Simple algorithms for online dial-a-ride problems on the line

van den Broek, Steven W.

Award date:
2021

Eindhoven University of Technology

Bachelor Thesis

Simple algorithms for online dial-a-ride problems on the line

Author:
Steven van den Broek

Supervisors:
Dr. Kelin Luo
Prof. Dr. Frits Spieksma

Combinatorial Optimization
Department of Mathematics and Computer Science

July, 2021

Abstract

In the online dial-a-ride problem, requests for rides arrive over time. Each request has a release time and specifies two points in a metric space representing the pick-up and drop-off locations for the ride. These requests are handled by a server with a certain capacity indicating the number of requests it can handle simultaneously. At time 0, the server is at a distinguished point in the metric space called the origin, and it needs to return there after having served all requests. The goal is to find a so-called online algorithm for controlling the server which performs well according to a specified objective.

We consider special cases of the problem where the metric space is the real line, the server has unit or infinite capacity, and all pick-up locations or all drop-off locations are the origin. Our interest lies in the performance of two simple algorithms: HOMESICK and ALTERNATE. HOMESICK is analyzed for the unit capacity problem settings and ALTERNATE for the infinite capacity settings. HOMESICK serves requests one-by-one in the order that they arrive and never lets the server stay idle when there are unserved requests. The ALTERNATE algorithm behaves similarly as HOMESICK, but makes use of the server's infinite capacity to serve all requests on the same side of the line at once, moving to the side that contains the earliest released request. For both algorithms, we conduct a competitive analysis with the objectives of minimizing the make-span and minimizing the maximum flow time. This yields four unit capacity settings and four infinite capacity settings for which the HOMESICK and ALTERNATE algorithms are analyzed respectively. The HOMESICK algorithm performs well, for three out of the four unit capacity settings, the HOMESICK algorithm has the best possible competitive ratio. For all the infinite capacity settings, there are gaps between the best lower bound we constructed and the ALTERNATE algorithm. Nevertheless, ALTERNATE is a simple algorithm and provides a basis for research into the performance of more complex algorithms.

Aside from competitive analysis, we also briefly consider the problem of minimizing the make-span when all requests are known in advance, and we investigate the expected flow time of the two algorithms by making assumptions on the probability distribution of the input.

Contents

Chapter 1

Introduction

Dial-a-ride transit, also called demand-responsive transport, is a type of transportation service where people can request a ride without planning it in advance. A bus or taxi drives around in an area without a fixed schedule or route, and its task is to pick up people that requested a ride and drive them to their destinations. This service is used in places where traditional public transport is not financially feasible, or for providing transport to passengers with special needs. Because of the flexible and on-demand nature of this service, the question arises: how to determine an effective route that is efficient for both the driver of the vehicle and the passengers?

The dial-a-ride transportation service inspired two mathematical problems: the dial-a-ride problem (DARP) and the online dial-a-ride problem (OLDARP). The DARP assumes that all requests are known beforehand, and is about finding ways to efficiently calculate an optimal route or approximate one well. The OLDARP, on the other hand, more closely resembles the real-world dial-a-ride service; requests are not known in advance, but become known over time. In the OLDARP, there is a server that needs to carry out a sequence of requests; each request becomes known to the server at its release time and specifies a pick-up and a drop-off location. The server has the task to pick up* each request at its pick-up location and drop it off* at its drop-off location; the number of requests the server can carry simultaneously is constrained by the server's capacity. After serving all requests, the server is required to return to the place from which it started, which is called the origin.

Besides considering the problem in a 2-dimensional plane that models the dial-a-ride transportation service, other spaces like the line have been considered as well. The motivation for analyzing the problem on a line is that it models elevators. The elevator takes the role of the server, and people wanting to make use of the elevator arrive unannounced. The users of the elevator want to be picked up at their current floor and dropped off at their destination floor.

In this report, we will consider special cases of the OLDARP on the line. Namely, the cases where either the pick-up locations of all requests are the origin, or the drop-off locations of all requests are the origin. Furthermore, we consider two extremes regarding the capacity of the server: either the server can only carry one request, or it can carry infinitely many. To view these problems as a real-life situation, one can think of an elevator system at a company and consider the start and end of the workday. At the start of the workday, employees will all want to be picked up at the ground floor (the origin). Similarly, at the end of the workday, the destination of all employees will be the ground floor. One can also think of a waiter at a one-dimensional restaurant, as shown in Figure 1.1.

* We say a request is picked up or dropped off to mean that the corresponding object or person is picked up or dropped off.

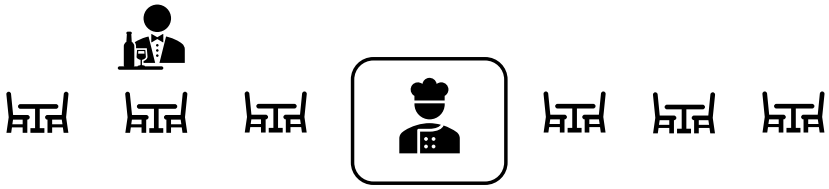
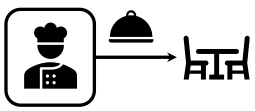
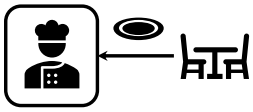


Figure 1.1: Waiter at the restaurant

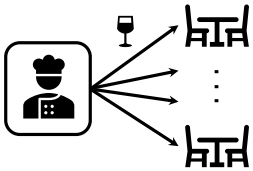
In this restaurant, the kitchen is at the origin and tables stand on both sides of it. Customers at these tables have requests for the waiter. For example, we can think of a waiter whose sole task is to bring meals from the kitchen to the tables (Figure 1.2a), or one who takes orders at the tables and notifies the cook in the kitchen (Figure 1.2d).



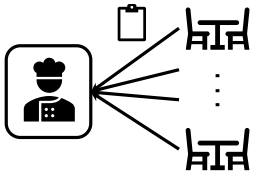
(a) Serving meals (all pick-up locations are the origin, unit capacity)



(b) Collecting dirty plates (all drop-off locations are the origin, unit capacity)



(c) Serving drinks (all pick-up locations are the origin, infinite capacity)



(d) Taking orders (all drop-off locations are the origin, infinite capacity)

Figure 1.2: Different problem settings

For these different problem settings, we consider simple algorithms for controlling the server and investigate their performance. Such a performance analysis requires a measure of performance: an objective. The focus in this report lies on the following two objectives:

- Minimizing the make-span. This objective minimizes the completion time of the server: the time it has served all requests and is back at the origin.
- Minimizing the maximum flow time. This objective minimizes the longest time a request needs to wait from the time it is released until it has been dropped off at its destination.

The intuition behind minimizing the make-span is that a server would like to be finished with all requests as soon as possible. On the other hand, minimizing the maximum flow time is desirable from the perspective of the people submitting a request. In the example of a waiter at a restaurant, people want their orders to arrive at the kitchen as soon as possible, and want minimal time between their meal being ready and them receiving it at their table.

In the subsequent sections, we will give the method used in this report for analyzing algorithms, a literature overview, and lastly an outline of the report.

1.1 Method

In this section, we introduce the method used in the main part of the report: competitive analysis. In competitive analysis, the quality of an algorithm is assessed by comparing the objective value of its solutions with the optimal objective value that can be computed with full knowledge of the input. We use the following concepts and conventions from [4] and [7] regarding competitive analysis of minimization problems:

- We call an algorithm *online* if it only becomes aware of a request at its release time. Similarly, we say an algorithm is *offline* if it is aware of all requests at any time.
- We say an algorithm is *randomized* if randomness may be used to determine the course of action. Similarly, an online algorithm is *deterministic* if no randomness is involved.
- Let ALG be a deterministic algorithm and σ a sequence of requests, then $\text{ALG}(\sigma)$ denotes the objective value that the server moved by ALG achieves for input σ. Since we consider multiple objectives in this report, subscripts will be used to clarify which objective is meant.
- We call a deterministic algorithm ALG_1 *optimal* if for all request sequences σ we have $\text{ALG}_1(\sigma) \leq \text{ALG}_2(\sigma)$ for any algorithm ALG_2.
- We use OPT to denote an arbitrary optimal algorithm. Note that the optimal algorithm can change depending on the constraints and objective function that is considered.
- A deterministic online algorithm is (strictly) ρ-competitive if for all sequences of requests σ we have that $\text{ALG}(\sigma) \leq \rho \cdot \text{OPT}(\sigma)$. Note that here $\rho \geq 1$ since by definition no algorithm can perform better than the optimal algorithm.
- The *competitive ratio* of a deterministic online algorithm ALG is the infimum over all ρ for which ALG is ρ-competitive.

This last piece of terminology, the competitive ratio, is the quality measure of an online algorithm in competitive analysis. The competitive ratio of an online algorithm gives a performance guarantee: no matter the input, the objective value of a solution constructed by the algorithm is never larger than the competitive ratio multiplied by the optimal objective value. Furthermore, the competitive ratio is the smallest value for which this property holds.

When doing a competitive analysis, one is often interested in finding the lowest possible competitive ratio and the corresponding algorithm that achieves it. For this, we define the best possible competitive ratio as the infimum over all competitive ratios of deterministic online algorithms (we do not consider randomized algorithms). Finding such a best possible competitive ratio is approached from two sides by constructing lower bounds and upper bounds.

A lower bound is proven by providing a set of instances on which no algorithm can perform well. If there is a lower bound of L, then that means that for any deterministic online algorithm ALG there exists an input σ for which $\text{ALG}(\sigma) \geq L \cdot \text{OPT}(\sigma)$. This sequence of requests σ can be different for each algorithm. Constructing such a lower bound can be seen as a game between the online algorithm and a malicious adversary. This adversary knows the workings of the online algorithm, and will construct an input for which the online algorithm performs badly, but the optimal algorithm performs well.

For an upper bound U on the best possible competitive ratio, one needs to find an algorithm and prove that it is U-competitive. Such a U is an upper bound because it shows a competitive ratio of U is achievable, therefore the best possible competitive ratio is then at most U.

1.2 Related work

In this section, we give an overview of research that has been done on problems related to the ones investigated in this report. As far as I am aware, no literature has been published on the exact problem variants of the OLDARP that are considered in this report. However, the OLDARP on the line with no constraints on the pick-up and drop-off locations is well-studied. Furthermore, the infinite capacity problems in this report are similar to the broadcast scheduling problem and the online travelling salesman problem (OLTSP), which we briefly discuss. The OLTSP is a special case of the OLDARP, just like the problem variants we consider in this report are special cases of the OLDARP. Note that lower bounds on the competitive ratio of any deterministic algorithm for a special case of the OLDARP also hold for the general OLDARP. Similarly, if an algorithm is ρ-competitive for the OLDARP then it is also ρ-competitive for special cases of the OLDARP, though it may have a lower competitive ratio.

The OLTSP on the line

In the OLTSP on the line, requests are released over time that specify points on the line that need to be visited by the server. This problem arises out of the OLDARP if the pick-up locations and drop-off locations of each request coincide. The OLTSP was first introduced in [3] by G. Ausiello et al. For the real line, the authors presented a lower bound of approximately 1.6404 on the competitive ratio of any deterministic algorithm, and introduced an algorithm that has a competitive ratio of 1.75. In [6], A. Bjelde et al. provided an algorithm with a competitive ratio of 1.6404, matching the lower bound and thereby settling the problem in the sense of traditional competitive analysis.

For the objective of minimizing the maximum flow time, Krumke et al. [17] showed that there is no randomized, and therefore also no deterministic, algorithm that can achieve a constant competitive ratio for the OLTSP on the real line. This result holds even in a "fair" setting where the optimal server may not move outside the convex hull of the released requests. The authors also study a "non-abusive" setting where the adversary is not allowed to move to the left or right if there are no unserved requests at that side. For this "non-abusive" setting, they introduce the algorithm DTO (detour) that obtains a competitive ratio of 8. The restriction does not make the problem trivial, a lower bound of 2 was given on the competitive ratio any deterministic algorithm can achieve.

The OLTSP has also been investigated for the objective of minimizing the weighted sum of completion times (see [16] and [13]). The problem for this objective, which is also referred to as minimizing latency, is often called the online travelling repairman problem.

The relevant results of the OLTSP from the literature are summarized in Table 1.1.

Objective	Lower bound	Upper bound
Minimize make-span	1.6404 [3]	**1.6404** 'Algorithm 1', [6]
Minimize maximum flow	∞ [17]	—

Table 1.1: Lower and upper bounds for the best possible competitive ratio of a deterministic algorithm for the closed OLTSP on the real line for different objectives. For every bound a reference is given, and for upper bounds the name of the algorithm that achieved the bound is given. An upper bound ρ is written in boldface when the given algorithm is not only ρ-competitive, but also has a competitive ratio of ρ on the real line.

The OLDARP on the line

To my knowledge, the algorithms we consider in this report have not been analyzed for the OLDARP. For different objective and capacities, we list the best lower bounds and the algorithms that achieve the best known competitive ratio.

For the OLDARP on the line, most research has been done regarding the objective of minimizing the make-span. For finite capacity, N. Ascheuer et al. [2] introduced the algorithm SMARTSTART. This result holds even for a generalization of the OLDARP with k servers with arbitrary finite capacities and on arbitrary metric spaces. Essentially, the SMARTSTART algorithm functions by computing optimal offline solutions for unserved requests. When the server is executing such a computed schedule, it ignores all incoming requests until it has finished its schedule. Furthermore, SMARTSTART lets the server be idle for some time when the computed schedule 'takes too long to serve'. In the same paper that SMARTSTART was introduced, the authors proved that it is 2-competitive. A. Birx and Y. Disser [5] gave a matching instance to prove that SMARTSTART has a competitive ratio of 2.

The best known lower bound, 1.7636, for the case where the server has finite capacity was proven in the thesis by A. Birx [4]. For infinite capacity, the best known lower bound follows from the lower bound from the OLTSP, namely 1.6404. The algorithm TIR (Temporarily Ignore Requests), introduced by E. Feuerstein and L. Stougie in [12], has the best known competitive ratio of 2 for infinite capacity on general metric spaces and on the real line. The TIR algorithms works as follows. When the TIR server is at the origin and there are unserved requests, the TIR algorithm computes an optimal offline route for serving all unserved requests. When following such a route, the TIR algorithm temporarily ignores requests if their pick-up and drop-off location are closer to the origin than the current position of the server. If either the pick-up or drop-off location are further from the origin than the server, then the algorithm orders the server to return to the origin and compute a new optimal offline route.

As the OLTSP is a special case of the OLDARP, also for the OLDARP no algorithms can exist with a constant competitive ratio for minimizing the maximum flow. As for the OLTSP, research has also been done regarding the latency objective for the OLDARP on the line [12] [16].

Table 1.2 summarizes the relevant results from the literature regarding the OLDARP on the line.

Objective	Constraint	Lower bound	Upper bound
Minimize make-span	$c < \infty$	1.7636 [4]	**2** SMARTSTART, [2]
	$c = \infty$	1.6404 [3]	**2** TIR, [12]
Minimize maximum flow		∞ [17]	—

Table 1.2: Lower and upper bounds for the best possible competitive ratio of a deterministic algorithm for the closed non-preemptive OLDARP on the real line for different objectives. For every bound a reference is given, and for upper bounds the name of the algorithm that achieved the bound is given. An upper bound ρ is written in boldface when the given algorithm is not only ρ-competitive, but also has a competitive ratio of ρ on the real line.

Broadcast scheduling

In broadcast scheduling there is a fixed number of different pages present at a server. Requests for pages arrive over time, and it is the job of the server to broadcast pages in order to complete the requests. When a server broadcasts a page, all requests for that page are satisfied. The main values of interest are the response times, i.e. the difference between the release time of a request and the time the corresponding page has been broadcast. The average response time objective has been investigated in [11] for the case where all pages have the same size. In [8], the maximum response time was investigated for unit sized pages. The authors give a proof that serving pages in a first-in-first-out order is 2-competitive. C. Chekuri et al. showed in [9] that the result holds even when page sizes are different.

Minimizing maximum response time in the broadcast problem has clear similarities to minimizing maximum flow time for the infinite capacity settings we consider in this report. Consider, for example, the case where all pick-up locations are the origin. One can view the negative part and positive part of the line as two different pages. When the server moves to serve requests on the positive part of the line, then it resembles the broadcasting of the respective page. The difference is that in the OLDARP, requests on the same part of the line can have different drop-off locations, while broadcasting a page takes a fixed amount of time. Furthermore, in the OLDARP the server needs to return to the origin after dropping off requests.

1.3 Report outline

The remaining part of the report is organized as follows.

In Chapter 2, we will give a more mathematical problem description and introduce the two main algorithms analysed in this report: HOMESICK and ALTERNATE. HOMESICK is analyzed for the unit capacity problem settings and ALTERNATE for the infinite capacity settings. HOMESICK serves requests one-by-one in the order that they arrive and never lets the server stay idle when there are unserved requests. The ALTERNATE algorithm behaves similarly as HOMESICK, but makes use of the server's infinite capacity to serve all requests on the same side of the line at once, moving to the side that contains the earliest released request.

In Chapter 3 we will use competitive analysis to analyze the HOMESICK and ALTERNATE algorithms. These algorithms are analyzed for different objectives, minimizing make-span and minimizing maximum flow, and for different settings, all pick-up locations are the origin or all drop-off locations are the origin. Furthermore, we investigate the offline problems for the objective of minimizing make-span. Algorithms are given that compute solutions with the optimal objective value.

In Chapter 4 the HOMESICK and ALTERNATE algorithms will be analyzed from a stochastic point of view. Assumptions will be made on the probability distribution of the inputs. In this way, the steady state behaviour of the HOMESICK and ALTERNATE algorithms can be investigated, and performance measures like expected flow time can be calculated.

Lastly, in Chapter 5 the report is concluded by discussing the results and providing directions for further research.

Chapter 2

Problem Description

In this chapter, we elaborate on the problem that is investigated in this report. In Section 2.1 we provide the formal problem definition of the OLDARP and give the variants that we explore in this report. Section 2.2 introduces the two algorithms that have our main interest: HOMESICK and ALTERNATE.

2.1 Formal problem definition

In the OLDARP, a *server* has to serve a sequence of *requests* $\sigma = (r_1, \ldots, r_n)$ on a metric space (X, d). Each request is a triple $r_i = (t_i, a_i, b_i)$, where $t_i \in \mathbb{R}_{\geq 0}$ is the time request r_i is released (becomes known to the server), and $a_i \in X$ and $b_i \in X$ are the *pick-up* and *drop-off* locations respectively. The server travels with unit speed, meaning it can cover one distance unit in one time unit, and starts at some unique position called the origin $o \in X$. The task of the server is to pick up each request at the pick-up location after its release time, and drop off the request at its drop-off location. The server can only serve a certain number of requests at the same time, which is determined by its *capacity* $c \in \mathbb{N} \cup \{\infty\}$. We are interested in finding algorithms for controlling the server that perform well according to a certain objective.

When a server has picked up a request, it is not allowed to drop it before it has reached the corresponding drop-off location. This restriction is called *non-preemptive* in the literature; we do not consider the *preemptive* variant here. Furthermore, we require the server to return to the origin after serving all requests. In the literature this is referred to as the *closed* OLDARP problem, as opposed to the *open* problem where this restriction is not present.

There is a light restriction on the metric spaces that are allowed, namely that for all pairs (x, y) of points in X, the shortest path from x to y is continuous, formed by points in X and has length $d(x, y)$. This restriction is adapted from a paper by Ausiello et al. [3]. A metric space corresponding to an undirected edge-weighted graph, where the set of points X are the vertices of the graph and the distance between two vertices is their weight, is an example of a metric space that does not satisfy the property. This is because there is no continuous path between some vertices x and y that are connected by an edge. A server moving on the graph would then either be at vertex x or vertex y, but it could not be at any point in between. Of course, one could consider the metric space induced by such an undirected edge-weighted graph, such that there does exist a continuous path between the vertices such that all the points lie in the metric space. When we speak of arbitrary or general metric spaces, a metric space with continuous paths between all pairs of points $x, y \in X$ with length $d(x, y)$ is meant.

In this report, we consider the real line with Euclidean distance; so $X = \mathbb{R}$ and $d(x, y) = |x - y|$.

We define the origin as $o := 0$, and we sometimes refer to the negative and positive parts of the line as the left and right side respectively. We will consider the problems where the server has unit capacity (UC) or infinite capacity (IC), and where all pick-up locations are the origin (PO), or all drop-off locations are the origin (DO). The main objectives we consider are minimizing make-span (MS) and minimizing maximum flow (MF). This yields eight different problem settings, which we refer to with the acronyms given in Table 2.1. The reason we do not consider any finite capacity greater than one, is because my supervisor, Kelin Luo, has unpublished negative results for this case. These results show that for any finite capacity greater than one, there can be no algorithm with a constant competitive ratio for the objective of minimizing maximum flow when all pick-up locations are the origin. It is likely that the result also holds for the case where all drop-off locations are the origin.

Acronym	Constraint
UCPO-MS	$c = 1,\ \forall i : a_i = o$
UCDO-MS	$c = 1,\ \forall i : b_i = o$
ICPO-MS	$c = \infty,\ \forall i : a_i = o$
ICDO-MS	$c = \infty,\ \forall i : b_i = o$

(a) Objective: minimizing make-span

Acronym	Constraint
UCPO-MF	$c = 1,\ \forall i : a_i = o$
UCDO-MF	$c = 1,\ \forall i : b_i = o$
ICPO-MF	$c = \infty,\ \forall i : a_i = o$
ICDO-MF	$c = \infty,\ \forall i : b_i = o$

(b) Objective: minimizing maximum flow

Table 2.1: Acronyms of the problem settings

In the OLDARP, the algorithm controlling the server has to make decisions based on input which is given item by item. Such problems are called *online* problems, opposed to *offline* problems where the algorithm has full knowledge of the input. The offline variant of the OLDARP is called the dial-a-ride problem, or DARP. The online and offline terminology is also used for algorithms; online and offline algorithms are algorithms for the online and offline problem respectively. In this report, we will briefly consider the DARP, but we will mainly focus on the OLDARP. In the next section, we introduce the two online algorithms that form the main topic of this report.

