Car-Sharing on a Star Network: On-Line Scheduling with k Servers

Kelin Luo
School of Management, Xi'an Jiaotong University, Xi'an, China
luokelin@stu.xjtu.edu.cn

Thomas Erlebach
Department of Informatics, University of Leicester, Leicester, United Kingdom
te17@leicester.ac.uk

Yinfeng Xu
School of Management, Xi'an Jiaotong University, Xi'an, China
yfxu@xjtu.edu.cn

Abstract

We study an on-line scheduling problem that is motivated by applications such as car-sharing for trips between an airport and a group of hotels. Users submit ride requests, and the scheduler aims to accept requests of maximum total profit using k servers (cars). Each ride request specifies the pick-up time, the pick-up location, and the drop-off location, where one of the two locations must be the airport. A request must be submitted a fixed amount of time before the pick-up time. The scheduler has to decide whether or not to accept a request immediately at the time when the request is submitted (booking time). In the unit travel time variant, the travel time between the airport and any hotel is a fixed value t. We give a 2-competitive algorithm for the case in which the booking interval (pick-up time minus booking time) is at least t and the number of servers is even. In the arbitrary travel time variant, the travel time between the airport and a hotel may have arbitrary length between t and Lt for some $L \geq 1$. We give an algorithm with competitive ratio $O(\log L)$ if the number of servers is at least $\lceil \log L \rceil$. For both variants, we prove matching lower bounds on the competitive ratio of any deterministic on-line algorithm.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Car-Sharing System, On-Line Scheduling, Competitive Analysis, Star Network

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.51

Funding *Kelin Luo*: This work was partially supported by the China Postdoctoral Science Foundation (Grant No. 2016M592811), and the China Scholarship Council (Grant No. 201706280058).

1 Introduction

In a car-sharing system, customers can hire a car from a company for a period of time. They can pick up a car in one location, drive it to another location, and return it there. Customer requests for car bookings arrive over time, and the decision about each request must be made immediately, without knowledge of future requests. The goal is to maximize the profit obtained from satisfied requests. We refer to this problem as the *car-sharing problem*. Similar problems arise in car rental or taxi dispatching. In this paper, we consider the setting where all car booking requests are for travel between a central location (e.g., an airport, shopping mall or central business district) and one of a group of nearby locations (e.g., hotels, or residential areas), but can be in either direction. The connections between the central location and the nearby locations can therefore be viewed as a star network.

The car-sharing problem bears some resemblance to interval scheduling, but in addition the pick-up and drop-off locations play an important role: The server (car) that serves a request must be at the pick-up location at the start time of the request and will be at the drop-off location at the end time of the request.

A server can serve two consecutive requests, where the pick-up time of the second is no earlier than the drop-off time of the first, only if the drop-off location of the first request is the same as the pick-up location of the second request, or if there is enough time to travel between the two locations otherwise. We allow *empty movements*, i.e., a server can be moved from one location to another while not serving a request. Such empty movements could be implemented by having company staff drive a car from one location to another, or in the future by self-driving cars.

1.1 Related work

On-line car-sharing problem. The car-sharing problem has been studied in several previous papers. In [9], we considered the special case with two locations and a single server, considering both fixed booking times and variable booking times, and presented tight results for the competitive ratio. The optimal competitive ratio was shown to be 2 for fixed booking times and 3 for variable booking times. In [10], we dealt with the car-sharing problem with two locations and two servers, considering only the case of fixed booking times, and showed that the optimal competitive ratio is 2. In [11], we studied the car-sharing problem with two locations and k servers, where k can be arbitrarily large. We considered both fixed booking times and variable booking times. The results showed that, surprisingly, 3 servers (in one case) and 5 servers (in another case) already allow us to get the best competitive ratio, and no improvement is possible with more servers. In contrast to the previous work on car-sharing that has only considered two locations, in this paper we study the car-sharing problem for fixed booking time in the setting with k servers and $m + 1$ locations that are arranged in a star network.

Off-line car-sharing problem. Böhmová et al. [4] showed that if all customer requests for car bookings are known in advance, the problem of maximizing the number of accepted requests is solvable in polynomial time. Furthermore, they considered the problem variant with two locations where each customer requests two rides (in opposite directions) and the scheduler must accept either both or neither of the two. They proved that this variant is NP-hard and APX-hard. In contrast to their work, we consider the on-line version of the problem with $m + 1$ locations.

On-line dial-a-ride problem. A closely related problem is the on-line dial-a-ride problem (OLDARP). Versions of OLDARP with the objective of serving all requests while minimizing the makespan [1, 3] or the maximum flow time [7] have been widely studied in the literature. Versions of OLDARP where not all requests need to be served include the setting where each request must be served before its deadline or rejected [12], and the setting with a given common time limit where the goal is to maximize the revenue from requests served before the time limit [6]. In OLDARP, transportation requests between locations in a metric space arrive over time, but typically it is assumed that requests want to be served "as soon as possible" rather than at a specific time as in our problem.

On-line interval scheduling problem. The on-line car-sharing problem can be interpreted as a variation of the on-line interval scheduling problem. If all the pick-up and drop-off locations are the same, the car-sharing problem becomes an on-line interval scheduling problem. Lipton and Tomkins [8] defined the basic on-line interval scheduling problem: intervals with a given length are presented in the order of their start time and the scheduler aims to accept intervals of maximum total length. The scheduler has to decide whether to accept each interval before the next interval is presented and ensure that no pair of accepted intervals overlap. They showed that no (randomized) algorithm can achieve competitive ratio $O(\log \Delta)$ (where Δ denotes the ratio between the longest and the shortest interval, and Δ is unknown to the algorithm), and gave an $O((\log \Delta)^{1+\varepsilon})$-competitive randomized algorithm.

1.2 Problem description and preliminaries

We consider a setting with $m+1$ locations in a star network and denote the central location by 0 and the other locations by i for $i \in \{1, 2, \ldots, m\}$. There are k servers, denoted by $S = \{s_1, s_2, \ldots, s_k\}$, that are initially located at 0. We assume that $m \geq 2$ since, if $m = 1$, the problem turns into the car-sharing problem between two locations that has been studied before [9, 10, 11]. The length of the edge between 0 and i, for $1 \leq i \leq m$, is denoted by $d(0, i) = d(i, 0)$. The travel time from 0 to i, $i \in \{1, 2, \ldots, m\}$, is $d(0, i) \cdot t$, where t is a fixed positive constant, and is the same as the travel time from i to 0, $d(i, 0) \cdot t$. In the variant with unit travel times, all edges have length 1 and the travel time between 0 and i is t for all $i \in \{1, 2, \ldots, m\}$ (see Fig. 1 for an example). In the variant with arbitrary travel times, we only assume that the edge lengths satisfy $1 \leq d(0, i) \leq L$ for all $1 \leq i \leq m$ (Fig. 2 shows an example).

