the requisite that for every $x$ and $y$, $d(x, y) = d(y, x)$. We review here the definitions. A set $\mathbb{M}$, equipped with a distance function $d : \mathbb{M}^2 \to \mathbb{R}_+$, is called a *quasi-metric space* if, for all $x, y, z \in \mathbb{M}$:

(i) $d(x, y) = 0$ if and only if $x = y$;

(ii) $d(x, y) \leq d(x, z) + d(z, y)$.

We call a space $\mathbb{M}$ an *admissible space* if $\mathbb{M}$ is a quasi-metric and, for any $x, y \in \mathbb{M}$, there is a function $f : [0, 1] \to \mathbb{M}$ such that $f(0) = x$, $f(1) = y$ and $f$ is continuous, in the following sense: $d(f(a), f(b)) = (b - a)d(x, y)$ for any $0 \leq a \leq b \leq 1$. Such a function represents a *shortest path* from $x$ to $y$. Notice that every admissible space is strongly connected.

We will use $\mathcal{M}$ to denote the class of admissible spaces. Notice that the discrete metric induced by a weighted graph is not admissible if we take $\mathbb{M}$ to be the set of vertices. However, we can always make such a space admissible by adding (an infinity of) extra points "along the arcs".

In particular, to see how a directed graph with positive weights on the arcs can define an admissible space, consider the all-pairs shortest paths matrix of the graph. This defines a finite quasi-metric. Now we add, for every arc $(x, y)$ of the graph, an infinity of points $\pi_\gamma$, indexed by a parameter $\gamma \in (0, 1)$. Let $\pi_0$ and $\pi_1$ denote $x$ and $y$ respectively. We extend the distance function $d$ so that:

$$d(\pi_\gamma, \pi_{\gamma'}) = (\gamma' - \gamma)d(x, y) \quad \text{for all } 0 \leq \gamma < \gamma' \leq 1.$$

It can be verified that the set $\pi = \{\pi_\gamma : \gamma \in [0, 1]\}$ represents a shortest path from $x$ to $y$. For $\gamma \notin \{0, 1\}$, the distance from a point $\pi_\gamma$ to a point $z$ not in $\pi$ is defined as $d(\pi_\gamma, z) = d(\pi_\gamma, y) + d(y, z)$; that is, the shortest path from $\pi_\gamma$ to $z$ passes through $y$. Vice versa, the distance from $z$ to $\pi_\gamma$ is defined as $d(z, \pi_\gamma) = d(z, x) + d(x, \pi_\gamma)$. Finally,

$$d(\pi_{\gamma'}, \pi_\gamma) = (1 - (\gamma' - \gamma))d(x, y) + d(y, x) \quad \text{for all } 0 \leq \gamma < \gamma' \leq 1.$$

We say that such a space is *induced* by the original directed weighted graph. We remark that this model, while still including the originally proposed one [17] as a special case, can also capture the situation in which the server is not allowed to do U-turns.

Finally, it will be useful to have a measure of the amount of asymmetry of a space. Define as the *maximum asymmetry* of a space $\mathbb{M} \in \mathcal{M}$ the value

$$\psi(\mathbb{M}) = \sup_{\substack{x,y \in \mathbb{M} \\ x \neq y}} \frac{d(x, y)}{d(y, x)}.$$

We will say that a space $\mathbb{M}$ has *bounded asymmetry* when $\psi(\mathbb{M}) < \infty$. Indeed, the spaces with bounded asymmetry are exactly those in which the server is always allowed to do a U-turn.

## 3.4 The homing online ATSP

In this section we consider the homing version of the online ATSP, in which the objective is to minimize the completion time required to serve all presented requests and return to the origin. We establish a lower bound of about 2.618 and a matching upper bound. Note that in the symmetric online TSP, the corresponding bounds are both equal to 2 [17, 66].

Let $\phi$ denote the golden ratio, that is, the unique positive solution to $x = 1+1/x$. In closed form, $\phi = \frac{1+\sqrt{5}}{2} \approx 1.618$.

**Theorem 3.3.** *The competitive ratio of any deterministic online algorithm for the homing online ATSP is at least* $1 + \phi$.

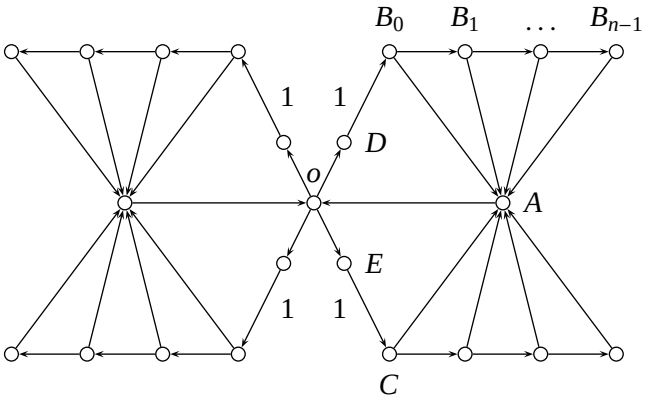


Figure 3.1: The graph used in the proof of Theorem 3.3.

*Proof.* Fix any $\epsilon > 0$. The space used in the proof is the one induced by the graph depicted in Figure 3.1. The graph has $7 + 4n$ nodes, where $n = 1 + \lceil \frac{\phi-1}{\epsilon} \rceil$, and the length of every arc is $\epsilon$, except for those labeled otherwise. Observe that the space is symmetric with respect to an imaginary vertical axis passing through $o$. Thus, we can assume without loss of generality that, at time 1, no request being released yet, the online server is in the left half of the space. Then at time 1 a request is given in point $A$, in the other half. Now let $t$ be the first time at which the online server reaches point $D$ or $E$.

If $t \geq \phi$, no further request is given. In this case $(\sigma) \geq t + 1 + 2\epsilon$ while $(\sigma) \leq 1 + 3\epsilon$ so that, when $\epsilon$ approaches zero, $(\sigma)/(\sigma)$ approaches $1 + t \geq 1 + \phi$.

Otherwise, if $t \in [1, \phi]$, at time $t$, we can assume that the online server has just reached $E$ (again, by symmetry). At this time, the adversary gives a request in $B_i$, where $i = \lceil \frac{t-1}{\epsilon} \rceil$. Now the online server has to traverse the entire arc $EC$ before it can serve $B_i$, thus

$$(\sigma) \geq t + 1 + 3\epsilon + 1 + \epsilon \left\lceil \frac{t-1}{\epsilon} \right\rceil + 2\epsilon \geq 2t + 1 + 5\epsilon.$$

Instead, the adversary server will have moved from $o$ to $B_i$ in time at most $t + 2\epsilon$ and then served $B_i$ and $A$, achieving the optimal cost $(\sigma) \leq t + 4\epsilon$. Thus, when $\epsilon$ approaches zero, $(\sigma)/(\sigma)$ approaches $2 + \frac{1}{t} \geq 1 + \phi$. □

To prove a matching upper bound on the competitive ratio, we use a variation of algorithm , described in Section 2.3.2. Here we give a different, less formal description of the algorithm.

As we will soon see, the best value of $\alpha$ is $\alpha^* = \phi$.

**Theorem 3.4.** *is $(1 + \phi)$-competitive for the homing online ATSP when $\alpha = \phi$.*

**Algorithm 3**

The algorithm keeps track, at every time $t$, of the length of an optimal schedule $S^*(t)$ over the unserved requests, starting at and returning to the origin. At the first instant $t'$ such that $t' \geq \alpha\ell(S^*(t'))$, the server starts following at full speed the currently optimal schedule, ignoring temporarily every new request. When the server is back at the origin, it stops and returns monitoring the value $\ell(S^*(t))$, starting as before when necessary.

*Proof.* We distinguish two cases depending on whether the last request arrives while the server is waiting at the origin or not.

In the first case, let $t$ be the release time of the last request. If the server starts immediately at time $t$, it will follow a schedule of length $\ell(S^*(t)) \leq t/\alpha$, ending at time at most $(1 + 1/\alpha)t$, while the adversary pays at least $t$, so the competitive ratio is at most $1 + 1/\alpha$. Otherwise, the server will start at a time $t' > t$ such that $t' = \alpha\ell(S^*(t))$ (since $S^*$ does not change after time $t$) and pay $(1 + \alpha)\ell(S^*(t))$, so the competitive ratio is at most $1 + \alpha$.

In the second case, let $S^*(t)$ be the schedule that the server is following while the last request arrives; that is, we take $t$ to be the starting time of that schedule. Let $S'(t)$ be an optimal schedule over the requests released *after* time $t$. If the server has time to wait at the origin when it finishes following $S^*(t)$, the analysis is the same as in the first case. Otherwise, the completion time of is $t + \ell(S^*(t)) + \ell(S'(t))$. Since has started following $S^*(t)$ at time $t$, we have $t \geq \alpha\ell(S^*(t))$. Then

$$t + \ell(S^*(t)) \leq (1 + 1/\alpha)t.$$

Also, if $\sigma_f = (r_f, x_f)$ is the first request served by the adversary having release time at least $t$, we have that $\ell(S'(t)) \leq d(o, x_f) + (\sigma) - t$, since a possibility for $S'$ is to go to $x_f$ and then do the same as the adversary (subtracting $t$ from the cost since we are computing a length, not a completion time, and on the other hand the adversary will not serve $\sigma_f$ at a time earlier than $t$).

By putting everything together, we have that pays at most

$$(1 + 1/\alpha)t + d(o, x_f) + (\sigma) - t$$

and since two obvious lower bounds on $(\sigma)$ are $t$ and $d(o, x_f)$, this is easily seen to be at most $(2 + 1/\alpha)$ $(\sigma)$.

Now $\max\{1 + \alpha, 2 + \frac{1}{\alpha}\}$ is minimum when $\alpha = \alpha^* = \phi$. For this value of the parameter the competitive ratio is $1 + \phi$. □

### 3.4.1 Zealous algorithms

In the previous section we have seen that the optimum performance is achieved by an algorithm that, before starting to serve requests, waits until a convenient

starting time is reached. In this section we consider instead the performance that can be achieved by *zealous* algorithms. A zealous algorithm does not change the direction of its server unless a new request becomes known, or the server is at the origin or at a request that has just been served; furthermore, a zealous algorithm moves its server always at full (that is, unit) speed when there are unserved requests (see Section 2.4.7 for a precise definition).

We show that, for zealous algorithms, the competitive ratio has to be at least 3 and, on the other hand, we give a matching upper bound.

**Theorem 3.5.** *The competitive ratio of any zealous deterministic online algorithm for the homing online ATSP is at least* 3*.*

*Proof.* We use the same space used in the lower bound for general algorithms (Figure 3.1). At time 1, the server has to be at the origin and the adversary gives a request in $A$. Thus, at time $1 + \epsilon$ the server will have reached wlog $E$ (by symmetry) and the adversary gives a request in $B_0$. The completion time of the online algorithm is at least $3 + 6\epsilon$, while $(\sigma) \leq 1 + 3\epsilon$. The result follows by taking a sufficiently small $\epsilon$. □

Using the previous theorem, we can prove that the Ignore algorithm (Algorithm 4) is best possible among the zealous algorithms for the homing online ATSP.

**Algorithm 4**

Whenever the server is at the origin and there are unserved requests, the algorithm computes a shortest schedule over the set of unserved requests starting and ending at $o$. Then the server starts following the schedule, ignoring temporarily every new request, until it finishes at the origin. Then it waits at the origin until unserved requests are available, as before.

**Theorem 3.6.** *is zealous and* 3*-competitive for the homing online ATSP.*

*Proof.* Let $t$ be the release time of the last request. If $S$ is the schedule that was following at time $t$, we have that finishes following $S$ at time $t' \leq t + \ell(S)$. At that time, it will eventually start again following a schedule over the requests which remain unserved at time $t'$. Let us call $S'$ this other schedule. The total cost paid by will be then at most $t + \ell(S) + \ell(S')$. But $t \leq (\sigma)$, since even the offline adversary cannot serve the last request before it is released, and on the other hand both $S$ and $S'$ have length at most $(\sigma)$, since the offline adversary has to serve all of the requests served in $S$ and $S'$. Thus, $t+\ell(S)+\ell(S') \leq 3 (\sigma)$. □

## 3.5 The nomadic online ATSP

In this section we consider the nomadic version of the online ATSP, in which the server can end its schedule anywhere in the space. We show that no online algorithm can have a constant competitive ratio (that is, independent of the underlying

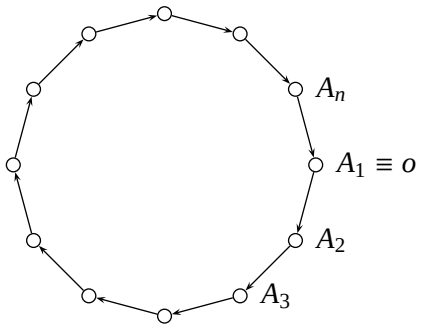


Figure 3.2: The graph used in the proof of Theorem 3.7.

space). Then we show, for spaces with a maximum asymmetry $\psi$, a lower bound $\sqrt{\psi}$ and an upper bound $1 + \sqrt{\psi + 1}$. Note that in the symmetric nomadic online TSP the best lower and upper bounds are 2.03 and $1 + \sqrt{2}$, respectively [75].

**Theorem 3.7.** *For every $L > 0$, there is a space $\mathbb{M} \in \mathcal{M}$ such that the competitive ratio of any online algorithm for nomadic online ATSP on $\mathbb{M}$ is at least $L$.*

*Proof.* For a fixed $\epsilon > 0$, consider the space induced by a directed cycle on $n = \lceil L/\epsilon \rceil$ nodes, where every arc has length $\epsilon$ (Figure 3.2). At time 0 a request is given in node $A_3$. Let $t$ be the first time the online algorithm reaches node $A_2$.

Now if $t \geq 1$, the adversary does not release any other request so that $(\sigma) = 2\epsilon$, $(\sigma) \geq 1 + \epsilon$ and $(\sigma)/(\sigma) \geq \frac{1}{2\epsilon} + \frac{1}{2}$.

Otherwise, if $t \leq 1$, at time $t$ the adversary releases a request at the origin. It is easily seen that $(\sigma) \leq t + 2\epsilon$ and $(\sigma) \geq t + \epsilon(\lceil \frac{L}{\epsilon} \rceil - 1) \geq t + 2\epsilon + L - 3\epsilon$ so that

$$(\sigma)/(\sigma) \geq 1 + \frac{L - 3\epsilon}{t + 2\epsilon} \geq 1 + \frac{L - 3\epsilon}{1 + 2\epsilon}.$$

By taking $\epsilon$ close to zero we see that in the first case the competitive ratio grows indefinitely while in the second case it approaches $L + 1$. □

**Corollary 3.8.** *There is no deterministic online algorithm for nomadic online ATSP on all spaces $\mathbb{M} \in \mathcal{M}$ with a constant competitive ratio.*

We also observe that the same lower bound can be used when the objective function is the sum of completion times.

Thus, we cannot hope for an online algorithm which is competitive for all spaces in $\mathcal{M}$. Indeed, we will now show that the amount of asymmetry of a space is related to the competitive ratio of any online algorithm for that space.

**Theorem 3.9.** *For every $\psi \geq 1$, there is a space $\mathbb{M} \in \mathcal{M}$ with maximum asymmetry $\psi$ such that any deterministic online algorithm for nomadic online ATSP on $\mathbb{M}$ has competitive ratio at least $\sqrt{\psi}$.*

*Proof.* Consider a set of points $\mathbb{M} = \{x_\gamma : \gamma \in [0,1]\}$ with a distance function

$$d(x_\gamma, x_{\gamma'}) = \begin{cases} \gamma' - \gamma & \text{if } \gamma \leq \gamma' \\ \psi(\gamma - \gamma') & \text{if } \gamma \geq \gamma'. \end{cases}$$

The origin is $x_0$. The adversary releases a request at time 1 in point $x_1$. Let $t$ be the time the online algorithm serves this request. If $t \geq \sqrt{\psi}$, no more requests are released and $\quad(\sigma) \geq \sqrt{\psi}, \quad (\sigma) = 1, \quad (\sigma)/\quad (\sigma) \geq \sqrt{\psi}$.

Otherwise, if $t \leq \sqrt{\psi}$, at time $t$ a request is given at the origin. Now $\quad(\sigma) \geq t + \psi, \quad (\sigma) \leq t + 1$ and

$$(\sigma)/\quad (\sigma) \geq \frac{t+\psi}{t+1} = 1 + \frac{\psi - 1}{t+1} \geq 1 + \frac{\psi - 1}{\sqrt{\psi}+1} = \sqrt{\psi}.$$

□

A natural algorithm, along the lines of the best known algorithm for the symmetric version of the problem [75], gives a competitive ratio which is asymptotically the same as that of this lower bound.

**Algorithm 5** -

At any moment at which a new request is released, the server returns to the origin via the shortest path. Once at the origin at time $t$, it computes an optimal schedule $S$ over all requests presented up to time $t$ and then starts following this schedule, staying within distance $\beta t'$ of the origin at any time $t'$, by reducing the speed at the latest possible time.