2.2 Algorithms

For both the unit capacity and infinite capacity settings, we analyze a simple algorithm. For the unit capacity case, we analyze a first-in first-out (FIFO) algorithm which we denote with HOMESICK (Algorithm 1). Essentially, HOMESICK serves requests as soon as they become known in the order they were released, and makes the server return to the origin immediately after each request.

Algorithm 1: HOMESICK

Input : server that starts at o; requests arrive over time
$Q \leftarrow$ empty FIFO queue

repeat
  if a new request r_i becomes known **then**
    add r_i to Q
  if server is at the origin and Q is not empty **then**
    $r_i \leftarrow$ pop request from Q
    order the server to serve request r_i and return to the origin via the shortest route

For infinite capacity we introduce the algorithm ALTERNATE, described in Algorithm 2. The algorithm exploits the server's infinite capacity to serve all unserved requests on the same side of the line at once, moving to the side that contains the earliest released request.

The algorithm functions by keeping track of the released but yet unserved requests. These are separated into the multisets R^- and R^+; multisets are used instead of sets because two requests may have exactly the same release time, pick-up location and drop-off location. Multiset R^- keeps track of the requests with negative pick-up or drop-off location depending on the problem we consider. Similarly, multiset R^+ contains the requests with non-negative pick-up or drop-off location.

Algorithm 2: ALTERNATE

Input : server that starts at o; requests arrive over time
$R^- \leftarrow$ empty multiset
$R^+ \leftarrow$ empty multiset

repeat
  if a new request r_i becomes known **then**
    if $a_i + b_i < 0$ **then**
      add r_i to R^-
    else
      add r_i to R^+
  if server is at the origin and $R^- \cup R^+$ is not empty **then**
    $r_f \leftarrow$ a request with minimum release time from $R^- \cup R^+$ (break ties arbitrarily)
    if $r_f \in R^-$ **then**
      order the server to serve requests R^- and return to the origin via the shortest route
      $R^- \leftarrow$ empty multiset
    else
      order the server to serve requests R^+ and return to the origin via the shortest route
      $R^+ \leftarrow$ empty multiset

Note that HOMESICK and ALTERNATE are general in the sense that they are valid algorithms for the OLDARP on the line with no restrictions on the pick-up and drop-off locations of requests.

Chapter 3

Competitive Analysis

3.1 Overview of results

An overview of the results for the online problems is given in Table 3.1.

Objective	Constraint	Lower bound	Upper bound
Minimize make-span	$c = 1,\ \forall i : a_i = o$	1	**1** HOMESICK, Theorem 3.4
	$c = 1,\ \forall i : b_i = o$	1.5 Theorem 3.9	**1.5** HOMESICK, Theorem 3.12
	$c = \infty,\ \forall i : a_i = o$	1.5 Theorem 3.17	**2** ALTERNATE, Theorem 3.21
	$c = \infty,\ \forall i : b_i = o$	1.6404* Theorem 3.34, [3]	**2.5**† ALTERNATE, Theorem 3.39
Minimize maximum flow	$c = 1,\ \forall i : a_i = o$	1.414 Theorem 3.5	**2** HOMESICK, Theorem 3.8
	$c = 1,\ \forall i : b_i = o$	2 Theorem 3.13	**2** HOMESICK, Theorem 3.16
	$c = \infty,\ \forall i : a_i = o$	1.7208 Theorem 3.29	**5** ALTERNATE, Theorem 3.32
	$c = \infty,\ \forall i : b_i = o$	2 Theorem 3.40	**6** ALTERNATE, Theorem 3.43

Table 3.1: Lower and upper bounds for the best competitive ratios of deterministic algorithms for closed non-preemptive OLDARP on the real line for different objectives. For every non-trivial bound the corresponding lemma or theorem is given, and for upper bounds the name of the algorithm that achieved the bound is given. An upper bound is written in boldface when it is proven to be tight.

Observe that the HOMESICK algorithm has the best possible competitive ratio for 3 out of the 4 unit capacity problem settings. Only for the UCPO-MF setting, the HOMESICK algorithm may not be best possible (it cannot yet be ruled out that there exists a lower bound of 2 on the competitive ratio of any deterministic algorithm, however we deem it highly unlikely that this is the case).

* This is not our own result, but instead follows from an equivalence of the ICDO-MS and OLTSP problems.

† 'Algorithm 1' from [6] has the best possible competitive ratio, hence there is an upper bound matching the lower bound of 1.6404.

Among the four infinite capacity problems, the ICDO-MS problem stands out. The reason is that the ICDO-MS problem is equivalent to the closed OLTSP problem, therefore the results for that problem transfer to the ICDO-MS problem. 'Algorithm 1' from [6] has the best possible competitive ratio for closed OLTSP, and by extension ICDO-MS. The ALTERNATE algorithm has a competitive ratio of 2.5 for this problem. For each of the infinite capacity problem settings, we see there are gaps between the competitive ratio of ALTERNATE and the lower bound we found for the problem. In particular, the gaps are large for the settings where the objective is to minimize the maximum flow time.

The results of the offline problems are summarized in Table 3.2. The offline unit capacity problems have very simple solutions. The infinite capacity problems are a bit more difficult, for these the structure of solutions can be exploited to compute a solution using a dynamic program.

Objective	Constraint	Time complexity of best known optimal offline algorithm
Minimize make-span	$c = 1,\ \forall i : a_i = o$	$\mathcal{O}(n)^*$
	$c = 1,\ \forall i : b_i = o$	$\mathcal{O}(n \log n)$
	$c = \infty,\ \forall i : a_i = o$	$\mathcal{O}(n^4)^\dagger$
	$c = \infty,\ \forall i : b_i = o$	$\mathcal{O}(n^2)$

Table 3.2: Runtimes for solving the offline problems. Here, n is the number of requests.

* It takes $\mathcal{O}(n)$ to compute the trajectory with compatible loading matrix; the order in which requests need to be served can be computed in $\mathcal{O}(1)$ time.

† It is trivial to adapt the algorithm such that it runs in $\mathcal{O}(n^3)$ time.

3.2 Preliminaries

A list of problem specific assumptions and notation is given below.

- We assume that the sequence of requests σ is ordered by non-decreasing release times unless specified otherwise.
- An online algorithm receives requests from σ one-by-one. If multiple requests have identical release times, then they are provided to the online algorithm in order of σ.
- We use sets and sequences somewhat interchangeably. For example, for a request sequence $\sigma = (r_1, \ldots, r_n)$ we use
$$r_i \in \sigma \iff r_i \in \{r_1, \ldots, r_n\}.$$
Let $\rho = (s_1, \ldots, s_m)$ also be a sequence of requests, then we define the union of σ and ρ as
$$\sigma \cup \rho = (r_1, \ldots, r_n, s_1, \ldots, s_m) \text{ sorted by non-decreasing release time.}$$
- For a sequence of requests $\sigma = (r_1, \ldots, r_n)$ and $m \in \mathbb{N}$ we define for $m \leq n$ the subsequence until m with $\sigma_{\leq m} := (r_1, \ldots, r_m)$.
- For a sequence of requests $\sigma = (r_1, \ldots, r_n)$ and $T \in \mathbb{R}$ we define $\sigma_{\leq T} = \{r_i \in \sigma \mid t_i \leq T\}$; we define $\sigma_{<T}, \sigma_{=T}, \sigma_{>T}, \sigma_{\geq T}$ in a similar fashion. Note that this definition can conflict with the previous one since $\mathbb{N} \subset \mathbb{R}$, but it will be clear from the context which is meant.

It is useful to mathematically define what a solution to the dial-a-ride problem is. In the DARP, such a solution is what an algorithm should output. For the OLDARP, the solution is calculated piece by piece. A formal definition of a solution also allows us to formally define the objectives. We adapt the following definitions from the thesis by A. Birx [4]:

- The *trajectory* of a server is modelled as a sequence of M tuples $(q_j, x_j)_{j \in \{1,\ldots,M\}}$ where $q_j \in \mathbb{R}$ is a point on the line and $x_j \in \mathbb{R}_{\geq 0}$ is a waiting time at that position. Since the server starts and ends at the origin, the trajectory needs to satisfy $q_1 = o$ and $q_M = o$. Furthermore, $x_M = 0$ since there is no waiting time after reaching the final position.
- Aside from moving around, the server also has to pick up requests, and drop them off. For this the *loading matrix* $(L_i, U_i)_{i \in \{1,\ldots,n\}}$ is introduced, consisting of the times the server picks up and drops off every request $r_i \in \sigma$. Requests can only be dropped off after they have been picked up and the server has moved to the drop-off location, therefore $U_i \geq L_i + d(a_i, b_i)$. Furthermore, the capacity constraint of the server need to be satisfied, so for all requests r_i we have $|\{k \in \{1, \ldots, n\} \setminus \{i\} \mid L_i \in [L_k, U_k)\}| < c$.
- A trajectory $(q_j, x_j)_{j \in \{1,\ldots,M\}}$ is *compatible* with loading matrix $(L_i, U_i)_{i \in \{1,\ldots,n\}}$ if the server is at position a_i at time L_i and at position b_i at time U_i for all requests r_i.
- An offline solution for OLDARP is a trajectory with a compatible loading matrix, and we call such a solution a (feasible) *walk*.
- The completion time of a walk with trajectory $(q_j, x_j)_{j \in \{1,\ldots,M\}}$ is
$$\sum_{j=1}^{M-1} x_j + d(q_j, q_{j+1}).$$

We can now define the following things:

- We say a server is *idle* in a trajectory at time t if it is waiting at time t. Similarly, a server is *busy* in a trajectory at time t if it is not waiting at time t.

- The *completion time* of a request r_i in a walk is the time that it is dropped off by the server, so U_i. We often denote this completion time by C_i for the walk of the online algorithm we are considering, and we use C_i^* for the optimal algorithm OPT.
- The *flow time* of a request r_i in a walk is the difference between its completion time and its release time. This is denoted by F_i and F_i^*. So

$$F_i = C_i - t_i, \quad \text{and,} \quad F_i^* = C_i^* - t_i.$$

- The *make-span* of a solution, is the completion time of the corresponding walk. For a certain input σ and an algorithm ALG, the make-span is denoted by $\text{ALG}_{C_{\max}}(\sigma)$. We sometimes use the shorthands $C_{\max}$ and $C_{\max}^*$ for the online algorithm we are considering and OPT respectively. Note that $C_{\max}$ is *not* equivalent to $\max_{r_i \in \sigma} C_i$ when not all drop-off locations are the origin, since then the server may need to return to the origin after finishing all requests. Still, we use this notation since it is often used in the literature for denoting the make-span, both for machine scheduling problems and dial-a-ride problems (see e.g. [10], [15]).
- The *maximum flow time* of a walk is the maximum of all the flow times of the different requests. This is denoted by $\text{ALG}_{F_{\max}}(\sigma)$ for some algorithm ALG and input σ. Similarly as before, we use $F_{\max}$ and $F_{\max}^*$ as shorthands. We have

$$F_{\max} = \max_{r_i \in \sigma} F_i, \quad \text{and,} \quad F_{\max}^* = \max_{r_i \in \sigma} F_i^*.$$

3.2.1 How to read lower bound figures

Position-time diagrams, such as Figure 3.1, are used throughout the report for illustrating lower bounds on the competitive ratio of algorithms.

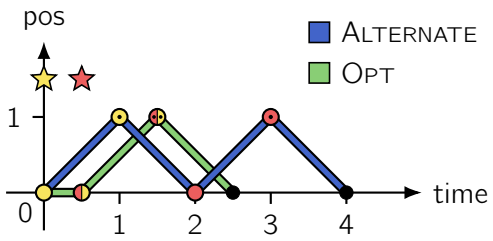


Figure 3.1: Example of position time diagram

These figures are inspired by similar position-time diagrams in the thesis by A. Birx [4]. A diagram is made for a particular input sequence of requests, and shows how different algorithms control the server for that input. The route the server travels when controlled by a particular algorithm is represented by a line, which is coloured differently for each algorithm. In addition, circles represent the moments and locations at which requests are picked up or dropped off by the server. Lastly, stars indicate the times at which requests are released; the height of a star has no meaning, they are simply positioned above the lines in the graph.

As an example, Figure 3.1 plots the workings of the ALTERNATE and OPT algorithms on the input $\sigma = (r_1, r_2)$, with $r_1 = (0, o, 1)$ and $r_2 = (0.5, o, 1)$. The symbols in Figure 3.1 are listed below, together with their meaning.

- ☆ Request r_1 is released, and is indicated using the color yellow.
- ★ Request r_2 is released, and is indicated using the color red.
- ○ Request r_1 is picked up by the ALTERNATE server.

- Request r_1 is dropped off by the ALTERNATE server.
- The OPT server picks up requests r_1 and r_2 at the same time.
- The OPT server drops off requests r_1 and r_2 at the same time.
- Request r_2 is picked up by the ALTERNATE server.
- Request r_2 is dropped off by the ALTERNATE server.

In the figure, the make-spans of the two algorithms are the times at which the lines end. Hence, we see that $\text{ALTERNATE}_{C_{\max}}(\sigma) = 4$ and $\text{OPT}_{C_{\max}}(\sigma) = 2.5$. For the flow times, we see that the flow time of request r_1 in the walk of ALTERNATE is $F_1 = 1$; similarly $F_2 = 2.5$. The flow times of requests r_1 and r_2 in the walk of OPT are $F_1^* = 1.5$ and $F_2^* = 1$ respectively. The maximum flow times of the algorithms are therefore $\text{ALTERNATE}_{F_{\max}}(\sigma) = 2.5$ and $\text{OPT}_{F_{\max}}(\sigma) = 1.5$.

3.2.2 Observations

Before considering specific problem settings and algorithms, we make three observations that will be useful throughout the competitiveness proofs. These observations hold for the general OLDARP, and therefore also for all special cases we consider. With slight abuse of notation, we use $\text{OPT}_{C_{\max}}(\sigma)$ and $\text{OPT}_{F_{\max}}(\sigma)$ to denote the optimal make-span and maximum flow time for input σ for any problem setting.

Our first observation is the following.

> *Observation* 3.1. Let σ be an arbitrary sequence of request, then
>
> $$\text{OPT}_{C_{\max}}(\sigma) \geq d(o, a_i) + d(a_i, b_i) + d(b_i, o), \qquad \text{for any request } r_i \text{ in } \sigma. \tag{3.1}$$

The above holds because the server starts and ends at the origin, and has to serve all requests. Hence, since request r_i must be dropped off at some moment in time, the optimal walk needs to include the movement from the pick-up location of request r_i to the drop-off location of r_i. Furthermore, since the server starts and ends at the origin, the walk must include the movement from the origin to the pick-up location of r_i, and from the drop-off location of r_i to the origin.

One can observe a similar trivial lower bound by making use of the fact that a request can only be picked up after its release time. After having picked up the request, the server will at least need to move to the drop-off location and then the origin in order to finish. Hence:

> *Observation* 3.2. Let σ be an arbitrary sequence of request, then
>
> $$\text{OPT}_{C_{\max}}(\sigma) \geq t_i + d(a_i, b_i) + d(b_i, o), \qquad \text{for any request } r_i \text{ in } \sigma. \tag{3.2}$$

Lastly, in order to complete a request, the server needs to at least wait until it is released and then move from the pick-up location to the drop-off location of the request. This yields the following observation on the flow time of requests.

> *Observation* 3.3. Let σ be an arbitrary sequence of requests and consider any algorithm serving input σ. Then for any request $r_i \in \sigma$ served by the algorithm we have:
>
> $$F_i = C_i - t_i \geq d(a_i, b_i), \tag{3.3}$$
>
> where F_i and C_i denote the flow and completion time of request r_i respectively.

3.2.3 Offline problems

In the offline setting, we are interested in finding optimal algorithms for the DARP, with which we mean algorithms that return solutions obtaining the optimal objective value. Such algorithms are also often called exact algorithms, but we use optimal for consistency with the definition in competitive analysis. For the four problem settings with the objective of minimizing make-span we consider the offline problem. For these problem settings, the offline problem will be discussed at the end of the corresponding sections. For the ICPO-MS problem a detailed analysis is given, for the other problem settings the solution method is only briefly mentioned.

Formally, a solution to the DARP can be represented by a trajectory with a compatible loading matrix, as introduced earlier in this section. The constraints of each of the different problem settings allow, however, for easier representation of solutions. For the UCPO-MS, UCDO-MS and ICDO-MS settings, a sequence denoting the order in which requests need to be served is sufficient for representing optimal solutions. Similarly, in the ICPO-MS setting a sequence of sets of requests is a simpler and sufficient representation of optimal solutions. We focus on the solution method; we deem the form of output less important.

3.3 The HOMESICK algorithm

In this section we perform a competitive analysis of the HOMESICK algorithm for the different problem settings. Recall that HOMESICK serves requests one-by-one in the order that they arrive and never lets the server stay idle when there are unserved requests. We will start by showing how the HOMESICK algorithm behaves on an example input sequence.

Example. Consider the sequence of requests $\sigma = (r_1, r_2, r_3)$ with $r_1 = (2, o, 1)$, $r_2 = (5, o, -1)$ and $r_3 = (6, o, 0.5)$. The server controlled by the HOMESICK algorithm starts at the origin at time 0. The server will stay there until a request is released. This happens at time $t_1 = 2$, at this point HOMESICK orders the server to pick up request r_1, drop it off, and return to the origin via the shortest route. The server arrives back at the origin at time 4, and will be ordered to wait there since there are no unserved requests. At time $t_2 = 5$ the second request is released, and the server will serve r_2. When the server is back at the origin at time 7, it can immediately serve request r_3. After serving request r_3 and the server is back at the origin at time 8, no more requests are released.

We see that the HOMESICK algorithm yields completion times $C_1 = 3$, $C_2 = 6$ and $C_3 = 7.5$ for requests r_1, r_2 and r_3 respectively. The flow times of the requests are calculated by subtracting the release times from the corresponding completion times; letting F_i denote the flow time of request r_i, we find

$$F_1 = C_1 - t_1 = 3 - 2 = 1; \quad F_2 = C_2 - t_2 = 6 - 5 = 1; \quad F_3 = C_3 - t_3 = 7.5 - 6 = 1.5.$$

At time 8, the server has served all requests and is back at the origin, therefore the make-span is $\text{HOMESICK}_{C_{\max}}(\sigma) = 8$. The maximum flow time of HOMESICK for input σ is

$$\text{HOMESICK}_{F_{\max}}(\sigma) = \max\{F_1, F_2, F_3\} = 1.5.$$

The behaviour of the HOMESICK server on input σ is shown in Figure 3.2. Two illustrations are given: a position-time diagram and a timeline. These two types of figures are used throughout the report to provide a visual representation of the behaviour of algorithms. Idle and busy periods of the server are also indicated in the figure. Recall that idle means that the server waits, not necessarily that there are no unserved requests. A server controlled by the Homesick algorithm, however, never stays idle when there are unserved requests.

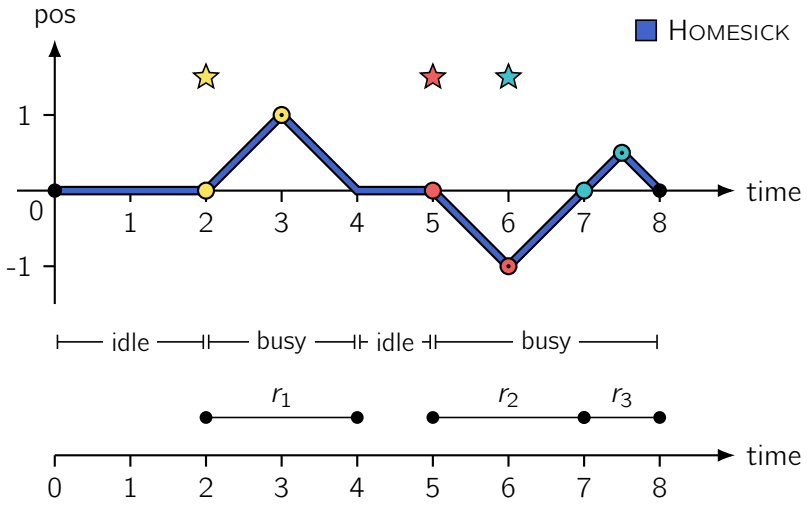


Figure 3.2: Example behaviour of the HOMESICK algorithm in the UCPO setting.

3.3.1 UCPO-MS

For this problem, HOMESICK is optimal, as is proven in the following theorem. This result also provides the solution to the offline problem, as is discussed at the end of the section.

Theorem 3.4. *HOMESICK is optimal for UCPO-MS.*

Proof. Let $\sigma = (r_1, \ldots, r_n)$ be an arbitrary sequence of requests. We use L_i to denote the time at which the HOMESICK server picks up request r_i (this corresponds to the loading matrix from the preliminaries in Section 3.2). Note that we have $L_1 = t_1$ since the server picks up and serves r_1 as soon as it is released. Furthermore, for any request r_i we have $L_i \geq t_i$ because requests cannot be picked up before they are released. Since the HOMESICK server picks up requests as soon as possible, $L_i > t_i$ implies that the server was not idle between time t_i and L_i. Now, define

$$f := \arg\max_{1 \leq i \leq n} \{L_i \mid L_i = t_i\} \quad \text{(if there is no unique index, pick the largest).}$$

Hence, r_f is the last request served by HOMESICK that was picked up at exactly the time it was released. Clearly f exists since $L_1 = t_1$.