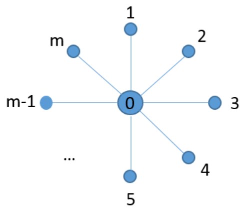

Figure 1 Unit travel times.

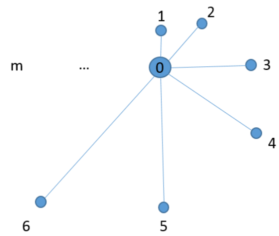

Figure 2 Arbitrary travel times.

Let R denote a sequence of requests that are released over time. The requests with the same release time are released one by one in arbitrary order. The i-th request is denoted by $r_i = (\tilde{t}_{r_i}, t_{r_i}, p_{r_i}, \dot{p}_{r_i})$ and specifies the *booking time* or *release time* $\tilde{t}_{r_i}$, the *start time* or *pick-up time* t_{r_i}, the pick-up location $p_{r_i} \in \{0, 1, \ldots, m\}$, and the drop-off location $\dot{p}_{r_i} \in \{0, 1, \ldots, m\}$. We also say that request r_i *drops off* at $\dot{p}_{r_i}$. We require $\dot{p}_{r_i} \neq p_{r_i}$ and $\min\{p_{r_i}, \dot{p}_{r_i}\} = 0$, i.e., for all requests $r_i \in R$, either the pick-up location p_{r_i} or the drop-off location $\dot{p}_{r_i}$ is 0. We assume that the *booking interval* $t_{r_i} - \tilde{t}_{r_i}$ is equal to a fixed value a for all requests $r_i \in R$. For the variant with unit travel times (resp. the variant with arbitrary travel times), if r_i is accepted, a server must pick up the customer at p_{r_i} at time t_{r_i} and drop off the customer at $\dot{p}_{r_i}$ at time $\dot{t}_{r_i} = t_{r_i} + t$ (resp. at time $\dot{t}_{r_i} = t_{r_i} + d(p_{r_i}, \dot{p}_{r_i}) \cdot t$), the *end time* (or *drop-off time*) of the request.

Each server can only serve one request at a time. If two requests are such that they cannot both be served by one server, we say that the requests are *in conflict*. For the variant with unit travel times (resp. arbitrary travel times), serving a request r_i yields profit $P_{r_i} = r$ (resp. $P_{r_i} = d(p_{r_i}, \dot{p}_{r_i}) \cdot r$). An empty movement has no cost. We denote the requests accepted by an algorithm by R'. The i-th request in R', in order of request start times, is denoted by r'_i. We denote the profit of serving the requests in R' by $P_{R'} = \sum_{i=1}^{|R'|} P_{r'_i}$. The goal of the car-sharing problem is to accept a set of requests R' that maximizes the profit $P_{R'}$.

We use *kSmL-U* to refer to the problem with k servers and $m+1$ locations with unit travel times. The problem variant with arbitrary travel times is called the *kSmL-A* problem. For the kSmL-A problem, we assume that $a \geq 2Lt$, where Lt is the travel time of the longest edge of the star. This ensures that any free server always has enough time to travel from its current location to the pick-up location of a newly accepted request. We do not require that the algorithm assigns an accepted request to a server immediately, provided that it ensures that one of the k servers will serve the request.

We forbid "unprompted" moves, i.e., the algorithm is allowed to make an empty move to another location only if it does so in order to serve a request that was accepted before the current time and whose pick-up location is the other location. If the length of the booking interval (recall that the booking interval is the interval between booking time and start time) is greater than the maximum travel time of any two locations in the two problems defined above, we observe that there is never a need for a server to make an unprompted movement. Therefore, if $a \geq 2t$ for kSmL-U or $a \geq 2Lt$ for kSmL-A, whether or not we forbid unprompted movements affects neither lower bounds nor the algorithm performance.

The performance of an algorithm for kSmL-U or kSmL-A is measured using competitive analysis [5]. For any request sequence R, let P_{R^A} denote the objective value produced by an on-line algorithm A, and P_{R^*} that obtained by an optimal scheduler OPT that has full information about the request sequence in advance. Like for the algorithm, we also require that OPT does not make unprompted moves, i.e., OPT is allowed to make an empty move starting at time t_0 with some server s_j from location p to location q only if there is an accepted request r_i assigned to s_j with $\tilde{t}_{r_i} \leq t_0$, $p_{r_i} = q$ and $t_{r_i} \geq t_0 + d(p,q) \cdot t$. The competitive ratio of A is defined as $\rho_A = \sup_R \frac{P_{R^*}}{P_{R^A}}$. We say that A is ρ-competitive if $P_{R^*} \leq \rho \cdot P_{R^A}$ for all request sequences R. Let ON be the set of all on-line algorithms for a problem. We only consider deterministic algorithms in this paper. A value β is a *lower bound* on the best possible competitive ratio if $\rho_A \geq \beta$ for all A in ON.

The asymptotic competitive ratio (asymptotic performance ratio) of A is defined to be $\rho'_A = \lim_{n\to\infty} \sup_R \{\frac{P_{R^*}}{P_{R^A}} | P_{R^*} = n\}$. A value β' is a *lower bound* on the best possible asymptotic competitive ratio if $\rho'_A \geq \beta'$ for all A in ON. We write $\mathbb{N} = \{0, 1, 2, \ldots\}$.

1.3 Paper outline

Table 1 Lower and upper bounds on the competitive ratio for the kSmL problem.

Problem	Booking constraint	Lower bound	Upper bound
kSmL-U	$a < t$	$\frac{k}{\lfloor k/m \rfloor}$	$\frac{k}{\lfloor k/m \rfloor}$
kSmL-U	$t \leq a < 2t$	2	2 (for even k)
kSmL-U	$a \geq 2t$	$2 - \frac{1}{2m-1}$	2 (for even k)
kSmL-A	$a \geq 2Lt$	$\Omega(\log L)$	$O(\log L)$ (for $k \geq \log L$)

In Section 2, we present lower bounds on the competitive ratio for the kSmL-U problem. In Section 3, we propose two greedy algorithms, the m-partition greedy algorithm and the bi-partition greedy algorithm, that achieve the best possible competitive ratio for kSmL-U for different ranges of a. In Section 4, we study kSmL-A and give an algorithm with competitive ratio $O(\log L)$ and show that no deterministic on-line algorithm can achieve competitive ratio smaller than $\Omega(\log L)$. Section 5 concludes the paper. An overview of our results is shown in Table 1. All our lower bounds hold even in the seemingly simpler case where the start time of every request is a multiple of t.

2 Lower bounds for kSmL-U

In this section, we present lower bounds for the kSmL-U problem. We use ALG to denote any on-line algorithm and OPT to denote an optimal scheduler. The set of requests accepted by ALG is denoted by R', and the set of requests accepted by OPT by R^*.