**Theorem 3.10.** *For every $\psi \geq 1$, there is a value of $\beta$ such that* - *is $(1 + \sqrt{\psi + 1})$-competitive on every space $\mathbb{M} \in \mathcal{M}$ with maximum asymmetry $\psi$.*

*Proof.* There are two cases to be considered. In the first case - does not need to reduce its speed after the last request is released. In this case, if $t$ is the release time of the last request, we have

$$\begin{aligned} - \quad (\sigma) &\leq t + \psi\beta t + \ell(S) \\ &\leq \quad (\sigma) + \psi\beta \quad (\sigma) + \quad (\sigma) = (2 + \psi\beta) \quad (\sigma). \end{aligned}$$

In the second case, let $t$ be the last time - is moving at reduced speed. At that time, - must be serving some request; let $x$ be the location of that request. Since - was moving at reduced speed towards $x$, we must have $d(o, x) = \beta t$; afterwards - will follow the remaining part $S_x$ of the schedule at full speed. Thus

$$- \quad (\sigma) \leq t + \ell(S_x) = (1/\beta)d(o, x) + \ell(S_x).$$

On the other hand, $\quad(\sigma) \geq \ell(S) \geq d(o,x) + \ell(S_x)$. Thus, in this case, the competitive ratio is at most $1/\beta$.

The value $\beta^* = \frac{\sqrt{\psi+1}-1}{\psi}$ minimizes $\max\{2+\psi\beta, 1/\beta\}$ and yields the competitive ratio of the theorem. □

### 3.5.1 Zealous algorithms

Also in the case of the nomadic version of the online ATSP, we wish to consider the performance of zealous algorithms. Of course, no zealous algorithm will be competitive for spaces with unbounded asymmetry. Here we show that the gap between non-zealous and zealous algorithms is much higher than in the homing case, the competitive ratio increasing from $\Theta(\sqrt{\psi})$ to $\Theta(\psi)$.

**Theorem 3.11.** *For every $\psi \geq 1$, there is a space $\mathbb{M} \in \mathcal{M}$ with maximum asymmetry $\psi$ such that the competitive ratio of any zealous deterministic online algorithm for nomadic online ATSP on $\mathbb{M}$ is at least $\frac{1}{2}(\psi+1)$.*

*Proof.* We use the same space used in the proof of Theorem 3.9. At time 0, the adversary releases a request in point $x_1$. The online server will be at point $x_1$ exactly at time 1. Then, at time 1, the adversary releases a request in point $x_0$. It is easy to see that $\quad(\sigma) \geq 1+\psi$, while $\quad(\sigma) = 2$. □

We finally observe that - with $\beta = 1$ is a zealous algorithm for nomadic online ATSP and, by the proof of Theorem 3.10, it has competitive ratio $\psi + 2$.

## 3.6 Polynomial time algorithms

None of the algorithms that we have proposed in the previous sections run in polynomial time, since all of them need to compute optimal schedules on some subsets of the requests. On the other hand, a polynomial time online algorithm with a constant competitive ratio could be used as an approximation algorithm for the ATSP, and thus we do not expect to find one easily. However, our algorithms use offline optimization as a black box and thus can use approximation algorithms as subroutines in order to give polynomial time online algorithms, the competitive ratio depending of course on the approximation ratio. In particular, in the homing version we need to solve instances of the offline ATSP. The best polynomial time algorithm known for this problem has an approximation ratio of $0.842 \log n$ [63]. For the nomadic version, the corresponding offline problem is the shortest asymmetric hamiltonian path, which also admits $O(\log n)$ approximation in polynomial time [35].

We do not repeat here the proofs of our theorems taking into account the approximation ratio of the offline solvers, since they are quite straightforward. However, we give the competitive ratio of our algorithms as a function of $\rho$, the approximation ratio, and $\psi$, the maximum asymmetry of the space, in Table 3.2. Note that,

| Problem | Algorithm | Competitive ratio |
|---|---|---|
| Homing ATSP | | $(1 + 2\rho + \sqrt{1 + 4\rho})/2$ |
| Homing ATSP (zealous) | | $1 + 2\rho$ |
| Nomadic ATSP | - | $(1 + \rho + \sqrt{(1 + \rho)^2 + 4\psi})/2$ |
| Nomadic ATSP (zealous) | - | $1 + \rho + \psi$ |

Table 3.2: The competitive ratio as a function of $\rho$ and $\psi$.

with respect to the values in Table 3.1, the competitive ratio becomes worse by a factor that is strictly less than the approximation ratio. In the case of and - , this is also due to the fact that the algorithms can adapt to the approximation ratio by suitably choosing the parameters $\alpha$ and $\beta$. For , the optimal choice is

$$\alpha^*_\rho = \frac{1}{2\rho}\left(1 + \sqrt{1 + 4\rho}\right),$$

while for - it is

$$\beta^*_\rho = \frac{1}{2\psi}\left[\sqrt{(1 + \rho)^2 + 4\psi} - (1 + \rho)\right].$$

In those special cases in which the ATSP with triangle inequality is approximable within a constant factor in polynomial time, our results yield polynomial time constant competitive algorithms. Notably, these cases include the ATSP with *strengthened triangle inequality* [24].

## 3.7 Conclusions

We have examined some of the online variations of the asymmetric traveling salesman problem. Our results confirm that the asymmetric problems are indeed harder than and not simply extensions of their symmetric counterparts.

The main conclusion is that, as usual in online vehicle routing when minimizing the completion time, waiting can improve the competitive ratio substantially. This is particularly evident in the case of nomadic ATSP on spaces with bounded asymmetry, where zealous algorithms have competitive ratio $\Omega(\psi)$ while - is $O(\sqrt{\psi})$-competitive.

We expect the competitive ratio of the homing online ATSP to be somewhat lower than $1 + \phi$ when the space has bounded asymmetry. Also, since the proof that no online algorithm can have a constant competitive ratio in the nomadic case is also valid when the objective function is the sum of completion times (the traveling repairman problem, see Sections 2.2.3 and 2.3.4), it would be interesting to investigate this last problem in spaces with bounded asymmetry.

Finally, we remark that the existence of polynomial time $O(1)$-competitive algorithms for the homing version is indissolubly tied to the existence of an $O(1)$-approximation algorithm for the offline ATSP.

# Chapter 4

# The online prize-collecting traveling salesman problem

## 4.1 Introduction

In the Traveling Salesman Problem, a salesman has to visit a set of cities to sell his merchandise, and his goal is to minimize the length of the tour. Let us consider the more general case in which each city has both a *penalty* and a *weight* associated with it; now the commitment of the salesman is to collect a given *quota* of weights, by visiting a sufficient number of cities; the final cost will be the length of the tour plus the penalty of every city that was not visited. This problem is known as the *Prize-Collecting Traveling Salesman Problem* (PCTSP) [21]. If all the penalties are equal to zero, then the PCTSP reduces to the special case known as the Quota Traveling Salesman Problem [16, 19, 27], also called sometimes the Quorum-Cast problem [37]. On the other hand, when the quota is zero, i.e. there is no requirement to visit any city at all, we call the resulting problem the Penalty Traveling Salesman Problem[1]. Related to the PCTSP is also the so called $k$-TSP problem, i.e. the problem of finding a tour of minimum length which visits $k$ cities among the given ones. Moreover, the problem is also related to the $k$-MST problem, that is the problem of finding a minimum weight tree which spans $k$ nodes in a graph. Thus, the PCTSP generalizes a number of interesting routing problems.

In this chapter we address the online version of the PCTSP, in which the requests arrive over time in a metric space and a server (the traveling salesman) has to decide which requests to serve and in what order to serve them, without yet knowing the whole sequence of requests; the goal is, as in the offline PCTSP, to collect the quota while minimizing the sum of the time needed to complete the tour and the penalties associated to the requests not in the tour. We study the online PCTSP in the usual framework of *competitive analysis*, where the performance of the online algorithm is matched against the performance of an optimum offline server (see

[1] Although the name Penalty TSP is not standard in the literature, we use it to avoid ambiguity, since some authors use the name PCTSP to refer to this special case.

Chapter 1 for details).

The rest of the chapter is structured as follows. We begin by providing a formal definition of the offline problem and an overview of approximation results in Section 4.2. We introduce the online model of the problem in Section 4.3. In Section 4.4, we observe that a lower bound on the competitive ratio of any algorithm for the online PCTSP is 2 and we give a $7/3$-competitive algorithm. In Section 4.5 we give lower and upper bounds for the special case when the underlying metric space is the halfline. Finally, we give our conclusions in Section 4.6.

## 4.2 The offline PCTSP

We begin by formally defining the offline PCTSP.

**Definition 4.1.** An instance of the *Prize-Collecting Traveling Salesman Problem*, is given by a metric $d$ over a finite space $\{1, \ldots, n\}$. Moreover, the instance specifies $2n + 1$ nonnegative integers $Q, w_1, \ldots, w_n, \pi_1, \ldots, \pi_n$. A feasible solution is given by a *tour*, that is a cyclic permutation $\varphi$ on some set $S \subseteq \{1, \ldots, n\}$ such that $1 \in S$ and $\sum_{i \in S} w_i \geq Q$. The cost of the solution is $c(\varphi) = \ell(\varphi) + \pi(\varphi)$ where $\ell(\varphi) = \sum_{i \in S} d_{i\varphi(i)}$ and $\pi(\varphi) = \sum_{i \notin S} \pi_i$.

Several other problems can be defined in terms of the PCTSP.

**Definition 4.2.** An instance of the *Penalty Traveling Salesman Problem* is a PCTSP instance in which $Q = 0$ and $w_i = 0$ for all $i \in \{1, \ldots, n\}$. Feasible solutions and costs are defined exactly as in the PCTSP.

**Definition 4.3.** An instance of the *Quota Traveling Salesman Problem* is a PCTSP instance in which $\pi_i = 0$ for all $i = \{1, \ldots, n\}$. Feasible solutions and costs are defined exactly as in the PCTSP.

**Definition 4.4.** An instance of the *k-Traveling Salesman Problem* is a PCTSP instance in which $\pi_i = 0$ and $w_i = 1$ for all $i \in \{1, \ldots, n\}$. The quota $Q$ is customarily denoted by $k$. Feasible solutions and costs are defined exactly as in the PCTSP.

Notice that the standard Traveling Salesman Problem is a special case of both the Penalty TSP (by letting every penalty be sufficiently high) and of the $k$-TSP (when $k = n$). Thus, the TSP is a special case of the Prize-Collecting TSP in *two* different ways. The lattice of relations between all these problems is given in Figure 4.1, where an arrow from problem $A$ to problem $B$ means that $A$ is a special case of $B$.

In the general form that we have described, the PCTSP was first formulated by Balas [21], who gave structural properties of the PCTSP polytope as well as heuristics. The problem arose during the task of developing daily schedules for a steel rolling mill.

The only results on guaranteed heuristics for the PCTSP are due to Awerbuch et al. [19]. They give a polynomial time algorithm with an approximation ratio of

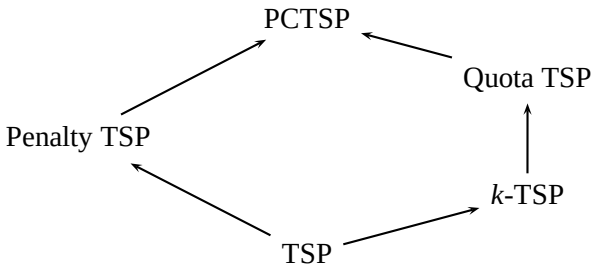


Figure 4.1: Relations between the TSP, the $k$-TSP, the Quota TSP, the Penalty TSP and the PCTSP.

$O((\log \min(Q, n))^2)$, where $n$ is the number of cities and $Q$ is the required quota. However, the PCTSP contains as special cases both the Penalty TSP and the $k$-TSP, which received more attention in the literature.

The Penalty TSP has been considered by Bienstock et al. [23], who give a $5/2$-approximation algorithm. A better result, due to Goemans and Williamson [53], is the following.

**Theorem 4.1** ([53])**.** *The Penalty TSP admits a* 2*-approximation algorithm.*

For the $k$-TSP, the best bound to date is a 2-approximation due to Garg [51]. Apart from the result of Awerbuch et al. [19] on the more general PCTSP, the Quota TSP was not addressed directly in the literature. However, a constant approximation easily follows from a result of Ausiello et al. [18].

**Theorem 4.2** ([18])**.** *The Quota TSP admits a* 10*-approximation algorithm.*

*Proof.* Ausiello et al. [18] give a 5-approximation for the Quota Minimum Spanning Tree problem. A 10-approximation for the Quota TSP easily follows by the standard technique of doubling a tree obtained by that algorithm and constructing a tour out of an Eulerian closed walk of the Eulerian multigraph thus obtained. □

From these results for the Penalty TSP and the Quota TSP, we show in the following how a technique used by Awerbuch et al. [19] allows to derive a constant approximation algorithm for the PCTSP.

**Theorem 4.3.** *The PCTSP admits a constant approximation algorithm.*

*Proof.* A simple idea exploited by the algorithm of Awerbuch et al. [19] is that, for a given instance $I$ of the PCTSP, the following quantities constitute lower bounds on the cost of an optimal solution:

1. the cost $_q$ of an optimal solution to a Quota TSP instance $I_q$ defined on the same graph and having the same weights and quota as in $I$ (since every feasible solution to $I$ can be turned into a feasible solution to $I_q$ of at most the same cost).

2. the cost $\quad_p$ of an optimal solution to a Penalty TSP instance $I_p$ defined on the same graph and having the same penalties as in $I$ (since a feasible solution to $I$ is also feasible for $I_p$ and has the same cost);

Thus, to approximate an optimal solution to the PCTSP instance $I$ we can:

1. Run a $\rho_q$-approximation algorithm for Quota TSP on $I_q$ to obtain a tour $T_q$ such that $\ell(T_q) \leq \rho_q \cdot \quad_q$. While $T_q$ is feasible for the PCTSP instance $I$, its cost might be high when we take into account penalties.

2. Run a $\rho_p$-approximation algorithm for Penalty TSP on $I_p$ to obtain a tour $T_p$ such that $c(T_p) \leq \rho_p \cdot \quad_p$. Notice that this tour $T_p$ might not be feasible for $I$.

3. Concatenate $T_q$ and $T_p$ to obtain a tour $T$ feasible for the PCTSP instance $I$ and of cost

$$c(T) \leq \ell(T_q) + c(T_p) \leq \rho_q \cdot \quad_q + \rho_p \cdot \quad_p \leq (\rho_q + \rho_p) \cdot \quad .$$

This means that, by using the algorithms of Theorems 4.1 and 4.2, we can obtain a constant-factor approximation to the Prize-Collecting TSP. □

## 4.3 The online model

Let us define the online PCTSP in a formal way. Let $\mathbb{M}$ be a metric space, with a distinguished point $o$ called the *origin*. As is customary (see Section 2.3.1), we only require $\mathbb{M}$ to be continuous and path-metric; for example, any weighted graph induces a space satisfying this requirements. We will also look at the special case of the halfline $\mathbb{R}_+$, with the origin at zero.

The input is given by a pair $(Q, \sigma)$, where $Q \in \mathbb{Q}_+$ is called the *quota* and $\sigma = \sigma_1 \cdots \sigma_n$ is a sequence of *requests*. Every request $\sigma_i$ is a quadruple $(r_i, x_i, w_i, \pi_i)$, where $r_i \in \mathbb{R}_+$ is the *release date* of the request, $x_i \in \mathbb{M}$ its *location*, $w_i \in \mathbb{Q}_+$ its *weight* and $\pi_i \in \mathbb{Q}_+$ its *penalty*. We also assume that the sequence is ordered such that $i < j$ implies $r_i \leq r_j$. All the information about a request, including its existence, becomes known only at its release date. On the other hand, the quota is revealed immediately to the algorithm.

The algorithm controls a single *server* (traveling salesman), initially located in the origin. The server can move around the space at speed at most 1. To *serve* a request, the server must visit the location of the request not earlier than its release date.

A feasible solution for instance $(Q, \sigma)$ is a *schedule*, that is, a sequence of moves of the server such that the following conditions are satisfied: (1) the server starts in the origin, (2) the total weight of the served requests is at least $Q$, and (3) the server ends in the origin. Let $S$ be a schedule for $(Q, \sigma)$. The time at which the server returns permanently at the origin is called the *makespan* of the schedule

| Problem | Lower Bound | Upper Bound | References |
|---|---|---|---|
| PCTSP | 2 | 7/3 | Th. 4.4, 4.8 |
| PCTSP ($\mathbb{R}_+$) | 1.89 | 2 | Th. 4.10, 4.13 |

Table 4.1: The competitive ratio of prize-collecting traveling salesman problems.

and we denote it by $m(S)$. The sum of the penalties of the requests which were not served is denoted by $\pi(S)$. The objective of the online PCTSP is to construct online a schedule $S$ for the given instance that minimizes $m(S) + \pi(S)$.

In the following, we denote by $(Q, \sigma)$ and $(Q, \sigma)$ the total cost incurred by an online and an optimal offline server respectively on input $(Q, \sigma)$.

For a sequence $\sigma$, we let $\pi(\sigma) = \sum_{\sigma_i \in \sigma} \pi_i$. By $\sigma^{\leq t}$ we denote the subsequence of $\sigma$ including all the requests having release date less than or equal to $t$, and similarly $\sigma^{>t}$ is the suffix of $\sigma$ consisting of all the requests having release date strictly larger than $t$. We also denote by $(t)$ the total cost incurred by the optimal offline server over the sequence $\sigma^{\leq t}$, i.e., $(Q, \sigma^{\leq t})$. When $\sigma^{\leq t}$ has no feasible solution (that is, $\sum_{r_i \in \sigma^{\leq t}} w_i < Q$), we conventionally define $(t) = \infty$. Note that according to these definitions we have $(Q, \sigma) = (r_n)$. Finally, we use $S^*(t)$ to denote an optimal offline schedule for $(Q, \sigma^{\leq t})$. Thus, $(t) = m(S^*(t)) + \pi(S^*(t))$. A summary of our results is contained in Table 4.1.