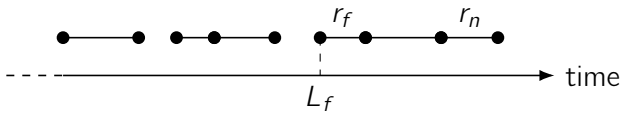


Figure 3.3: Example timeline of HOMESICK where L_f is the last time the server picked up a request the same time it was released.

Now, because for all $i > f$ we have $L_i > t_i$, the server was not idle after t_f. Therefore,

$$\text{HOMESICK}_{C_{\max}}(\sigma) = t_f + 2\sum_{i=f}^{n} d(o, a_i). \tag{3.4}$$

Next, we investigate the make-span of the OPT algorithm. Aiming for a similar expression as in Equation 3.4, we consider the requests $\sigma_{\geq f}$. Let r_h be the first request from $\sigma_{\geq f}$ that the OPT server serves. We know the OPT server can only pick up r_h after it has been released at time t_h. After the server has picked up r_h, it has to move to the drop-off location and back to the origin which takes at least $2d(o, a_h)$ time. Similarly, for any other requests in $\sigma_{\geq f}$ the server has to pick up the request at the origin, drop it off at the drop-off location, and return to the origin. Therefore we have

$$\text{OPT}_{C_{\max}}(\sigma) \geq t_h + 2\sum_{i=f}^{n} d(o, a_i).$$

Now, since $r_h \in \sigma_{\geq f}$ we have $t_h \geq t_f$ because σ is ordered on non-decreasing release times. Therefore,

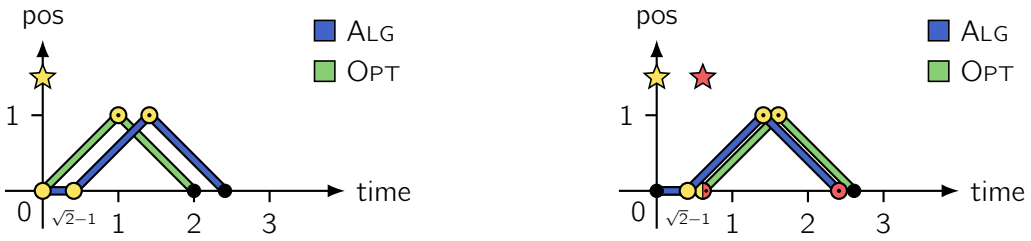
$$\text{HOMESICK}_{C_{\max}}(\sigma) = t_f + 2\sum_{i=f}^{n} d(o, a_i) \leq t_h + 2\sum_{i=f}^{n} d(o, a_i) = \text{OPT}_{C_{\max}}(\sigma).$$

□

Offline We have seen that serving requests as soon as they are released is optimal. Since the input sequence of requests is already sorted on release time, the offline solution can be 'computed' in $\mathcal{O}(1)$ time by simply returning the input. Calculating the exact trajectory and loading matrix can be done in $\mathcal{O}(n)$ time by stepping through the input sequence σ, keeping track of the necessary variables and doing basic arithmetic.

3.3.2 UCPO-MF

For the UCPO-MS problem, the HOMESICK algorithm is optimal. When minimizing the maximum flow time, no deterministic online algorithm can be optimal, as is apparent from the following theorem.



(a) Best ALG can do: adversary releases one request

(b) Best ALG can do: adversary releases two requests

Figure 3.4: Illustration of Theorem 3.5 with $\varepsilon = 0.2$

Theorem 3.5. *No deterministic online algorithm ALG for UCPO-MF has a competitive ratio lower than* $\sqrt{2} \approx 1.4142$.

Proof. Let ALG be an arbitrary deterministic online algorithm. The adversary releases request $r_1 = (0, o, 1)$ at time 0. The server operated by ALG has to pick up request r_1 at some time $T \geq 0$. We perform a case distinction on the value of T.

Case 1: $0 \leq T < \sqrt{2} - 1$. The adversary releases one other request $r_2 = (T + \varepsilon, o, o)$ for some small $\varepsilon \in (0, 1 - \sqrt{2}/2)$. In this case, the complete request sequence is $\sigma = (r_1, r_2)$. The server operated by ALG has picked up request r_1 at time T and cannot drop it to pick up request r_2 because we do not allow preemption. Hence, r_1 is dropped off at time at least $T + 1$ yielding flow time $F_1 = T + 1 < \sqrt{2}$, and r_2 is dropped off at time at least $T + 2$ yielding flow time $F_2 \geq T + 2 - (T + \varepsilon) = 2 - \varepsilon$. Hence $\text{ALG}_{F_{\max}}(\sigma) \geq 2 - \varepsilon$. The optimal algorithm OPT first serves r_2 at time $T + \varepsilon$, and then serves r_1. This yields flow times $F_1^* = T + \epsilon + 1$ and $F_2^* = 0$; hence $\text{OPT}_{F_{\max}}(\sigma) = T + \varepsilon + 1$. Hence ALG has competitive ratio at least

$$\frac{\text{ALG}_{F_{\max}}(\sigma)}{\text{OPT}_{F_{\max}}(\sigma)} \geq \frac{2 - \varepsilon}{T + \varepsilon + 1} \geq \frac{2 - \varepsilon}{\sqrt{2} + \varepsilon}.$$

Taking ε small enough, the lower bound can be made arbitrarily close to $\sqrt{2}$.

Case 2: $T \geq \sqrt{2} - 1$. The adversary releases no other requests, so the complete sequence is $\sigma = (r_1)$. The server operated by ALG picks up r_1 at time T, then moves to the drop-off location. Therefore $\text{ALG}_{F_{\max}}(\sigma) \geq T + 1$. An optimal algorithm OPT would pick up r_1 directly at time 0 yielding $\text{OPT}_{F_{\max}}(\sigma) = 1$. Hence ALG has competitive ratio at least

$$\frac{\text{ALG}_{F_{\max}}(\sigma)}{\text{OPT}_{F_{\max}}(\sigma)} \geq \frac{T + 1}{1} \geq \sqrt{2}.$$

□

In the previous theorem we have seen that no deterministic online algorithm can exist which is optimal in this problem setting. Now, we will analyze the HOMESICK algorithm and prove it has competitive ratio 2. First, we prove a lower bound of 2 on the competitive ratio of HOMESICK using a similar instance as in the previous theorem. Afterwards, we give a 2-competitiveness proof in which we use the optimality result of HOMESICK in the UCPO-MS problem setting.

Lemma 3.6. *HOMESICK has a competitive ratio of at least 2 for UCPO-MF.*

Proof. Consider instance $\sigma = (r_1, r_2)$ with requests $r_1 = (0, o, 1)$ and $r_2 = (\varepsilon, o, o)$ for some small $\varepsilon \in (0, 1/2)$. Then HOMESICK immediately serves request r_1, and directly afterwards r_2. This yields flow times $F_1 = 1$, $F_2 = 2 - \varepsilon$. Hence $\text{HOMESICK}_{F_{\max}}(\sigma) = 2 - \varepsilon$. The optimal algorithm OPT would first serve r_2, then r_1. This yields flow times $F_1^* = 1 + \varepsilon$, $F_2^* = 0$, therefore $\text{OPT}_{F_{\max}}(\sigma) = 1 + \varepsilon$. We can take ε arbitrarily small, hence HOMESICK has a competitive ratio of at least 2. □

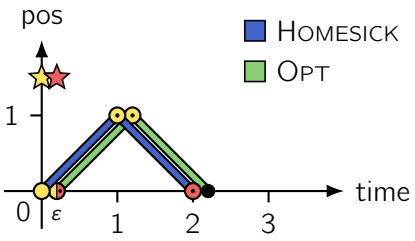


Figure 3.5: Illustration of Lemma 3.6 with $\varepsilon = 0.2$

Lemma 3.7. *HOMESICK is 2-competitive for UCPO-MF.*

Proof. Let $\sigma = (r_1, \ldots, r_n)$ be an arbitrary sequence of requests. For all $m \leq n$, let $\mathcal{P}(m)$ be the property that $\text{HOMESICK}_{F_{\max}}(\sigma_{\leq m}) \leq 2\text{OPT}_{F_{\max}}(\sigma_{\leq m})$. By using induction, we will prove that $\mathcal{P}(n)$ holds, thereby proving the lemma.

Base case: $m = 1$. As soon as r_1 is released the HOMESICK server will pick up the request, serve it, and return to the origin. Since the server operated by HOMESICK starts and stays at the origin, and the pick-up locations of all requests are the origin, the server can immediately pick up r_1. Therefore the flow time of r_1 is $F_1 = d(o, b_1)$. The optimal algorithm cannot do better than this, therefore

$$\text{HOMESICK}_{F_{\max}}(\sigma_{\leq 1}) = d(o, b_1) \overset{(3.3)}{\leq} \text{OPT}_{F_{\max}}(\sigma_{\leq 1}) \leq 2\text{OPT}_{F_{\max}}(\sigma_{\leq 1}),$$

and $\mathcal{P}(1)$ holds.

Induction hypothesis (IH). Suppose we have some $m < n$ such that property $\mathcal{P}(m)$ holds.

Inductive step. We prove that $\mathcal{P}(m+1)$ holds by using the induction hypothesis. Since HOMESICK serves requests in the same order as σ, the flow times $F_1, \ldots, F_m$ are the same on input $\sigma_{\leq m+1}$ and $\sigma_{\leq m}$. Therefore

$$\text{HOMESICK}_{F_{\max}}(\sigma_{\leq m+1}) = \max\{F_1, \ldots, F_{m+1}\} = \max\{\text{HOMESICK}_{F_{\max}}(\sigma_{\leq m}), F_{m+1}\}.$$

We will consider the two terms of the maximum separately. For the first term we know from the induction hypothesis that

$$\text{HOMESICK}_{F_{\max}}(\sigma_{\leq m}) \overset{(IH)}{\leq} 2\text{OPT}_{F_{\max}}(\sigma_{\leq m}).$$

Also note that adding request r_{m+1} to the input sequence will not decrease the optimal maximum flow, therefore

$$\text{OPT}_{F_{\max}}(\sigma_{\leq m}) \leq \text{OPT}_{F_{\max}}(\sigma_{\leq m+1}).$$

By combining the above two inequalities we find

$$\text{HOMESICK}_{F_{\max}}(\sigma_{\leq m}) \leq 2\text{OPT}_{F_{\max}}(\sigma_{\leq m+1}), \tag{3.5}$$

therefore property $\mathcal{P}(m+1)$ holds in case the first term is largest.

Now the second term of the maximum, F_{m+1}. First observe that from Theorem 3.4 we know the make-span $C_{\max} := \textsc{Homesick}_{C_{\max}}(\sigma_{\leq m+1})$ is the best possible. Hence, when defining $C^*_{\max} := \textsc{Opt}_{C_{\max}}(\sigma_{\leq m+1})$, we have $C_{\max} \leq C^*_{\max}$. Using these make-spans, the completion time of request r_{m+1} can be bounded. The following trivial upper bound is sufficient:

$$C_{m+1} \leq C_{\max}.$$

We assume that Opt returns to the origin immediately after serving its last request. We can make this assumption without loss of generality because this does not make a difference for the flow times of either Opt or Homesick, and $C_{\max} \leq C^*_{\max}$ holds regardless. Letting r_l be the last request served by Opt, we then have

$$C^*_{\max} = C^*_l + d(b_l, o).$$

Combining these two equations, we find

$$C_{m+1} \leq C^*_l + d(b_l, o).$$

With this inequality in hand, the flow time F_{m+1} can be bounded. We have

$$F_{m+1} = C_{m+1} - t_{m+1} \leq C^*_l + d(b_l, o) - t_{m+1} = t_l + F^*_l + d(b_l, o) - t_{m+1}.$$

Noting that $t_l - t_{m+1} \leq 0$ because r_{m+1} cannot be before r_l in the request sequence, we end up with

$$F_{m+1} \leq F^*_l + d(b_l, o) \overset{(3.3)}{\leq} 2F^*_l \leq 2\textsc{Opt}_{F_{\max}}(\sigma_{\leq m+1}). \tag{3.6}$$

By obtaining Inequalities 3.5 and 3.6 we can now bound $\textsc{Homesick}_{F_{\max}}(\sigma_{\leq m+1})$ to get

$$\textsc{Homesick}_{F_{\max}}(\sigma_{\leq m+1}) = \max\{\textsc{Homesick}_{F_{\max}}(\sigma_{\leq m}), F_{m+1}\} \leq 2\textsc{Opt}_{F_{\max}}(\sigma_{\leq m+1}),$$

and we see that property $\mathcal{P}(m+1)$ holds.

Thus, by the principle of induction $\mathcal{P}(m)$ holds for all $m \leq n$. In particular it holds for $m = n$, therefore $\textsc{Homesick}_{F_{\max}}(\sigma) \leq 2\textsc{Opt}_{F_{\max}}(\sigma)$ for any σ, meaning Homesick is 2-competitive.

□

Theorem 3.8. *Homesick has a competitive ratio of 2 for UCPO-MF.*

Proof. The result follows directly from Lemma 3.6 and 3.7. □

3.3.3 UCDO-MS

We start by proving a lower bound on the competitive ratio of any deterministic online algorithm for the UCDO-MS problem. The idea behind the proof is that the optimal offline algorithm knows the pick-up locations of future requests, therefore it can already move towards the pick-up location of a request before it is released. The lower bound is illustrated in Figure 3.6, and formalized in the following theorem.

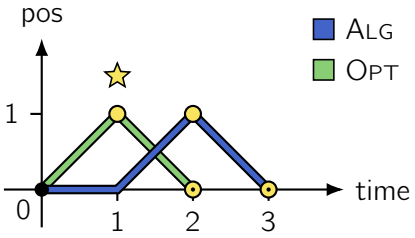


Figure 3.6: Illustration of Theorem 3.9

Theorem 3.9. *No deterministic online algorithm ALG has a competitive ratio smaller than* 1.5 *for UCDO-MS.*

Proof. Consider the input sequence $\sigma = (r_1)$, where $r_1 = (1, 1, o)$. Without loss of generality, we assume the server operated by ALG is at a non-positive location x at time 1. The ALG server has to pick up request r_1 and move back to the origin at some point, therefore we find a make-span of

$$\text{ALG}_{C_{\max}}(\sigma) \geq 1 + d(x, 1) + d(1, o) \geq 3.$$

The OPT server starts moving directly towards the pick-up location 1 at time 0, picks up the request and returns to the origin to drop it off. Therefore, the optimal make-span is

$$\text{OPT}_{C_{\max}}(\sigma) = d(o, 1) + d(1, o) = 2,$$

and we find

$$\frac{\text{ALG}_{C_{\max}}(\sigma)}{\text{OPT}_{C_{\max}}(\sigma)} \geq \frac{3}{2}.$$

For a server that is at a positive location at time 1 consider $r_1 = (1, -1, o)$ instead. □

We now turn our attention to the HOMESICK algorithm. We will prove that HOMESICK has a competitive ratio of 1.5, matching the lower bound of the previous theorem. The key observation for proving this is that OPT can only ever be ahead of HOMESICK by $d(o, a_i)$ for some request r_i. This is formalized and proven in Lemma 3.10, which will also prove useful in the UCDO-MF setting. The proof of the lemma follows the same structure as Theorem 3.4.

Lemma 3.10. *Let σ be an arbitrary sequence of requests, then*

$$\text{HOMESICK}_{C_{\max}}(\sigma) \leq \text{OPT}_{C_{\max}}(\sigma) + \max_{r_i \in \sigma} d(o, a_i).$$

Proof. Let $\sigma = (r_1, \ldots, r_n)$ be an arbitrary sequence of requests. Denote the time at which the server operated by HOMESICK leaves the origin to serve request r_i by O_i. Note that we have $O_1 = t_1$ since the server leaves the origin to serve r_1 as soon as it is released. Furthermore, for any request

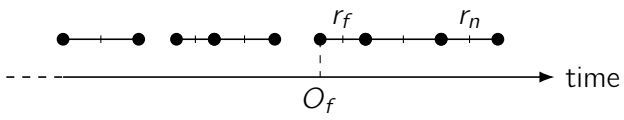


Figure 3.7: Example timeline of HOMESICK where O_f is the last time the server left the origin to serve a request the same time it was released. Vertical lines represent requests being picked up by the server.

r_i we have $O_i \geq t_i$. Since HOMESICK serves requests as soon as possible, if $O_i > t_i$ then HOMESICK was not idle between O_i and t_i. Now, define

$$f := \arg\max_{1 \leq i \leq n} \{O_i \mid O_i = t_i\} \quad \text{(if there is no unique index, pick the largest)}.$$

Hence, r_f is the last request for which HOMESICK left the origin at exactly the time it was released. Clearly f exists since $O_1 = t_1$.

Now, because for all $i > f$ we have $O_i > t_i$, the server was not idle after t_f. Therefore

$$\text{HOMESICK}_{C_{\max}}(\sigma) = t_f + 2\sum_{i=f}^{n} d(o, a_i). \tag{3.7}$$

Next, we investigate the make-span of the OPT algorithm. Aiming for a similar expression as in Equation 3.7, we consider the requests $\sigma_{\geq f}$. Let r_h be the first request from $\sigma_{\geq f}$ that the OPT server serves. We know the OPT server can only pick up r_h after it has been released at time t_h. After the server has picked up r_h, it has to move to the drop-off location which takes at least $d(o, a_h)$ time. For any other requests in $\sigma_{\geq f}$ the server first has to move to the pick-up location, and then drop if off. Therefore we have

$$\text{OPT}_{C_{\max}}(\sigma) \geq t_h + 2\sum_{i=f}^{n} d(o, a_i) - d(o, a_h).$$

Now, since $r_h \in \sigma_{\geq f}$ we have $t_h \geq t_f$ because σ is ordered on non-decreasing release times. Combining this fact with the previous equation we see

$$\text{OPT}_{C_{\max}}(\sigma) \geq t_f + 2\sum_{i=f}^{n} d(o, a_i) - d(o, a_h). \tag{3.8}$$

Finally, using Inequality 3.8 the make-span $\text{HOMESICK}_{C_{\max}}(\sigma)$ can be bounded:

$$\text{HOMESICK}_{C_{\max}}(\sigma) = t_f + 2\sum_{i=f}^{n} d(o, a_i) \overset{(3.8)}{\leq} \text{OPT}_{C_{\max}}(\sigma) + d(o, a_h) \leq \text{OPT}_{C_{\max}}(\sigma) + \max_{r_i \in \sigma} d(o, a_i).$$

□

Corollary 3.11. *HOMESICK is 1.5-competitive for UCDO-MS.*

Proof. Let σ be an arbitrary sequence of requests. Applying Lemma 3.10 to input σ gives

$$\text{HOMESICK}_{C_{\max}}(\sigma) \leq \text{OPT}_{C_{\max}}(\sigma) + \max_{r_i \in \sigma} d(o, a_i).$$

We then apply Inequality 3.1 to obtain the desired result:

$$\text{HOMESICK}_{C_{\max}}(\sigma) \leq \text{OPT}_{C_{\max}}(\sigma) + \max_{r_i \in \sigma} d(o, a_i) \overset{(3.1)}{\leq} \frac{3}{2}\text{OPT}_{C_{\max}}(\sigma).$$

□

Theorem 3.12. *HOMESICK has a competitive ratio of* 1.5 *for UCDO-MS, and this is the best possible competitive ratio.*

Proof. Follows directly from Theorem 3.9 and Corollary 3.11. □

Offline As we have seen, the strategy of HOMESICK is not optimal for UCDO-MS. In this problem setting, knowing the entire sequence of requests allows a server to move to pick-up locations of requests before they are released. Intuitively, an optimal algorithm for the offline problem would be one that exploits this knowledge best.

For serving some request $r_i \in \sigma$, leaving the origin at time $t_i - d(o, a_i)$ and travelling directly to a_i would let a server arrive at a_i exactly at the request's release time. When aiming to pick up request r_i, leaving the origin earlier than $t_i - d(o, a_i)$ would therefore never be beneficial. These values can thus be seen as the earliest times the server can start working on the corresponding request. Serving requests in non-decreasing order of these values hence lets the server work on serving a request whenever it can, and in this way minimizes the make-span.

The UCDO-MS problem can thus be solved optimally by sorting requests $r_i \in \sigma$ in non-decreasing order of $t_i - d(o, a_i)$. This sorting can be done in $\mathcal{O}(n \log n)$ time. For computing the trajectory and loading matrix for the server, $\mathcal{O}(n)$ time is needed, which does not increase the time complexity.

3.3.4 UCDO-MF

We will start by giving a lower bound on the competitive ratio of any deterministic online algorithm for UCDO-MF; the same set of instances is used as in the lower bound for UCDO-MS.

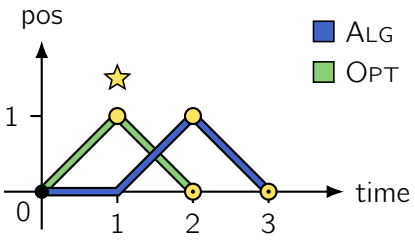


Figure 3.8: Illustration of Theorem 3.13

Theorem 3.13. *No deterministic online algorithm ALG has a competitive ratio smaller than 2 for UCDO-MF.*

Proof. Consider $\sigma = (r_1)$, where $r_1 = (1, 1, o)$. Without loss of generality, we assume the server operated by ALG is at a non-positive location x at time 1. The ALG server has to pick up request r_1 and move back to the origin at some point. At time 1 the server is at location $x \leq 0$, therefore the server drops off request r_1 at time

$$C_1 = 1 + d(x, 1) + d(1, o) \geq 3,$$

and the maximum flow time is

$$\text{ALG}_{F_{\max}}(\sigma) = F_1 = C_1 - t_1 \geq 2.$$

The OPT server starts moving directly towards the pick-up location 1 at time 0, picks it up and returns to the origin. Therefore the completion of request r_1 is

$$C_1^* = d(o, 1) + d(1, o) = 2,$$

and the maximum flow time is

$$\text{OPT}_{F_{\max}}(\sigma) = F_1^* = C_1^* - t_1 = 1.$$

Therefore

$$\frac{\text{ALG}_{F_{\max}}(\sigma)}{\text{OPT}_{F_{\max}}(\sigma)} \geq \frac{2}{1} = 2.$$

For a server that is at a positive location at time 1 consider $r_1 = (1, -1, o)$ instead. □

Next, we will prove that HOMESICK has a competitive ratio of 2. First, we prove that HOMESICK is 2-competitive under a restriction on the input. Afterwards we generalize the result to any input, thereby proving HOMESICK is 2-competitive.