▶ **Theorem 1.** *For $a < t$, no deterministic on-line algorithm for kSmL-U can achieve asymptotic competitive ratio smaller than $\frac{k}{\lfloor k/m \rfloor}$.*

Proof. Consider a sequence of requests that consists of γ phases where phase i, for $1 \le i \le \gamma$, consists of l_i groups of requests, with each group consisting of k identical requests. Let $\sigma(u, v)$ be the number of request groups that the adversary has released by the time when the requests in phase u, group v have just been released, i.e., $\sigma(u, v) = \sum_{i=1}^{u-1} l_i + v$.

The adversary releases requests based on the release rule for kSmL-U shown in Algorithm 1.

Algorithm 1 Release Rule for kSmL-U with $a < t$.

Initialization: The adversary presents the requests in phase 1 group 1: k copies of the request $(\nu \cdot t - a, \nu \cdot t, 0, 1)$ for some ν such that $\nu \in \mathbb{N}$ and $\nu \cdot t - a \ge 0$.
$i = 1$, $j = 1$. Let l be a large, positive, odd integer.
While $i \le m$ do
  if $j < l$ then
    if $|R'_{i,j}| > \lfloor k/m \rfloor$ then
      $l_i = j$, $i = i + 1$, $j = 1$ and the adversary releases the requests in $R_{i,j}$;
    else if $|R'_{i,j}| \le \lfloor k/m \rfloor$ then
      $j = j + 2$ and the adversary releases the requests in $R_{i,j-1}$ and $R_{i,j}$;
  if $j \ge l$ then
      break.
Output: $\gamma = i$ and $l_i = j$.
(1) Let $R_{i,j}$ denote the set of requests in phase i group j. If $i > 1$ and $j = 1$, $R_{i,j}$ consists of k copies of the request $(\tilde{t}_{r_{\sigma(i-1,l_{i-1})k}} + t, t_{r_{\sigma(i-1,l_{i-1})k}} + t, 0, i)$; if $i > 0$, $j > 1$ and $j = 2e$ where $e \in \mathbb{N}$, $R_{i,j}$ consists of k copies of the request $(\tilde{t}_{r_{\sigma(i,j-1)k}} + t, t_{r_{\sigma(i,j-1)k}} + t, i, 0)$; if $i > 0$, $j > 1$ and $j = 2e + 1$ where $e \in \mathbb{N}$, $R_{i,j}$ consists of k copies of the request $(\tilde{t}_{r_{\sigma(i,j-1)k}} + t, t_{r_{\sigma(i,j-1)k}} + t, 0, i)$.
(2) Let $R'_{i,j}$ denote the set of requests accepted by ALG in phase i group j.

We make four observations:

(a) For each $i < \gamma$, $l_i < l$. This holds because, as soon as j reaches value l, the While-loop is exited and γ is set to i.

(b) For each $i \le \gamma$, ALG accepts no more than $\lfloor k/m \rfloor (l_i - 1)$ requests in total among the requests in phase i excluding the requests in phase i group l_i. This can be seen as follows: The algorithm accepts at most $\lfloor k/m \rfloor$ requests from phase i group j for any odd j, $j < l_i$. Moreover, the total number of requests from phase i group j for all even j together cannot be larger than the total number of requests from phase i group j for all odd j, $j < l_i$.

(c) ALG accepts no more than k requests in total among the requests in phase 1 group l_1, phase 2 group l_2, …, phase γ group l_γ. This holds because any server accepting a request in phase i group l_i will remain at i and not be able to serve any further requests.

(d) ALG accepts more than $(\lfloor k/m \rfloor + 1)(\gamma - 1)$ requests in total among the requests in phase 1 group l_1, phase 2 group l_2, …, phase $\gamma - 1$ group $l_{\gamma-1}$. This holds because the algorithm accepts strictly more than $\lfloor k/m \rfloor$ requests in each of these $\gamma - 1$ groups.

According to (d), more than $(\lfloor k/m \rfloor + 1) \cdot (\gamma - 1)$ servers are not in location 0 or γ when the requests in phase γ are released. These servers cannot accept requests in phase γ because the release time of a request is too late for such a server to be able to serve it with empty movement. If $\gamma = m$, $k - (\lfloor k/m \rfloor + 1)(\gamma - 1) \leq \lfloor k/m \rfloor$, and hence the adversary stops to release requests in phase m only after group l. Therefore, $l_\gamma = l$ no matter whether $\gamma < m$ or $\gamma = m$.

By (b) and (c), we have that ALG accepts no more than $(\lfloor k/m \rfloor \sum_{i=1}^{\gamma}(l_i - 1)) + k$ requests.

OPT accepts all the requests except the requests in phase 1 group l_1, phase 2 group l_2, …, phase $\gamma - 1$ group $l_{\gamma-1}$. We have $P_{R^*} = (kr \sum_{i=1}^{\gamma-1}(l_i - 1)) + krl$. Since $P_{R'} \leq kr + \lfloor k/m \rfloor \sum_{i=1}^{\gamma}(l_i - 1)r$ and $\lim_{l\to\infty} \inf_{l_i \leq l} \frac{kl + k\sum_{i=1}^{\gamma-1}(l_i-1)}{k + \lfloor k/m \rfloor \sum_{i=1}^{\gamma}(l_i-1)} = \frac{k}{\lfloor k/m \rfloor}$, where the infimum is taken over all possible values of l_i for $1 \leq i \leq \gamma - 1$, we get $\lim_{P_{R^*}\to\infty} P_{R^*}/P_{R'} \geq \frac{k}{\lfloor k/m \rfloor}$. ◀

▶ **Theorem 2.** *For $t \leq a < 2t$, no deterministic on-line algorithm for kSmL-U can achieve asymptotic competitive ratio smaller than* 2.

Proof. Let l be a large, positive integer. Consider a sequence of requests that consists of 2 phases where phase i, for $i = 1, 2$, consists of l_i groups of requests, with each group consisting of k identical requests. Let $R_{i,j}$ denote the set of requests in phase i group j. Initially, the adversary releases $R_{i,j}$ with $i = 1$ and $j = 1$, consisting of k copies of the request $r_1 = (\nu \cdot t - a, \nu \cdot t, 0, 1)$ for some ν such that $\nu \in \mathbb{N}$ and $\nu \cdot t - a \geq 0$. Let $R'_{i,j}$ denote the set of requests accepted by ALG in phase i group j. The adversary releases further requests based on the following rules: If $|R'_{1,j}| \leq \frac{k}{2}$ and $j < l$, let $j = j + 1$ and release $R'_{1,j}$ consisting of k copies of the request $(\tilde{t}_{r_1} + 2(j-1)t, t_{r_1} + 2(j-1)t, 0, 1)$; otherwise, set $l_1 = j$ and stop to release requests in phase 1. Note that either $l_1 = l$, or $l_1 < l$ and $|R'_{1,l_1}| > \frac{k}{2}$. We distinguish two cases.