## 4.4 A competitive algorithm for the online PCTSP

In this section we give a 7/3-competitive algorithm for the online PCTSP. Notice that a lower bound of 2 on the competitive ratio of the online PCTSP is inherited from simpler problems.

**Theorem 4.4.** *The competitive ratio of any deterministic online algorithm for the online PCTSP is at least 2, even if $Q = 0$ or if $\pi_i = 0$ for all requests $\sigma_i$.*

*Proof.* When $Q = 0$, we can give requests with arbitrarily high penalties so that the problem becomes equivalent to the standard online TSP, for which a lower bound of 2 is known [17].

When $\pi_i = 0$ for all requests $r_i$, the problem is equivalent to the online Quota TSP, for which a lower bound of 2 is also known, even for fixed $Q$ [16]. □

In order to get our competitiveness result, we begin by proving a lemma on the properties of $(t)$ as a function of $t$. Notice that the fact that a request is served in the optimal solution for a sequence $\sigma$ does not imply that the request is served in an optimal solution for $\sigma^{\leq t}$; this is why the following lemma is non-trivial.

**Lemma 4.5.** *Consider an instance $(Q, \sigma)$ of the online PCTSP and let $\sigma_l$ be a request with maximum release date among the requests served in an optimal offline solution. Then*

*(a)* $(t) + \pi(\sigma^{>t}) =$ $(r_n)$ *for all* $t \in [r_l, r_n]$;

*(b)* $(r_l) \geq r_l$.

*Proof.* (a) Since a feasible solution for $\sigma$ is to first serve optimally $\sigma^{\leq t}$ and then pay penalties for all successive requests, we have

$$(r_n) \leq \quad (t) + \pi(\sigma^{>t}). \tag{4.1}$$

On the other hand, a feasible solution for $\sigma^{\leq t}$ is to follow the optimal offline schedule for $\sigma$, of course saving on the penalty cost of the requests released after time $t$, so we have

$$(t) \leq \quad (r_n) - \pi(\sigma^{>t}). \tag{4.2}$$

The claim follows by combining (4.1) and (4.2).

(b) Assume by contradiction that $(r_l) < r_l$. By (a),

$$\begin{aligned} (r_n) &= (r_l) + \pi(\sigma^{>r_l}) \\ &< r_l + \pi(\sigma^{>r_l}). \end{aligned}$$

But since there is an optimal schedule for $\sigma$ that serves $\sigma_l$ and no request with a release date later than $r_l$,

$$(r_n) \geq r_l + \pi(\sigma^{>r_l})$$

which gives a contradiction.

□

Consider the following algorithm for the online PCTSP (Algorithm 6).

**Algorithm 6** - Wait and Go with Restart

The algorithm has two states, WAIT and GO. The initial state is WAIT.

- WAIT. Wait until a time $t$ such that $(t) = t$, then enter state GO.
- GO. Let $t_g$ be the time this state was entered. Return the server to the origin at full speed, then start following schedule $S^*(t_g)$; when done, return to state WAIT. Meanwhile, if a new request arrives at time $t$, compute $(t)$. If $(t) \geq t$, make a *restart*, that is, stop the server at its current location and return to state WAIT.

The intuition behind the algorithm is the following. In order to be competitive, the algorithm tries to guess which requests are ignored and which are served by the

optimal offline server. The condition $(t) \geq t$ is used to ensure that if a restart occurs, only a short time has elapsed, compared to the optimal cost, while if a restart does not occur, the new requests can be safely ignored.

To move from intuition to proof, we need the following important property of : after the algorithm restarts, it eventually enters state GO another time. This property is non-trivial, because $(t)$ can be a discontinuous function of $t$. We formalize the property in the following lemma.

**Lemma 4.6.** *If enters state WAIT at time $t_w$ and $(t_w) \geq t_w$, then at some time $t_g \geq t_w$ enters state GO.*

*Proof.* If no requests are released after time $t_w$, for any $t \geq t_w$ we have $(t) = (t_w)$ and thus at time $t_g = (t_w) \geq t_w$ enters state GO.

Otherwise, consider the sequence $\sigma^{>t_w}$. If no request in $\sigma^{>t_w}$ is served in an optimal solution for $\sigma$, by Lemma 4.5(a),

$$(t) = (t_w) + \pi(\sigma^{>t_w}) - \pi(\sigma^{>t})$$

for any $t \in [t_w, r_n]$. Thus, for $t \geq t_w$, $(t)$ can only increase as a function of $t$ (until time $r_n$), thus at some time $t_g$ it must intersect the identity function, since $\sigma$ has finite length.

The last case to consider is when some request in $\sigma^{>t_w}$ is served in an optimal solution for $\sigma$. Let $\sigma_l$ be a request with maximum release date among such requests. Then by Lemma 4.5(b), $(r_l) \geq r_l$ and since $(t)$ can only increase for $t \in [r_l, r_n]$, at some time $t_g$ it will intersect the identity function. □

We need a last ingredient in order to prove the competitiveness of .

**Lemma 4.7.** *At any time $t$, the server controlled by is at most at distance $t/3$ from the origin.*

*Proof.* We prove the claim by induction on the sequence of states entered by the algorithm. The claim is trivially true at time 0. Also, by induction, it is true while the algorithm is in state WAIT since in that state the server does not move.

When the algorithm enters state GO, the server is, by inductive hypothesis, at some distance $d_g \in [0, t_g/3]$ from the origin. first has to move the server to the origin; if it changes state before arriving, then the claim is true. Otherwise, at time $t_g + d_g$, starts following schedule $S^*(t_g)$, which takes time at most $(t_g) = t_g$, so at any later moment $t_g + d_g + \Delta$ the server cannot be at a distance greater than $\min\{\Delta, t_g/2\}$ from the origin. Now $\min\{\Delta, t_g/2\} \leq (t_g + \Delta)/3 \leq (t_g + d_g + \Delta)/3$, which proves the lemma. □

**Theorem 4.8.** *is 7/3-competitive for online PCTSP.*

*Proof.* Let $t_g = (t_g)$ be the last time state GO was entered (if state GO is never entered, is easily seen to be optimal). According to Lemma 4.7, the online server is at distance at most $t_g/3$ from the origin. Then the server will pay, for

the sequence $\sigma^{\leq t_g}$, at most $t_g + t_g/3 +$ $(t_g) = (7/3)$ $(t_g)$. Notice that this also includes the penalties of all requests in $\sigma^{\leq t_g}$ that were not served by $S^*(t_g)$. Moreover, the last request $\sigma_l$ served by an optimal offline solution must have been released at a time $r_l \leq t_g$, otherwise, since by Lemma 4.5(b) $(r_l) \geq r_l$, would have restarted and by Lemma 4.6 state GO would have been entered one more time.

The online server also pays the penalties of all the requests released after time $t_g$, but notice that these penalties are also paid by the optimal solution. More specifically,

$$(Q, \sigma) \leq \frac{7}{3} \quad (t_g) + \pi(\sigma^{>t_g})$$

while by Lemma 4.5(a),

$$(Q, \sigma) = \quad (r_l) + \pi(\sigma^{>r_l}) = \quad (t_g) + \pi(\sigma^{>t_g}).$$

The competitive ratio is then at most $7/3$. □

To see that the ratio is tight, consider the real line with $Q = 1$. Fix some sufficiently large $B$ and sufficiently small $\epsilon$. At time $r_1 = 1$ a new request having $x_1 = 1$, $w_1 = 1$ and $\pi_1 = B$ arrives. waits until time 2, then begins moving towards the request. At time $r_2 = 3 - \epsilon$ a new request arrives having $x_2 = -\epsilon$, $w_2 = 1$ and $\pi_2 = B$; no more requests arrive so $(Q, \sigma) =$ $(r_2) = 3$. When the second request arrives, (while the server is in $1 - \epsilon$) recomputes $(r_2)$ and makes a restart since $(r_2) > r_2$. At time $r_2 + 1 = 4 - \epsilon$ reaches $o$ and then starts a schedule which ends at time $4 - \epsilon + 3 = 7 - \epsilon$. For $\epsilon$ approaching 0 the competitive ratio approaches $7/3$.

## 4.5 The online PCTSP on the halfline

In this section we consider the online PCTSP when the underlying metric space is the real halfline. For this case, we prove a lower bound of 1.89 on the competitive ratio of any algorithm and we give a 2-competitive online algorithm.

### 4.5.1 Lower bound

**Lemma 4.9.** *No competitive algorithm for online PCTSP with $Q = 0$ can leave the origin until at least one request with positive penalty has been released.*

*Proof.* Suppose that the online server is not at the origin at some time $t > 0$. If no requests at all are released, the optimal cost is zero while the online algorithm pays at least $t > 0$. Even if we insist that at least one request should be released, the same result follows by considering a single request with a penalty that approaches zero. □

**Theorem 4.10.** *The competitive ratio of any deterministic online algorithm for the online PCTSP on the halfline is at least* $(3 + \sqrt{21})/4 \approx 1.89$.

*Proof.* In what follows, let $\gamma = (3 + \sqrt{21})/4$. Since we assume that $Q = 0$, the weights of the requests will not be relevant for the proof. Using Lemma 4.9 we can assume that the server will remain at the origin until the first request is released in $x_1 = 1$ at time $r_1 = 1$. This request has penalty $\pi_1 = 3/\gamma$. If the online algorithm eventually serves $\sigma_1$, then obviously

$$\begin{aligned} (Q,\sigma) &\geq 3, \\ (Q,\sigma) &\leq \pi_1, \\ \frac{(Q,\sigma)}{(Q,\sigma)} &\geq \frac{3}{\pi_1} = \gamma. \end{aligned}$$

Otherwise, let $t_c$ be the makespan of the online algorithm on $\sigma^{\leq r_1}$. If $t_c \geq 3 - \pi_1$, we have

$$\frac{(Q,\sigma)}{(Q,\sigma)} \geq \frac{3 - \pi_1 + \pi_1}{\pi_1} \geq \frac{3}{\pi_1} = \gamma.$$

Finally, if $t_c < 3 - \pi_1$, at time $r_2 = 1 + \frac{\pi_1}{2}$ ($\geq t_c$), while the online server is at the origin, the adversary releases a new request in $x_2 = 2 - r_2$ with a very high penalty. Thus this request $\sigma_2$ must be necessarily served for the algorithm to be competitive.

If the online algorithm only serves $\sigma_2$, we have

$$(Q,\sigma) \geq r_2 + 2x_2 + \pi_1 = 4 - r_2 + \pi_1 = 3 + \frac{\pi_1}{2}.$$

On the other hand, if the algorithm serves both $\sigma_1$ and $\sigma_2$, the cost will be

$$(Q,\sigma) \geq r_2 + 2x_1 = 1 + \pi_1/2 + 2 = 3 + \frac{\pi_1}{2}.$$

Thus, regardless of the behavior of the algorithm, a lower bound on the online cost is $3+\pi_1/2$. Instead, the optimal offline solution is to serve both requests by moving immediately to 1, serving $\sigma_1$ as soon as it is released, and serving $\sigma_2$ on the way back to the origin. The cost of this solution is 2. Thus, the competitive ratio is at least

$$\frac{1}{2}\left(3 + \frac{\pi_1}{2}\right) = \gamma.$$

□

### 4.5.2 Upper bound

The algorithm we consider in this section is a variation of Algorithm 6 designed specifically for the halfline metric space. For the analysis of the competitive ratio, it will be useful to make the further assumption that, among the optimal schedules, we consider one that minimizes the makespan.

We can now state the algorithm (Algorithm 7).

In the competitiveness proof for we will need the following two lemmas, which strengthen Lemma 4.5 and 4.6, respectively.

**Algorithm 7** - Wait and Go on the Halfline

The algorithm has two states, WAIT and GO. The initial state is WAIT.

- WAIT. Wait until a time $t$ such that $t = m(S^*(t))$, then enter state GO.
- GO. Let $t_g$ be the time this state was entered. Let $x_*$ be the point of $S^*(t_g)$ which is farthest from the origin. Move the server to $x_*$ at full speed, then return it to the origin, while serving every request whose location is visited; when done, return to state WAIT. Meanwhile, if a new request arrives at time $t$, compute $m(S^*(t))$; if $m(S^*(t)) \geq t$, stop the server at its current location and return to state WAIT.

**Lemma 4.11.** *Consider an instance $(Q, \sigma)$ of the online PCTSP and let $\sigma_l$ be a request with maximum release date among the requests served by an optimal offline solution. Then*

*(a)* $m(S^*(t)) = m(S^*(r_l))$ *for all* $t \in [r_l, r_n]$*;*

*(b)* $m(S^*(r_l)) \geq r_l$.

*Proof.* (a) First, notice that

$$m(S^*(r_l)) \geq m(S^*(t)) \text{ for all } t \in [r_l, r_n]. \tag{4.3}$$

Otherwise, by using schedule $S^*(r_l)$ on $\sigma^{\leq t}$, we could achieve a shorter makespan while not increasing the total cost which, by Lemma 4.5(a), is $(r_l) + \pi(\sigma^{>r_l}) - \pi(\sigma^{>t})$. By a similar argument,

$$m(S^*(t)) \geq m(S^*(r_n)) \text{ for all } t \in [r_l, r_n]. \tag{4.4}$$

Finally,

$$m(S^*(r_n)) \geq m(S^*(r_l)) \tag{4.5}$$

as if that was not the case, since schedule $S^*(r_n)$ does not serve any request released later than $r_l$, by using it on $\sigma^{\leq r_l}$ we could achieve a shorter makespan while not increasing the total cost which, by Lemma 4.5(a), is $(r_n) - \pi(\sigma^{>r_l})$. The claim follows by combining (4.3), (4.4) and (4.5), as equality must hold throughout.

(b) Follows from (a) and the fact that $S^*(r_n)$ serves $\sigma_l$ and thus must have makespan at least $r_l$.

□

**Lemma 4.12.** *If enters state WAIT at time $t_w$ and $m(S^*(t_w)) \geq t_w$, then at some time $t_g \geq t_w$ enters state GO.*

*Proof.* If no requests are released after time $t_w$, for any $t \geq t_w$ we have $m(S^*(t)) = m(S^*(t_w))$ and thus at time $t_g = m(S^*(t_w)) \geq t_w$ enters state GO.

Otherwise, consider the sequence $\sigma^{>t_w}$. If no request in $\sigma^{>t_w}$ is served in an optimal solution for $\sigma$, by Lemma 4.11(a),

$$m(S^*(t)) = m(S^*(t_w))$$

for any $t \in [t_w, r_n]$. Thus, for $t \geq t_w$, $m(S^*(t))$ remains constant as a function of $t$, and the claim follows with $t_g = m(S^*(t_w))$.

The last case to consider is when some request in $\sigma^{>t_w}$ is served in an optimal solution for $\sigma$. Let $\sigma_l$ be a request with maximum release date among such requests. Then by Lemma 4.11(b), $m(S^*(r_l)) \geq r_l$ and since by Lemma 4.11(a) $m(S^*(t))$ remains constant for $t \geq r_l$, the claim follows with $t_g = m(S^*(r_l))$. □

**Theorem 4.13.** *is 2-competitive for online PCTSP on the halfline.*

*Proof.* Let $\sigma_l$ be a request that, among the ones served by the optimal offline solution, has maximum release date. By Lemma 4.11(a),

$$m(S^*(r_l)) = m(S^*(r_n)) \geq r_l,$$

thus from Lemma 4.12, eventually after time $r_l$ the algorithm enters state GO. Let $t_g (\geq r_l)$ be the last time state GO is entered. From Lemma 4.5(a) we have

$$(r_n) = \quad (t_g) + \pi(\sigma^{>t_g}). \tag{4.6}$$

Recall that $x_*$ is the point of the optimal schedule $S^*(t_g)$ which is farthest from the origin, thus

$$m(S^*(t_g)) \geq 2x_*. \tag{4.7}$$

Also,

$$m(S^*(t_g)) = t_g, \tag{4.8}$$

because state GO was entered at time $t_g$.

Let $s(t_g)$ be the position of at time $t_g$. If $s(t_g) \leq x_*$, we have

$$\begin{aligned}
(Q, \sigma) &\leq t_g + 2x_* + \pi(S^*(t_g)) + \pi(\sigma^{>t_g}) \\
&\leq m(S^*(t_g)) + m(S^*(t_g)) + \pi(S^*(t_g)) + \pi(\sigma^{>t_g}) && \text{by (4.7),(4.8)} \\
&= m(S^*(t_g)) + \quad (r_n) && \text{by (4.6)} \\
&\leq 2 \quad (r_n) && \text{by (4.6)} \\
&= 2 \quad (Q, \sigma).
\end{aligned}$$

If $s(t_g) \geq x_*$, we have

$$\begin{aligned}
(Q, \sigma) &\leq t_g + s(t_g) + \pi(S^*(t_g)) + \pi(\sigma^{>t_g}) \\
&\leq 2t_g + \pi(S^*(t_g)) + \pi(\sigma^{>t_g}) \\
&\leq 2m(S^*(t_g)) + \pi(S^*(t_g)) + \pi(\sigma^{>t_g}) && \text{by (4.8)} \\
&= m(S^*(t_g)) + \quad (r_n) && \text{by (4.6)} \\
&\leq 2 \quad (r_n) && \text{by (4.6)} \\
&= 2 \quad (Q, \sigma).
\end{aligned}$$

□

## 4.6 Conclusions

We formulated an online version of the Prize-Collecting Traveling Salesman Problem. For general spaces, we gave a $7/3$-competitive algorithm which is not far from best possible, the competitive ratio of any online algorithm being at least 2. We also discussed the special case of the halfline, for which we gave a 1.895 lower bound and a 2-competitive algorithm.