Lemma 3.14. *Let $\sigma = (r_1, \ldots, r_n)$ be a sequence of requests such that the flow time of request r_n is maximal in the walk of HOMESICK on input σ, i.e. $F_n = \text{HOMESICK}_{F_{\max}}(\sigma)$. Then*

$$\text{HOMESICK}_{F_{\max}}(\sigma) \leq 2\text{OPT}_{F_{\max}}(\sigma).$$

Proof. By assumption the maximum flow time of HOMESICK is attained by request r_n, therefore we compute its flow time F_n. To compute this, note that the completion time of request r_n is equal to the make-span of HOMESICK. This is because HOMESICK serves request r_n last, thus the server is finished as soon as it drops off r_n at the origin. Hence, we have

$$\text{HOMESICK}_{F_{\max}}(\sigma) = F_n = C_n - t_n = \text{HOMESICK}_{C_{\max}}(\sigma) - t_n. \tag{3.9}$$

We bound the make-span of HOMESICK by applying Lemma 3.10 to input σ. Let OPT′ denote the optimal algorithm for minimizing make-span, then

$$\text{HOMESICK}_{C_{\max}}(\sigma) \leq \text{OPT}'_{C_{\max}}(\sigma) + \max_{r_i \in \sigma} d(o, a_i) \leq \text{OPT}_{C_{\max}}(\sigma) + \max_{r_i \in \sigma} d(o, a_i). \tag{3.10}$$

Let r_l denote the request that is dropped off last by OPT; this means its completion time $C_l^* = \text{OPT}_{C_{\max}}(\sigma)$. Note that $t_l \leq t_n$ because r_n is the last request in the sequence, therefore

$$\text{OPT}_{C_{\max}}(\sigma) - t_n = C_l^* - t_n \leq C_l^* - t_l = F_l^* \leq \text{OPT}_{F_{\max}}(\sigma). \tag{3.11}$$

Combining Inequalities 3.3, 3.9, 3.10 and 3.11 yields the desired result

$$\begin{aligned}
\text{HOMESICK}_{F_{\max}}(\sigma) &\overset{(3.9)}{=} \text{HOMESICK}_{C_{\max}}(\sigma) - t_n \\
&\overset{(3.10)}{\leq} \text{OPT}_{C_{\max}}(\sigma) - t_n + \max_{r_i \in \sigma} d(o, a_i) \\
&\overset{(3.11)}{\leq} \text{OPT}_{F_{\max}}(\sigma) + \max_{r_i \in \sigma} d(o, a_i) \\
&\overset{(3.3)}{\leq} 2\text{OPT}_{F_{\max}}(\sigma).
\end{aligned}$$

□

Lemma 3.15. *HOMESICK is 2-competitive for UCDO-MF.*

Proof. Let σ be an arbitrary sequence of requests. Denote with r_f a request with maximal flow in the walk of HOMESICK on input σ. Consider now the subsequence $\sigma_{\leq f}$ obtained by omitting all requests with index greater than f from σ. Observe that the maximum flow time of HOMESICK is still F_f for input $\sigma_{\leq f}$ because the algorithm serves requests in order of σ. Clearly, sequence $\sigma_{\leq f}$ satisfies the condition in Lemma 3.14, therefore we have

$$\text{HOMESICK}_{F_{\max}}(\sigma) = \text{HOMESICK}_{F_{\max}}(\sigma_{\leq f}) \leq 2\text{OPT}_{F_{\max}}(\sigma_{\leq f}) \leq 2\text{OPT}_{F_{\max}}(\sigma).$$

The last inequality follows from the observation that adding more requests to an input will never decrease the maximum flow time obtained on that input. □

Theorem 3.16. *HOMESICK has a competitive ratio of 2 for UCDO-MF, and this is the best possible competitive ratio.*

Proof. The result follows directly from Theorem 3.13 and Lemma 3.15. □

3.4 The ALTERNATE algorithm

The main part of this section is about proving the competitive ratio of the ALTERNATE algorithm for the various problem settings. Furthermore, for the ICPO-MS setting a detailed analysis of the offline problem is performed, and an algorithm is given that solves it. Recall that ALTERNATE serves all unserved requests on the same side of the line at once, moving to the side that contains the earliest released request. The ALTERNATE algorithm is never idle when there are unserved requests, and ignores requests that are released while it is away from the origin.

Before considering specific problem settings, we show the workings of the ALTERNATE algorithm by giving an example.

> *Example.* Consider the sequence of requests $\sigma = (r_1, r_2, r_3, r_4)$ with $r_1 = (1, o, 1)$, $r_2 = (2, o, -1)$, $r_3 = (2.5, o, 0.5)$ and $r_4 = (3, o, 1)$. The server controlled by the ALTERNATE algorithm is at the origin at time 0. The server will stay there until a request is released. This happens at time $t_1 = 1$; at this point ALTERNATE orders the server to move to point 1 to pick up request r_1 and return to the origin directly to drop it off. The server arrives back at the origin at time 3; at this point there are three unserved requests: r_2, r_3 and r_4. The request with earliest release time is r_2, therefore since r_2 has a negative pick-up location ALTERNATE decides to serve all requests with negative pick-up locations. This means the ALTERNATE server moves to pick up r_2 and drops it off at the origin at time 4. At time 4, the request with earliest release time is r_3, therefore ALTERNATE decides to serve requests with non-negative pick-up locations, which are r_3 and r_4. After the ALTERNATE server has served these requests and is back at the origin, no more requests are released.
>
> The ALTERNATE algorithm has completion times $C_1 = 3$, $C_2 = 4$ and $C_3 = C_4 = 6$, where C_i is the completion time of request r_i. The flow times follow from the completion times; letting F_i denote the flow time of request r_i, we find
>
> $$F_1 = 3 - 1 = 2; \quad F_2 = 4 - 2 = 2; \quad F_3 = 6 - 2.5 = 3.5; \quad F_4 = 6 - 3 = 3.$$
>
> At time 6, the ALTERNATE server has served all requests and is back at the origin, therefore the make-span is $\text{ALTERNATE}_{C_{\max}}(\sigma) = 6$. The maximum flow time of ALTERNATE for input σ is
>
> $$\text{ALTERNATE}_{F_{\max}}(\sigma) = \max\{F_1, F_2, F_3, F_4\} = 3.5.$$

Figure 3.9 shows the behaviour of the ALTERNATE server on input σ. Such figures are provided for all lower bounds and it is recommended to consult these while reading the corresponding proof. Some figures in this section use a slightly different color coding of requests; if there are many requests, the requests with negative pick-up or drop-off location are colored red and the others green.

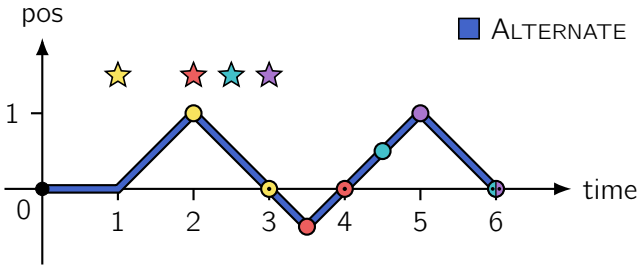


Figure 3.9: Example behaviour of the ALTERNATE algorithm in the ICDO setting.

3.4.1 ICPO-MS

First we will construct a lower bound on the competitive ratio that can be achieved by any deterministic online algorithm, then we will prove that ALTERNATE has a competitive ratio of 2 for this problem.

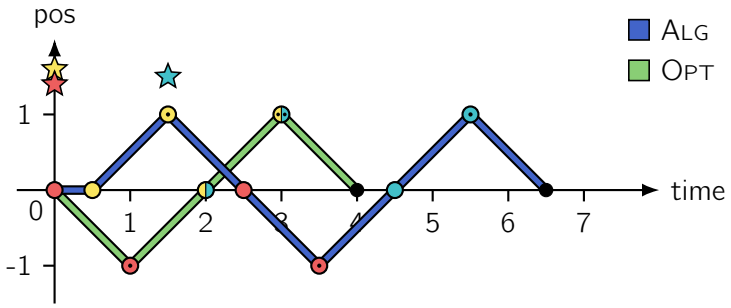


Figure 3.10: Illustration of Theorem 3.17; example with $T = 1.5$

Theorem 3.17. *No deterministic online algorithm ALG has a competitive ratio smaller than* 1.5 *for ICPO-MS.*

Proof. Let ALG be an arbitrary deterministic online algorithm. The adversary starts by releasing requests $r_1 = (0, o, 1)$ and $r_2 = (0, o, -1)$. Since these two requests are symmetric and they have to be dropped off at some point, we can assume, without loss of generality, that the server operated by ALG drops off r_1 before r_2 at some time $T \geq 1$. We perform a case distinction on the value of T.

Case 1: $T \in [1, 4]$. The adversary releases one other request $r_3 = (T, o, 1)$ just when the server controlled by ALG has dropped off r_1. The complete request sequence in this case is $\sigma = (r_1, r_2, r_3)$. Next, the ALG server has to serve r_2 and r_3. Using the knowledge that the server was at location 1 at time T, we see that the make-span is $\text{ALG}_{C_{\max}}(\sigma) \geq T + 5$. The optimal algorithm will start by serving request r_2, after which it is back at the origin at time 2. To determine the actions of the optimal algorithm after time 2, we again perform a case distinction on the value of T.

Case 1.1: $T \in [1, 2]$. In this case, request r_3 has been released at time 2, therefore OPT can directly leave the origin to serve requests r_1 and r_3 together. This yields the optimal make-span $\text{OPT}_{C_{\max}}(\sigma) = 4$. In this case, we see that ALG has competitive ratio at least

$$\frac{\text{ALG}_{C_{\max}}(\sigma)}{\text{OPT}_{C_{\max}}(\sigma)} \geq \frac{T+5}{4} \geq \frac{6}{4} = 1.5.$$

Case 1.2: $T \in (2, 4]$. The OPT server will wait for $T - 2$ time units until request r_3 is released, and then serve r_1 and r_3 together. In this case $\text{OPT}_{C_{\max}}(\sigma) = T + 2$, and ALG has competitive ratio at least

$$\frac{\text{ALG}_{C_{\max}}(\sigma)}{\text{OPT}_{C_{\max}}(\sigma)} \geq \frac{T+5}{T+2} \geq \frac{9}{6} = 1.5.$$

Case 2: $T > 4$. The adversary releases no other requests, hence the complete input is $\sigma = (r_1, r_2)$. After the ALG server has dropped off request r_1 at location 1, it takes the ALG server at least 3 time units to serve r_2 and return to the origin. Hence $\text{ALG}_{C_{\max}}(\sigma) = T + 3$. The optimal algorithm will serve requests r_1 and r_2 immediately, giving $\text{OPT}_{C_{\max}}(\sigma) = 4$. Hence, the competitive ratio of ALG is at least

$$\frac{\text{ALG}_{C_{\max}}(\sigma)}{\text{OPT}_{C_{\max}}(\sigma)} \geq \frac{T+3}{4} \geq \frac{7}{4} > 1.5.$$

□

Remark. Theorem 3.17 only uses the fact that the capacity is at least 2, therefore the lower bound also holds for the ICPO-MS problem where instead of infinite capacity the server has any finite capacity greater than 1.

We continue by investigating the performance of the ALTERNATE algorithm. We start with a lower bound.

Lemma 3.18. *ALTERNATE has a competitive ratio of at least 2 for ICPO-MS.*

Proof. Consider the sequence of requests $\sigma = (r_1, r_2)$ with $r_1 = (0, o, 1)$ and $r_2 = (\varepsilon, o, 1)$ for some small $\varepsilon < 2$. The ALTERNATE server first serves r_1 and ignores the release of request r_2. Only after finishing request r_1, the request r_2 will be served. The make-span is therefore $\text{ALTERNATE}_{C_{\max}}(\sigma) = 4$. The optimal server waits until time ε and then serves r_1 and r_2 in one visit to location 1, yielding a make-span of $\text{OPT}_{C_{\max}}(\sigma) = 2 + \varepsilon$. Taking ε small enough, the lower bound on ALTERNATE's competitive ratio can be made arbitrary close to 2. □

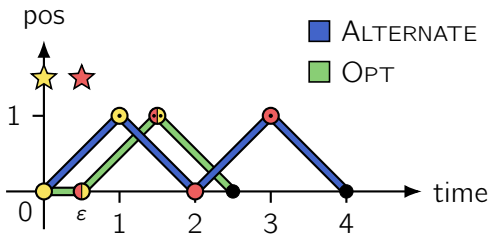


Figure 3.11: Illustration of Lemma 3.18 with $\varepsilon = 0.5$

Before we prove ALTERNATE is 2-competitive, we introduce some notation.

Definition. Let S be a non-empty set or sequence of requests on one side of the line. Then let $L(S)$ denote the length of the shortest tour visiting all drop-off locations from requests in S, starting and ending at the origin. We have

$$L(S) = 2 \max_{r_i \in S} |b_i|.$$

Note that if $S_1 \subseteq S$ then $L(S_1) \leq L(S)$.

Definition. For some set or sequence of requests ρ, define

$$\rho^+ = \{r_i \in \rho \mid b_i \geq o\}, \quad \text{and,} \quad \rho^- = \{r_i \in \rho \mid b_i < o\}.$$

Note that $\rho = \rho^+ \cup \rho^-$.

Observation 3.19. Any algorithm has to serve the left-most request and the right-most request while starting and ending at the origin. Hence,

$$\text{OPT}_{C_{\max}}(\sigma) \geq L(\sigma^-) + L(\sigma^+). \tag{3.12}$$

Lemma 3.20. *ALTERNATE is 2-competitive for ICPO-MS.*

Proof. We distinguish cases based on what the ALTERNATE server is doing at the time t_n at which the last request r_n becomes known. If the ALTERNATE server is idle at the origin then it simply

serves the released request r_n immediately giving

$$\textsc{Alternate}_{C_{\max}}(\sigma) = t_n + 2d(o, b_n) \overset{(3.2)}{\leq} \textsc{Opt}_{C_{\max}}(\sigma),$$

and the lemma holds.

Now, we consider the case where the ALTERNATE server is not idle at the origin at time t_n, but instead it is following some schedule $\mathcal{S}$ serving requests at one side of the line. The arrival of request r_n does not stop the ALTERNATE server from following $\mathcal{S}$; it ignores this request and continues with its schedule. Let T be the time the server started schedule $\mathcal{S}$, and let R denote the set of unserved requests at that time T. The requests served by the ALTERNATE server in schedule $\mathcal{S}$ are either the requests R^- or R^+, whichever set has the request with minimum release time. Without loss of generality we assume that R^+ contains the requests that are served in schedule $\mathcal{S}$. Now, the make-span of ALTERNATE can be expressed as:

$$\textsc{Alternate}_{C_{\max}}(\sigma) = T + L(R^+) + L(R^- \cup \sigma^-_{\geq T}) + L(\sigma^+_{\geq T}). \tag{3.13}$$

Indeed, the ALTERNATE server finishes schedule $\mathcal{S}$ at time $T + L(R^+) \geq t_n$; this is later than t_n because recall the ALTERNATE server was performing schedule $\mathcal{S}$ when request r_n was released. This means no new requests arrived after schedule $\mathcal{S}$ was finished. Therefore, the last task for the ALTERNATE server is serving the remaining requests with drop-off location on the negative part of the line $R^- \cup \sigma^-_{\geq T}$, and the remaining requests at the positive part $\sigma^+_{\geq T}$.

For the OPT server, consider the first request r_f from $\sigma_{\geq T}$ that is served by the OPT server. Since r_f is in $\sigma_{\geq T}$, by definition $t_f \geq T$. The OPT server has to at least wait until this release time t_f, and then serve all requests in $\sigma_{\geq T}$, therefore

$$\textsc{Opt}_{C_{\max}}(\sigma) \geq t_f + L(\sigma^+_{\geq T}) + L(\sigma^-_{\geq T}) \geq T + L(\sigma^+_{\geq T}) + L(\sigma^-_{\geq T}). \tag{3.14}$$

Using Equations 3.13 and 3.14 we can bound $\textsc{Alternate}_{C_{\max}}(\sigma)$ by $\textsc{Opt}_{C_{\max}}(\sigma)$. First, from Equation 3.13 we see

$$\textsc{Alternate}_{C_{\max}}(\sigma) \overset{(3.13)}{\leq} T + L(R^+) + L(R^- \cup \sigma^-_{\geq T}) + L(\sigma^+_{\geq T}).$$

We rearrange and add and subtract appropriate terms to get

$$\textsc{Alternate}_{C_{\max}}(\sigma) \leq T + L(\sigma^+_{\geq T}) + L(\sigma^-_{\geq T}) + L(R^+) + L(R^- \cup \sigma^-_{\geq T}) - L(\sigma^-_{\geq T}).$$

Next, we apply Equation 3.14, which yields

$$\begin{aligned}\textsc{Alternate}_{C_{\max}}(\sigma) &\overset{(3.14)}{\leq} \textsc{Opt}_{C_{\max}}(\sigma) + L(R^+) + L(R^- \cup \sigma^-_{\geq T}) - L(\sigma^-_{\geq T}) \\ &\leq \textsc{Opt}_{C_{\max}}(\sigma) + L(R^+) + L(R^- \cup \sigma^-_{\geq T}).\end{aligned}$$

Note that $R^+ \subseteq \sigma^+$ and $R^- \cup \sigma^-_{\geq T} \subseteq \sigma^-$, therefore using Inequality 3.12 we have

$$\textsc{Alternate}_{C_{\max}}(\sigma) \overset{(3.12)}{\leq} \textsc{Opt}_{C_{\max}}(\sigma) + \textsc{Opt}_{C_{\max}}(\sigma) = 2\textsc{Opt}_{C_{\max}}(\sigma),$$

and the lemma also holds in this case. □

Theorem 3.21. *ALTERNATE has a competitive ratio of 2 for ICPO-MS.*

Proof. Follows directly from Lemma 3.18 and 3.20. □

We also briefly consider the online algorithm TIR (Temporarily Ignore Requests) introduced by E. Feuerstein and L. Stougie [12]. The algorithm TIR was originally constructed for a more general version of ICPO-MS where pick-up locations need not be the origin. In this setting TIR has a competitive ratio of 2. This implies that in the more restricted setting of ICPO-MS algorithm TIR is still 2-competitive (see Theorem 3.22), but its competitive ratio may be lower. Indeed, the instance given in [12] to prove tightness does not hold in the ICPO-MS setting. We found, however, a different set of instances which shows that TIR has a competitive ratio of 2 for the ICPO-MS problem (see Lemma 3.23).

The way the algorithm TIR works is as follows. When the TIR server is at the origin and there are unserved requests, the algorithm orders the server to serve all the unserved requests via the shortest route, and to return to the origin at the end. When the TIR serve receives a request while it is busy, it acts based on how far away the drop-off location of the incoming request is from the origin. If the drop-off location is further from the origin than the server itself, then the server returns to the origin; otherwise the server continues with its task. In Algorithm 3, TIR is described in more detail.

Algorithm 3: TIR

Input : server that starts at o; requests arrive over time
$R \leftarrow$ empty multiset

repeat
- **if** a new request r_i becomes known **then**
  - add r_i to R
  - $p \leftarrow$ position of server
  - **if** $d(b_i, o) > d(p, o)$ **then**
    - order server to return to the origin immediately
  - **else**
    - let server continue with its schedule
- **if** server drops off request r_i **then**
  - remove r_i from R
- **if** server is at the origin and R is not empty **then**
  - order the server to serve requests R and return to the origin via the shortest route

Theorem 3.22. (Theorem 2.3 in [12]) *TIR is 2-competitive for ICPO-MS.*

Lemma 3.23. *TIR has a competitive ratio of at least 2 for ICPO-MS.*

Proof. Let $N \in \mathbb{N}_{\geq 1}$ and let δ be a small positive number. Consider the sequence of requests $\sigma = (p_1, m, p_2, \ldots, p_N)$, where $m = (0, o, -1)$, $p_1 = (0, o, \frac{1}{N})$, and $p_i = ((2i-3) \cdot (\frac{1}{N} - \delta), o, \frac{1}{N})$ for $i \in \{2, \ldots, N\}$. Requests p_1 and m have the same release time, but recall that requests are presented to the online algorithm one-by-one in order of σ. When request p_1 becomes known to TIR, it orders the server to serve p_1 and return to the origin via the shortest route. However, immediately afterwards request m becomes known, which causes TIR to return to the origin. This results in the server being at the origin at time 0, and TIR ordering it to serve requests p_1 and m and return to the origin via the shortest route. The algorithm does not specify which shortest route it uses; without loss of generality we can thus assume that it first serves requests with drop-off locations on the positive side of the line, and then moves to the negative side.

At time $\frac{1}{N} - \delta$, the server is at point $\frac{1}{N} - \delta$ and request p_2 is released. The release of p_2 causes the server to return to the origin. When back at the origin at time $\frac{2}{N} - 2\delta$, the TIR algorithm will decide to go to the positive side of the line to server requests p_1 and p_2, and afterwards serve m. At time $\frac{3}{N} - 3\delta$, request p_3 is released and the TIR algorithm will again return to the origin. This process of the server moving to $\frac{1}{N}$, and returning to the origin just before it gets there, repeats itself N times. Finally, at time $2 - \frac{1}{N} - 2N\delta$ the server drops off all requests p_i for $i \in \{1, \ldots, N\}$. The server then proceeds to serve request m, yielding a make-span of $\text{TIR}_{C_{\max}}(\sigma) = 4 - 2N\delta$.