Case 1: $l_1 = l$. Observe that $|R'_{1,j}| \leq \frac{k}{2}$ for all $1 \leq j < l_1$. In this case, OPT accepts all requests in $R_{1,j}$ for all $1 \leq j \leq l$. We have $P_{R^*} = l \cdot kr$ and $P_{R'} = \sum_{j=1}^{l_1} |R'_{1,j}|r \leq \frac{k}{2} \cdot (l-1)r + kr$, and hence $\lim_{P_{R^*}\to\infty} P_{R^*}/P_{R'} \geq 2$.

Case 2: $l_1 < l$ **and** $|R'_{1,l_1}| > \frac{k}{2}$. Observe that $|R'_{1,j}| \leq \frac{k}{2}$ for all $1 \leq j < l_1$. The adversary then releases $R_{2,j}$ for all $1 \leq j \leq l^2$, where each $R_{2,j}$ consists of k copies of the request $(\tilde{t}_{r_1} + 2(l_1 - 1 + j)t, t_{r_1} + 2(l_1 - 1 + j)t, 2, 0)$. Observe that $|R'_{1,l_1}|$ servers of the algorithm are in location 1 when the requests in $R_{2,j}$ for all $1 \leq j \leq l^2$ are released. The release time of a request with pick-up location 2 is too late for a server in location 1 to be able to serve it with empty movement because the travel time between location 1 and the pick-up location 2 is $2t$, which is greater than the booking interval a. From this it follows that the $|R'_{1,l_1}|$ servers of ALG cannot accept any requests in phase 2. OPT accepts all requests in phase 2. We have $P_{R^*} \geq l^2 \cdot kr$. Since $P_{R'} \leq \sum_{j=1}^{l_1-1} |R'_{1,j}|r + |R'_{1,l_1}|r + (k - |R'_{1,l_1}|)l^2 r \leq \frac{k}{2} \cdot (l^2 + l_1 - 1)r + |R'_{1,l_1}|r$, we have $\lim_{P_{R^*}\to\infty} P_{R^*}/P_{R'} \geq 2$. ◀

▶ **Theorem 3.** *For $a \geq 2t$, no deterministic on-line algorithm for kSmL-U can achieve competitive ratio smaller than $2 - \frac{1}{2m-1}$. Furthermore, if $k < 2(m-1)$, no deterministic on-line algorithm for kSmL-U can achieve competitive ratio smaller than* 2.

Proof. The adversary releases a number of request sequences. We use k_i $(0 \leq k_i \leq k)$ to denote the number of requests that ALG accepts from the i^{th} request sequence.

Initially, the adversary releases the 1^{st} request sequence, consisting of k copies of the request $(\nu \cdot t - a, \nu \cdot t, 0, 1)$ for some ν such that $\nu \in \mathbb{N}$ and $\nu \cdot t - a \geq 0$. There are two options that the adversary can adopt:

Option 1. The adversary does not release any more requests. In this case, OPT accepts all requests in the 1^{st} request sequence. We have $P_{R^*} = k \cdot r$ and $P_{R'} = k_1 \cdot r$, and hence $P_{R^*}/P_{R'} \geq \frac{k}{k_1}$.

Option 2. The adversary releases the 2^{nd} request sequence, consisting of k copies of the request $(\tilde{t}_{r_1}, t_{r_1}, 1, 0)$, and then releases $m-1$ further request sequences (from the 3^{rd} request sequence to the $(m+1)^{th}$ request sequence), where the i^{th} $(3 \leq i \leq m+1)$ request sequence consists of k copies of the request $(\tilde{t}_{r_1}+t, t_{r_1}+t, 0, i-1)$. Then, the adversary releases the $(m+2)^{th}$ request sequence, consisting of k copies of the request $(\tilde{t}_{r_1}+2t, t_{r_1}+2t, \varrho-1, 0)$ where $\varrho = \arg\min\{k_i, 3 \leq i \leq m+1\}$. Since the requests in the 1^{st} request sequence are in conflict with the requests in the i^{th} $(2 \leq i \leq m+2)$ request sequence, k_1 servers of ALG accept at most one request each. OPT accepts all requests in the 2^{nd} request sequence, the ϱ^{th} request sequence and the $(m+2)^{th}$ request sequence, i.e., $P_{R^*} = 3kr$.

Observe that $k_\varrho \leq \frac{k-k_1}{m-1}$. Furthermore, the requests in the $(m+2)^{th}$ request sequence are in conflict with the requests in the i^{th} $(3 \leq i \leq m+1$ and $i \neq \varrho)$ request sequence. Therefore, at most k_ϱ servers accept requests both in the $(m+2)^{th}$ request sequence and the i^{th} $(3 \leq i \leq m+1)$ request sequence. From this it follows that k_ϱ servers accept at most three requests each (in the 2^{nd}, the ϱ^{th}, and the $(m+2)^{th}$ request sequence) and the remaining servers of ALG, i.e., $k-k_1-k_\varrho$ servers, each accept at most two requests (in the 2^{nd} and the i^{th} request sequence where $3 \leq i \leq m+1$ and $i \neq \varrho$). Thus, we have $P_{R'} \leq k_1 r + 2(k-k_1)r + k_\varrho \cdot r$ and hence $P_{R'} \leq k_1 r + 2(k-k_1)r + \frac{k-k_1}{m-1} \cdot r$. Since $P_{R^*} = 3kr$, $P_{R^*}/P_{R'} \geq \frac{3k}{2k-k_1+(k-k_1)/(m-1)}$.

If we choose the option (from Option 1 and Option 2) that maximizes $\frac{P_{R^*}}{P_{R'}}$, we have $\frac{P_{R^*}}{P_{R'}} \geq \max\{\frac{k}{k_1}, \frac{3k}{2k-k_1+(k-k_1)/(m-1)}\}$. As k_1 increases, $\frac{k}{k_1}$ decreases and $\frac{3k}{2k-k_1+(k-k_1)/(m-1)}$ increases. Since $\frac{3k}{2k-k_1+(k-k_1)/(m-1)} = \frac{k}{k_1} = 2 - \frac{1}{2m-1}$ when $k_1 = \frac{2m-1}{4m-3} \cdot k$, we have $P_{R^*}/P_{R'} \geq 2 - \frac{1}{2m-1}$. The claimed lower bound of $2 - \frac{1}{2m-1}$ follows.

Furthermore, if $k < 2(m-1)$, we can argue as follows. If $k_1 \leq \frac{k}{2}$, we choose Option 1 and get $P_{R^*}/P_{R'} = \frac{k}{k_1} \geq 2$. If $k_1 > \frac{k}{2}$, then $k_\varrho \leq \frac{k-k_1}{m-1} < 1$ and hence $k_\varrho = 0$. We choose Option 2 and have $P_{R'} \leq k_1 r + 2(k-k_1)r$ and thus $P_{R^*}/P_{R'} \geq \frac{3k}{2k-k_1} > 2$. The claimed lower bound of 2 follows. ◀

3 Upper bounds for kSmL-U

In this section, we prove the upper bounds for the kSmL-U problem. Denote the requests accepted by OPT by $R^* = \{r_1^*, r_2^*, \ldots, r_{|R^*|}^*\}$ and the requests accepted by an algorithm by $R' = \{r_1', r_2', \ldots, r_{|R'|}'\}$, indexed in the order in which they are released (and hence also in order of non-decreasing pick-up times).