Other than improving the bounds, an interesting open problem is to develop a competitive algorithm for the online PCTSP that runs in polynomial time. The standard technique of combining the online algorithm with an offline approximation subroutine does not seem to work directly here, as that requires the subroutine to be used as a black box, while our analysis relied more than once on the properties of the optimal solution. Thus, a different approach may be required.

# Chapter 5

# Online $k$-server routing problems

## 5.1 Introduction

In a $k$-server routing problem, $k$ servers (vehicles) move in a metric space in order to visit a set of points (cities). Given a schedule, that is, a sequence of movements of the servers, the time at which a city is visited for the first time by one of the servers is called the *completion time* of the city. The objective is to find a schedule that minimizes some function of the completion times.

We study $k$-server routing problems in their *online* version. Prior to this work, there was essentially no result on online multiserver routing problems, except for some isolated algorithms [11, 25]. We give competitive algorithms and negative results for online multiserver routing problems, with the objective of minimizing either *makespan* or *total completion time*. In the case of makespan we consider the variant known as *nomadic*, in which the servers are not required to return at the origin after serving all requests; the above cited previous results apply to the other variant, known as the *homing* traveling salesman problem. Apart from being the first work dedicated to multiserver online routing problems, the results are somewhat unexpected. We give the first results of online problems for which multiple server versions admit lower competitive ratios than their single server counterparts. This is typically not the case for problems in the one-by-one model; for example, it is known that in the famous *k-server* problem [78] the competitive ratio necessarily grows linearly with $k$.

It may also be useful to draw a comparison with machine scheduling, which is closer to routing problems in many ways. In scheduling a lot of research has been conducted to online multiple machine problems [85]. In the one-by-one model competitive ratios increase with increasing number of machines. In real time online scheduling nobody has been able to show smaller competitive ratios for multiple machine problems than for the single machine versions, though here lower bounds do not exclude that such results exist (and indeed people suspect they do) [36, 41].

The rest of this chapter is structured as follows. We discuss the approximability of offline $k$-server problems in Section 5.2. Then, after introducing the online

model in Section 5.3, we give in Section 5.4 competitive algorithms and lower bounds for both the $k$-Traveling Salesman and the $k$-Traveling Repairman in general spaces. In Section 5.5 we show that in the case of the real line we have an almost optimal algorithm for large $k$. The same result cannot hold in the Euclidean plane, as we show in Section 5.6. We give our conclusions in Section 5.7.

## 5.2 Offline $k$-server routing problems

Routing problems with multiple servers have been studied often in the literature. Here we focus on approximation results related to the problems that we will study in the online setting. We begin by defining the $k$-server version of the Traveling Salesman Problem.

**Definition 5.1.** An instance of the *$k$-Traveling Salesman Problem*[1] is given by a metric $d$ over a finite space $\{1,\ldots,n\}$. A feasible solution is given by a *$k$-tour*, that is a set of $k$ cyclic permutations $\varphi_1,\ldots,\varphi_k$ where $\varphi_j : X_j \to X_j$ and the $X_j \subseteq \{1,\ldots,n\}$ are such that $1 \in X_1 \cap \ldots \cap X_k$ and $X_1 \cup \ldots \cup X_k = \{1,\ldots,n\}$. The cost of a solution is

$$c(\varphi_1,\ldots,\varphi_k) = \max_{j\in\{1,\ldots,k\}} \sum_{i=1}^{|X_j|} d(\varphi_j^{(i-1)}(1), \varphi_j^{(i)}(1)).$$

Frederickson et al. [48] consider the $k$-TSP and give an approximation preserving reduction from the $k$-TSP to the standard TSP.

**Theorem 5.1** ([48])**.** *The $k$-TSP admits a $(\rho + 1 - 1/k)$-approximation algorithm, where $\rho$ is the ratio of any approximation algorithm for the TSP.*

By using Christofides' heuristic [38] one obtains the following result, that has not been improved yet.

**Corollary 5.2.** *The $k$-TSP admits a $(5/2 - 1/k)$-approximation algorithm.*

We now consider the multiple server version of the Traveling Repairman Problem.

**Definition 5.2.** An instance of the *$k$-Traveling Repairman Problem* is given by a metric $d$ over a finite space $\{1,\ldots,n\}$. A feasible solution is given by a *$k$-tour*, that is a set of $k$ cyclic permutations $\varphi_1,\ldots,\varphi_k$ where $\varphi_j : X_j \to X_j$ and the $X_j \subseteq \{1,\ldots,n\}$ are such that $1 \in X_1 \cap \ldots \cap X_k$ and $X_1 \cup \ldots \cup X_k = \{1,\ldots,n\}$. The cost of a solution is

$$c(\varphi_1,\ldots,\varphi_k) = \sum_{j=1}^{k} \sum_{i=1}^{|X_j|} \sum_{l=1}^{i-1} d(\varphi_j^{(l-1)}(1), \varphi_j^{(l)}(1)).$$

[1] It should be noted that this is a different problem from the TSP with $k$ *clients* defined in Chapter 4. Unfortunately, the name $k$-TSP is standard in the literature for both problems. In this chapter we only consider the TSP with $k$ *servers*.

The first approximation algorithm for the $k$-TRP has been given by Fakcharoenphol et al. [44]. It has an approximation ratio of $8.497\rho$, where $\rho$ is the best approximation factor for the problem of finding the least cost rooted tree spanning $i$ vertices ($i$-MST). The currently best known factor for the $i$-MST is 2 [51], which gives an approximation ratio of 16.994 for the $k$-TRP. This ratio has been improved by Chaudhuri et al. [33], who reduce it to 8.497, which is currently the best known factor for the problem.

**Theorem 5.3** ([33])**.** *The $k$-TRP admits a* 8.497*-approximation algorithm.*

Jothi and Raghavachari [60] consider a generalization of the $k$-TRP in which nodes have associated repair times. They present a $\rho + 2$-approximation algorithm for this problem, where $\rho$ is the best approximation factor obtainable for the basic $k$-Traveling Repairman Problem. Chekuri and Kumar [34] consider instead a variant of the $k$-TRP in which different servers may have different departure points, or depots. They give a 24-approximation algorithm for this problem. Chaudhuri et al. [33] improve this factor to 12.

To our knowledge, a multiple server version of the Wandering Salesman Problem (defined in Chapter 2) has never been considered in the literature from the point of view of approximation algorithms.

## 5.3 The online model

The model we use is an extension of the single-server online routing model introduced in Section 2.3. Here we only highlight the differences.

A $k$-server routing algorithm controls $k$ vehicles or *servers*. Initially, at time 0, all these servers are located in the same point $o$ of the metric space $\mathbb{M}$. The algorithm can then move the servers around the space at speed at most 1. (We do not consider the case in which servers have different maximum speeds; in compliance with machine scheduling vocabulary we could say that the servers are identical and work in parallel.) As in the single-server framework, we consider continuous path-metric spaces.

Defining the *completion time* of a request as the time at which the request has been served, the *nomadic $k$-traveling salesman problem* ($k$-TSP) has objective minimizing the *maximum completion time*, the *makespan*, and the *$k$-traveling repairman problem* ($k$-TRP) has objective minimizing the *total completion time*.

We use $s_1, \ldots, s_k$ to denote the $k$ servers, and write $s_j(t)$ for the position of server $s_j$ at time $t$, and $d_j(t)$ for $d(s_j(t), o)$. Finally, given a path $P$ in $\mathbb{M}$, we denote its length by $\ell(P)$.

All the lower bounds we prove hold for randomized algorithms against an oblivious adversary. In order to prove these results, we frequently resort to Yao's principle (cf. Section 1.6).

A summary of our results is presented in Tables 5.1 and 5.2, which also report the known results for the *homing* $k$-traveling salesman problem, the multiserver

| Problem | Lower Bound | Upper Bound | References |
|---|---|---|---|
| Homing $k$-TSP | 2 | 2 | [11, 76] |
| Nomadic $k$-TSP | 2 | $1+\sqrt{2}$ | Th. 5.11, 5.6 |
| $k$-TRP | 2 | $(1+\sqrt{2})^2$ | Th. 5.11, 5.10 |
| Homing $k$-TSP ($\mathbb{R}$) | 3/2 | 3/2 | [25] |
| Nomadic $k$-TSP ($\mathbb{R}$) | $1+\Omega(1/k)$ | $1+O((\log k)/k)$ | Th. 5.16, 5.12 |
| $k$-TRP ($\mathbb{R}$) | $1+\Omega(1/k)$ | $1+O((\log k)/k)$ | Th. 5.16, 5.12 |

Table 5.1: The competitive ratio of multiserver routing problems.

| Problem | Lower Bound | References |
|---|---|---|
| Homing $k$-TSP ($\mathbb{R}^2$) | 3/2 | [25] |
| Nomadic $k$-TSP ($\mathbb{R}^2$) | 4/3 | Th. 5.17 |
| $k$-TRP ($\mathbb{R}^2$) | 5/4 | Th. 5.18 |

Table 5.2: Lower bounds in the Euclidean plane.

generalization of the homing TSP defined in Section 2.3.2 (in Table 5.1, the entries for the homing $k$-TSP on the real line assume $k \geq 2$).

## 5.4 Algorithms for general metric spaces

In the following, we give competitive algorithms and lower bounds for the nomadic $k$-TSP and the $k$-TRP in general spaces. Our results will be formulated in a more general *resource augmentation* framework [62]. We define the nomadic $(k, k^*)$-TSP and $(k, k^*)$-TRP exactly as the nomadic $k$-TSP and the $k$-TRP, except that we measure the performance of an online algorithm with $k$ servers relative to an optimal offline algorithm with $k^* \leq k$ servers.

Sections 5.4.1 and 5.4.2 give an algorithm for the $(k, k^*)$-TSP and the $(k, k^*)$-TRP respectively. A lower bound for both problems is proved in Section 5.4.3.

### 5.4.1 The $k$-Traveling Salesman Problem

**Theorem 5.4.** *There is a deterministic online algorithm for the nomadic* $(k, k^*)$*-TSP with competitive ratio*

$$1 + \sqrt{1 + 1/2^{\lfloor k/k^* \rfloor - 1}}.$$

The algorithm achieving this bound is called Group Return Home (Algorithm 8). Define the *distance of a group to the origin* at time $t$ as the maximum distance of a server in the group to $o$ at time $t$.

**Algorithm 8** - Group Return Home

Divide the servers into $g = \lfloor k/k^* \rfloor$ disjoint sets (*groups*) of $k^*$ servers each. Any remaining server is not used by the algorithm.
Initially, all servers wait at $o$. Every time a new request arrives, all servers not at $o$ return to the origin at full speed. Once all of the servers in one of the groups, say group $G$ (ties broken arbitrarily), are at $o$, compute a set of $k^*$ paths $\{P_1, \ldots, P_{k^*}\}$ starting at $o$, covering all unserved requests and minimizing $\max_i \ell(P_i)$. Then, for $i = 1, \ldots, k^*$, the $i$-th server in $G$ follows path $P_i$ at the highest possible speed while remaining at a distance at most $\alpha t$ from $o$ at any time $t$, for some constant $\alpha \in (0, 1]$. Servers in other groups continue to head towards $o$ (or wait there) until a new request is released.

**Lemma 5.5.** *At any time $t$, in the schedule generated by , let $G_1(t), \ldots, G_g(t)$ be the $g$ groups in order of nondecreasing distance to $o$. Then the distance of $G_i(t)$ to $o$ is at most $2^{i-g}\alpha t$.*

*Proof.* We prove the lemma by induction on the number of requests. That is, we show that if the lemma holds at the release date $t$ of some request, it will hold until the release date $t + \delta$ of the next request. Obviously, the lemma is true up to the time the first request is given, since all servers remain at $o$.

Suppose a request is given at time $t$. By induction, we know that there are groups $G_1(t), \ldots, G_g(t)$ such that each server of group $G_i(t)$ is at distance at most $2^{i-g}\alpha t$ from $o$. For the rest of the proof we fix the order of the groups as the order they have at time $t$ and write $G_i$ instead of $G_i(t)$. Let $D_i(\tau) = \max_{s \in G_i} d(s(\tau), o)$.

Between time $t$ and $t' = t + D_1(t)$, the lemma holds since all servers are getting closer to $o$. We show that the lemma holds at $t' + \delta$ for all $\delta > 0$. Notice that $D_1(t' + \delta) \leq \delta$ since every server moves at most at unit speed.

If $\delta \in (0, 2^{1-g}\alpha t]$, we know that $D_1(t' + \delta) \leq 2^{1-g}\alpha t$, so the lemma holds with the groups in the same order as before.

Now, let $\delta \in (2^{i-1-g}\alpha t, 2^{i-g}\alpha t]$ for $2 \leq i \leq g$. Then at time $t' + \delta$, group $G_j$ is already at $o$ for each $1 < j < i$. For group $G_i$, $D_i(t' + \delta) \leq 2^{i-g}\alpha t - 2^{i-1-g}\alpha t = 2^{i-1-g}\alpha t$. For group $G_1$, $D_1(t' + \delta) \leq 2^{i-g}\alpha t$. For groups $G_{i+1}$ through $G_g$, $D_{i+1}(t' + \delta) \leq 2^{i+1-g}\alpha t, \ldots, D_g(t' + \delta) \leq 2^0\alpha t$. So the lemma holds for these values of $\delta$.

The last case is $\delta > \alpha t$. In this case all groups except $G_1$ are at $o$, and because of the speed constraint $D_1(t' + \delta) \leq \alpha(t' + \delta)$. Thus the lemma holds. □

*Proof of Theorem 5.4.* Let $t$ be the release date of the last request and let $G_1$ be the group minimizing the distance to the origin at time $t$. Using Lemma 5.5 we know that $D_1(t) \leq 2^{1-g}\alpha t$. Group $G_1$ will return to the origin and then follow the offline set of paths $\{P_1, \ldots, P_{k^*}\}$. Notice that $(\sigma) \geq t$, since no schedule can end before the release date of a request, and $(\sigma) \geq \max_i \ell(P_i)$ because of the optimality of the $P_i$.

Let $s$ be the server in $G_1$ that achieves the makespan. If $s$ does not limit its speed after time $t$, we have $(\sigma) \leq t + D_1(t) + \max_i \ell(P_i) \leq (2 + 2^{1-g}\alpha)$ $(\sigma)$.

Otherwise, let $t'$ be the last time at which $s$ is moving at limited speed. It is not difficult to see that $s$ must serve some request at that time. Let $x_0$ be the location of this request. Then $t' = (1/\alpha)d(x_0, o)$ and $s$ continues following the remaining part of its path, call it $P'$, at full speed. Hence, $(\sigma) = t' + \ell(P')$. Since $(\sigma) \geq \max_i \ell(P_i) \geq d(o, x_0) + \ell(P')$ this yields $(\sigma) \leq (1/\alpha)$ $(\sigma)$.

Thus, the competitive ratio is at most $\max\{2 + 2^{1-g}\alpha, 1/\alpha\}$ and choosing $\alpha$ in order to minimize it gives $\alpha = \sqrt{2^{g-1}(2^{g-1}+1)} - 2^{g-1}$ and the desired competitive ratio. □

**Corollary 5.6.** *There is a deterministic* $(1 + \sqrt{2})$*-competitive online algorithm for the nomadic k-TSP.*

### 5.4.2 The $k$-Traveling Repairman Problem

**Theorem 5.7.** *There is a deterministic online algorithm for the* $(k, k^*)$*-TRP with competitive ratio* $2 \cdot 3^{1/\lfloor k/k^* \rfloor}$.

**Algorithm 9** - Group Interval

Divide the servers into $g = \lfloor k/k^* \rfloor$ disjoint sets (groups) of $k^*$ servers each. Any remaining server is not used by the algorithm.
Let $L$ be the earliest time that any request can be completed (wlog $L > 0$). For $i = 0, 1, \ldots$, define $B_i = \alpha^i L$ where $\alpha \in (1, 3^{1/g}]$ will be fixed in the analysis.
At time $B_i$, compute a set of paths $S_i = \{P^i_1, \ldots, P^i_{k^*}\}$ for the set of yet unassigned requests released up to time $B_i$ with the following properties:
(i) every $P^i_j$ starts at the origin $o$;
(ii) $\max_j \ell(P^i_j) \leq B_i$;
(iii) $S_i$ maximizes the number of requests served among all schedules satisfying the first two conditions.
The requests in $S_i$ are now considered assigned.
Let $\beta = 2/(\alpha^g - 1)$. Starting at time $\beta B_i$, the $j$-th server in the $(i \bmod g)$-th group follows path $P^i_j$, then returns to $o$ at full speed.

We call the algorithm achieving the bound Group Interval (Algorithm 9), as it can be seen as a multiserver generalization of algorithm Interval [68]. The algorithm is well defined since the time between two departures of the same group is enough for the group to complete its first schedule and return to the origin: $\beta B_{i+g} - \beta B_i = \beta(\alpha^g - 1)B_i = 2B_i$. It is an online algorithm since $\beta \geq 1$ for any $\alpha \in (1, 3^{1/g}]$.