The optimal algorithm would first serve request m, and afterwards all requests p_i for $i \in \{1, \ldots, N\}$ in one visit to $\frac{1}{N}$. This yields the optimal make-span $\text{OPT}_{C_{\max}}(\sigma) = 2 + \frac{2}{N}$. We have therefore

$$\frac{\text{TIR}_{C_{\max}}(\sigma)}{\text{OPT}_{C_{\max}}(\sigma)} = \frac{4 - 2N\delta}{2 + \frac{2}{N}}.$$

For any $\varepsilon > 0$, consider $N = \left\lceil \frac{2}{\varepsilon} \right\rceil$ and $\delta = \frac{\varepsilon^2}{2N}$. Then, we have:

$$\frac{\text{TIR}_{C_{\max}}(\sigma)}{\text{OPT}_{C_{\max}}(\sigma)} = \frac{4 - \varepsilon^2}{2 + \frac{2}{\left\lceil \frac{2}{\varepsilon} \right\rceil}} \geq \frac{(2 - \varepsilon)(2 + \varepsilon)}{2 + \varepsilon} = 2 - \varepsilon.$$

Since this holds for any $\varepsilon > 0$, the competitive ratio of TIR is at least 2.

□

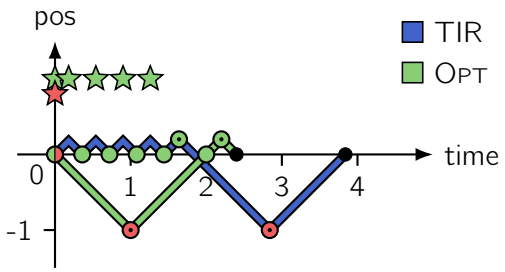


Figure 3.12: Illustration of Lemma 3.23 with $\delta = 0.02$ and $N = 5$

Corollary 3.24. *TIR has a competitive ratio of 2 for ICPO-MS.*

Proof. Follows directly from Theorem 3.22 and Lemma 3.23. □

Offline

It was stated by De Paepe et al. [10] that there exists a polynomial time algorithm for solving the offline ICPO-MS problem (using Lemma 9 in the paper the problem ICPO-MS can be rewritten to the 'easy' problem E7 in Table 1). The algorithm however has been lost to time (personal communication with one of the authors revealed that the current location of the file containing the algorithm is unknown), therefore we reinvent it. Inspired by De Paepe et al. we follow a dynamic programming approach and exploit the structure of optimal walks for offline ICPO-MS.

We use similar notation and assumptions as in [6]:

- There is a request $r_0 = (0, o, o)$. Note that if such a request r_0 is not present in a request sequence, then it can be added without altering the make-span of any solution.
- Requests are sorted and indexed by non-decreasing drop-off location. We denote the requests as $r_{-L}, \ldots, r_0, \ldots, r_R$ with $b_{-L} \leq \cdots \leq b_0 \leq \cdots \leq b_R$. Therefore r_{-L} denotes the leftmost request, and r_R the rightmost request.
- A request r_i is dropped off the last time the server visits its drop-off location b_i; this assumption does not affect the make-span of a solution.

The dynamic program we introduce relies on the fact that an optimal walk has the kind of spiral structure as shown in Figure 3.13. The length of the visits to the positive part of the line are non-increasing over time, similarly with the visits to the negative part of the line. To see this, suppose there was an increase in amplitude, then on the smaller amplitude part of the walk no requests are dropped off. This is because we assume requests are dropped off the last time their drop-off location is visited, therefore the first departure was redundant. This is formalized in Lemma 3.25 which is adapted from a similar lemma in [6].

> **Lemma 3.25.** *At any time in a feasible walk, the set of served requests is the union of two disjoint sets $S_1 = \{r_{-L}, \ldots, r_i\}$ and $S_2 = \{r_j, \ldots, r_R\}$ for some $i \leq 0$ and $j \geq 0$.*

Proof. Suppose there is a time T at which the set of requests does not have the claimed structure. Then there is some request r_k that is dropped off at time T, and a request r_l that is still unserved at time T with $l > k \geq 0$ or $l < k \leq 0$. Without loss of generality assume the first to be true. At time $T' > T$ when request r_l is dropped off, the server is at position b_l. Since the walk ends at the origin $o = 0$ and $0 \leq b_k < b_l$, the server passes drop-off location b_k at some time after T. Since a request is dropped off the last time the server visits its drop-off location, request r_k is dropped off at some time after T. This contradicts that r_k was dropped off at time T, therefore the lemma holds. □

At the start of an optimal solution we see therefore that a set of left-most requests or right-most requests is served, after which we return to the origin and arrive in a similar situation as before.

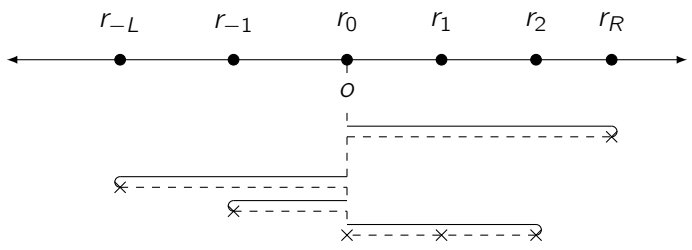


Figure 3.13: Spiral structure of an optimal solution to offline ICPO-MS. Crosses indicate a request being dropped off.

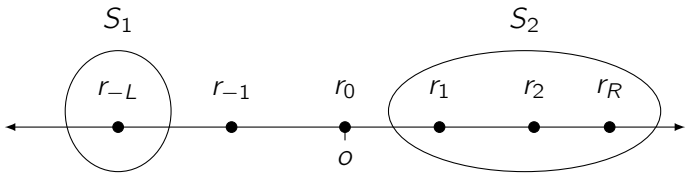


Figure 3.14: Example of two contiguous sets S_1, S_2 in the context of Lemma 3.25.

At this time, there are fewer unserved requests, and again the server will serve a set of left-most requests or right-most requests. This process continues until all requests have been served. Hence, this problem has optimal substructure. Based on this we give the following definition.

> *Definition.* Let $i \in [-L-1, 0]$ and $j \in [0, R+1]$, then we define $V(i,j)$ as the optimal make-span of a walk starting and ending at the origin, serving requests $\{r_l \mid l \leq i \vee l \geq j\}$.

Note that the optimal make-span for the entire request sequence is then $V(0,0)$. In the following lemma, a recurrence is given for $V(i,j)$.

> **Lemma 3.26.** *Given an instance of ICPO-MS with requests $r_{-L}, \ldots, r_R$, some indices $i \in [-L-1, 0], j \in [0, R+1]$, and make-spans $V(h,g)$ for $h \in [-L-1, i), g \in (j, R+1]$; then the makespan $V(i,j)$ is given by the following recurrence:*
>
> $$V(i,j) = \begin{cases} 0 & \text{if } i = -L-1 \text{ and } j = R+1 \\ t_R + 2d(o, b_R) & \text{if } i = -L-1 \text{ and } j = R \\ t_{-L} + 2d(o, b_{-L}) & \text{if } i = -L \text{ and } j = R+1 \\ \min\{V^-(i,j), V^+(i,j)\} & \text{if } i \in [-L, 0] \text{ and } j \in [0, R] \end{cases},$$
>
> *where*
>
> $$V^-(i,j) = \min_{-L-1 \leq h < i} \{\max\{V(h,j), \max_{h < k \leq i} t_k\} + 2d(o, b_{h+1})\},$$
> $$V^+(i,j) = \min_{j < g \leq R+1} \{\max\{V(i,g), \max_{j \leq k < g} t_k\} + 2d(o, b_{g-1})\}.$$

Proof. We start by proving the first three base cases. For the first case, note that

$$\{r_l \mid l \leq -L-1 \vee l \geq R+1\} = \emptyset,$$

therefore no requests need to be served, and the optimal make-span is 0. For the second base case, the set of requests that need to be served is

$$\{r_l \mid l \leq -L-1 \vee l \geq R\} = \{r_R\},$$

hence only request r_R needs to be served. The optimal make-span for this is obtained by a server that waits until time t_R, picks up the request at the origin, drops it off at b_R and moves back to the origin immediately afterwards. This yields $V(-L-1, R) = t_R + 2d(o, b_R)$. The third base case is symmetrical.

We prove the recurrence of $V(i,j)$ by proving $\leq$ and $\geq$.

$\leq$: We prove a make-span of value $\min\{V^-(i,j), V^+(i,j)\}$ is feasible for serving requests r_l with $l \leq i$ or $l \geq j$. Regarding $V^-(i,j)$, we construct a route consisting of two parts. First, for any value of h the server starts by executing the route attaining $V(h,j)$ in which requests $\{r_l \mid l \leq h \vee l \geq j\}$ are served. After this route the server is back at the origin at which the server will wait until

all requests r_k with $h < k \leq i$ are released, and then will pick up all of them. Next, the server moves to b_{h+1} and back to the origin, dropping of requests along the way. Note that since $b_{h+1} \leq b_k \leq b_i \leq 0$ for all $h < k \leq i$, all drop-off locations for requests r_k are visited. In this second part of the route we see therefore that requests $\{r_k \mid h < k \leq i\}$ are served. Hence, combining the two parts of the route, we find that requests

$$\{r_l \mid l \leq h \vee l \geq j\} \cup \{r_k \mid h < k \leq i\} = \{r_l \mid l \leq i \vee l \geq j\}$$

are served in the constructed route. Therefore $V^-(i,j)$ is feasible. Symmetrically we have feasibility for $V^+(i,j)$, therefore $\min\{V^-(i,j), V^+(i,j)\}$ is feasible.

$\geq$: We prove a make-span of value $\min\{V^-(i,j), V^+(i,j)\}$ is minimal for serving requests r_l with $l \leq i$ or $l \geq j$. From Lemma 3.25 follows that between two consecutive visits to the origin a server serving requests r_l with $l \leq i$ or $l \geq j$ either drops off requests $\{r_k \mid h \leq k \leq i\}$ last for some $h < i$ or requests $\{r_k \mid j \leq k \leq g\}$ for some $g > j$. Therefore the make-span is at least

$$\min\{\min_{-L-1 \leq h < i}\{V(h,j) + 2d(o, b_{h+1})\}, \min_{j < g \leq R+1}\{V(i,g), +2d(o, b_{g-1})\}\}.$$

Furthermore, we can only leave the origin in order to drop off certain requests after they have been released. Therefore $\min\{V^-(i,j), V^+(i,j)\}$ is the minimal make-span for serving requests r_l with $l \leq i$ or $l \geq j$.

□

We can make use of the recurrence from Lemma 3.26 to calculate the optimal make-span for a given sequence of requests. Computing $V(0,0)$ directly from the recursion is not efficient because the same computation would be done multiple times. Therefore, we describe a dynamic program which stores previously computed values, and in this way is more efficient. For constructing this dynamic program, we use the fact that given $i \in [-L-1, 0]$ and $j \in [0, R+1]$, the recurrence for $V(i,j)$ only uses values $V(h,j)$ with $-L-1 \leq h < i$ and values $V(i,g)$ with $j < g \leq R+1$. This is more easily understood in a picture; in Figure 3.15 a table is depicted, each cell in the table representing a values of V. In the table, the calculation of the value in a cell requires the values of the cells above it and to the right of it. To calculate $V(0,0)$, we therefore need to start at the top-right and move to the bottom-left, either row-wise or column-wise. This calculation procedure is described in Algorithm 4, of which the correctness and time complexity is proven in Theorem 3.27.

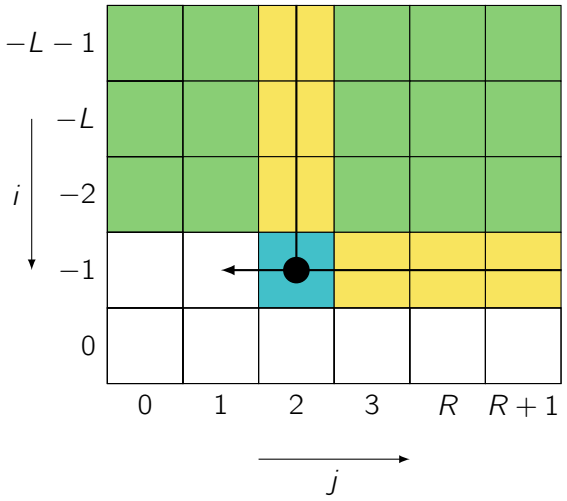


Figure 3.15: An in-progress dynamic program calculation with $L = 3$ and $R = 4$

Algorithm 4: Dynamic Program for Offline ICPO-MS.

Input : Sequence of requests σ
Output : The minimum make-span

1: $r_{-L}, \ldots, r_R \leftarrow \sigma$ sorted in non-decreasing order of drop-off location
2: $V(-L-1, R+1) \leftarrow 0$
3: $V(-L-1, R) \leftarrow t_R + 2d(o, b_R)$
4: $V(-L, R+1) \leftarrow t_{-L} + 2d(o, b_{-L})$
5: **for** $i \leftarrow -L-1$ **to** 0 **do**
6: **for** $j \leftarrow R+1$ **to** 0 **do**
7: $V^-(i,j) \leftarrow \min_{-L-1 \le h < i}\{\max\{V(h,j), \max_{h<k\le i} t_k\} + 2d(o, b_{h+1})\}$
8: $V^+(i,j) \leftarrow \min_{j<g\le R+1}\{\max\{V(i,g), \max_{j\le k<g} t_k\} + 2d(o, b_{g-1})\}$
9: $V(i,j) \leftarrow \min\{V^-(i,j), V^+(i,j)\}$
10: **return** $V(0,0)$

Theorem 3.27. *Algorithm 4 computes the minimum make-span of a walk for ICPO-MS in time $\mathcal{O}(n^4)$, where n is the number of requests.*

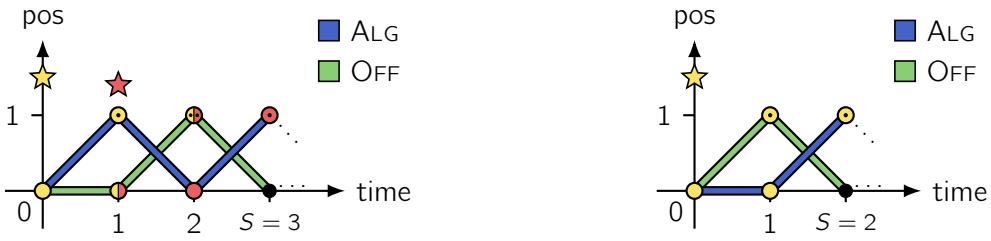
Proof. Correctness of the program follows from Lemma 3.26 and the fact that values of V are referenced only after they have been computed. For the runtime, first note that the sorting in line 1 can be done in $\mathcal{O}(n \log n)$ time and lines 2-4 are performed in $\mathcal{O}(1)$ time. For the nested loop, note that $L \le n$ and $R \le n$, therefore lines 7-9 are executed $\mathcal{O}(n^2)$ times. For line 7, since $i \le 0$, the minimum is taken over $\mathcal{O}(n)$ values. For calculating each value, the maximum has to be taken over $\mathcal{O}(n)$ release times. Hence line 10 takes in total at most $\mathcal{O}(n^2)$ time. Similarly, line 8 takes at most $\mathcal{O}(n^2)$ time and line 9 takes $\mathcal{O}(1)$ time. The nested for loop can hence be executed in $\mathcal{O}(n^4)$ time, and since this is the most computationally expensive part of the program, the entire program can be executed in $\mathcal{O}(n^4)$ time. □

Remark. By precomputing the maximums of release times, the algorithm can be adjusted to run in $\mathcal{O}(n^3)$ time.

Remark. Algorithm 4 computes the make-span of an optimal solution. To compute the optimal solution itself, an extra table can be introduced that stores the choices leading to the optimal solution, or alternatively the choices may be reconstructed from the table of value V.

3.4.2 ICPO-MF

As in most other sections, we start by providing a lower bound on the competitive ratio any deterministic algorithm can have. First we prove a lemma that will help us in constructing a lower bound.



(a) Example with $T = 1$

(b) Example with $T = 2$

Figure 3.16: Illustration of Lemma 3.28

Lemma 3.28. *Let ALG be an arbitrary deterministic online algorithm. The adversary can release requests such that there is a value $S \in \mathbb{R}_{\geq 0}$ and an offline algorithm OFF satisfying the following properties:*

(i) At time S the ALG server is at location 1 and has no unserved requests.

(ii) At time S the OFF server is at the origin and has no unserved requests. Furthermore, the flow times of the requests it has served do not exceed 3.

Proof. The adversary starts by releasing request $r_1 = (0, o, 1)$. This request is dropped off by the ALG server at time $T \geq 1$. We perform a case distinction based on the value of T.

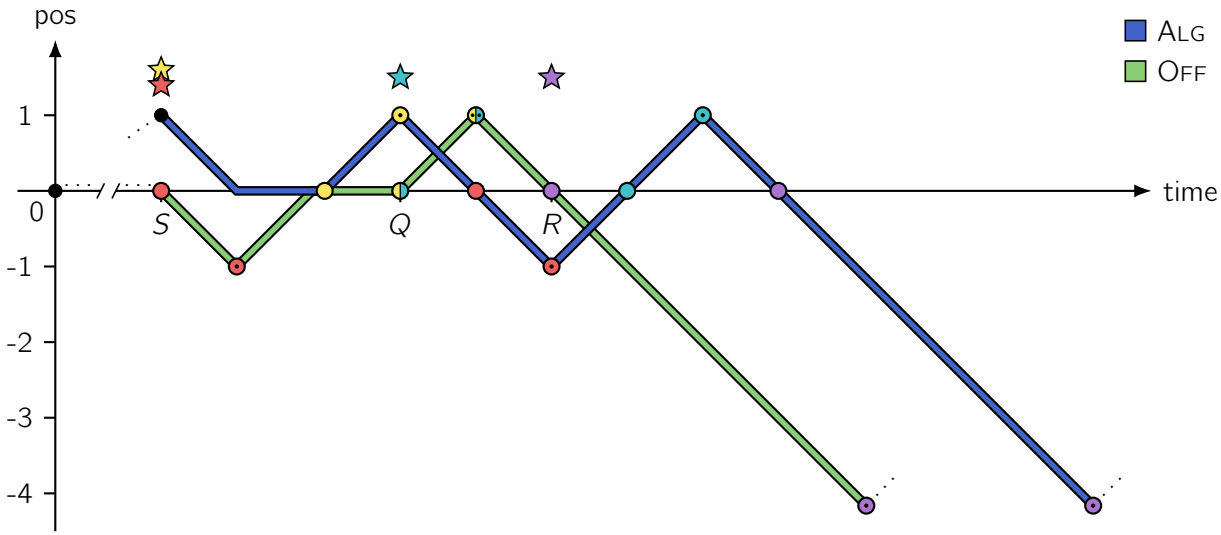
Case 1: $1 \leq T < 2$. The adversary releases a second request $r_2 = (T, o, 1)$. Define S as the time r_2 is dropped off, therefore $S \geq T + 2$. By construction, property (i) holds. Let the OFF server wait until time T, and then serve both request r_1 and r_2, returning to the origin immediately afterwards. This results in the OFF server being at the origin with no unserved requests at time $T + 2 \leq S$ at which it stays until time S. Furthermore, the flow times of requests r_1 and r_2 are $F_1 = T + 1 < 3$ and $F_2 = 1$ respectively, none exceed 3. Therefore, property (ii) holds and thereby the lemma holds.

Case 2: $T \geq 2$. The adversary releases no other requests until time S. Define $S := T \geq 2$; it is easily seen that property (i) holds. Let the OFF server pick up and serve request r_1 immediately at time 0. Afterwards, the server can return to the origin directly such that it is back at time $2 \leq S$. If needed, the server can wait until time S. Since the flow time of request r_1 is 1, we see that property (ii) and therefore the lemma holds. □

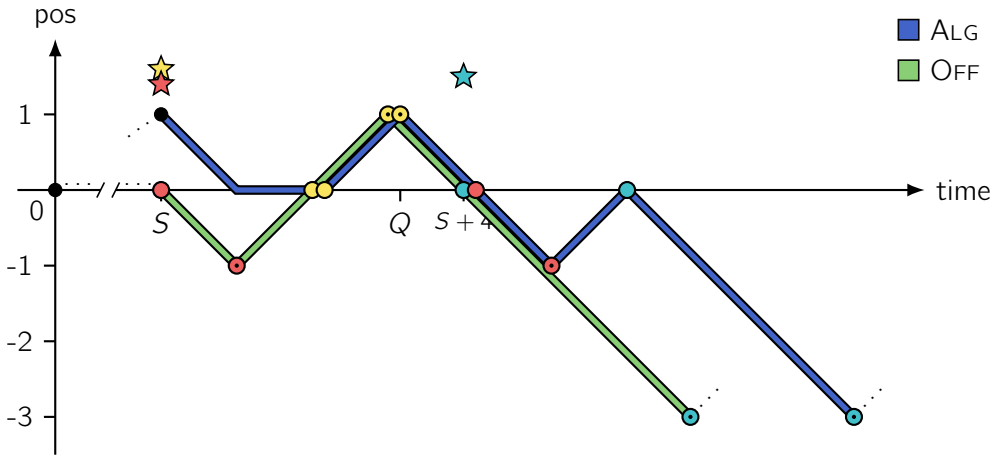
Theorem 3.29. *No deterministic online algorithm ALG for ICPO-MF has a competitive ratio lower than $(\sqrt{10} + 2)/3 \approx 1.7208$.*

Proof. Let ALG be an arbitrary deterministic online algorithm. From Lemma 3.28, we have S and OFF satisfying the listed properties. Let ρ be the sequence of requests released by the adversary before time S.