Let $R^*(e,p,d)$ denote the set of requests in R^* which start at time e at location p and drop off at location d. Observe that $\forall e,p,d$, $|R^*(e,p,d)| \leq k$. Let $R^*(p,d)$ denote the set of requests in R^* which start at location p and drop off at location d. Furthermore, let $R^*(e,0,X)$ denote the set of requests in R^* which start at time e at location 0, i.e., $R^*(e,0,X) = \bigcup_{d=1}^{m} R^*(e,0,d)$, and let $R^*(e,X,0)$ denote the set of requests in R^* which start at time e and drop off at location 0, i.e., $R^*(e,X,0) = \bigcup_{d=1}^{m} R^*(e,d,0)$. Similarly, define $R^*(0,X) = \bigcup_{d=1}^{m} R^*(0,d)$ and $R^*(X,0) = \bigcup_{d=1}^{m} R^*(d,0)$. The subsets $R'(e,p,d)$, $R'(p,d)$, $R'(e,0,X)$, $R'(e,X,0)$ $R'(0,X)$ and $R'(X,0)$ of R' are defined analogously.

3.1 Upper bound for $0 \leq a < t$

We propose an m-Partition Greedy Algorithm (m-PGA) for the kSmL-U problem when $0 \leq a < t$, shown in Algorithm 2. The k servers are divided into m groups $S_1, S_2, \ldots, S_m$. From group S_1 to group S_{m-1}, each group has $\lfloor k/m \rfloor$ servers; group S_m has $k - \lfloor k/m \rfloor(m-1) \geq \lceil k/m \rceil$ servers. The servers in group S_i, $1 \leq i \leq m$, only serve requests whose pick-up or drop-off location is i.

Algorithm 2 m-Partition Greedy Algorithm (m-PGA).

Input: k servers, requests arrive over time.

Step: When request r_i arrives, if it is acceptable to a server in group $S_{g(r_i)}$, where $g(r_i) = \max\{p_{r_i}, \dot{p}_{r_i}\}$, assign it to that server; otherwise, reject it.

(1) $r^n_{i,j}$ denotes the newest request which is assigned to s_j before r_i is released. Set $\dot{p}_{r^n_{i,j}} = 0$ and $\dot{t}_{r^n_{i,j}} = 0$ if server s_j has not accepted any request before r_i is released.
(2) r_i is acceptable to a server s_j ($s_j \in S$) if and only if $p_{r_i} = \dot{p}_{r^n_{i,j}}$ and $t_{r_i} \geq \dot{t}_{r^n_{i,j}}$.

We refer to the servers of m-PGA as $s'_1, s'_2, \ldots, s'_k$, and the servers of OPT as $s^*_1, s^*_2, \ldots, s^*_k$. For an arbitrary request sequence $R = (r_1, r_2, \ldots, r_n)$, note that we have $t_{r_i} \leq t_{r_{i+1}}$ for $1 \leq i < n$ because $t_{r_i} - \tilde{t}_{r_i} = a$ is fixed.

▶ **Observation 4.** m-PGA only accepts requests without empty movement because the release time of a request is too late for a server to be able to serve it with empty movement in kSmL-U with $a < t$. Therefore, each m-PGA server accepts requests with alternating pick-up location, starting with a request with pick-up location 0.

▶ **Observation 5.** OPT only accepts requests without empty movement because the release time of a request is too late for a server to be able to serve it with empty movement in kSmL-U with $a < t$. Therefore, each OPT server accepts requests starting with a request with pick-up location 0, and any two consecutive requests accepted by a server of OPT have the following property: the drop-off location of the first request is the pick-up location of the second request.

For simplification of the analysis, we suppose that for each d, $1 \leq d \leq m$, OPT has k separate servers for serving requests for travel between location 0 and location d, and those k servers only serve such requests. This simplification does not decrease the profit gained by OPT. In this way we can analyse the requests for travel between location 0 and location d for different d independently. In the following, we focus on an arbitrary value of d and assume that S_d contains $\lfloor k/m \rfloor$ servers.

The analysis of the algorithm is divided into two parts. First, we reassign the requests in $R'(0,d)$, $R'(d,0)$, $R^*(0,d)$ and $R^*(d,0)$ by repeated application of two reassignment rules, and then we show that the profit accrued by the algorithm is within a certain factor of the profit accrued by OPT.

Suppose OPT accepts k^*_0 requests that start at location 0 and drop off at location d, i.e., $R^*(0,d) = \{r^{*0}_1, r^{*0}_2, \ldots, r^{*0}_{k^*_0}\}$, and OPT accepts k^*_1 requests that start at location d and drop off at location 0, i.e., $R^*(d,0) = \{r^{*d}_1, r^{*d}_2, \ldots, r^{*d}_{k^*_1}\}$. The subsets $R'(0,d) = \{r'^0_1, r'^0_2, \ldots, r'^0_{k'_0}\}$ and $R'(d,0) = \{r'^d_1, r'^d_2, \ldots, r'^d_{k'_1}\}$ of R' are defined analogously.

Reassignment Rule 1. Consider the case that requests r^{*0}_o and r^{*d}_o are both assigned to the same server for $o < i$ and r^{*0}_i and r^{*d}_i are assigned to different servers. Suppose r^{*0}_i is assigned to s^*_j and r^{*d}_i is assigned to s^*_l where $l \neq j$. We reassign r^{*d}_i to server s^*_j, reassign all requests in $R^* \setminus (\{r_1^{*0}, r_2^{*0}, \ldots, r_i^{*0}\} \bigcup \{r_1^{*d}, r_2^{*d}, \ldots, r_i^{*d}\})$ that are assigned to s_j^* (denote the set of these requests by $\Re_j$) to server s_l^*, and reassign all requests in $R^* \setminus (\{r_1^{*0}, r_2^{*0}, \ldots, r_i^{*0}\} \bigcup \{r_1^{*d}, r_2^{*d}, \ldots, r_i^{*d}\})$ that are assigned to s_l^* (denote them by $\Re_l$) to server s_j^*.

As each server accepts requests with alternating pick-up location, starting with a request with pick-up location 0, we have $t_{r_i'^0} \le t_{r_i'^d}$ (for all $i \le k_1'$) and $t_{r_i^{*0}} \le t_{r_i^{*d}}$ (for all $i \le k_1^*$). Thus, for $i \le k_1^*$, r_i^{*0} and r_i^{*d} are not in conflict, and hence reassigning r_i^{*d} to server s_j^* is valid. Furthermore, any two consecutive requests in $\Re_l$ are not in conflict, so reassigning all requests of $\Re_l$ to server s_j^* is valid. Observe that server s_l^* is at location d at time $t_{r_i^{*d}}$. Because the first request in $\Re_j$, say, request x, has pick-up location d and starts after $t_{r_i^{*d}}$, reassigning request x to server s_l^* is valid. As any two consecutive requests in $\Re_j$ are not in conflict, reassigning all requests of $\Re_j$ to server s_l^* is valid. From this it follows that $R^*(0,d)$ and $R^*(d,0)$ are still a valid solution with the same profit after the reassignment.