To sketch the proof of Theorem 5.7, we start with two auxiliary lemmas.

**Lemma 5.8** ([68])**.** *Let* $a_i, b_i \in \mathbb{R}$ *for* $i = 1, \ldots, p$*, for which*

*(i)* $\sum_{i=1}^{p} a_i = \sum_{i=1}^{p} b_i$*, and*

*(ii)* $\sum_{i=1}^{p'} a_i \geq \sum_{i=1}^{p'} b_i$ *for all* $1 \leq p' \leq p$.

*Then the* $\sum_{i=1}^{p} \tau_i a_i \leq \sum_{i=1}^{p} \tau_i b_i$ *for any nondecreasing sequence of real numbers* $0 \leq \tau_1 \leq \tau_2 \leq \ldots \leq \tau_p$.

**Lemma 5.9.** *Let* $R_i$ *be the set of requests served by the set of paths* $S_i$ *computed by at time* $B_i$, $i = 1, 2, \ldots$ *and let* $R_i^*$ *be the set of requests in the optimal offline solution that are completed in the time interval* $(B_{i-1}, B_i]$. *Then*

$$\sum_{i=1}^{q} |R_i| \geq \sum_{i=1}^{q} |R_i^*| \quad \text{for all } q = 1, 2, \ldots.$$

*Proof.* We omit the proof, as it is basically the same as that of Lemma 4 in [68]. □

*Proof of Theorem 5.7.* Let $\sigma = \sigma_1 \ldots \sigma_m$ be any sequence of requests. By construction of , each request in $R_i$ is served at most at time $(1 + \beta)B_i$. Now, let $p$ be such that the optimal offline schedule completes in the interval $(B_{p-1}, B_p]$. Summing over all phases $1, \ldots, p$ yields

$$(\sigma) \leq (1 + \beta) \sum_{i=1}^{p} B_i |R_i| = (1 + \beta) \cdot \alpha \sum_{i=1}^{p} B_{i-1} |R_i|. \tag{5.1}$$

From Lemma 5.9 we know that $\sum_{i=1}^{q} |R_i| \geq \sum_{i=1}^{q} |R_i^*|$ for $q = 1, 2, \ldots$ We also know that $\sum_{i=1}^{p} |R_i| = \sum_{i=1}^{p} |R_i^*|$. Applying Lemma 5.8 to the sequences $a_i := |R_i|$, $b_i := |R_i^*|$, $\tau_i := B_{i-1}$, $i = 1, \ldots, p$ yields in (5.1)

$$(\sigma) \leq (1 + \beta) \cdot \alpha \sum_{i=1}^{p} B_{i-1} |R_i| \leq (1 + \beta) \cdot \alpha \sum_{i=1}^{p} B_{i-1} |R_i^*|. \tag{5.2}$$

Let $C_j^*$ be the optimal offline completion time of request $\sigma_j$. For each $\sigma_j$ denote by $(B_{\phi_j}, B_{\phi_{j+1}}]$ the interval that contains $C_j^*$. This inserted in (5.2) yields

$$(\sigma) \leq (1 + \beta) \cdot \alpha \sum_{j=1}^{m} B_{\phi_j} \leq (1 + \beta) \cdot \alpha \sum_{j=1}^{m} C_j^* = (1 + \beta) \cdot \alpha \cdot \quad (\sigma).$$

Choosing $\alpha = 3^{1/g}$ (so that $\beta = 1$) gives the theorem. □

**Corollary 5.10.** *There is a deterministic* $(1 + \sqrt{2})^2$*-competitive online algorithm for the k-TRP.*

*Proof.* When $g = 1$, the analysis in Theorem 5.7 can be improved slightly by choosing $\alpha = 1 + \sqrt{2}$ (however, simple calculus shows that no similar improvement is possible when $g \geq 2$). This improved ratio matches the $(1 + \sqrt{2})^2$-competitive algorithm [68] for the TRP with a single server. □

### 5.4.3 Lower bounds

**Theorem 5.11.** *The competitive ratio of any randomized online algorithm for the nomadic* $(k, k^*)$*-TSP or the* $(k, k^*)$*-TRP is at least* 2*.*

*Proof.* Consider the metric space induced by a star graph with $m$ unit-length rays, the origin being the center of the star. No request is given until time 1. At time 1, the adversary gives a request on an edge chosen uniformly at random, at distance 1 from the origin. The expected makespan for the adversary is 1. For the online algorithm, we say that a server *guards* a ray if at time 1 the server is located on the ray, but not at the center of the star. Then the makespan is at least 2 if no server guards the ray where the request is released, and at least 1 otherwise. But $k$ servers can guard at most $k$ rays, so

$$\mathbb{E}[\ \ (\sigma)] \geq 2 \cdot \left(1 - \frac{k}{m}\right) + 1 \cdot \frac{k}{m} \geq 2 - \frac{k}{m}$$

and the result follows by Yao's principle, since $m$ can be arbitrarily large. □

Notice that this lower bound is independent of the values $k$ and $k^*$. A consequence of this is that the upper bounds of Sections 5.4.1 and 5.4.2 are essentially best possible when $k \gg k^*$, as in that case they both approach 2.

## 5.5 Algorithms for the real line

### 5.5.1 An asymptotically optimal algorithm

**Theorem 5.12.** *There is a deterministic online algorithm with competitive ratio* $1 + O((\log k)/k)$ *for both the nomadic k-TSP and the k-TRP on the real line.*

As a preliminary, we prove a similar result on the *halfline*.

**Algorithm 10** - Geometric Progression Speeds

As a preprocessing step, the algorithm delays every request $(r, x)$ for which $x \geq r$ to time $x$; that is, the release date of each request $(r, x)$ is reset at $r' := \max\{r, x\}$ (the *modified release date*).
Then, let $g_k$ be the unique root greater than 1 of the equation $g_k^k = \frac{3g_k - 1}{g_k - 1}$ and define $\alpha_j = g_k^{j-k-1}$ for $j \in \{2, 3, \ldots, k\}$. For every $j > 1$, server $s_j$ departs at time 0 from $o$ at speed $\alpha_j$ and never turns back. The first server $s_1$ waits in $o$ until the first request $(r_0, x_0)$ is released with $0 < x_0 < s_2(r_0')$. For $i \geq 0$, define $t_i = g_k^i r_0'$. During any interval $[t_{i-1}, t_i]$, $s_1$ moves at full speed first from $o$ to $(g_k - 1)t_{i-1}/2$ and then back to $o$.

**Lemma 5.13.** *(Algorithm 10) is* $g_k$*-competitive for nomadic k-TSP and k-TRP on the halfline.*

*Proof.* First, notice that the modified release date of a request is a lower bound on its completion time. Thus it is enough to prove that, for every request $(r, x)$, the time at which it is served is at most $g_k r'$.

For $1 < j < k$, we say that a request $(r, x)$ is in *zone* $j$ if $\alpha_j \leq x/r' < \alpha_{j+1}$. We also say that a request is in zone 1 if $x/r' < \alpha_2$, and that it is in zone $k$ if $x/r' \geq \alpha_k$. By construction, every request is in some zone and a request in zone $j$ will be eventually served by server $s_j$.

For a request $(r, x)$ in a zone $j$ with $1 < j < k$, since the request is served by server $s_j$ at time $x/\alpha_j$ and since $x \leq \alpha_{j+1} r$, the ratio between completion time and modified release date is at most $\alpha_{j+1}/\alpha_j = g_k$. Similarly, for a request in zone $k$, since $x \leq r'$, the ratio between completion time and modified release date is at most $1/\alpha_k = g_k$.

It remains to give a bound for requests in zone 1. Take any such request, i.e., a request $(r, x)$ such that $x < \alpha_2 r'$ and suppose it is served at time $\tau \in [t_{i-1}, t_i]$ for some $i$. If $r' \geq t_{i-1}$, then, since $\tau \leq t_i$, the ratio between $\tau$ and $r'$ is at most $g_k$ by definition of $t_i$, $i \geq 0$.

If $r' < t_{i-1}$, then, since $\tau > t_{i-1}$, only two possible cases remain. First, the situation that $x > \frac{g_k - 1}{2} t_{i-2}$. Since $\tau = t_{i-1} + x$ and $r' \geq x/\alpha_2$, we have

$$\frac{\tau}{r'} \leq \frac{x + t_{i-1}}{x/\alpha_2} \leq \alpha_2 \left(1 + \frac{2 g_k t_{i-2}}{(g_k - 1) t_{i-2}}\right) = \alpha_2 \frac{3 g_k - 1}{g_k - 1} = \alpha_2 g_k^k = g_k.$$

In the second situation, $x \leq \frac{g_k - 1}{2} t_{i-2}$. Then $r'$ must be such that $s_1$ was already on its way back to 0 during $[t_{i-2}, t_{i-1}]$, in particular $r' \geq g_k t_{i-2} - x$. Thus,

$$\tau / r' \leq \frac{g_k t_{i-2} + x}{g_k t_{i-2} - x} \leq \frac{3 g_k - 1}{g_k + 1} \leq g_k.$$

□

The algorithm for the real line simply splits the $k$ servers evenly between the two halflines, and uses on each halfline.

**Algorithm 11** - Split Geometric Progression Speeds

Arbitrarily assign $\lceil k/2 \rceil$ servers to $\mathbb{R}_+$ and $\lfloor k/2 \rfloor$ servers to $\mathbb{R}_-$. On each of the two halflines, apply Algorithm 10 independently (i.e., ignoring the requests and the servers in the other halfline).

**Lemma 5.14.** *For any $k \geq 2$, (Algorithm 11) is $g_{\lfloor k/2 \rfloor}$-competitive for the nomadic $k$-TSP and the $k$-TRP on the line.*

*Proof.* The only lower bounds on the offline cost that we used in the proof of Lemma 5.13 were the distance of every request from $o$ and the release date of every request. They are valid independent of the number of offline servers. In particular, they hold if the number of offline servers is twice the number of online servers. Thus, we can analyze the competitiveness of the online servers on each of the two halflines separately and take the worst of the two competitive ratios. □

**Lemma 5.15.** *For any $k \geq 1$, $g_k \leq 1 + \frac{2\log k+3}{k}$.*

*Proof.* We defined $g_k$ as the unique root greater than 1 of $z^k = 1 + \frac{2z}{z-1}$. Since $\lim_{z\to\infty} z^k > \lim_{z\to\infty} 1 + \frac{2z}{z-1}$, it suffices to prove that $z_0 := 1 + \frac{2\log k+3}{k}$ satisfies $z_0^k \geq 1 + \frac{2z_0}{z_0-1}$. The binomial theorem and the standard fact that $\binom{k}{j} \geq \frac{k^j}{j^j}$ yield

$$\begin{aligned} z_0^k - 1 &= \sum_{j=1}^{k} \binom{k}{j} \frac{(2\log k+3)^j}{k^j} \geq \sum_{j=1}^{k} \frac{(2\log k+3)^j}{j^j} \geq \sum_{j=1}^{\lfloor \log k \rfloor + 1} \left(\frac{2\log k+3}{j}\right)^j \\ &\geq \sum_{j=1}^{\lfloor \log k \rfloor + 1} 2^j \geq 2^{\log k+1} - 2 = 2k - 2. \end{aligned}$$

Now it can be verified that for all $k > 2$, $2k - 2 > \frac{2k}{2\log k+3} + 2 = \frac{2z_0}{z_0-1}$. Finally, the bound also holds for $k \in \{1, 2\}$ as seen by explicitly finding $g_1$ and $g_2$. □

Theorem 5.12 now follows from Lemma 5.14 and Lemma 5.15.

### 5.5.2 Lower bounds

**Theorem 5.16.** *The competitive ratio of any randomized online algorithm for the nomadic $(k, k^*)$-TSP or the $(k, k^*)$-TRP on the line is at least $1 + 1/2k$.*

*Proof.* The adversary gives a single request at time 1, in a point drawn uniformly at random from the interval $[-1, 1]$. The expected optimal cost is obviously 1. Thus, by Yao's principle it suffices to show that $\mathbb{E}[\ \ (\sigma)] \geq 1 + 1/2k$.

In order to bound $\mathbb{E}[\ \ (\sigma)]$, let $f(x) = \min_{j\in\{1,\ldots,k\}} d(x, s_j(1))$. Notice that $1 + f(x)$ is a lower bound on the cost paid by the online algorithm, assuming that the request was given at $x$. In terms of expected values,

$$\mathbb{E}[\ \ (\sigma)] \geq \mathbb{E}[1 + f(x)] = 1 + \frac{1}{2}\int_{-1}^{1} f(x)dx.$$

Thus, we want to find the minimum value of the area below $f$ in $[-1, 1]$. Basic calculus shows that this area is minimized when the servers are evenly spread inside the interval and at distance $1/k$ from the extremes, in which case its value is $1/k$. □

## 5.6 Lower bounds on the plane

Comparing the results in Section 5.4 with those in Section 5.5, we see that while in general spaces the competitive ratio of both the nomadic $k$-TSP and the $k$-TRP always remains lower bounded by 2, on the real line we can achieve $1 + o(1)$ asymptotically. A natural question is whether on a low-dimensional space like the Euclidean plane we can also achieve $1 + o(1)$ competitiveness. In this section we answer this question negatively.

**Theorem 5.17.** *The competitive ratio of any randomized online algorithm for the nomadic k-TSP on the plane is at least* $4/3$.

*Proof.* As a crucial ingredient of the proof we introduce a new kind of request, which is located in a single point $x$ of the space but has an arbitrarily long processing time $p$ (this processing time can be divided among the servers processing the request). We show how this can be emulated in the Euclidean plane with arbitrarily good precision by giving a high enough number of requests packed inside an arbitrarily small square around $x$.

Fix some arbitrary $\epsilon > 0$. Consider a square with side length $s = \sqrt{\epsilon p}$ centered around $x$. The square can be partitioned in $s^2/\epsilon^2$ smaller squares of side length $\epsilon$. In the center of each of these smaller squares we give a request. Notice that the distance between any pair of such requests is at least $\epsilon$. Thus, the sum of the times required for any $k$ servers to serve all requests is at least $(\frac{s^2}{\epsilon^2} - k)\epsilon$, no matter where the servers start (the $-k\epsilon$ term reflects the possible saving each server could have by starting arbitrarily close to the *first* request he serves).

For $\epsilon$ tending to zero, the requests converge to the point $x$ and the total processing time needed converges to $p$. If the starting points of the servers are most favorable, an algorithm could finish serving all requests in time $p/k$.

We show how to use such a "long" request to achieve our lower bound. At time 1, the adversary gives a long request of processing time $p = 2k$ in a point drawn uniformly at random from $\{(1,0),(-1,0)\}$. The expected optimal cost is $1 + p/k = 3$. By Yao's principle, it remains to prove that $\mathbb{E}[\quad(\sigma)] \geq 4$.

Since there is a single long request, we can assume wlog that all the online servers will move to the request and contribute to serving it. Since $p = 2k$, the server that will contribute most to the service will have to spend time at least $2k/k = 2$ in $x$, and this is enough for any other server to arrive and give a contribution (since at time 1 no server can be farther than 2 from $x$).

Suppose wlog that the servers are numbered in order of nondecreasing distance to $x$ and let $d_i = d(x, s_i(1))$. We have $\quad(\sigma) \geq 1 + t_0$, with $t_0$ the time needed for the servers to completely serve the request, i.e., the time when its remaining processing time is zero. Thus, $t_0$ satisfies $\sum_{i=1}^{k-1} i(d_{i+1} - d_i) + k(t_0 - d_k) = p$ , since during interval $[d_i, d_{i+1})$ exactly $i$ servers are processing the request. Hence,

$$kt_0 = p + kd_k - \sum_{i=1}^{k-1} i(d_{i+1} - d_i) = p + \sum_{i=1}^{k} d_i.$$

Now consider the positions of the online servers at time 1 inside the ball of radius 1 around the origin. Regarding points as vectors in $\mathbb{R}^2$, $d_i$ can be written as $\|s_i(1) - x\|$ (here $\|\cdot\|$ denotes the Euclidean norm). Then

$$\begin{aligned}\sum_{i=1}^{k} d_i &= \sum_i \|s_i(1) - x\| \geq \left\| \sum_i (s_i(1) - x) \right\| \\ &= k\left\| \frac{1}{k} \sum_i s_i(1) - x \right\| = k\|b - x\| = k \cdot d(b, x),\end{aligned}$$

where $b = \frac{1}{k}\sum_i s_i(1)$ is the centroid of the $s_i(1)$. Hence,

$$\begin{aligned}\mathbb{E}[\quad(\sigma)] \geq 1 + \mathbb{E}[t_0] &\geq 1 + p/k + \mathbb{E}[d(b,x)] = \\ &= 3 + (1/2)\,d(b,(1,0)) + (1/2)\,d(b,(-1,0)) \\ &\geq 3 + (1/2)\,d((1,0),(-1,0)) = 4.\end{aligned}$$

□

A similar technique gives an analogous lower bound for the $k$-TRP on the plane.

**Theorem 5.18.** *The competitive ratio of any randomized online algorithm for the $k$-TRP on the plane is at least* $5/4$.

*Proof.* We use the same input distribution used in the proof of Theorem 5.17. For the costs, it is equivalent but easier to consider average completion time instead of the sum of completion times. Then, the expected optimal cost can be easily seen to be 2 since the $k$ servers can uniformly process the request of length $p$ (recall that $p = 2k$) during time interval $[1, 3]$.