At time S the adversary releases two new requests $s_1 = (S, o, 1)$ and $s_2 = (S, o, -1)$. Without loss of generality, we can assume that the ALG server drops off request s_1 before s_2 at time $Q \geq S + 2$ (otherwise the signs of all subsequent requests can be flipped). We perform a case distinction on the value of $Q - S$.



(a) Best ALG can do: Case 1.2.2



(b) Best ALG can do: Case 2.2

Figure 3.17: Illustration of Theorem 3.29

Case 1: $2 \leq Q - S \leq \sqrt{10}$. As s_1 is dropped off, the adversary releases another request $s_3 = (Q, o, 1)$. We perform a case distinction on whether s_2 or s_3 is dropped off first by ALG.

Case 1.1: ALG drops off s_3 before s_2. The completion time of s_2 is at least $Q+1+2+1 = Q+4$, therefore the maximum flow time of ALG is at least $Q - S + 4$.

Case 1.2: ALG drops off s_2 before s_3. Denote the time request s_2 is dropped off by $R \geq Q + 2$. At that time, the adversary releases its last request $s_4 = (R, o, -(Q - S + 1))$. We distinguish cases based on the order in which the ALG server drops off s_3 and s_4.

Case 1.2.1: ALG drops off s_4 before s_3. The completion time of s_3 is at least $R + 1 + 2(Q - S + 1) + 1 \geq R + 8$ giving a maximum flow time of at least $R + 8 - Q \geq 10 \geq Q - S + 4$.

Case 1.2.2: ALG drops off s_3 before s_4. The flow time of s_4 is at least $R+1+2+Q-S+1-R = Q - S + 4$. Therefore also the maximum flow time of ALG is $Q - S + 4$.

Hence, we see that in all cases the maximum flow time of ALG is at least $Q - S + 4$. From property (ii) of S we know that the flow times of the requests served by OFF before time S do not exceed 3. From time S onward, the OFF algorithm can serve request s_2, wait until time Q and then serve s_1 and s_3 together. The yields flow times of 1, 1 and $Q - S + 1$ for r_1, r_2 and r_3 respectively. The

OFF server is back at the origin at time $Q + 2$.

In Case 1.1, OFF is finished and has a maximum flow time of $Q - S + 1$. In Case 1.2, the OFF server will wait at the origin until time R and then serve request s_4 as it is released, yielding a flow time of $Q - S + 1$. Therefore, also in Case 1.2 the maximum flow time of OFF is $Q - S + 1$. No further requests are released, therefore the complete request sequence is $\sigma = \rho \cup (s_1, s_2, s_3)$ in Case 1.1 and $\sigma = \rho \cup (s_1, s_2, s_3, s_4)$ in Case 1.2. In conclusion, in Case 1 the competitive ratio of ALG is at least

$$\frac{\text{ALG}(\sigma)}{\text{OPT}(\sigma)} \geq \frac{\text{ALG}(\sigma)}{\text{OFF}(\sigma)} \geq \frac{Q - S + 4}{Q - S + 1} \geq \frac{\sqrt{10} + 4}{\sqrt{10} + 1} = \frac{\sqrt{10} + 2}{3}.$$

Case 2: $\sqrt{10} < Q - S$. The adversary releases its last request $s_3 = (S + 4, o, -3)$. We distinguish cases based on the order in which the ALG server drops off s_2 and s_3.

Case 2.1: ALG drops off s_3 before s_2. The completion time of s_2 is at least $Q+1+6+1 = Q+8$ giving a maximum flow time of at least $8 + \sqrt{10} \geq 2 + \sqrt{10}$.

Case 2.2: ALG drops off s_2 before s_3. We see that the flow time of s_3 is $Q+1+2+3-(S+4) = Q - S + 2 \geq 2 + \sqrt{10}$.

The OFF server first serves s_1 and s_2; it is back at the origin at time $S + 4$. At this time, s_4 is released and will directly be served, yielding a maximum flow time of 3. Therefore:

$$\frac{\text{ALG}(\sigma)}{\text{OPT}(\sigma)} \geq \frac{\text{ALG}(\sigma)}{\text{OFF}(\sigma)} \geq \frac{\sqrt{10} + 2}{3}.$$

□

Next, we will investigate the performance of the ALTERNATE algorithm in this problem setting. Lower and upper bounds of 5 on the competitive ratio of ALTERNATE are proven, after which we conclude that ALTERNATE has a competitive ratio of 5.

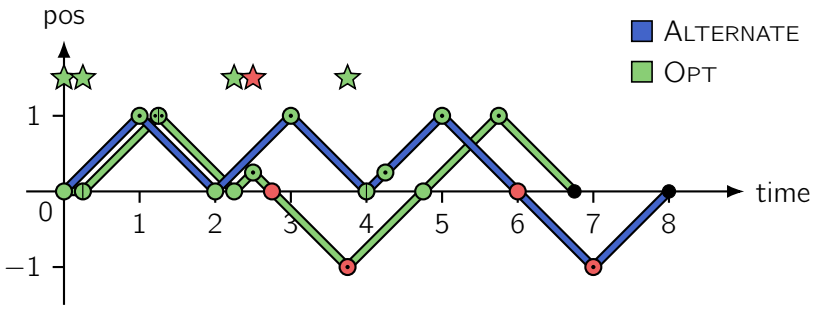


Figure 3.18: Illustration of Lemma 3.30 with $\varepsilon = 0.25$

Lemma 3.30. *ALTERNATE has a competitive ratio of at least 5 for ICPO-MF.*

Proof. Consider the sequence of requests $\sigma = (r_1, r_2, r_3, r_4, r_5)$ with $r_1 = (0, o, 1)$, $r_2 = (\varepsilon, o, 1)$, $r_3 = (2 + \varepsilon, 0, \varepsilon)$, $r_4 = (2 + 2\varepsilon, 0, -1)$, and $r_5 = (4 - \varepsilon, 0, 1)$; here ε is a sufficiently small positive number.

The ALTERNATE server first serves r_1 and ignores the release of request r_2. The server finishes request r_1 at $C_1 = 1$, and is back at the origin at time 2. Next request r_2 is served, which is dropped off at time $C_2 = 3$ and the server has returned to the origin at time 4. At this time there are three released but yet unserved requests. The ALTERNATE server will choose to go to the positive

part to serve requests r_3 and r_5 since r_3 has the earliest release time among the three. This yields completion times $C_3 = 4 + \varepsilon$ and $C_5 = 5$. Lastly, the ALTERNATE server picks up request r_4 and drops it off at time $C_4 = 7$. From the completion times and release times, the flow times of the requests follow:

$$F_1 = 1, \qquad F_2 = 3 - \varepsilon, \qquad F_3 = 2,$$
$$F_4 = 5 - 2\varepsilon, \qquad F_5 = 1 + \varepsilon.$$

Therefore $\text{ALTERNATE}_{F_{\max}}(\sigma) = 5 - 2\varepsilon$.

The optimal server starts by waiting ε time units until request r_2 is released, then it picks up both r_1 and r_2 and drops them off at time $C_1^* = C_2^* = 1 + \varepsilon$. When the server is back at the origin it serves request r_3 that has arrived, yielding completion time $C_3^* = 2 + 2\varepsilon$. At time $2 + 3\varepsilon$, the server is back at the origin and it serves requests r_4 and r_5 consecutively. This gives completion times $C_4^* = 3 + 3\varepsilon$ and $C_5^* = 5 + 3\varepsilon$. From the completion times and release times the following flow times can be calculated:

$$F_1^* = 1 + \varepsilon, \qquad F_2^* = 1, \qquad F_3^* = \varepsilon,$$
$$F_4^* = 1 + \varepsilon, \qquad F_5^* = 1 + 4\varepsilon.$$

Therefore $\text{OPT}_{F_{\max}}(\sigma) = 1 + 4\varepsilon$, and

$$\frac{\text{ALTERNATE}_{F_{\max}}(\sigma)}{\text{OPT}_{F_{\max}}(\sigma)} = \frac{5 - 2\varepsilon}{1 + 4\varepsilon}.$$

By making ε small enough, the lower bound on ALTERNATE's competitive ratio can be made arbitrarily close to 5.

□

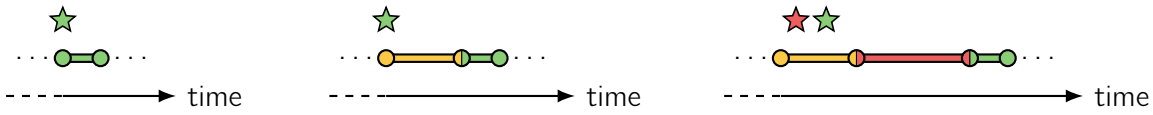
Lemma 3.31. *ALTERNATE is 5-competitive for ICPO-MF.*

Proof. Let σ be an arbitrary sequence of requests. We consider an arbitrary request $r_i \in \sigma$ and will prove its flow time F_i is not larger than 5 times the maximum flow F^* of the optimal algorithm OPT. Without loss of generality we can assume $b_i \geq 0$ as the proof for $b_i < 0$ is symmetric.

There are three ways the ALTERNATE server can serve request r_i, and these are listed below.

1. As soon as request r_i is released the ALTERNATE server leaves the origin to serve it.
2. When request r_i is released the ALTERNATE server is or begins serving requests at the negative or positive part of the line, and serves requests on the positive part including r_i as soon as its done.
3. When request r_i is released the ALTERNATE server is or begins serving requests at the negative or positive of the line, serves requests on the negative part when its done, and after that it serves requests on the positive part.

Note that these are indeed all possible cases. In particular, it is not possible that after r_i is released the ALTERNATE server would leave the origin twice without intending to drop off r_i. Indeed, if the ALTERNATE server leaves to the positive part of the origin it will also drop off r_i. The option that is left is that the server leaves to the negative part of the line two times in a row after r_i is released, but this is not possible. One can see this by noting that during the first excursion to the negative part of the line, all requests with release time no later than t_i are served. Therefore after that first excursion, r_i would be the request with earliest release time and the ALTERNATE server would decide to serve requests on the positive part of the line.



(a) Case 1 **(b)** Case 2 **(c)** Case 3

Figure 3.19: The three ways the ALTERNATE server can serve r_i. Red and green indicate requests with drop-off location on the negative and positive part of the line respectively. Orange indicates that requests were served either on the negative or the positive part of the line.

Consider the third case. Let r_f be the request in the first excursion with the furthest drop-off location. Similarly define r_m for the second excursion, the one to the negative part of the line. Now, note that for any request r_i we have $F^* \geq d(o, b_i)$ (Equation 3.3), therefore the flow time of request r_i is

$$F_i \leq 2d(o, b_f) + 2d(o, b_m) + d(o, b_i) \overset{(3.3)}{\leq} 5F^*.$$

Case 1 and 2 can be proven similarly to yield bounds of F^* and $3F^*$ respectively, which are both stricter than $5F^*$.

Since r_i was arbitrary we have $F_i \leq 5F^*$ for all requests r_i, hence $\text{ALTERNATE}_{F_{\max}}(\sigma) \leq 5\text{OPT}_{F_{\max}}(\sigma)$.

□

Theorem 3.32. *ALTERNATE has a competitive ratio of 5 for ICPO-MF.*

Proof. The result follows directly from Lemma 3.30 and 3.31. □

3.4.3 ICDO-MS

First we will show that this problem setting is equivalent to closed OLTSP, allowing us to use existing results. Afterwards, we investigate the performance of the ALTERNATE algorithm in this setting.

> **Lemma 3.33.** (part of Lemma 6 of [10]) *The ICDO-MS problem is equivalent to minimizing make-span for closed OLTSP on the real line.*

Proof. Let $\rho = (s_1, \ldots, s_n)$ be an arbitrary sequence of requests for closed OLTSP on the real line. Here $s_i = (t'_i, p_i)$ denotes a request released at time t'_i specifying a point p_i that has to be visited. We construct a sequence of requests for ICDO-MS: let $\sigma = (r_1, \ldots, r_n)$ with $r_i = (t_i, a_i, o)$, and set $t_i = t'_i$ and $a_i = p_i$. Any walk that solves ICDO-MS for request sequence σ is also valid for the OLTSP problem with input ρ since all positions are visited by the server and the walk ends at the origin. Also, the walk has the same make-span for both problems.

This conversion can also be done the other way around. In the same way, any route that solves the OLTSP problem for an input ρ is also valid for the ICDO-MS problem for the corresponding input σ. Any time the server is at a pick-up location, the request is picked up. Now because the route visits all points p_i for the OLTSP, also all pick-up locations are visited for the ICDO-MS problem. Since the server returns to the origin at the end, the server can drop off all requests at that moment.

□

Applying Lemma 3.33, we can use existing results of closed OLTSP on the line for the ICDO-MS setting.

> **Theorem 3.34.** (Theorem 3.3 of [3]) *No deterministic online algorithm has a competitive ratio smaller than* $(9 + \sqrt{17})/8 \approx 1.6404$ *for ICDO-MS.*

> **Theorem 3.35.** (Theorem 3.5 of [6]) *'Algorithm 1' has a competitive ratio of* $(9 + \sqrt{17})/8 \approx 1.6404$ *for ICDO-MS, and this is the best possible competitive ratio.*

The authors of [6] that introduce 'Algorithm 1' note that at any time t, sufficient input for an online algorithm is the position of the server at time t, and the left-most and right-most unserved requests at that time. Based on this observation, the authors describe 'Algorithm 1' which makes decisions only when a new left-most or right-most unserved request is released. 'Algorithm 1' uses a complicated set of rules to calculate how the server should serve all remaining requests, and return to the origin. This involves determining whether the server should wait, and whether it should first go to the left side of the line and then the right, or vice versa.

'Algorithm 1' as described in [6] runs in constant time, however it is only run when a new extreme request (a new left-most or right-most request) is released. One way of determining whether a released request is an extreme request, would be to simply go through all unserved requests. This would take $\Theta(|R|)$ time where $|R|$ is the number of unserved requests. Other approaches may allow checking whether a request is extreme in constant time, but this most likely requires keeping a data structure updated. In any case, it is likely that the lowest possible time complexity of 'Algorithm 1' is non-constant, but instead depends on the number of unserved requests in some way.

We now return to our analysis of simple algorithms, and investigate the performance of the ALTERNATE algorithm for this problem. Even though an algorithm with a best-possible competitive ratio for this problem is already known, analyzing ALTERNATE is still interesting. In particular,

ALTERNATE is simpler, both in terms of understanding and implementing it, as most likely in time complexity.

First, we give a lower bound on the competitive ratio of ALTERNATE in Lemma 3.36.

Lemma 3.36. *ALTERNATE has a competitive ratio of at least 2.5 for ICDO-MS.*

Proof. Consider the sequence of requests $\sigma = (r_1, r_2)$ with $r_1 = (1, o, 1)$ and $r_2 = (\varepsilon, o, 1)$ for some small $\varepsilon < 2$. The ALTERNATE server picks up r_1 when it is released at time 1 and serves it. While serving r_1 the ALTERNATE server ignores the release of request r_2. After the server is back at the origin it will serve request r_2, resulting in a make-span of $\text{ALTERNATE}_{C_{\max}}(\sigma) = 5$. The OPT server would start driving towards point 1 at time 0, wait ε time units, and serve both request r_1 and r_2 at the same time. This yields an optimal make-span of $\text{OPT}_{C_{\max}}(\sigma) = 2 + \varepsilon$. Now, taking ε small enough, we see that the lower bound on ALTERNATE's competitive ratio can be made arbitrarily close to $\frac{5}{2} = 2.5$. □

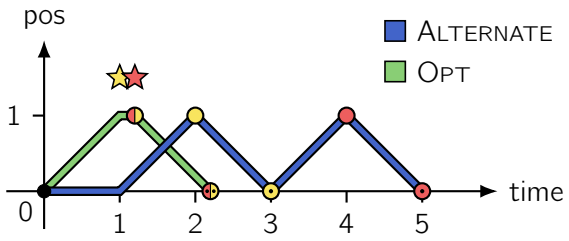


Figure 3.20: Illustration of Lemma 3.36 with $\varepsilon = 0.2$

Now that we have a lower bound on the competitive ratio of ALTERNATE, we match this with a corresponding upper bound in Lemma 3.38. The competitiveness proof is very similar to that of the ICPO-MS problem, Lemma 3.20. We also use similar notation, but in the definitions the drop-off locations are substituted by the pick-up location. For completeness, the definitions are given below.

Definition. Let S be a non-empty set or sequence of requests on one side of the line. Then let $L(S)$ denote the length of the shortest tour visiting all pick-up locations from requests in S, starting and ending at the origin. We have

$$L(S) = 2 \max_{r_i \in S} |a_i|.$$

Definition. For some set or sequence of requests ρ, define

$$\rho^+ = \{r_i \in \rho \mid a_i \geq o\}, \quad \text{and,} \quad \rho^- = \{r_i \in \rho \mid a_i < o\}.$$

Also, the same observation can be made as before:

Observation 3.37. Any algorithm has to serve the left-most request and the right-most request while starting and ending at the origin. Hence,

$$\text{OPT}_{C_{\max}}(\sigma) \geq L(\sigma^-) + L(\sigma^+). \tag{3.15}$$

Lemma 3.38. *ALTERNATE is 2.5-competitive for ICDO-MS.*

Proof. We distinguish cases based on what the ALTERNATE server is doing at the time t_n at which the last request r_n becomes known. If the ALTERNATE server is idle at the origin then we simply serve the request giving

$$\textsc{Alternate}_{C_{\max}}(\sigma) = t_n + 2d(o, b_n) \overset{(3.2)}{\leq} \textsc{Opt}_{C_{\max}}(\sigma) + d(o, b_n) \overset{(3.1)}{\leq} \frac{3}{2}\textsc{Opt}_{C_{\max}}(\sigma),$$

and the lemma holds.

Now, we consider the case where the ALTERNATE server is not idle at the origin at time t_n, but instead it is following some schedule $\mathcal{S}$ serving requests at one side of the line. The arrival of request r_n does not stop the ALTERNATE server from following $\mathcal{S}$; it ignores this request and continues with its schedule. Let T be the time it started schedule $\mathcal{S}$, and let R denote the set of unserved requests at that time T. The requests served by the ALTERNATE server in schedule $\mathcal{S}$ are either the requests R^- or R^+, whichever set has the request with minimum release time. Without loss of generality we assume that R^+ contains the requests that are served in schedule $\mathcal{S}$. Now, the make-span of ALTERNATE can be expressed as:

$$\textsc{Alternate}_{C_{\max}}(\sigma) = T + L(R^+) + L(R^- \cup \sigma^-_{\geq T}) + L(\sigma^+_{\geq T}). \tag{3.16}$$

Indeed, the ALTERNATE server finishes schedule $\mathcal{S}$ at time $T + L(R^+) \geq t_n$; this is later than t_n because recall the ALTERNATE server was performing schedule $\mathcal{S}$ when request r_n was released. This means no new requests arrive after schedule $\mathcal{S}$ is finishes. Therefore, the last task for the ALTERNATE server is to serve the remaining requests with drop-off location on the negative part of the line $R^- \cup \sigma^-_{\geq T}$, and the remaining requests at the positive part $\sigma^+_{\geq T}$.

For the OPT server, consider the first request r_f from $\sigma_{\geq T}$ that is picked up by the OPT server. Since r_f is in $\sigma_{\geq T}$, by definition $t_f \geq T$. The OPT server has to at least wait until this release time t_f, and then serve all requests in $\sigma_{\geq T}$. The OPT server therefore needs to move to the furthest negative request and positive request in $\sigma_{\geq T}$, and return to the origin. If the server would start at the origin, then this would take at least $L(\sigma^+_{\geq T}) + L(\sigma^-_{\geq T})$ time; a distance of $d(o, a_f)$ may have already been covered though, therefore this can be subtracted. This yields

$$\textsc{Opt}_{C_{\max}}(\sigma) \geq t_f + L(\sigma^+_{\geq T}) + L(\sigma^-_{\geq T}) - d(o, a_f) \geq T + L(\sigma^+_{\geq T}) + L(\sigma^-_{\geq T}) - d(o, a_f). \tag{3.17}$$

Using Equations 3.16 and 3.17 we can bound $\textsc{Alternate}_{C_{\max}}(\sigma)$ by $\textsc{Opt}_{C_{\max}}(\sigma)$. First, from Equation 3.16 we see

$$\textsc{Alternate}_{C_{\max}}(\sigma) \overset{(3.16)}{\leq} T + L(R^+) + L(R^- \cup \sigma^-_{\geq T}) + L(\sigma^+_{\geq T}).$$

We rearrange and add and subtract appropriate terms to get

$$\textsc{Alternate}_{C_{\max}}(\sigma) \leq T + L(\sigma^+_{\geq T}) + L(\sigma^-_{\geq T}) + L(R^+) + L(R^- \cup \sigma^-_{\geq T}) - L(\sigma^-_{\geq T}).$$

Next, we apply Equation 3.17, which yields

$$\begin{aligned} &\overset{(3.17)}{\leq} \textsc{Opt}_{C_{\max}}(\sigma) + d(o, a_f) + L(R^+) + L(R^- \cup \sigma^-_{\geq T}) - L(\sigma^-_{\geq T}) \\ &\leq \textsc{Opt}_{C_{\max}}(\sigma) + d(o, a_f) + L(R^+) + L(R^- \cup \sigma^-_{\geq T}). \end{aligned}$$

The optimal make-span is at least $2d(o, a_f)$ (Observation 3.1), therefore

$$\overset{(3.1)}{\leq} \frac{3}{2}\textsc{Opt}_{C_{\max}}(\sigma) + L(R^+) + L(R^- \cup \sigma^-_{\geq T}).$$

Note that $R^+ \subseteq \sigma^+$ and $R^- \cup \sigma^-_{\geq T} \subseteq \sigma^-$, therefore using Inequality 3.15 we have

$$\textsc{Alternate}_{C_{\max}}(\sigma) \overset{(3.15)}{\leq} \frac{3}{2}\textsc{Opt}_{C_{\max}}(\sigma) + \textsc{Opt}_{C_{\max}}(\sigma) = \frac{5}{2}\textsc{Opt}_{C_{\max}}(\sigma),$$

and the lemma also holds in this case. □

Theorem 3.39. *ALTERNATE has a competitive ratio of 2.5 for ICDO-MS.*

Proof. Follows directly from Lemma 3.36 and 3.38. □

Offline setting

The dynamic programming approach of Algorithm 5 from [6] solves the closed offline travelling salesman problem (TSP) in $\mathcal{O}(n^2)$ time, where n is the number of requests. In Lemma 3.33 we proved the online ICDO-MS problem is equivalent to OLTSP, the same argument applies to the offline setting. Therefore, Algorithm 5 from [6] solves offline ICDO-MS in $\mathcal{O}(n^2)$ time. As noted before, for proving the offline algorithm for ICPO-MS we used a similar structure as in [6], therefore the approach outlined in the paper is similar to the offline discussion in Section 3.4.1. First, a property of feasible tours is identified and proven, this structure reveals optimal substructure which is then exploited to obtain a recurrence for the optimal make-span. Lastly, the dynamic program is given for calculating the optimal make-span in Algorithm 5 in [6].