Reassignment Rule 2. Consider the case that requests r_o^{*0} and r_o^{*d} are both assigned to the server $s_{o \bmod k}$ for $o < i$ and r_i^{*0} and r_i^{*d} are not assigned to the server $s_{i \bmod k}$ (the case where r_i^{*d} does not exist can be handled similarly). Suppose r_i^{*0} and r_i^{*d} are assigned to s_j^*, $j \neq i \bmod k$. We reassign r_i^{*0} and r_i^{*1} to server $s_{i \bmod k}$, reassign all requests in $R^* \setminus (\{r_1^{*0}, r_2^{*0}, \ldots, r_i^{*0}\} \bigcup \{r_1^{*d}, r_2^{*d}, \ldots, r_i^{*d}\})$ that are assigned to s_j^* (denote the set of these requests by $\Re_j$) to server $s_{i \bmod k}^*$, and reassign all requests in $R^* \setminus (\{r_1^{*0}, r_2^{*0}, \ldots, r_i^{*0}\} \bigcup \{r_1^{*d}, r_2^{*d}, \ldots, r_i^{*d}\})$ that are assigned to $s_{i \bmod k}^*$ (denote them by $\Re_{i \bmod k}$) to server s_j^*.

Since the requests r_o^{*0} and r_o^{*d} are both assigned to the server $s_{o \bmod k}$ for $o < i$, the last request r_l accepted by $s_{i \bmod k}^*$ whose pick-up time is earlier than $t_{r_i^{*0}}$ ends not later than the last request accepted by s_j^* whose pick-up time is earlier than $t_{r_i^{*0}}$ if $j \neq i \bmod k$. Reassigning all requests of $\Re_j$ to server $s_{i \bmod k}^*$ is valid. Because the first request in $\Re_{i \bmod k}$ accepted by $s_{i \bmod k}^*$ starts later than $t_{r_i^{*0}}$ and starts at location $p_{r_i^{*0}}$, and any two consecutive requests in $\Re_{i \bmod k}$ are not in conflict, reassigning all requests of $\Re_{i \bmod k}$ to server s_j^* is valid. From this it follows that $R^*(0,d)$ and $R^*(d,0)$ are still a valid solution with the same profit after the reassignment.

Similarly, we reassign the requests in $R'(0,d)$ and $R'(d,0)$ based on the above process until both requests $r_i'^0$ and $r_i'^d$ are assigned to $s'_{i \bmod \lfloor k/m \rfloor}$ for $i \le k_1'$. Note that this reassignment does not affect the validity of $R'(0,d)$ and $R'(d,0)$, and $P_{R'(0,d)}$ and $P_{R'(d,0)}$ do not change.

▶ **Theorem 6.** *m-PGA is $\frac{k}{\lfloor k/m \rfloor}$-competitive for kSmL-U if $0 \le a < t$.*

Proof. We bound the competitive ratio of m-PGA by analyzing the requests between 0 and d for each d independently. As $\bigcup_{d=1}^m R'(0,d) \cup \bigcup_{d=1}^m R'(d,0) = R'$ and $\bigcup_{d=1}^m R^*(0,d) \cup \bigcup_{d=1}^m R^*(d,0) = R^*$, it is clear that, for any $\alpha \ge 1$, $P_{R^*}/P_{R'} \le \alpha$ follows if we can show that $P_{R^*(0,d)}/P_{R'(0,d)} \le \alpha$ and $P_{R^*(d,0)}/P_{R'(d,0)} \le \alpha$ for all d, $1 \le d \le m$. Consider an arbitrary value of d from now on.

The proof uses similar ideas to the one used for the case $a < t$ of car-sharing between two locations (Theorem 2 in [9] and Theorem 2 in [10]), but there one can easily show that R^* can be transformed into R' without reducing its profit, whereas we encounter the additional difficulty that m-PGA has only $\lfloor k/m \rfloor$ servers while *OPT* has k servers.

Let $\bar{R}^*(0,d)$ (resp. $\bar{R}^*(d,0)$) be the subset of $R^*(0,d)$ (resp. $R^*(d,0)$) that contains the requests from the $(ik+1)^{th}$ to the $(ik + \lfloor k/m \rfloor)^{th}$, for all $0 \le i \le \frac{|R^*(0,d)|}{k}$ (resp. $0 \le i \le \frac{|R^*(d,0)|}{k}$). In other words, $\bar{R}^*(0,d)$ and $\bar{R}^*(d,0)$ contain the requests that are accepted by the first $\lfloor k/m \rfloor$ servers of *OPT*. Suppose the first $\lfloor k/m \rfloor$ servers of *OPT* accept k_0^* requests that start at location 0 and drop off at location d, i.e., $\bar{R}^*(0,d) = \{r_1^{*0}, r_2^{*0}, \ldots, r_{k_0^*}^{*0}\}$, and the first $\lfloor k/m \rfloor$ servers of OPT accept k_1^* requests that start at location d and drop off at location 0, i.e., $\bar{R}^*(d,0) = \{r_1^{*d}, r_2^{*d}, \ldots, r_{k_1^*}^{*d}\}$. Suppose m-PGA accepts k_0 requests that start at location 0 and drop off at location d, i.e., $R'(0,d) = \{r_1'^0, r_2'^0, \ldots, r_{k_0}'^0\}$, and m-PGA accepts k_1 requests that start at location d and drop off at location 0, i.e., $R'(d,0) = \{r_1'^d, r_2'^d, \ldots, r_{k_1}'^d\}$. We claim that $\bar{R}^*(0,d)$ (resp. $\bar{R}^*(d,0)$) can be transformed into $R'(0,d)$ (resp. $R'(d,0)$) without reducing its profit, thus showing that $P_{\bar{R}^*(0,d)} \leq P_{R'(0,d)}$ (resp. $P_{\bar{R}^*(d,0)} \leq P_{R'(d,0)}$), and hence $P_{R^*(0,d)} \leq \frac{k}{\lfloor k/m \rfloor} P_{R'(0,d)}$ (resp. $P_{R^*(d,0)} \leq \frac{k}{\lfloor k/m \rfloor} P_{R'(d,0)}$).

By Observation 5, when $\bar{R}^*(d,0)$ consists of w requests, $\bar{R}^*(0,d)$ consists of at least w requests and of at most $w + \lfloor k/m \rfloor$ requests, i.e., $k_1^* \leq k_0^* \leq k_1^* + \lfloor k/m \rfloor$.