To lower bound the online cost, consider $w \in [0, p]$ and let $C(w)$ be the first time at which a total work of $w$ has been completed by the online algorithm. Then

$$(\sigma) = \frac{1}{p}\int_0^p C(w)dw.$$

Now if we define $t_0$ as in the proof of Theorem 5.17 we have $C(p) = 1 + t_0$, while $C(\epsilon) > 1$ for any $\epsilon > 0$. Moreover, the function $C$ is concave, since the speed at which the long request is processed can only increase as more servers arrive, so the value of the integral above is at least $p(1 + t_0/2)$. Thus

$$\mathbb{E}[\quad(\sigma)] \geq \mathbb{E}[\frac{1}{p} \cdot p(1 + \frac{t_0}{2})] \geq \mathbb{E}[1 + t_0/2] = 5/2$$

since we already showed in the proof of Theorem 5.17 that $\mathbb{E}[t_0] \geq 3$. The claim follows by Yao's principle. □

## 5.7 Conclusions and open problems

After analyzing the differences between multiple and single server variants, we can conclude that sometimes having multiple servers is more beneficial to the online algorithm than to the offline adversary. In some cases, including the traveling repairman problem on the line, the online algorithms can approach the offline cost when there are enough servers. In more general spaces, these extremely favorable situation cannot occur. Still in some intermediate cases, like the Euclidean plane, it is conceivable that the competitive ratios become lower than those of the corresponding single server problems. We leave the analysis of the competitive ratio in these situations as an open problem.

# Chapter 6

# An adversarial queueing model for online server routing

## 6.1 Introduction

Consider the following model of a computer harddrive: while the disk is rotating, read and write requests arrive for data lying on specific sectors of the disk. Thus, the head located on the arm of the disk has to move in order to align itself to the correct track, and wait for the disk to rotate onto the sector holding the data. If we ignore the rotational delay, we can view the problem as an online server routing problem on a finite chain (the head's arm).

A sensible algorithm for controlling the disk's head should be able to cope with requests in a *stable* manner: the number of unserved requests should not increase indefinitely as time passes. This requires of course that the rate of incoming requests does not overflow the head's service speed, but that condition alone is not sufficient; an algorithm that moves the head back and forth between far away locations while serving few requests will easily lead to an unstable system even when the arrival rate is relatively low. Notice that this stability requirement can be seen as requiring the algorithm to have an optimal *throughput*, since no algorithm with suboptimal throughput can keep the system stable if the load is high for a sufficient amount of time. Apart from this minimal stability requirement, one would like the delay of each individual request to be as small as possible and definitely within a fixed, predictable range.

The optimization part of this problem could be seen as an online server routing problem with the *maximum flow time* objective. However, it is well-known from the online algorithm literature that there cannot be constant competitive algorithms for this problem. Thus, pure competitive analysis is unable to distinguish the behavior of different algorithms. Restrictions to the offline adversary and alternative models have been proposed that try to overcome this issue [56, 70].

In this chapter we propose a different approach that, while abandoning competitive analysis, still assumes worst case behavior of the inputs. The approach

is based on *adversarial queueing theory*, which has been recently proposed as a mean to analyze online packet routing problems [7, 32]. We propose an adversarial queueing theory model specific for online server routing, and in this framework we analyze the stability and the performance of several natural algorithms. This allows us to distinguish between algorithms that would fare equally badly from the point of view of competitive analysis.

## 6.2 Related work

The adversarial queueing theory framework was first formulated by Borodin et al. [32], who consider a model of packet routing problems in networks with continuous packet arrivals. The model replaces the probabilistic assumptions usual in queueing theoretical analyses with worst case inputs. Thus the name *adversarial queueing theory* was proposed to stress that while the issue studied is that of stability – the crucial issue of queueing theory – the approach is of adversarial nature. Following the work of Borodin et al., Andrews et al. [7] consider several natural algorithms in this framework and give many stability and instability results.

In the context of online server routing problems, the model that comes closer to the one we propose is the *reasonable load* model by Hauptmeier et al. [56]. Roughly speaking, they assume that every set of requests that come up in a sufficiently large time period can be served in a time period of at most the same length. Under this assumption, they consider the dial-a-ride problem for the minimization of maximum resp. total flow time, and they distinguish the behavior of two algorithms, Replan and Ignore, that would be indistinguishable by pure competitive analysis. We also give similar results for Replan, Ignore and several other algorithms. Other results in related directions of research, albeit in quite different models, are given by Alborzi et al. [4] and Irani et al. [58].

Experimental results for different disk scheduling policies were given by Teorey and Pinkerton [88]. More recently, new algorithms for disk scheduling have been proposed by Andrews et al. [8].

## 6.3 The model

An *online server routing system* consists of a triple $(G, A, P)$, where $G$ is a graph, $A$ is an adversary and $P$ is an online protocol[1]. We now further detail each of these components.

The undirected, connected graph $G = (V, E)$ represents the space where requests are injected by the adversary and where the server operated by the online protocol moves. A special vertex $o \in V$ is marked as the origin. We let $n$ be the number of vertices of $G$ and $\delta$ its diameter.

[1] In network management applications [32], the term protocol is used to stress the fact that the algorithm is distributed, i.e. it acts based only on local information. Since we do not have this issue here, we use interchangeably the terms *protocol* and *algorithm*.

We use a discrete time model. At every time step, the server operated by the protocol can either cross an edge or serve a single request from its current location (but not both). At time 0, the server is located at the origin.

We consider two types of adversaries, a stronger one used in all the positive results and a weaker one used for the negative results; this difference can only strengthen the results. A *strong adversary* of rate $\lambda \in (0, 1]$ and burst $\mu > 0$ can, during any time interval $I$, release at most $\lambda|I| + \mu$ requests overall anywhere on the nodes of the graph. A *weak adversary* of rate $\lambda \in (0, 1]$ can, during any time interval $I$, release at most $\lceil \lambda|I| \rceil$ requests overall anywhere on the nodes of the graph. The sequence generated by the adversary is denoted by $\sigma = \sigma_1\sigma_2 \cdots$. Every request is a pair $(r_j, x_j) \in \mathbb{Z}_+ \times V$, where $r_j$ is the release date of the request (the time it becomes available) and $x_j$ its location (a vertex of $G$). We denote by $C_j$ the completion time of request $\sigma_j$, i.e., the time unit following the one during which the server started processing $\sigma_j$.

More formally, the model can be described as follows. At every time step $t$, the current *configuration* $C_t$ of the system is a vertex $s(t) \in V$ plus a collection of sets $\{U_v^t : v \in V\}$, such that $s(t)$ is the position of the server at time $t$ and $U_v^t$ is the set of requests waiting at $v$ at the time $t$. From the configuration $C_t$ we obtain the configuration $C_{t+1}$ as follows. The adversary adds new requests to some of the sets $U_v^t$; then the protocol either chooses $s(t + 1)$ such that $\{s(t), s(t + 1)\} \in E$ or it removes a request from $U_{s(t)}^t$ and leaves $s(t + 1) = s(t)$. A *time-evolution* of $G$, of rate $\lambda$ and burst $\mu$, is a sequence of such configurations $C_0, C_1, \ldots$, such that for all intervals $I$, no more than $\lambda|I| + \mu$ requests are introduced during $I$ in $G$. By the *system* $(G, A, P)$ we mean the time-evolution of $G$ induced by adversary $A$ and protocol $P$ with initial configuration $s(0) = o$ and $U_v^0 = \emptyset$ for all $v \in V$.

Our results are centered around the following concepts.

**Definition 6.1.** An online server routing system $(G, A, P)$ is *stable* if there exists a constant $u_{\max}$ (which may depend on the system) such that

$$\sum_{v \in V} |U_v^t| \leq u_{\max}$$

for all $t \in \mathbb{Z}_+$, that is, the total number of unserved requests is bounded by $u_{\max}$ at all times. Otherwise we say that the system is *unstable*.

**Definition 6.2.** A protocol $P$ is *universally stable* if for every graph $G$ and every strong adversary $A$ of rate $\lambda$ and burst $\mu$ with $\lambda < 1$, the system $(G, A, P)$ is stable.

**Definition 6.3.** A system $(G, A, P)$ has *bounded flow time* if there is a constant $F$ such that for any request $\sigma_j$ released by $A$, $C_j - r_j \leq F$. We say that a protocol has bounded flow time if for every graph $G$ and every strong adversary $A$ of rate $\lambda$ and burst $\mu$ with $\lambda < 1$, the system $(G, A, P)$ has bounded flow time.

Clearly, if a protocol is not universally stable, it cannot have bounded flow time. The converse implication is false in general; compare, for example, Theorems 6.6 and 6.11.

## 6.4 Stability results

In this section we consider several natural algorithms for the online server routing model we have introduced, and we classify each of them according to Definitions 6.2 and 6.3. A summary of the results is presented in Table 6.1.

As a preliminary result, we remark that, in Definition 6.2, it is necessary to consider only adversaries with rate $\lambda$ strictly less than 1, as otherwise for any protocol the system can be made unstable.

**Lemma 6.1.** *Let $P$ be any protocol and $G$ any nontrivial graph. Then there exists an adversary $A$ of rate 1 such that the system $(G, A, P)$ is unstable.*

*Proof.* The adversary gives at every time step a request on a vertex that is distinct from the current location of the server. The server must change location infinitely often, but every time it does, the number of unserved requests increases by one. □

Since the requests are qualitatively identical, we assume without loss of generality that at every node of the graph there is a *queue* holding the requests pending at that node; inside a queue, requests are ordered by release date. The protocol always serves the oldest request of a queue.

A general term used in the algorithms is that of "emptying" the queue at a given vertex $v$ of the graph. By this we mean that the server serves a request in $v$ at every time step until no more requests in $v$ are available. This process takes a finite amount of time because the rate of the adversary is strictly less than one. More precisely, we have the following.

**Lemma 6.2.** *Suppose that at time $t$ a server is located at a node $v$ where $u_v$ requests are pending unserved. Against a strong adversary of rate $1-\epsilon$ and burst $\mu$, emptying $v$ requires at most $(u_v+\mu)/\epsilon$ time steps.*

*Proof.* Suppose the claim is false. Then, a request will be served in $v$ on each of the following $1+(u_v+\mu)/\epsilon$ time steps, but there are at most $u_v$ requests in $v$ at time $t$ and no more than

$$(1-\epsilon)\left(1+\frac{u_v+\mu}{\epsilon}\right)+\mu = 1+\frac{u_v+\mu}{\epsilon}-\epsilon-u_v-\mu+\mu < 1+\frac{u_v+\mu}{\epsilon}-u_v$$

new requests can be released until time $t+1+(u_v+\mu)/\epsilon$. Thus we get a contradiction, because the server would serve more requests than what are available. □

Some of the algorithms we consider are undefined in the case of ties. In those cases, we assume the worst tie-breaking rule for the positive results and the best for the negative ones.

The following technical lemma is useful when establishing instability results. Given two adversaries $A_1, A_2$, their *union* $A_1 \cup A_2$ is the adversary that releases the requests of both adversaries.

| Algorithm | Univ. stable? | Bounded flow time? | References |
|---|---|---|---|
| | no | no | Th. 6.4 |
| | no | no | Th. 6.5 |
| | yes | no | Th. 6.6, 6.11 |
| | yes | yes | Th. 6.7, 6.12 |
| | no | no | Th. 6.8 |
| | yes | yes | Th. 6.9, 6.12 |
| - | yes | yes | Th. 6.10, 6.12 |

Table 6.1: Universal stability of different server routing algorithms.

**Lemma 6.3.** *Let $A_1$ be a weak adversary of rate $\lambda \leq 1/2$. Then for every $v \in V$ and every sufficiently small $\epsilon > 0$, there exists a weak adversary $A_2$ of rate $\epsilon$ releasing requests in $v$ such that $A_1 \cup A_2$ is a weak adversary of rate at most $2/3 + \epsilon$.*

*Proof.* We define $A_2$ as follows: it releases requests in $v$ only at time steps during which $A_1$ does not release any request, and so that its rate is $\epsilon$; this is always possible by taking $\epsilon$ sufficiently small. Apart from these constraints, the precise release dates of the requests of $A_2$ are irrelevant for the lemma.

We have to prove that for any interval $I$ of length $t$, the requests released by $A_1 \cup A_2$ are at most $\lceil (2/3 + \epsilon)t \rceil$.

Consider any such interval $I$. If $t = 1$, the claim holds simply because $A_1$ and $A_2$ never release a request during the same time step.

For all other $t$, notice that the number of requests released by $A_1$ (resp. $A_2$) is at most $\lceil \lambda t \rceil$ (resp. $\lceil \epsilon t \rceil$). We prove the claim by showing that $\lceil \lambda t \rceil + \lceil \epsilon t \rceil \leq \lceil (2/3+\epsilon)t \rceil$. We assume that $\epsilon \leq 2/3 - \lambda$. When $t \geq 1/(2/3 - \lambda)$,

$$\lceil \lambda t \rceil + \lceil \epsilon t \rceil \leq \lceil \lambda t + 1 + \epsilon t \rceil \leq \lceil \lambda t + (2/3 - \lambda)t + \epsilon t \rceil \leq \lceil (2/3 + \epsilon)t \rceil.$$

When $t < 1/(2/3 - \lambda)$, notice that $\lceil \epsilon t \rceil = 1$. If $t$ is even, say $t = 2q$,

$$\lceil \lambda t \rceil \leq \lceil t/2 \rceil = q \leq (4/3)q = (2/3)t.$$

If $t$ is odd, $t = 2q + 1$ where $q \geq 1$,

$$\lceil \lambda t \rceil \leq \lceil t/2 \rceil = q + 1 \leq (4/3)q + 2/3 = (2/3)t.$$

Thus, in both cases,

$$\lceil \lambda t \rceil \quad \leq \quad (2/3)t < (2/3)t + \epsilon t$$

from which it follows

$$\lceil \lambda t \rceil + \lceil \epsilon t \rceil = \lceil \lambda t \rceil + 1 \quad \leq \quad \lceil (2/3)t + \epsilon t \rceil.$$

□

**Algorithm 12** - First In First Out

The server processes the requests in the same order as their release dates.

In the context of single-machine scheduling, the protocol (Algorithm 12) is an optimal algorithm for minimizing the maximum flow time of jobs. However, a similar approach fails in server routing because the costs of moving between distant requests are not recovered in any way.

**Theorem 6.4.** *For every nontrivial graph $G$, there is a weak adversary $A$ such that $(G, A, )$ is unstable.*

*Proof.* Consider any edge $\{v, w\}$ of the graph, and suppose that the server starts in $w$. Fix $\lambda \in (1/2, 1)$. The requests are given alternatively in $v$ and $w$ at rate $\lambda$. Note that serves requests at a rate of at most $1/2$, since it needs to move after serving each request. But $\lambda > 1/2$, thus the system is unstable. In general, if the graph has diameter $\delta$, is unstable at every rate greater than $1/(\delta + 1)$. □

Another natural algorithm is Shortest Seek Time First (Algorithm 13). The algorithm attempts to minimize the distance traveled between the service of any two requests.

**Algorithm 13** - Shortest Seek Time First

works in phases. At the beginning of each phase, let $v$ be the node with a nonempty queue that is nearest to the current position of the server. At every step during the phase, the server moves along a shortest path from the current node to $v$. When it reaches $v$, it proceeds to empty $v$. When $v$ has been emptied, the phase ends.

**Theorem 6.5.** *For every graph $G$ of diameter at least 3 there is a weak adversary $A$ such that $(G, A, )$ is unstable.*

*Proof.* Consider any chain $v_0v_1v_2v_3$ of length 3 in $G$, and suppose that the server starts in $v_1$. A first adversary $A_1$ gives requests alternatively in $v_0$ and $v_1$, at a rate of $1/2$, so that as soon as a request is served in $v_i$ a new request appears in $v_{1-i}$. By Lemma 6.3, for any $\epsilon > 0$ there exists an adversary $A_2$ of rate $\epsilon$ releasing requests in $v_3$ such that $A_1 \cup A_2$ has rate at most $2/3 + \epsilon$. Notice that requests of $A_1$ keep the server between $v_0$ and $v_1$, so that requests of $A_2$ will never be served. Thus the system $(G, A_1 \cup A_2, )$ is unstable. □

Another reasonable strategy tries to maximize the work to be done on the next queue before leaving it, by serving the largest queue (Algorithm 14).

**Theorem 6.6.** *is universally stable.*

**Algorithm 14** - Empty the Largest Queue

works in phases. At the beginning of each phase, let $v \in V$ be the node with the largest queue in the system. At every step during the phase, the server moves along a shortest path from the current node to $v$, until it reaches $v$. When it reaches $v$, the server proceeds to empty it. When $v$ has been emptied, the phase ends.