In Algorithm 5 in [6], the authors use the assumption that $t_i \geq |a_i|$ (using the notation from this report). This assumption can be made, without loss of generality, because the server cannot reach $|a_i|$ before time t_i, therefore transforming input to match this assumption does not change the set of feasible solutions. This does entail that before applying Algorithm 5 the input has to be modified to match the assumption, or the base cases need to be changed from t_{-L} and t_R to $\max\{t_{-L}, |a_{-L}|\}$ and $\max\{t_R, |a_R|\}$ so that the algorithm works for any input.

3.4.4 ICDO-MF

The lower bound on the competitive ratio of any deterministic online algorithm in the UCDO-MF section only used one request, therefore it also holds for the ICDO-MF problem setting. This is stated in the following theorem.

> **Theorem 3.40.** *No deterministic online algorithm ALG has a competitive ratio smaller than 2 for ICDO-MF.*

Proof. Identical to the proof of Theorem 3.13. □

The analysis of the ALTERNATE algorithm for ICDO-MF follows the same structure as the one for ICPO-MF. In Lemma 3.41 we give a lower bound of 6 on the competitive ratio of ALTERNATE for ICDO-MF, after which we give the 6-competitive proof thereby proving a competitive ratio of 6.

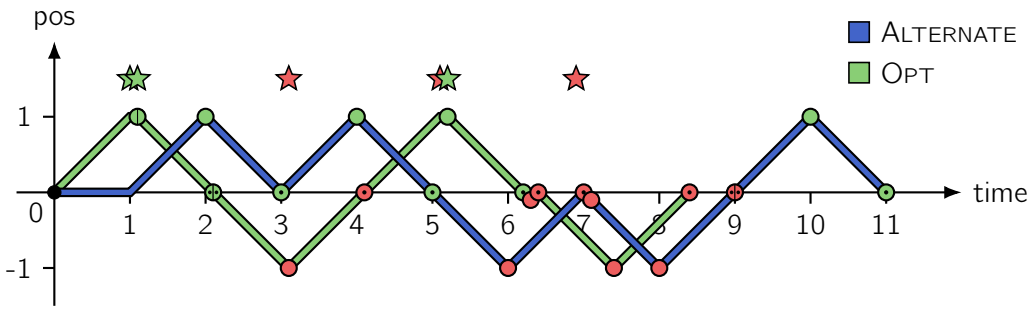


Figure 3.21: Illustration of Lemma 3.41 with $\varepsilon = 0.1$

> **Lemma 3.41.** *ALTERNATE has a competitive ratio of at least 6 for ICDO-MF.*

Proof. Consider the sequence of requests $\sigma = (r_1, r_2, r_3, r_4, r_5, r_6)$, with $r_1 = (1, 1, 0)$, $r_2 = (1+\varepsilon, 1, 0)$, $r_3 = (3+\varepsilon, -1, 0)$, $r_4 = (5+\varepsilon, -\varepsilon, 0)$, $r_5 = (5+2\varepsilon, 1, 0)$ and $r_6 = (7-\varepsilon, -1, 0)$ for some small positive ε.

The ALTERNATE server leaves the origin at time 1 to serve request r_1, and ignores the release of request r_2. After having dropped off r_1 at the origin at time $C_1 = 3$, the server operated by ALTERNATE will serve request r_2 and drop it off at $C_2 = 5$. Next the server goes to the negative part of the line to serve request r_3, and drops it off at time $C_3 = 7$. Back at the origin at time 7, the released but yet unserved requests are r_4, r_5 and r_6. Request r_4 with pick-up location at the negative part of the line has the earliest release time, hence the ALTERNATE server serves requests r_4 and r_6 yielding $C_4 = C_6 = 9$. Lastly request r_5 is served, dropping it off at time $C_5 = 11$. From the completion times and release times the following flow times follow:

$$F_1 = 2, \qquad F_2 = 4-\varepsilon, \qquad F_3 = 4-\varepsilon,$$
$$F_4 = 4-\varepsilon, \qquad F_5 = 6-2\varepsilon, \qquad F_6 = 2+\varepsilon.$$

Therefore $\text{ALTERNATE}_{F_{\max}}(\sigma) = 6 - 2\varepsilon$.

The OPT server moves to position 1 at time 0, and waits there for ε time units until request r_2 is released. Then it picks both up and drops them off at time $C_1 = C_2 = 2+\varepsilon$. Afterwards, the server moves to point -1 to pick up request r_3 at time $3+\varepsilon$ and drops it off at the origin at time $C_3 = 4+\varepsilon$. Next, the server moves to point 1 and waits there for ε time units to pick up request r_5 when it is released, and drops it off at time $C_5 = 6+2\varepsilon$. Now, the server goes to location $-\varepsilon$ to pick up request r_4, and immediately returns to the origin to drop it off at time $C_4 = 6+4\varepsilon$. Finally

the OPT server picks up r_6 to then drop it off at time $C_6 = 8 + 4\varepsilon$. Subtracting each release time from the corresponding completion time results in the following flow times:

$$F_1^* = 1 + \varepsilon, \qquad F_2^* = 1, \qquad F_3^* = 1,$$
$$F_4^* = 1 + 3\varepsilon, \qquad F_5^* = 1, \qquad F_6^* = 1 + 5\varepsilon.$$

Hence the optimal maximum flow time for input σ is $\text{OPT}_{F_{\max}}(\sigma) = 1 + 5\varepsilon$. The lower bound on the competitive ratio of ALTERNATE is therefore

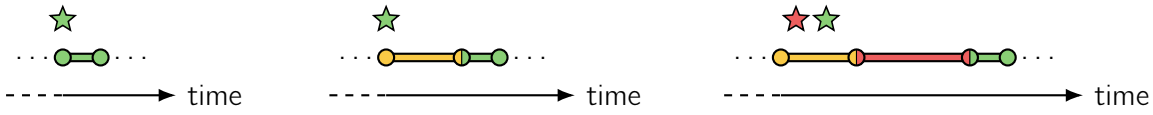
$$\frac{\text{ALTERNATE}_{F_{\max}}(\sigma)}{\text{OPT}_{F_{\max}}(\sigma)} = \frac{6 - 2\varepsilon}{1 + 5\varepsilon}.$$

By taking ε small enough, this lower bound on ALTERNATE's competitive ratio can be made arbitrarily close to 6. □

Lemma 3.42. *ALTERNATE is 6-competitive for ICDO-MF.*

Proof. Let σ be an arbitrary sequence of requests. We consider an arbitrary request $r_i \in \sigma$ and will prove its flow time F_i is not larger than 6 times the maximum flow F^* of the optimal algorithm OPT. Without loss of generality we can assume $a_i \geq 0$ as the proof for $a_i < 0$ is symmetric.

Similar to Lemma 3.31, there are three ways the ALTERNATE server can serve request r_i, which are depicted in Figure 3.22.



(a) Case 1 **(b)** Case 2 **(c)** Case 3

Figure 3.22: The three ways the ALTERNATE server can serve r_i. Red and green indicate requests with pick-up location on the negative and positive part of the line respectively. Orange indicates that requests were served either on the negative or the positive part of the line.

Consider case 3. In this case, at the time that r_i is released, the ALTERNATE server is busy serving requests at some part of the line, and serves requests at the negative part afterwards. Let r_f be the request in the first excursion with the furthest pick-up location, and define r_m similarly for the second excursion. Now, using the fact that for any request r_i we have $F^* \geq d(o, b_i)$ (Equation 3.3), the flow time of request r_i can be bounded:

$$F_i \leq 2d(o, b_f) + 2d(o, b_m) + 2d(o, b_i) \overset{(3.3)}{\leq} 6F^*.$$

Case 1 and 2 can be proven similarly to yield bounds of $2F^*$ and $4F^*$ respectively, which are both stricter than $6F^*$.

Since r_i was arbitrary we have $F_i \leq 6F^*$ for all requests r_i, hence $\text{ALTERNATE}_{F_{\max}}(\sigma) \leq 6\text{OPT}_{F_{\max}}(\sigma)$.

□

Theorem 3.43. *ALTERNATE has a competitive ratio of 6 for ICDO-MF.*

Proof. The result follows directly from Lemma 3.41 and 3.42. □

Chapter 4

Stochastic Analysis

In the previous chapter, we conducted a competitive analysis where a type of worst-case performance of online algorithms is considered. This chapter is about the average performance of HOMESICK and ALTERNATE. We make an assumption on the probability distribution of the input, and we will see that then the unit capacity problem settings have a very close connection to queueing systems. Queueing theory allows us to then compute the expected flow time of requests in steady state.

4.1 Preliminaries

Before outlining how the unit capacity problem settings can be viewed as a queueing system, we list some basic information about queueing systems.

Queueing systems

A queueing system, an example is depicted in Figure 4.1, consists of three main parts: arriving tasks (often called customers), a queue, and servers that perform tasks. There are several parameters that vary between queueing systems, five of which are listed below:

- The arrival process of tasks.
- The service times of tasks.
- The number of servers.
- The service discipline.
- The size of the queue.

Kendall [14] introduced a notation for classifying queueing systems based on these different parameters, namely a code consisting of three parts: $A/S/k$. Here A describes the arrival process, S the service time distribution and k the number of servers. There are also extensions of this notation where more parameters are described, such as the service discipline or size of the queue. Unless otherwise specified, in these queueing systems it is assumed that the queue has infinite size, and that the service discipline is first-come-first-served (FCFS).

Poisson process

The standard (homogeneous) Poisson process models random events occurring over time at a fixed rate $\lambda \in \mathbb{R}_{>0}$. We will not go into the formal definition of a Poisson process, but briefly list some of its convenient properties.

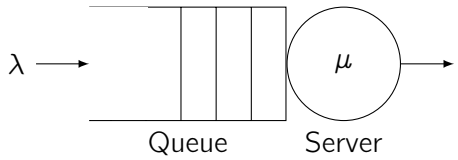


Figure 4.1: A standard queueing system

- The times between events are exponentially distributed with parameter λ.
- The number of events that occur in t time units is distributed according to a Poisson distribution with parameter $t\lambda$. More specifically, letting $N(t)$ denote the number of events that occur in the interval $[0, t]$, then

$$\mathbb{P}(N(t) = n) = e^{-t\lambda}\frac{(t\lambda)^n}{n!}.$$

  The Poisson process has independent increments. This implies that the distribution of the number of events in an interval of time only depends on the length of the interval, not on previous events. Therefore, for any $s, t \geq 0$ we have $\mathbb{P}(N(s+t) - N(s) = n) = \mathbb{P}(N(t) = n)$. Also note that $\mathbb{E}[N(t)] = \lambda t$, so λ can be viewed as the expected number of requests that arrive per time unit.
- Given that an event occurs within an interval of time $[s, t]$ with $t \geq s$, then the time T at which the event occurs is uniformly distributed in $[s, t]$.

Queueing systems that use Poisson arrivals have a useful property. Namely, the probability that the system is in some state upon arrival of a task is equal to the probability that the system is in that state in any given time. This is often referred to with the acronym PASTA which stands for Poisson Arrivals See Time Averages.

Little's law

Little's law expresses an intuitive but powerful relation. Namely, in a queueing system, the expected number of tasks in the queue is equal to the expected time a task spends in the queue multiplied by the arrival rate of tasks. Letting λ denote the arrival rate, L the length of the queue, and W the time spent in the queue (the waiting time), then

$$\mathbb{E}[L] = \lambda \cdot \mathbb{E}[W].$$

4.2 The HOMESICK algorithm

We can think of the OLDARP where the server has unit capacity and is controlled by the HOMESICK algorithm as a queueing system. We abstract away from HOMESICK serving requests in the UCPO and UCDO settings, and outline how these can more generally be seen as tasks. Such a task encapsulates all the work done by HOMESICK to serve a request: moving towards the pick-up location to pick up the request, dropping off the request at the drop-off location, and returning to the origin. A certain request $r_i = (t_i, a_i, b_i)$ can be considered as a task τ_i as follows:

- The task τ_i arrives at the request's release time t_i.
- The service time of task τ_i is $d(o, a_i) + d(a_i, b_i) + d(b_i, o)$; this is $2d(o, b_i) = 2|b_i|$ for the UCPO setting, and $2d(o, a_i) = 2|a_i|$ for the UCDO setting.

Notice that in the UCPO setting, if the drop-off locations of requests are distributed according to some distribution G, then the service time distribution of tasks is $2|G|$. The same holds for the UCDO setting, but then the pick-up locations are distributed according to G. Now that we have defined what tasks are and determined their service times, we need to determine the other parameters of the queueing system. For the arrival process, it is not too unreasonable to assume that requests, and therefore tasks, arrive according to a Poisson process. We make this assumption, and do not consider an arbitrary arrival process, because a Poisson process allows for an elegant mathematical analysis. For the service discipline it is clear that the HOMESICK algorithm translates to a FCFS service discipline. Lastly, in our OLDARP settings there is only one server, and there is no limit on the number of unserved requests. Therefore, in the queueing system there is also only one server, and the size of the queue is infinite. Combining the parameters, we find the UCPO and UCDO problems can be viewed as an $M/G/1$ queueing system. The M means Poisson arrivals and the G denotes that service times are identically and independently distributed (i.i.d.) according to some unspecified distribution G.

For analyzing the $M/G/1$ queue, we follow the mean value approach for analyzing $M/G/1$ queueing systems as described in the lecture notes written by Ivo Adan et al. for the course "Queueing Systems" at the TU/e [1]. We use λ to denote the arrival rate of the Poisson process. Furthermore, we use the continuous random variable B to denote the service time distribution, and ρ to denote the fraction of time the server is busy serving tasks. We assume $\rho < 1$; in this case $\rho = \lambda \cdot \mathbb{E}[B]$, the expected amount of 'work units' that arrive per time unit since HOMESICK serves tasks as soon as they arrive. Lastly, for a random variable X, we write f_X and F_X for the probability density function and cumulative distribution function respectively.

4.2.1 Waiting time of a task

Adapted from Section 7.6 of [1].

Consider a task τ arriving in the steady state of HOMESICK. We will investigate the time between the arrival of τ and the time the server leaves the origin to perform task τ. We call this time the waiting time of task τ, and denote it with the random variable W_τ.

To calculate W_τ, we use the law of total expectation and condition on whether the server is busy upon arrival of task τ:

$$\begin{aligned}\mathbb{E}[W_\tau] = \mathbb{E}[W_\tau \mid \text{server is busy upon arrival}] \cdot \mathbb{P}(\text{server is busy upon arrival}) \\ + \mathbb{E}[W_\tau \mid \text{server is idle upon arrival}] \cdot \mathbb{P}(\text{server is idle upon arrival}).\end{aligned}$$

If the server is idle at the moment task τ arrives, then the server will leave the origin immediately to perform the task. Therefore $\mathbb{E}[W_\tau \mid \text{server is idle upon arrival}] = 0$, and the second term vanishes. If instead, the server is busy upon arrival, then the server first needs to finish its current task, and then perform each task that is before τ in the queue. Let the random variable R denote the residual service time of the task that is currently being served by the HOMESICK server, and let L denote the number of requests that are in the queue upon arrival of task τ. Then,

$$\mathbb{E}[W_\tau \mid \underbrace{\text{server is busy upon arrival}}_{A}] = \mathbb{E}[R] + \mathbb{E}[L \mid A] \cdot \mathbb{E}[B].$$

To calculate $\mathbb{E}[L \mid A]$, we apply the law of total expectation to $\mathbb{E}[L]$ as follows:

$$\mathbb{E}[L] = \mathbb{E}[L \mid A] \cdot \mathbb{P}(A) + \underbrace{\mathbb{E}[L \mid \text{server is idle upon arrival}]}_{=0} \cdot \mathbb{P}(\text{server is idle upon arrival}).$$

Now, the PASTA property for queueing systems with Poisson arrivals tells us that arriving tasks encounter on average the same situation as an outside observer. Therefore, recalling that ρ denotes the fraction of time a server is busy, we have

$$\mathbb{P}(A) = \mathbb{P}(\text{server is busy}) = \rho, \qquad \text{and hence,} \qquad \mathbb{E}[L \mid A] = \frac{\mathbb{E}[L]}{\rho}.$$

Finally, by using the previous equations we find

$$\mathbb{E}[W_\tau] = \mathbb{E}[W_\tau \mid A] \cdot \mathbb{P}(A) = (\mathbb{E}[R] + \mathbb{E}[L \mid A] \cdot \mathbb{E}[B]) \cdot \rho = \rho \cdot \mathbb{E}[R] + \mathbb{E}[L] \cdot \mathbb{E}[B]. \tag{4.1}$$

Applying Little's law to the queue, we find that the mean number of tasks in the queue is the product of the mean time a task spends in the queue and the arrival rate of new tasks. This means we have $\mathbb{E}[L] = \mathbb{E}[W_\tau] \cdot \lambda$. Substituting this expression in Equation 4.1 yields

$$\mathbb{E}[W_\tau] = \rho \cdot \mathbb{E}[R] + \mathbb{E}[W_\tau] \cdot \underbrace{\lambda \cdot \mathbb{E}[B]}_{=\rho} \implies (1-\rho)\mathbb{E}[W_\tau] = \rho \cdot \mathbb{E}[R] \implies \mathbb{E}[W_\tau] = \frac{\rho \cdot \mathbb{E}[R]}{1-\rho}.$$

The expected residual service time $\mathbb{E}[R]$ is still unknown; we will calculate this in the next section.

4.2.2 Residual service time

Adapted from Section 7.7 of [1].

We now consider in more detail the case where a task arrives when the server is busy. Our aim is to compute the expected residual service time, i.e., the expected amount of time it takes for the server to finish its current task in service. To compute this residual service time, we first consider the total service time X of the task in service. The probability density function of X, $f_X(x)$, is not exactly equal to $f_B(x)$. Indeed, given that the server is busy, a task is more likely to arrive during longer service times, therefore $f_X(x)$ is directly proportional to x. Letting $C \in \mathbb{R}$ be a constant, we see

$$f_X(x) = C \cdot x \cdot f_B(x).$$

A density function should integrate to 1, therefore

$$\int_0^\infty C \cdot x \cdot f_B(x)\,\mathrm{d}x = C \cdot \mathbb{E}[B] = 1 \implies C = \frac{1}{\mathbb{E}[B]}, \quad \text{hence,} \quad f_X(x) = \frac{x \cdot f_B(x)}{\mathbb{E}[B]}.$$

Using the distribution of the total service time, we can calculate the distribution of the residual service time. First we condition on the total service time. Suppose the task we are considering arrives in a service time of length x, then its arrival time is uniformly distributed in that service interval. From this it is easily seen that the residual service time is uniformly distributed on $[0, x]$, hence

$$f_{R|X}(r \mid X = x) = \begin{cases} \frac{1}{x} & \text{if } r \in [0, x] \\ 0 & \text{if } r \notin [0, x] \end{cases}.$$

Using the above, we can calculate the probability density function of R to find

$$f_R(r) = \int_{x=r}^\infty f_{R|X}(r \mid X = x) f_X(x)\,\mathrm{d}x = \int_{x=r}^\infty \frac{1}{x} \cdot \frac{x \cdot f_B(x)}{\mathbb{E}[B]}\,\mathrm{d}x = \frac{1}{\mathbb{E}[B]} \int_{x=r}^\infty f_B(x)\,\mathrm{d}x = \frac{1 - F_B(r)}{\mathbb{E}[B]}.$$

Finally, the expected residual service time can be calculated:

$$\mathbb{E}[R] = \int_{r=0}^\infty r \cdot f_R(r)\,\mathrm{d}r = \frac{1}{\mathbb{E}[B]} \int_{r=0}^\infty r(1 - F_B(r))\,\mathrm{d}r$$

Using partial integration, we see

$$\begin{aligned} \int_{r=0}^\infty r(1 - F_B(r))\,\mathrm{d}r &= \left[\frac{1}{2} r^2 \cdot (1 - F_B(r))\right]_{r=0}^\infty + \frac{1}{2} \int_{r=0}^\infty r^2 \cdot f_B(r)\,\mathrm{d}r \\ &= \lim_{M \to \infty} \left(\frac{1}{2} M^2 \cdot (1 - F_B(M))\right) + \frac{1}{2}\mathbb{E}[B^2] \end{aligned}$$

We make the following assumption.