As m-PGA accepts the request r_γ which is the first acceptable request that starts at location 0 and the request r_δ which is the first acceptable request that starts at location d (r_δ is the first request in R that starts at location d and starts after $\dot{t}_{r_\gamma}$), it is clear that $t_{r_1'^0} \leq t_{r_1^{*0}}$ and $t_{r_1'^d} \leq t_{r_1^{*d}}$. If $r_1'^0 \neq r_1^{*0}$, we can replace r_1^{*0} by $r_1'^0$ in $\bar{R}^*(0,d)$, and if $r_1'^d \neq r_1^{*d}$, we can replace r_1^{*d} by $r_1'^d$ in $\bar{R}^*(d,0)$. Similarly, if $i \leq \lfloor k/m \rfloor$, request $r_i'^0$ (resp. $r_i'^d$) starts earlier than request r_i^{*0} (resp. r_i^{*d}). Otherwise, server s_i' accepts request r_i^{*0} and r_i^{*d} instead of $r_i'^0$ and $r_i'^d$. If $r_i'^0 \neq r_i^{*0}$, we can replace r_i^{*0} by $r_i'^0$ in $\bar{R}^*(0,d)$, and if $r_1'^d \neq r_i^{*d}$, we can replace r_i^{*d} by $r_i'^d$ in $\bar{R}^*(d,0)$.

Now assume, that the first i ($i \geq \lfloor k/m \rfloor$) requests in $\bar{R}^*(0,d)$ are identical to the first i requests in $R'(0,d)$, and the first i requests in $\bar{R}^*(d,0)$ are identical to the first i requests in $R'(d,0)$ where $1 \leq i \leq k_1^*$. Note that server $s'_{(i+1) \bmod \lfloor k/m \rfloor}$ and $s^*_{i+1 \bmod \lfloor k/m \rfloor}$ are at location 0 at time $\dot{t}_{r'_{(i+1)-\lfloor k/m \rfloor}}$. If there are two requests r_{i+1}^{*0} and r_{i+1}^{*d} accepted by server $s^*_{(i+1) \bmod \lfloor k/m \rfloor}$, there must also be two requests $r_{i+1}'^0$ and $r_{i+1}'^d$ accepted by $s'_{(i+1) \bmod \lfloor k/m \rfloor}$ and request $r_{i+1}'^0$ (resp. $r_{i+1}'^d$) starts earlier than request r_{i+1}^{*0} (resp. r_{i+1}^{*d}). If $r_{i+1}'^0 \neq r_{i+1}^{*0}$, we can replace r_{i+1}^{*0} by $r_{i+1}'^0$ in $\bar{R}^*(0,d)$, and if $r_1'^d \neq r_{i+1}^{*d}$, we can replace r_{i+1}^{*d} by $r_{i+1}'^d$ in $\bar{R}^*(d,0)$. If there are no such requests r_{i+1}^{*0} and r_{i+1}^{*d} accepted by server $s^*_{(i+1) \bmod \lfloor k/m \rfloor}$, then $i+1 > k_1^*$, and hence it follows that $\bar{R}^*(d,0)$ is identical to $R'(d,0)$ (or $R'(d,0)$ even contains additional requests).

If $k_0^* = k_1^*$, the claim thus follows. If $k_0^* \neq k_1^*$ ($k_0^* - k_1^* = \tau$ where $1 \leq \tau \leq \lfloor k/m \rfloor$), then $\bar{R}^*(d,0)$ is already identical to $R'(d,0)$, and the first k_1^* requests of $\bar{R}^*(0,d)$ are already identical to the first k_1^* requests of $R'(0,d)$ by the argument above. Observe that server s_j' and s_j^*, $1 \leq j \leq \lfloor k/m \rfloor$, are at location 0 at time $\dot{t}_{r'_{k_1^*+j-\lfloor k/m \rfloor}}$. If there is a request $r_{k_1^*+o}^{*0}$ ($1 \leq o \leq \tau$) accepted by server s_j^*, there must also be a request $r_{k_1^*+o}'^0$ accepted by server s_j' and it starts no later than request $r_{k_1^*+o}^{*0}$. If $r_{k_1^*+o}'^0 \neq r_{k_1^*+o}^{*0}$, we can replace $r_{k_1^*+o}^{*0}$ by $r_{k_1^*+o}'^0$ in $\bar{R}^*(0,d)$ making $\bar{R}^*(0,d)$ identical to $R'(0,d)$. If there is no request $r_{k_1^*+o}^{*0}$ accepted by server s_j^*, then $k_1^* + o > k_0^*$, and hence it follows that $\bar{R}^*(0,d)$ is identical to $R'(0,d)$ (or $R'(0,d)$ even contains additional requests). As $\bar{R}^*(d,0)$ is already identical to $R'(d,0)$, we have that $\bar{R}^*(0,d) \cup \bar{R}^*(d,0)$ is identical to $R'(0,d) \cup R'(d,0)$ (or $R'(0,d) \cup R'(d,0)$ even contains additional requests). ◀

3.2 Upper bound for $a \geq t$

We propose a Bi-Partition Greedy Algorithm (Bi-PGA) for the kSmL-U problem when $a \geq t$, shown in Algorithm 3. We assume that $k \geq 2$. The k servers are divided into two groups: a group S^c of $\lfloor k/2 \rfloor$ servers and a group S^n of $\lceil k/2 \rceil$ servers. The $\lfloor k/2 \rfloor$ servers in S^c serve requests that start at location 0, and the $\lceil k/2 \rceil$ servers in S^n serve requests that drop off at location 0.

Algorithm 3 Bi-Partition Greedy Algorithm (Bi-PGA).

Input: k servers, requests arrive over time.

Step: When request r_i arrives, if $p_{r_i} = 0$ and r_i is acceptable to a server in S^c, assign it to that server; otherwise, if $\dot{p}_{r_i} = 0$ and r_i is acceptable to a server in S^n, assign it to that server; otherwise, reject it.

(1) R'_j ($1 \leq j \leq k$) is the list of requests accepted by server s_j before r_i is released.

(2) r_i is acceptable to a server s_j if and only if r_i is not in conflict with the requests in R'_j, i.e., $\forall r'_q \in R'_j$, $|t_{r_i} - t_{r'_q}| \geq 2t$.

▶ **Theorem 7.** *Bi-PGA is $\frac{k}{\lfloor k/2 \rfloor}$-competitive for kSmL-U if $a \geq t$. In particular, Bi-PGA is 2-competitive for kSmL-U if $a \geq t$ and k is even.*

Proof. For simplification of the analysis, we suppose that OPT can use k servers to serve requests that start at location 0 and another k servers to serve requests that drop off at location 0. This simplification does not decrease the profit gained by OPT. With this we can analyse the requests in $R'(0, X)$ and $R'(X, 0)$ independently. In the following analysis, we focus on the requests that start at location 0. Let $R'(0, X) = \{r'_1, \ldots, r'_{|R'(0,X)|}\}$ and $R^*(0, X) = \{r^*_1, \ldots, r^*_{|R^*(0,X)|}\}$, indexed in the order in which the requests are released.