*Proof.* Consider a strong adversary of rate $1-\epsilon$ and burst $\mu$. First we prove by induction on phases that at the beginning of every phase the number of unserved request is at most

$$(1-\epsilon)\delta n/\epsilon + \mu n/\epsilon.$$

Consider some phase starting at $t$ and ending at $t'$, and let $z$ be the number of requests served during the phase and $u$ the number of requests unserved at time $t$. Then

$$t' - t \leq \delta + z, \tag{6.1}$$

since at every time step either the algorithm moves towards the target node of the phase, or it serves a request in $v$. Also,

$$z \geq u/n, \tag{6.2}$$

since $v$ was the node with the largest queue at time $t$, and thus holds at least the average number of requests per node. Then, the number of unserved requests at time $t'$ is at most

$$\begin{aligned} u + (1-\epsilon)(t'-t) + \mu - z &\leq (1-\epsilon)\delta + \mu + u - \epsilon z && \text{by (6.1)} \\ &\leq (1-\epsilon)\delta + \mu + (1-\epsilon/n)u && \text{by (6.2)} \\ &\leq (1-\epsilon)\delta n/\epsilon + \mu n/\epsilon && \text{by induction.} \end{aligned}$$

To conclude the proof we need to show that the duration of each phase is bounded by a constant. Obviously, traveling to the target node $v$ takes at most $\delta$ time steps. It remains to bound the time needed to empty $v$. When the server arrives at $v$, $v$ contains at most $u + \delta + \mu$ requests. By Lemma 6.2, emptying $v$ will require time at most $(u + \delta + 2\mu)/\epsilon$. □

An algorithm related to is (Algorithm 15). It can be seen as a variant of with an hysteresis mechanism.

**Algorithm 15** - Empty the Oldest Queue

works in phases. At the beginning of each phase, let $v$ be the node containing the oldest request in the system. At every step during the phase, the server moves along a shortest path from the current node to $v$. When it reaches $v$, it proceeds to empty $v$. When $v$ has been emptied, the phase ends.

**Theorem 6.7.** *is universally stable.*

*Proof.* We define an *epoch* as a sequence of $n$ phases. We first prove by induction that at the beginning of every epoch the number of unserved request is bounded by

$$(1-\epsilon)\delta n/\epsilon + \mu/\epsilon.$$

Then we show that the duration of every epoch is also bounded. Let $[t, t')$ be an epoch, let $z$ be the number of requests served during the epoch and let $u$ be the number of unserved requests at the beginning of the epoch. Then

$$t' - t \leq \delta n + z \tag{6.3}$$

by definition of , since at every time step either moves the server towards the target of the new phase (of which there are $n$) or it serves a request. On the other hand, let $X$ be the set of nodes that had a nonempty queue at time $t$. We claim that during the epoch, empties each node in $X$ at least once. This is true because in each phase of the epoch empties a node, and the nodes in $X$ have requests older than those in $V \setminus X$.

The fact that every node in $X$ is emptied at least once during the epoch implies that

$$z \geq u. \tag{6.4}$$

The number of requests unserved at time $t'$ is then at most

$$\begin{array}{rcll} u + (1-\epsilon)(t'-t) + \mu - z & \leq & (1-\epsilon)\delta n + \mu + u - \epsilon z & \text{by (6.3)} \\ & \leq & (1-\epsilon)\delta n + \mu + (1-\epsilon)u & \text{by (6.4)} \\ & \leq & (1-\epsilon)\delta n/\epsilon + \mu/\epsilon & \text{by induction.} \end{array}$$

To conclude the proof we need to show that the duration of each epoch is bounded by a constant. This is true because the duration of each phase is bounded (the proof, based on Lemma 6.2, is similar to that in the proof of Theorem 6.6). □

Two classical strategies for generic online service are and (Algorithms 16 and 17). Although for the purpose of minimizing the maximum flow time they perform equally badly from the point of view of competitive analysis, the following results establish as a more robust algorithm. These results are similar to those of Hauptmeier et al. [56].

**Algorithm 16**

maintains a shortest walk on the set of nodes that have unserved requests. Whenever the current node is nonempty, serves a request there. Otherwise, it moves the server along the shortest walk. Whenever a new request is released, the shortest walk is recomputed.

**Theorem 6.8.** *For every graph $G$ of diameter at least* 3 *there is a weak adversary $A$ such that* $(G, A, )$ *is unstable.*

**Algorithm 17**

works in phases. At the beginning of each phase, let $o$ be the current position of . computes a shortest schedule on the set of currently unserved requests starting and ending at $o$. During the phase, follows the schedule, ignoring temporarily requests released after the beginning of the phase. When the schedule has been completed, the phase ends.

*Proof.* Consider any chain $v_0v_1v_2v_3$ of length 3 in the graph, and suppose that the server starts in $v_0$. The first adversary $A_1$ gives the following requests: $(0, v_3)$, $(3, v_0)$, $(6, v_3)$, and $(9 + 3i, v_0)$ for all $i \geq 0$. The sequence is such that starting from time 9, the server will never reach $v_3$ again. The rate of $A_1$ is $1/3$; thus, by Lemma 6.3, for any sufficiently small $\epsilon > 0$ there exists an adversary $A_2$ of rate $\epsilon$ releasing requests in $v_3$ such that $A_1 \cup A_2$ has rate at most $2/3 + \epsilon$. The requests released by $A_2$ do not change the behavior of , since they are always scheduled after the requests released by $A_1$. Thus the requests of $A_2$ are never served and the system is unstable. □

**Theorem 6.9.** *is universally stable.*

*Proof.* We start by proving by induction that at the beginning of every phase the number of unserved request is bounded by

$$(1 - \epsilon)2n/\epsilon + \mu/\epsilon,$$

then we show that the duration of every phase is bounded. Let $[t, t')$ be a phase, let $z$ be the number of requests served during the phase and let $u$ be the number of unserved requests at the beginning of the phase. Then, by definition of ,

$$t' - t \leq 2(n - 1) + z \tag{6.5}$$

since when does not serve a request, its server moves along a shortest closed walk spanning all the requests unserved at time $t$, and this walk cannot be longer than twice the number of edges of a spanning tree of the graph. By definition, serves all those requests, which means that

$$z = u. \tag{6.6}$$

The number of requests unserved at time $t'$ is then at most

$$\begin{aligned} u + (1 - \epsilon)(t' - t) + \mu - z &\leq (1 - \epsilon)2n + \mu + u - \epsilon z && \text{by (6.5)} \\ &\leq (1 - \epsilon)2n + \mu + (1 - \epsilon)u && \text{by (6.6)} \\ &\leq (1 - \epsilon)2n/\epsilon + \mu/\epsilon && \text{by induction.} \end{aligned}$$

To conclude the proof we need to show that the duration of each phase is bounded by a constant. Again, this can be shown using Lemma 6.2. □

Finally, we propose an algorithm called - (Algorithm 18) that can be seen as a generalization of the algorithm Scan frequently used for disk scheduling [88]. Notice that - , differently from algorithms such as or , does not use phases and is greedy, in the sense that it always serves a request from the current location of the server if possible.

**Algorithm 18** -

Let $W$ be a closed Eulerian walk on the graph obtained by doubling all the edges of a spanning tree of $G$.
At every time step, if the current node has a nonempty queue, - serves a request on the current node. Whenever the current node $v$ has an empty queue, - moves the server to the node following $v$ in the walk $W$.

**Theorem 6.10.** - *is universally stable.*

*Proof.* The analysis is very similar to that of (Theorem 6.9), except that the proof is by induction on time steps instead of phases. Consider any time $t'$. Let $t$ be the latest time before $t'$ during which the server was located at $s(t')$ and such that between $t$ and $t'$ - visited the whole graph. Let $z$ be the number of requests served during $[t, t')$. Then

$$t' - t \leq z + 2(n-1), \tag{6.7}$$

since at each time step either the algorithm serves a request or it moves to the next node in the walk $W$. Also, let $u$ be the number of unserved requests at time $t$. The inductive hypothesis is that $u \leq 2(1-\epsilon)(n-1)/\epsilon + \mu/\epsilon$. All requests unserved at time $t$ are served at time $t'$ because the algorithm always serves requests when visiting a node and during the interval $[t, t')$ the server visited the entire graph; thus

$$z \geq u. \tag{6.8}$$

The number of requests unserved at time $t'$ is then at most

$$\begin{aligned} u + (1-\epsilon)(t'-t) + \mu - z &\leq 2(1-\epsilon)(n-1) + \mu + u - \epsilon z && \text{by (6.7)} \\ &\leq 2(1-\epsilon)(n-1) + \mu + (1-\epsilon)u && \text{by (6.8)} \\ &\leq 2(1-\epsilon)(n-1)/\epsilon + \mu/\epsilon && \text{by induction.} \end{aligned}$$

For the basis of the induction, we can consider a fictitious preliminary walk on the graph without pending requests. □

## 6.5 Results on maximum flow time

In this section, we investigate which algorithms among the ones we gave in Section 6.4 have bounded flow time. Since bounded flow time implies stability, the only

algorithms among the ones we have considered that may have bounded flow time are , , and - .

It is easy to see that does not have bounded flow time, since while its server is busy serving a pair of long queues, a single request can remain unserved in a stable queue.

**Theorem 6.11.** *has unbounded flow time.*

*Proof.* Consider a chain $v_0v_1v_2v_3v_4$ and suppose the server starts at $v_0$. Assume a weak adversary gives the cruel sequence $(0, v_2), (1, v_2), (2, v_0), (3, v_0), (4, v_4)$ followed by $(7+8i, v_2), (8+8i, v_2), (11+8i, v_0), (12+8i, v_0)$ for all $i \geq 0$. The longest queues are always at $v_0$ and $v_2$, and the single request at $v_4$ remains unserved forever. □

**Theorem 6.12.** , *and* - *have bounded flow time.*

*Proof.* : consider any request $\sigma_j$ entering the system at time $r_j$. Recall the proof of Theorem 6.7 and consider the time $t \geq r_j$ at which the next epoch begins in the execution of . During an epoch, all the nodes with nonempty queues are visited and emptied, so in particular $\sigma_j$ will be served. Since every epoch has bounded length, the claim follows.

: consider any request $\sigma_j$. Recall the proof of Theorem 6.9 and consider the time at which the next phase begins in the execution of . During a phase, all requests that were unserved at the beginning of the phase are served by , so in particular $\sigma_j$ will be served. Since every phase has bounded length, the claim follows.

- : consider any request $\sigma_j$ entering the system at time $r_j$. Let $k < 2n$ be the distance from the location of the server at time $r_j$ to $x_j$, the location of $\sigma_j$. By Theorem 6.10 and Lemma 6.2, there is a constant $u_{\max}$ such that - spends at most $(u_{\max} + \mu)/\epsilon$ time units on each node, where $1-\epsilon$ is the rate and $\mu$ the burst of the adversary. Thus, the server will reach and empty $x_j$ in at most $k(u_{\max} + \mu)/\epsilon$ time steps. □

## 6.6 A lower bound

In this section we give a lower bound on the number of unserved requests that any protocol may need to have at any time.

**Lemma 6.13.** *Let $P$ be any protocol and $G$ a graph of diameter $\delta$. For every $\epsilon \in (0, 1)$, there exists a weak adversary $A$ of rate $1-\epsilon$ such that eventually the number of requests pending at one of the nodes of $G$ is at least $(1-\epsilon)\delta/\epsilon$.*

*Proof.* Let $v, w \in V$ be two nodes at distance $\delta$ from each other. We break the construction of $A$ into phases. A *$v$-phase* (resp., *$w$-phase*) starts when the server moves to $v$ (resp., $w$) and ends when it moves to $w$ (resp., $v$). In a $v$-phase (resp., $w$-phase), $A$ gives requests in $w$ (resp., $v$) at rate $1-\epsilon$. We prove the claim by showing

that in any phase, either the total number of unserved request has to increase, or there are already more than $(1-\epsilon)\delta/\epsilon$ unserved requests.

Consider any $v$-phase. Let $u_v, u_w$ be the number of unserved requests at $v$ and $w$ at the start of the phase, and let $u'_v, u'_w$, be the number of unserved requests at $v$ and $w$ at the end of the same phase. If $z_v$ is the number of requests served during the phase at $v$, we have by construction

$$u'_v = u_v - z_v \tag{6.9}$$
$$u'_w = u_w + (1-\epsilon)(\delta + z_v) \tag{6.10}$$
$$z_v \leq u_v. \tag{6.11}$$

Suppose that the total number of unserved requests at $v$ and $w$ does not increase, that is

$$(u'_v + u'_w) - (u_v + u_w) \leq 0. \tag{6.12}$$

Combining (6.12) with (6.9) and (6.10),

$$(1-\epsilon)(\delta + z_v) - z_v \leq 0.$$

Solving for $z_v$, we get

$$z_v \geq (1-\epsilon)\delta/\epsilon$$

and by (6.11), we obtain

$$u_v \geq (1-\epsilon)\delta/\epsilon$$

so that the number of unserved requests at the beginning of the phase is already at least $(1-\epsilon)\delta/\epsilon$. The analysis for a $w$-phase is completely symmetrical. □

In case the graph $G$ is a chain, $\delta = n-1$, and the lemma shows that the upper bound for - (given in the proof of Theorem 6.10) is best possible up to a factor of 2.

## 6.7 Open problems

Our model suggests several open problems. First, the model can be easily generalized in several directions. One is that of directed (strongly connected) graphs. For directed graphs, an algorithm such as is still stable, but can we do better in terms of the worst case number of unserved requests? Notice that it is not obvious how to generalize - to directed graphs.

Another direction is that of considering the dial-a-ride problem, in which requests have both a source and a destination. In that case, of course, the rate constraint on the adversary should be reformulated appropriately. Such a model would be interesting, since in particular it could represent a quite fair model for elevator scheduling. Also, in our model the processing time of a request is directly related to the speed of the server: serving a request takes the same time as moving to a neighboring node. What about requests with arbitrary processing times?

Finally, in our opinion the most interesting problem suggested by our model is the following: consider the online server routing problem on a graph, where the objective is now *minimization of the maximum number of unserved requests at any time*. Can we find a constant competitive algorithm for this problem? The competitive ratio would necessarily depend on the characteristics of the graph (otherwise it is easy to prove that no such algorithm can exist). In any case, we think that such a competitive algorithm would be interesting because it would be able to maintain a near-optimal number of unserved requests not only under heavy load conditions (something that every stable algorithm already does), but even when the load is light.

# Bibliography


1. F. N. Afrati, S. S. Cosmadakis, C. H. Papadimitriou, G. Papageorgiou, and N. Papakostantinou. The complexity of the travelling repairman problem. *Theoretical Informatics and Applications*, 20(1):79–87, 1986.

2. S. Albers. On the influence of lookahead in competitive paging algorithms. *Algorithmica*, 18(3):283–305, 1997.

3. S. Albers. A competitive analysis of the list update problem with lookahead. *Theoretical Computer Science*, 197(1–2):95–109, 1998.

4. H. Alborzi, E. Torng, P. Uthaisombut, and S. Wagner. The *k*-client problem. *Journal of Algorithms*, 41(2):115–173, 2001.

5. L. Allulli, G. Ausiello, V. Bonifaci, and L. Laura. On-line algorithms, real time, the virtue of laziness, and the power of clairvoyance. In J.-Y. Cai, S. B. Cooper, and A. Li, editors, *Proc. 3rd Theory and Applications of Models of Computation*, volume 3959 of *Lecture Notes in Computer Science*, pages 1–20. Springer-Verlag, 2006.

6. L. Allulli, G. Ausiello, and L. Laura. On the power of lookahead in on-line vehicle routing problems. In *Proc. 11th Int. Computing and Combinatorics Conference*, pages 728–736, 2005.

7. M. Andrews, B. Awerbuch, A. Fernández, T. Leighton, Z. Liu, and J. M. Kleinberg. Universal-stability results and performance bounds for greedy contention-resolution protocols. *Journal of the ACM*, 48(1):39–69, 2001.

8. M. Andrews, M. A. Bender, and L. Zhang. New algorithms for disk scheduling. *Algorithmica*, 32(2):277–301, 2002.

9. S. Arora. Polynomial-time approximation schemes for euclidean TSP and other geometric problems. *Journal of the ACM*, 45(5):753–782, 1998.

10. S. Arora and G. Karakostas. Approximation schemes for minimum latency problems. *SIAM Journal on Computing*, 32(5):1317–1337, 2003.

11. N. Ascheuer, S. O. Krumke, and J. Rambau. Online dial-a-ride problems: Minimizing the completion time. In H. Reichel and S. Tison, editors, *Proc. 17th*

*Symp. on Theoretical Aspects of Computer Science*, volume 1770 of *Lecture Notes in Computer Science*, pages 639–650. Springer-Verlag, 2000.

12. G. Ausiello, V. Bonifaci, and L. Laura. The on-line asymmetric traveling salesman problem. In F. Dehne, A. López-Ortiz, and J. Sack, editors, *Proc. 9th Workshop on Algorithms and Data Structures*, volume 3608 of *Lecture Notes in Computer Science*, pages 306–317. Springer-Verlag, 2005. Submitted.

13. G. Ausiello, V. Bonifaci, and L. Laura. The online prize-collecting traveling salesman problem. Technical Report TR 08-06, Department of Computer and Systems Science, University of Rome "La Sapienza", Rome, Italy, 2006.

14. G. Ausiello, V. Bonifaci, S. Leonardi, and A. Marchetti-Spaccamela. Prize-collecting traveling salesman and related problems. In T. F. Gonzalez, editor, *Handbook of Approximation Algorithms and Metaheuristics*. CRC Press, to appear.

15. G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and approximation – Combinatorial optimization problems and their approximability properties*. Springer-Verlag, Berlin, 1999.

16. G. Ausiello, M. Demange, L. Laura, and V. Paschos. Algorithms for the on-line quota traveling salesman problem. *Information Processing Letters*, 92(2): 89–94, 2004.

17. G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, and M. Talamo. Algorithms for the on-line travelling salesman. *Algorithmica*, 29(4):560–581, 2001.