> *Assumption* 4.1. For the service time distribution B, we have
>
> $$\lim_{M \to \infty} \left(M^2 \cdot (1 - F_B(M))\right) = 0.$$

Assumption 4.1 holds for the most commonly used service time distributions, but not for all probability distributions (e.g. the Pareto distribution with $1 - F(x) = \frac{1}{x}$ for $x \geq 1$).

If Assumption 4.1 holds, then

$$\mathbb{E}[R] = \frac{\mathbb{E}[B^2]}{2\mathbb{E}[B]}.$$

4.2.3 Shortest-processing-time-first discipline

We have derived the expected waiting time for a task when the server uses the FCFS service discipline. This service discipline is not optimal with respect to the expected waiting time of tasks. A service discipline that performs better in this regard is the shortest-processing-time-first (SPTF) discipline. The SPTF discipline, just like the FCFS discipline, serves tasks as soon as they arrive. However, the disciplines differ in their actions when there are multiple tasks in the queue. FCFS will serve the task that arrived longest ago, the SPTF discipline, on the other hand, will serve the task that has the shortest service time. The SPTF makes the optimal choice, as proven in [1], and we make a brief intuitive argument based on [1]. To see SPTF makes the optimal choice, one can imagine n tasks in the queue that are served by the server in the order $\tau_1, \ldots, \tau_n$. Let b_i denote the service time of task τ_i. Suppose no new tasks arrive, then the total waiting time of the tasks is

$$\sum_{i=1}^{n} \sum_{j=1}^{i-1} b_j = \sum_{i=1}^{n} (n-i) b_i.$$

This is seen from the fact that task τ_i needs to wait until the tasks before it in the order, $\tau_1, \ldots, \tau_{i-1}$, are completed. The earlier a task is in the ordering, the larger the coefficient with which its service time is multiplied. Therefore, it is clear that to obtain the lowest total waiting time, the tasks should be ordered such that $b_1, b_2, \ldots b_n$ are in increasing order. Recalling that b_i denotes the service time of request r_i, we hence want to serve the task with shortest processing time first in order to minimize the total (and therefore average) waiting time.

When the server uses the SPTF discipline, the expected waiting time of tasks is

$$\mathbb{E}[W_\tau] = \rho \cdot \mathbb{E}[R] \cdot \int_{x=0}^{\infty} \frac{f_B(x)\,\mathrm{d}x}{(1 - \lambda \cdot \int_{y=0}^{y=x} y f_B(y)\,\mathrm{d}y)^2},$$

as derived in [1]. If Assumption 4.1 holds, then $\mathbb{E}[R]$ can be substituted to yield

$$\mathbb{E}[W_\tau] = \rho \cdot \frac{\mathbb{E}[B^2]}{2\mathbb{E}[B]} \cdot \int_{x=0}^{\infty} \frac{f_B(x)\,\mathrm{d}x}{(1 - \lambda \cdot \int_{y=0}^{y=x} y f_B(y)\,\mathrm{d}y)^2}.$$

4.2.4 Expected flow time of requests

In the previous sections we have considered W_τ, the waiting time of a task. We now return to the OLDARP setting and consider requests and their flow times. For deterministic inputs we investigated the maximum flow time (Chapter 3); in this section, we will calculate the expected flow time. The flow time of a request can be obtained simply by taking its waiting time and adding the time it takes to move from the request's pick-up location to the drop-off location. Therefore, when the server is controlled by the HOMESICK algorithm, we have

$$\mathbb{E}[F_r] = \mathbb{E}[W_r] + \frac{1}{2}\mathbb{E}[B].$$

For the UCPO problem, the server picks up a request r as soon as it starts the associated task τ, therefore the waiting time W_r of r equals W_τ. For the UCDO problem, driving towards the pick-up location of a request r takes half the service time of the associated task τ, therefore

$$\mathbb{E}[W_r] = \mathbb{E}[W_\tau + B] = \mathbb{E}[W_\tau] + \frac{1}{2}\mathbb{E}[B].$$

Next, we give an example.

Example. Consider the UCDO problem where pick-up locations are uniformly distributed on $[-1, 1]$ and requests arrive according to a Poisson process with rate λ. As described at the start of Section 4.2, this corresponds to tasks with service time $B \sim \text{Unif}([0, 2])$. Before we continue, we verify that Assumption 4.1 holds. Since $F_B(x) = 1$ for $x > 1$, we see easily

$$\lim_{M\to\infty} \left(M^2 \cdot (1 - F_B(M))\right) = 0.$$

Note that for the chosen service time distribution we have $\mathbb{E}[B] = 1$, $\mathbb{E}[B^2] = \frac{4}{3}$ and $\rho = \mathbb{E}[B] \cdot \lambda = \lambda$. Therefore, when the FCFS service discipline is used, the expected flow time of a request is

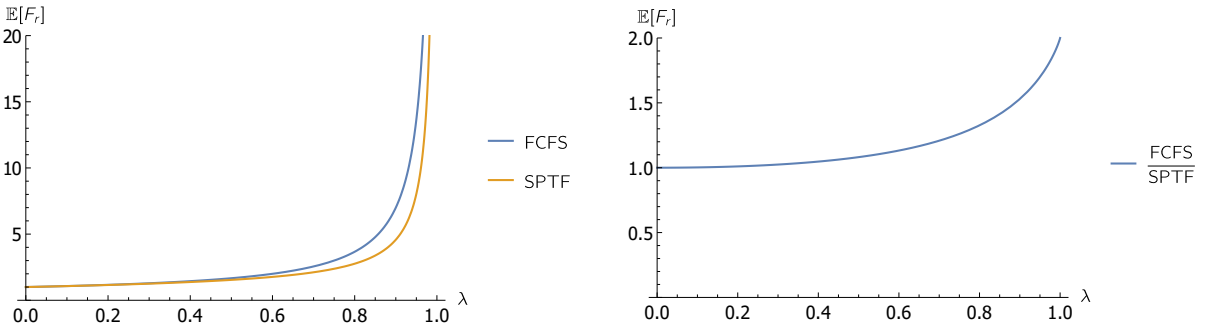
$$\mathbb{E}[F_r] = \mathbb{E}[W_\tau] + \mathbb{E}[B] = \frac{\lambda}{1-\lambda} \cdot \frac{2}{3} + 1.$$

For the SPTF service discipline, the expected flow time of a request is

$$\begin{aligned}
\mathbb{E}[F_r] &= \rho \cdot \frac{\mathbb{E}[B^2]}{2\mathbb{E}[B]} \cdot \int_{x=0}^{\infty} \frac{f_B(x)\,\mathrm{d}x}{(1 - \lambda \cdot \int_{y=0}^{y=x} y f_B(y)\,\mathrm{d}y)^2} + \mathbb{E}[B] \\
&= \frac{2}{3}\lambda \cdot \int_{x=0}^{2} \frac{\frac{1}{2}\,\mathrm{d}x}{(1 - \lambda \cdot \int_{y=0}^{y=x} \frac{1}{2} y\,\mathrm{d}y)^2} + 1 \\
&= \frac{2}{3}\lambda \cdot \int_{x=0}^{2} \frac{1}{2(1 - \frac{1}{4}\lambda x^2)^2}\,\mathrm{d}x + 1.
\end{aligned}$$

The expected flow times of requests are plotted in Figure 4.2. Figure 4.2a shows the expected flow times separately for the two service disciplines FCFS and SPTF; Figure 4.2b plots the ratio of the flow times of the two disciplines.

We have seen that the SPTF discipline results in lower expected flow times than FCFS. It is natural to wonder whether this can be quantified in terms of competitive ratios for the objective of minimizing the average flow time of a request. It turns out that, for the competitive ratio as we have defined it, the SPTF and FCFS algorithms cannot be distinguished in terms of competitive ratio. This is because there can be no deterministic online algorithm with a constant competitive ratio. Indeed in e.g. the UCPO setting, the adversary can start by releasing a request $r_1 = (0, o, 1)$.



(a) Expected flow times, tend to ∞ for λ to 1

(b) Ratio of expected flow times, tends to 2 for λ to 1

Figure 4.2: Expected flow times of the FCFS and SPTF disciplines for different values of λ. Here, the service time follows a uniform distribution on $[0, 2]$.

Just as the request is picked up by the online algorithm, the adversary releases N requests with drop-off location the origin. In this way, all requests served by the online algorithm have flow time at least 1, while the average flow time of requests served by the optimal offline algorithm goes to 0 as N goes to infinity.

One could also think of the SPTF discipline for the objective of minimizing maximum flow. For this objective, SPTF also has no constant competitive ratio. Indeed, if there is one request with a drop-off or pick-up location far from the origin (large request), and there are regular arrivals of requests that are close by (small requests), then SPTF will ignore the large request until there are no small requests anymore. In this way, the flow time of the large request can be made arbitrarily large.

In conclusion, for the objective of minimizing the average flow time, the SPTF algorithm performs just as bad as the FCFS algorithm in terms of competitive ratio. This fact highlights a shortcoming of competitive analysis when using it to decide which algorithm to use. Namely, two algorithms that cannot be distinguished based on competitive ratio may still have different average case performance. From the stochastic analysis we have performed in this chapter, we can conclude that SPTF is a better choice than FCFS when minimizing the expected or average flow time of requests. FCFS or HOMESICK, however, is a good choice for minimizing the maximum flow time, and much safer than the SPTF algorithm.

4.3 The ALTERNATE algorithm

For the infinite capacity problem settings and the ALTERNATE algorithm, there is no corresponding queueing system which closely resembles the problem. The techniques from queueing theory can, however, be used to provide formulas for the expected flow time of requests for a simplified version of the problem. A preliminary analysis of the ALTERNATE algorithm for the original problem was not promising; many case distinction were needed thereby creating complicated expressions.

4.3.1 Half-line with fixed service time

Suppose all requests have the exact same pick-up or drop-off location, depending on the problem. Then the problem with the ALTERNATE algorithm reduces to the situation in Exercise 73.(ii-iii) in [1]. We abstract away from requests and consider tasks in the same way as before. Let b denote the fixed service time of a task, and λ the arrival rate of the Poisson process.

We calculate the expected waiting time of an arriving task. Since the waiting time is zero when the server is idle, we find

$$\mathbb{E}[W_\tau] = \mathbb{E}[W_\tau \mid \text{server is busy upon arrival}] \cdot \mathbb{P}(\text{server is busy upon arrival}). \tag{4.2}$$

Since the server has infinite capacity, the arriving task only needs to wait until the current task is completed. Given that the task arrives when the server is busy, the arrival instant will be uniformly distributed within the service time. Therefore, since every service time is b, the expected residual service time is $\mathbb{E}[R] = \frac{b}{2}$. Hence,

$$\mathbb{E}[W_\tau \mid \text{server is busy upon arrival}] = \mathbb{E}[R] = \frac{b}{2} \tag{4.3}$$

Next, we calculate the probability that the server is busy upon arrival. Using the PASTA property this is equal to the fraction of time the server is busy. To calculate this, we consider a service time period followed by a possible idle period. Clearly, the service period is b since all tasks have service time b. If tasks arrived during the service period, then there is no idle period since the server continues work after it is done with its current tasks. If no tasks arrived during the service period, then upon return the server becomes idle. This idle period lasts until the next task arrives. Recall that the times between arrivals of tasks in a Poisson process are exponentially distributed with parameter λ, and that exponential distributions are memoryless. Therefore, the expected time until the next request arrives is $\frac{1}{\lambda}$. Hence,

$$\mathbb{P}(\text{server is busy upon arrival}) = \frac{b}{b + \mathbb{P}(\text{no tasks arrive during service period}) \cdot \frac{1}{\lambda}}.$$

Recall that the number of arriving tasks in a time period of length b is distributed according to a Poisson distribution with parameter $b \cdot \lambda$. Therefore,

$$\mathbb{P}(\text{server is busy upon arrival}) = \frac{b}{b + e^{-b\lambda} \cdot \frac{1}{\lambda}}. \tag{4.4}$$

Substituting Equations 4.3 and 4.4 in Equation 4.2 yields

$$\mathbb{E}[W_\tau] = \frac{b}{2} \cdot \frac{b}{b + e^{-b\lambda} \cdot \frac{1}{\lambda}} = \frac{b^2}{2b + e^{-b\lambda} \cdot \frac{2}{\lambda}}.$$

The expected flow times of requests can then be calculated in the same way as for the unit capacity problems (Section 4.2.4).

Chapter 5

Conclusion

In this report, we considered eight different settings of the closed non-preemptive OLDARP on the line. Our main interest was the performance of two simple algorithms we introduced: HOMESICK and ALTERNATE. We used competitive analysis to assess the performance of these two algorithms for the objectives of minimizing the make-span and minimizing the maximum flow time. For each problem setting, we have proven the competitive ratio of either the HOMESICK or ALTERNATE algorithm, and gave lower bounds on the competitive ratio any deterministic online algorithm can achieve. In addition, we briefly considered the offline problems for minimizing the make-span, and gave optimal algorithms for the different settings. Lastly, we investigated the performance of the HOMESICK algorithm when assumptions are made on the probability distribution of the input. Using the close resemblance of the problem to an M/G/1 queue, we gave an expression for the expected flow time of requests served by HOMESICK. We also compared HOMESICK to a similar algorithm SPTF that optimally chooses the request to serve among the waiting requests.

5.1 Discussion

In this section we discuss slight variations of the problem we considered in this report. For each variant and problem setting we indicate whether the results can be transferred.

The open OLDARP

We considered the closed variant of the OLDARP, meaning that after having served all requests, the server is required to return to the origin. Removing this requirement yields the open OLDARP. For the problem settings where all drop-off locations are the origin, lifting the restriction to return to the origin at the end makes no difference since any algorithm finishes at the origin as soon as it drops off its last request there. Furthermore, the restriction does not make a difference for the objective of minimizing maximum flow. However, for the cases where all pick-up locations are the origin and the objective is minimizing make-span, the restriction does make a difference.

For the UCPO-MS problem, lifting the restriction means HOMESICK is no longer optimal, instead it has a competitive ratio of 2. For the lower bound, one can consider the instance $\sigma = ((0, o, 1), (\varepsilon, o, \varepsilon))$ for a small positive ε. The competitiveness proof can also easily be obtained by adapting Theorem 3.4. For the lower bound on any deterministic online algorithm, a lower bound of $\frac{\sqrt{3}+1}{2} \approx 1.3660$ can easily be constructed. The adversary can release request $r_1 = (0, o, 1)$; the online algorithms picks it up at some time T. If $T < \sqrt{3}$, the adversary releases a second request $r_2 = (T + \varepsilon, o, o)$ just as the first one is picked up, otherwise no further requests are released.

For the ICPO-MS problem, the instance in Lemma 3.18 yields a lower bound of 3 for ALTERNATE when considering the open variant. By slightly modifying the competitiveness proof in Lemma 3.20, it can then be shown that ALTERNATE has a competitive ratio of 3. The lower bound of 1.5 in Lemma 3.17 for arbitrary deterministic online algorithms yields a lower bound of $\frac{5}{3} \approx 1.6667$ for the open ICPO-MS setting.

Problem setting	Lower bound	Upper bound
UCPO-MS	1.3660	**2** HOMESICK
ICPO-MS	1.6667	**3** ALTERNATE

Table 5.1: Lower and upper bounds for the open problem variants. The results for the problem settings not listed in this table are the same as for the closed problem.

The preemptive OLDARP

Aside from the restriction discussed above, we also did not allow the server to drop off requests at any point other than their drop-off location. When this restriction is present, the problem is called non-preemptive. Clearly, preemption does not matter when the server has infinite capacity. For unit capacity, it does make a difference since it allows online algorithms to change plans. Preemption provides the server the ability to free up its capacity, and in this way allowing it to pick up a different request. Note that the optimal algorithm does not gain any advantage when allowing preemption; it knows the future, therefore nothing unexpected can happen that requires it to change plans. This implies that the competitive ratio of HOMESICK for the different problem settings is the same for the preemptive case as when disallowing preemption. Hence, the only difference between the non-preemptive and preemptive results may be the lower bounds for the problem settings where the server has unit capacity. It is easily seen that lower bounds for UCDO-MS and UCDO-MF still hold in the preemptive case, but the UCPO-MF lower bound does not. Recall that the lower bound for UCPO-MS was the trivial lower bound of 1, therefore this also stays the same.

The OLDARP with service times

The OLDARP is a simplified model of real-world problems. To make the model more realistic, one can consider adding service times to each request. A service time can be added for picking up a request and for dropping off a request. Then, when picking up a request, the server needs to wait at the pick-up location for the specified service time. Similarly, the server would need to wait for a specified amount of time at the drop-off location before the request is completed. Thinking back to the example given in the introduction, a waiter at a restaurant, this is a natural extension. For example, it takes the waiter some time to take an order, and the service time varies per table.

Clearly, all lower bounds given in this report also hold for the variant with service times: simply set the service times of each request to zero. For the upper bounds, the competitiveness proofs of all but the ICPO-MF and ICDO-MF settings can easily be adapted.

5.2 Future work

For the unit capacity problems, we have seen that HOMESICK performs well. In fact, for three out of the four settings it has the best possible competitive ratio. For the infinite capacity problems, on the other hand, there are gaps between the best lower bound we found and the competitive ratio of ALTERNATE. The gaps are largest for the objective of minimizing the maximum flow time. Therefore, a natural direction for future work is closing these gaps by constructing better lower bounds and analyzing more complex algorithms. One exception is the ICPO-MS problem; we have seen that this problem is equivalent to the OLTSP problem for which a best-possible algorithm is known.

Besides continuing the competitive analysis of this report, modifications of the model, like the variants discussed in the previous section, could also be investigated. Other possible variants are the problem where requests arrive that have either the pick-up or the drop-off location as the origin, or the problem variants in this report but on different metric spaces like the circle or star. The offline problems could be further explored as well. In this report only the offline problems with the objective of minimizing make-span were considered. It would be interesting to construct offline algorithms for the objective of minimizing the maximum flow time, or prove that the problems are NP-hard. One could also consider further studying the problems and algorithms from a stochastic point of view. Lastly, a computational analysis of algorithms could be performed. In this way more complex algorithms could be compared with one another without much difficulty. This would also allow for analysis of more realistic problems where a mathematical analysis would take much effort.

Bibliography

[1] Ivo Adan et al. *Queueing Systems*. Technische Universiteit Eindhoven, 2021.

[2] Norbert Ascheuer, Sven O. Krumke, and Jörg Rambau. "Online Dial-a-Ride Problems: Minimizing the Completion Time". In: *STACS 2000*. Ed. by Horst Reichel and Sophie Tison. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 639–650. ISBN: 978-3-540-46541-6.

[3] Giorgio Ausiello et al. "Algorithms for the on-line travelling salesman". In: *Algorithmica* 29.4 (2001), pp. 560–581.

[4] Alexander Birx. "Competitive analysis of the online dial-a-ride problem". PhD thesis. Darmstadt: Technische Universität, Sept. 2020. URL: http://tuprints.ulb.tu-darmstadt.de/14134/.

[5] Alexander Birx and Yann Disser. "Tight Analysis of the Smartstart Algorithm for Online Dial-a-Ride on the Line". In: *SIAM Journal on Discrete Mathematics* 34.2 (2020), pp. 1409–1443. DOI: 10.1137/19M1268513. eprint: https://doi.org/10.1137/19M1268513. URL: https://doi.org/10.1137/19M1268513.

[6] Antje Bjelde et al. "Tight Bounds for Online TSP on the Line". In: *ACM Trans. Algorithms* 17.1 (Dec. 2020). ISSN: 1549-6325. DOI: 10.1145/3422362. URL: https://doi.org/10.1145/3422362.

[7] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. cambridge university press, 1998.

[8] Jessica Chang et al. "Broadcast scheduling: algorithms and complexity". In: *ACM Transactions on Algorithms (TALG)* 7.4 (2011), pp. 1–14.

[9] Chandra Chekuri, Sungjin Im, and Benjamin Moseley. "Minimizing maximum response time and delay factor in broadcast scheduling". In: *European Symposium on Algorithms*. Springer. 2009, pp. 444–455.

[10] Willem E. De Paepe et al. "Computer-aided complexity classification of dial-a-ride problems". In: *INFORMS Journal on Computing* 16.2 (2004), pp. 120–132. ISSN: 08991499. DOI: 10.1287/ijoc.1030.0052.

[11] Jeff Edmonds and Kirk Pruhs. "A Maiden Analysis of Longest Wait First". In: *ACM Trans. Algorithms* 1.1 (July 2005), pp. 14–32. ISSN: 1549-6325. DOI: 10.1145/1077464.1077467. URL: https://doi.org/10.1145/1077464.1077467.

[12] Esteban Feuerstein and Leen Stougie. "On-line single-server dial-a-ride problems". In: *Theoretical Computer Science* 268.1 (2001), pp. 91–105. ISSN: 03043975. DOI: 10.1016/S0304-3975(00)00261-9.

[13] Dawsen Hwang and Patrick Jaillet. "Online scheduling with multi-state machines". In: *Networks* 71.3 (2018), pp. 209–251.

[14] David G Kendall. "Stochastic processes occurring in the theory of queues and their analysis by the method of the imbedded Markov chain". In: *The Annals of Mathematical Statistics* (1953), pp. 338–354.

[15] Sven O Krumke. “Online optimization: Competitive analysis and beyond”. Habilitationsschrift. Technische Universität Berlin, 2002.

[16] Sven O. Krumke et al. *News from the online traveling repairman*. English. SPOR-Report : reports in statistics, probability and operations research. Technische Universiteit Eindhoven, 2002.

[17] Sven O. Krumke et al. “Non-abusiveness helps: An O(1)-competitive algorithm for minimizing the maximum flow time in the online traveling salesman problem”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 2462.1 (2002), pp. 200–214. ISSN: 16113349.