18. G. Ausiello, S. Leonardi, and A. Marchetti-Spaccamela. On salesmen, repairmen, spiders, and other traveling agents. In G. Bongiovanni, G. Gambosi, and R. Petreschi, editors, *Proc. 4th Italian Conference on Algorithms and Complexity*, volume 1767 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, 2000.

19. B. Awerbuch, Y. Azar, A. Blum, and S. Vempala. New approximation guarantees for minimum-weight $k$-trees and prize-collecting salesmen. *SIAM Journal on Computing*, 28(1):254–262, 1998.

20. B. Awerbuch, Y. Azar, S. Leonardi, and O. Regev. Minimizing the flow time without migration. *SIAM Journal on Computing*, 31(5):1370–1382, 2002.

21. E. Balas. The prize collecting traveling salesman problem. *Networks*, 19: 621–636, 1989.

22. S. Ben-David, A. Borodin, R. M. Karp, G. Tardos, and A. Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11(1):2–14, 1994.

23. D. Bienstock, M. X. Goemans, D. Simchi-Levi, and D. Williamson. A note on the prize collecting traveling salesman problem. *Mathematical Programming*, 59(3):413–420, 1993.

24. M. Bläser, B. Manthey, and J. Sgall. An improved approximation algorithm for the asymmetric TSP with strengthened triangle inequality. *Journal of Discrete Algorithms*, to appear.

25. M. Blom, S. O. Krumke, W. E. de Paepe, and L. Stougie. The online TSP against fair adversaries. *INFORMS Journal on Computing*, 13(2):138–148, 2001.

26. A. Blum, P. Chalasani, D. Coppersmith, W. R. Pulleyblank, P. Raghavan, and M. Sudan. The minimum latency problem. In *Proc. 26th Symp. on Theory of Computing*, pages 163–171, 1994.

27. A. Blum, R. Ravi, and S. Vempala. A constant-factor approximation algorithm for the *k*-MST problem. In *Proc. 28th Symp. on Theory of Computing*, pages 442–448, 1996.

28. V. Bonifaci. An adversarial queueing model for online server routing. Technical Report TR 07-06, Department of Computer and Systems Science, University of Rome "La Sapienza", Rome, Italy, 2006.

29. V. Bonifaci and L. Stougie. Online *k*-server routing problems. In T. Erlebach and C. Kaklamanis, editors, *Proc. 4th Workshop on Approximation and Online Algorithms*. Springer-Verlag, to appear. Invited to a special issue of Theory of Computing Systems.

30. A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

31. A. Borodin and R. El-Yaniv. On randomization in online computation. *Information and Computation*, 150:244–267, 1999.

32. A. Borodin, J. M. Kleinberg, P. Raghavan, M. Sudan, and D. P. Williamson. Adversarial queuing theory. *Journal of the ACM*, 48(1):13–38, 2001.

33. K. Chaudhuri, B. Godfrey, S. Rao, and K. Talwar. Paths, trees, and minimum latency tours. In *Proc. 44th Symp. on Foundations of Computer Science*, pages 36–45, 2003.

34. C. Chekuri and A. Kumar. Maximum coverage problem with group budget constraints and applications. In K. Jansen, S. Khanna, J. D. P. Rolim, and D. Ron, editors, *Proc. 7th Int. Workshop on Approximation Algorithms for Combinatorial Optimization*, volume 3122 of *Lecture Notes in Computer Science*, pages 72–83. Springer-Verlag, 2004.

35. C. Chekuri and M. Pál. An $O(\log n)$ approximation ratio for the asymmetric traveling salesman path problem. In *Proc. 9th Int. Workshop on Approximation Algorithms for Combinatorial Optimization*, to appear.

36. B. Chen and A. P. A. Vestjens. Scheduling on identical machines: How good is LPT in an on-line setting? *Operations Research Letters*, 21(4):165–169, 1997.

37. S. Y. Cheung and A. Kumar. Efficient quorumcast routing algorithms. In *Proc. of INFOCOM '94*, volume 2, pages 840–847, 1994.

38. N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA, 1976.

39. J. Cirasella, D. S. Johnson, L. A. McGeoch, and W. Zhang. The asymmetric traveling salesman problem: Algorithms, instance generators, and tests. In *Proc. 3rd Int. Workshop on Algorithm Engineering and Experimentation*, pages 32–59, 2001.

40. S. A. Cook. The importance of the P versus NP question. *Journal of the ACM*, 50(1):27–29, 2003.

41. J. R. Correa and M. R. Wagner. LP-based online scheduling: From single to parallel machines. In *Integer programming and combinatorial optimization*, volume 3509 of *Lecture Notes in Computer Science*, pages 196–209. Springer-Verlag, 2005.

42. G. B. Dantzig, D. R. Fulkerson, and S. M. Johnson. Solution of a large-scale traveling salesman problem. *Operations Research*, 2:393–410, 1954.

43. W. E. de Paepe. *Complexity Results and Competitive Analysis for Vehicle Routing Problems*. PhD thesis, Technical University Eindhoven, The Netherlands, 2002.

44. J. Fakcharoenphol, C. Harrelson, and S. Rao. The $k$-traveling repairman problem. In *Proc. 14th Symp. on Discrete Algorithms*, pages 655–664, 2003.

45. E. Feuerstein and L. Stougie. On-line single-server dial-a-ride problems. *Theoretical Computer Science*, 268(1):91–105, 2001.

46. A. Fiat, R. Karp, M. Luby, L. A. McGeoch, D. Sleator, and N. E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, 1991.

47. A. Fiat and G. J. Woeginger, editors. *Online Algorithms: The State of the Art*. Springer-Verlag, 1998.

48. G. N. Frederickson, M. S. Hecht, and C. E. Kim. Approximation algorithms for some routing problems. *SIAM Journal on Computing*, 7(2):178–193, 1978.

49. A. M. Frieze, G. Galbiati, and F. Maffioli. On the worst-case performance of some algorithms for the asymmetric traveling salesman problem. *Networks*, 12(1):23–39, 1982.

50. M. R. Garey and D. S. Johnson. *Computers and intractability – An introduction to the theory of NP-completeness*. W. H. Freeman, 1979.

51. N. Garg. Saving an epsilon: a 2-approximation for the *k*-MST problem in graphs. In *Proc. 37th Symp. on Theory of Computing*, pages 396–402, 2005.

52. M. Goemans and J. Kleinberg. An improved approximation ratio for the minimum latency problem. *Mathematical Programming*, 82(1):111–124, 1998.

53. M. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, 1995.

54. R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

55. G. Gutin and A. P. Punnen, editors. *The Traveling Salesman Problem and its Variations*. Kluwer, Dordrecht, The Nederlands, 2002.

56. D. Hauptmeier, S. O. Krumke, and J. Rambau. The online dial-a-ride problem under reasonable load. In G. Bongiovanni, G. Gambosi, and R. Petreschi, editors, *Proc. 4th Italian Conference on Algorithms and Complexity*, volume 1767 of *Lecture Notes in Computer Science*, pages 125–136. Springer-Verlag, 2000.

57. J. A. Hoogeveen. Analysis of Christofides' heuristic: Some paths are more difficult than cycles. *Operations Research Letters*, 10(5):291–295, 1991.

58. S. Irani, X. Lu, and A. Regan. On-line algorithms for the dynamic traveling repair problem. *Journal of Scheduling*, 7(3):243–258, 2004.

59. P. Jaillet and M. Wagner. Online routing problems: Value of advanced information and improved competitive ratios. *Transportation Science*, 40:200–210, 2006.

60. R. Jothi and B. Raghavachari. Minimum latency tours and the *k*-traveling repairmen problem. In M. Farach-Colton, editor, *Proc. 6th Symp. Latin American Theoretical Informatics*, volume 2976 of *Lecture Notes in Computer Science*, pages 423–433. Springer-Verlag, 2004.

61. M. Jünger, G. Reinelt, and G. Rinaldi. The traveling salesman problem. In M. O. Ball, T. Magnanti, C. L. Monma, and G. Nemhauser, editors, *Network Models, Handbook on Operations Research and Management Science*, volume 7, pages 225–230. Elsevier, 1995.

62. B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47(4):214–221, 2000.

63. H. Kaplan, M. Lewenstein, N. Shafrir, and M. Sviridenko. Approximation algorithms for asymmetric TSP by decomposing directed regular multigraphs. *Journal of the ACM*, 52(4):602–626, 2005.

64. H. Kellerer, T. Tautenhahn, and G. J. Woeginger. Approximability and nonapproximability results for minimizing total flow time on a single machine. *SIAM Journal on Computing*, 28(4):1155–1166, 1999.

65. E. Koutsoupias and C. H. Papadimitriou. Beyond competitive analysis. *SIAM Journal on Computing*, 30(1):300–317, 2000.

66. S. O. Krumke. Online optimization: Competitive analysis and beyond. Habilitation Thesis, Technical University of Berlin, 2001.

67. S. O. Krumke, W. E. de Paepe, D. Poensgen, M. Lipmann, A. Marchetti-Spaccamela, and L. Stougie. On minimizing the maximum flow time in the online dial-a-ride problem. In T. Erlebach and G. Persiano, editors, *Proc. 3rd Workshop on Approximation and Online Algorithms*, volume 3879 of *Lecture Notes in Computer Science*, pages 258–269. Springer, 2005.

68. S. O. Krumke, W. E. de Paepe, D. Poensgen, and L. Stougie. News from the online traveling repairman. *Theoretical Computer Science*, 295(1-3):279–294, 2003.

69. S. O. Krumke, W. E. de Paepe, D. Poensgen, and L. Stougie. Erratum to "news from the online traveling repairman". *Theoretical Computer Science*, 352(1-3):347–348, 2006.

70. S. O. Krumke, L. Laura, M. Lipmann, A. Marchetti-Spaccamela, W. E. de Paepe, D. Poensgen, and L. Stougie. Non-abusiveness helps: an $O(1)$-competitive algorithm for minimizing the maximum flow time in the online traveling salesman problem. In K. Jansen, S. Leonardi, and V. V. Vazirani, editors, *Proc. 5th Int. Workshop on Approximation Algorithms for Combinatorial Optimization*, volume 2462 of *Lecture Notes in Computer Science*, pages 200–214. Springer-Verlag, 2002.

71. S. O. Krumke, N. Megow, and T. Vredeveld. How to whack moles. In K. Jansen and R. Solis-Oba, editors, *Proc. 1st Workshop on Approximation and Online Algorithms*, volume 2909 of *Lecture Notes in Computer Science*, pages 192–205. Springer, 2003.

72. H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics*, 2:83–97, 1955.

73. E. L. Lawler, J. K. Lenstra, A. Rinnooy Kan, and D. B. Shmoys, editors. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Wiley, Chichester, England, 1985.

74. S. Leonardi and D. Raz. Approximating total flow time on parallel machines. In *Proc. 29th Symp. on Theory of Computing*, pages 110–119, 1997.

75. M. Lipmann. *On-Line Routing*. PhD thesis, Technical University Eindhoven, The Netherlands, 2003.

76. M. Lipmann, 2005. Personal communication.

77. M. Lipmann, X. Lu, W. E. de Paepe, R. A. Sitters, and L. Stougie. On-line dial-a-ride problems under a restricted information model. *Algorithmica*, 40 (4):319–329, 2004.

78. M. Manasse, L. A. McGeoch, and D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11(2):208–230, 1990.

79. C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization, Algorithms and Complexity*. Prentice-Hall, Englewood Cliffs, NJ, 1982.

80. C. H. Papadimitriou and S. Vempala. On the approximability of the traveling salesman problem. In *Proc. 32nd Symp. on Theory of Computing*, pages 126–133, 2000. See also a corrected version available at http://www.cs.berkeley.edu/~christos/.

81. C. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation. *Algorithmica*, 32(2):163–200, 2002.

82. D. Poensgen. *Facets of Online Optimization*. PhD thesis, Technical University Berlin, Germany, 2003.

83. G. Reinelt. *The Traveling Salesman: Computational Solutions for TSP Applications*, volume 840 of *Lecture Notes in Computer Science*. Springer, Berlin, 1994.

84. S. Sahni and T. F. Gonzalez. P-complete approximation problems. *Journal of the ACM*, 23(3):555–565, 1976.

85. J. Sgall. On-line scheduling. In A. Fiat and G. J. Woeginger, editors, *Online Algorithms: the State of the Art*, pages 196–231. Springer-Verlag, 1998.

86. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

87. L. Stougie and A. P. A. Vestjens. Randomized on-line scheduling: How low can't you go? *Operations Research Letters*, 30(2):89–96, 2002.

88. T. J. Teorey and T. B. Pinkerton. A comparative analysis of disk scheduling policies. *Communications of the ACM*, 15(3):177–184, 1972.

89. P. Toth and D. Vigo, editors. *The Vehicle Routing Problem*. Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2002.

90. V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, Berlin, 2001.

91. A. C. C. Yao. Probabilistic computations: towards a unified measure of complexity. In *Proc. 18th Symp. on Foundations of Computer Science*, pages 222–227, 1977.

# Summary

## Models and Algorithms for Online Server Routing

Combinatorial optimization is the discipline that studies problems in which one seeks to minimize or maximize an objective function by appropriately choosing the values of some variables from within an allowed finite set. In a typical combinatorial optimization problem, the feasibility of a solution can be efficiently verified, but the number of feasible solutions is so large that an exhaustive search of an optimal solution is doomed to failure. Thus, efficient combinatorial optimization algorithms need to exploit the structure of the problem being solved.

While the classical approach to a combinatorial optimization problem is to assume that all relevant data are available before a solution method is applied, it has recently become more and more evident that in many applications data arrive step by step and a partial solution needs to be maintained at every step. Typical examples of online problems are the scheduling of processes in an operating system, or the trade of stocks in a financial market. In these applications the data is arriving over time and the algorithm that solves the problem has to be online, meaning that it has to keep at every time a solution that has been produced without knowledge of future data. It is quite clear that, because of this lack of information about the future, an online algorithm will not in general be able to produce the optimal solution. Competitive analysis is a theoretical framework that allows to quantify the worst-case suboptimality of the solutions found by an online algorithm. An online algorithm is called competitive if it produces solutions whose cost is always within a constant factor of the optimal solution.

In this thesis we study competitive algorithms for server routing problems. In a server routing problem, one or more servers move in a metric space in order to visit some requested points in the space. The objective is to minimize some function of the movement of the servers. An important example is the traveling salesman problem, in which a salesman has to find a round-trip tour through a set of cities in order to minimize the total length of the tour. We consider online versions of this and other server routing problem, in which the points to be visited are released over time.

After giving a brief introduction to the field of online optimization in the first chapter of this thesis, in Chapter 2 we review the basic complexity results for offline server routing problems, we introduce formally the online server routing frame-

work and we survey the state of the art. We show the basic proof techniques and we discuss several attempts in the literature to extend the basic competitive analysis setting.

In Chapter 3, we consider the online asymmetric traveling salesman problem from the point of view of competitive analysis. For the homing version, where the server has to return to its starting point, we give an algorithm that has the best possible competitive ratio. We also consider the nomadic version (where returning to the starting point is not mandatory) and prove that it does not admit constant competitive algorithms. However, for the nomadic version we prove a competitive ratio as a function of the amount of asymmetry of the space. We also consider the competitiveness of zealous algorithms, in which, intuitively, the server is not allowed to remain idle when there are outstanding requests. Finally we discuss the issue of polynomial time online algorithms for the problem.

In Chapter 4, we study the online prize-collecting traveling salesman problem. After discussing the approximation ratio of the offline version, we give a 7/3-competitive algorithm. We also consider the special case of the halfline as the metric space, for which we prove lower and upper bounds of 1.89 and 2, respectively, on the competitive ratio of deterministic algorithms.

In Chapter 5, we consider the online nomadic traveling salesman and the online traveling repairman with $k$ servers. We give competitive algorithms whose competitive ratios match the ones for the single server variants. For the special case of the real line, we prove the existence of algorithms with competitive ratio $1 + O((\log k)/k)$, meaning that we can approach the optimal cost as $k$ grows. We also show that this phenomenon is limited to the one dimensional case, since already in the Euclidean plane, we prove a lower bound of 4/3 for the online nomadic TSP and of 5/4 for the online TRP independently of the number of servers. Finally, we give resource augmentation results that are asymptotically best possible as the number of online servers grows beyond the number of offline servers.

In Chapter 6, in order to address the limits of competitive analysis, we introduce a new model for online server routing based on adversarial queueing theory. The model addresses the stability of online algorithms that are continuously operating. We call an online algorithm stable if there exists an upper bound on the number of unserved requests at any time that does not depend on the time the system has been running. We consider a number of natural algorithms in this model and we prove the existence of algorithms that are stable and such that the maximum flow time of a request also does not depend on the time the system has been running.

# Curriculum Vitae

Vincenzo Bonifaci was born on March 5th 1978 in Rome, Italy. In 1997, he received his diploma from the Liceo Scientifico Nomentano in Rome. In September of the same year he started studying Computer Engineering at the University of Rome "La Sapienza". He graduated cum laude in March 2003, on the subject of online algorithms for metrical service systems, under the supervision of Giorgio Ausiello.

In November 2003 Vincenzo started as a Ph.D. student at the University of Rome "La Sapienza" in a joint Ph.D. project with Eindhoven University of Technology. He worked under the supervision of Giorgio Ausiello, Jan Karel Lenstra and Leen Stougie. The joint Ph.D. project was partially supported by the Dutch Ministry of Education, Culture and Science through a Christiaan Huygens scholarship. The results of his research are presented in this thesis.