Similar to the proof of Theorem 6, the analysis of the algorithm is divided into two parts. First, we reassign the requests in $R'(0, X)$ and $R^*(0, X)$ by repeated application of the following reassignment rule so that servers are assigned to the accepted requests in round-robin fashion. Then we show that the profit gained by the algorithm is within a certain factor of the profit accrued by OPT.

Reassignment Rule Assume that request r^*_o is assigned to server $s^*_{o \bmod k}$ for $o < i$ and r^*_i is not assigned to the server $s^*_{i \bmod k}$. Suppose r^*_i is assigned to s^*_j, $j \neq i \bmod k$. We reassign r^*_i to server $s_{i \bmod k}$, reassign all requests in $R^* \setminus \{r^*_1, r^*_2, \ldots, r^*_i\}$ that are assigned to s^*_j (denote the set of these requests by $\Re_j$) to server $s^*_{i \bmod k}$, and reassign all requests in $R^* \setminus \{r^*_1, r^*_2, \ldots, r^*_i\}$ that are assigned to $s^*_{i \bmod k}$ (denote them by $\Re_{i \bmod k}$) to server s^*_j.

Since request r^*_o is assigned to server $s^*_{o \bmod k}$ for $o < i$, the latest request r_l with pick-up time earlier than $t_{r^*_i}$ that is accepted by $s^*_{i \bmod k}$ ends no later than the latest request with pick-up time earlier than $t_{r^*_i}$ that is accepted by s^*_j if $j \neq i \bmod k$. Reassigning all requests of $\Re_j$ to server $s^*_{i \bmod k}$ is valid. Because the first request in $\Re_{i \bmod k}$ accepted by $s^*_{i \bmod k}$ starts no earlier than $t_{r^*_i}$ and any two consecutive requests in $\Re_{i \bmod k}$ are not in conflict, reassigning all requests of $\Re_{i \bmod k}$ to server s^*_j is valid as well. From this it follows that $R^*(0, X)$ is still a valid solution with the same profit after the reassignment. Similarly, we reassign the requests in $R'(0, X)$ based on the above process until request r'_i is assigned to $s'_{i \bmod \lfloor k/2 \rfloor}$ for $i \leq |R'(0, X)|$. Note that this reassignment does not affect the validity of $R'(0, X)$, and $P_{R'(0,X)}$ does not change.

The remainder of the proof proceeds similarly as the proof of Theorem 6, but here we have that Bi-PGA has $\lfloor k/2 \rfloor$ servers while OPT has k servers and all requests accepted by OPT and Bi-PGA start at location 0.

Let $\bar{R}^*(0, X)$ be the subset of $R^*(0, X)$ that contains the requests from the $(ik + 1)^{th}$ to the $(ik + \lfloor k/2 \rfloor)^{th}$, for all i. In other words, $\bar{R}^*(0, X)$ contains the requests that are accepted by the first $\lfloor k/2 \rfloor$ servers of OPT. Suppose the first $\lfloor k/2 \rfloor$ servers of OPT accept k^*_0 requests that start at location 0, i.e., $\bar{R}^*(0, X) = \{r^*_1, r^*_2, \ldots, r^*_{k^*_0}\}$. Suppose Bi-PGA accepts k_0 requests that start at location 0, i.e., $R'(0, X) = \{r'_1, r'_2, \ldots, r'_{k_0}\}$. We claim that $\bar{R}^*(0, X)$ can be transformed into $R'(0, X)$ without reducing its profit, thus showing that $P_{\bar{R}^*(0,X)} \leq P_{R'(0,X)}$, and hence $P_{R^*(0,X)} \leq \frac{k}{\lfloor k/2 \rfloor} P_{R'(0,X)}$.

As Bi-PGA accepts the request r_γ which is the first acceptable request that starts at location 0, it is clear that $t_{r'_1} \le t_{r^*_1}$. If $r'_1 \ne r^*_1$, we can replace r^*_1 by r'_1 in $\bar{R}^*(0,X)$. Similarly, if $i \le \lfloor k/2 \rfloor$, request r'_i starts earlier than request r^*_i. Otherwise, server s'_i accepts request r^*_i instead of r'_i. If $r'_i \ne r^*_i$, we can replace r^*_i by r'_i in $\bar{R}^*(0,X)$.

Now assume, that the first i ($i \ge \lfloor k/2 \rfloor$) requests in $\bar{R}^*(0,X)$ are identical to the first i requests in $R'(0,X)$, where $1 \le i \le k^*_0$. Note that server $s'_{(i+1) \bmod \lfloor k/2 \rfloor}$ and $s^*_{i+1 \bmod \lfloor k/2 \rfloor}$ are at location $\dot{p}_{r'_{(i+1)-\lfloor k/2 \rfloor}}$ ($\dot{p}_{r'_{(i+1)-\lfloor k/2 \rfloor}} \ne 0$) at time $\dot{t}_{r'_{(i+1)-\lfloor k/2 \rfloor}}$. If there is a request r^*_{i+1} accepted by server $s^*_{(i+1) \bmod \lfloor k/2 \rfloor}$, there must also be a request r'_{i+1} accepted by $s'_{(i+1) \bmod \lfloor k/2 \rfloor}$ and request r'_{i+1} starts no later than request r^*_{i+1}. If $r'_{i+1} \ne r^*_{i+1}$, we can replace r^*_{i+1} by r'_{i+1} in $\bar{R}^*(0,X)$. If there is no such request r^*_{i+1} accepted by server $s^*_{(i+1) \bmod \lfloor k/2 \rfloor}$, then $i+1 > k^*_0$, and hence it follows that $\bar{R}^*(0,X)$ is identical to $R'(0,X)$ (or $R'(0,X)$ even contains additional requests). ◀

4 kSmL-A: Arbitrary travel times

▶ **Theorem 8.** *For $a \ge 2Lt$ and an arbitrary number k of servers, no deterministic on-line algorithm for kSmL-A can achieve competitive ratio smaller than $\frac{1}{2}\ln L$.*

Proof. Consider a star with $m+1$ nodes and $d(0,v) = v$ for $1 \le v \le m$. Note that $L = m$ and hence $\ln L = \ln m$. The adversary presents requests in γ phases, where phase i, for $1 \le i \le \gamma$, consists of k identical requests. The requests are released based on the release rule for kSmL-A shown in Algorithm 4. All requests appear at the same time.

Algorithm 4 Release Rule for kSmL-A.

Initialization: The adversary presents the requests in phase 1: k copies of the request $(\nu \cdot t - a, \nu \cdot t, 1, 0)$ for some ν such that $\nu \in \mathbb{N}$ and $\nu \cdot t - a \ge 0$.
$i = 1$;
While $i < m$ do
  Let k_i be the number of servers used in phase i.
  if $\sum_{j=1}^{i}(j \cdot k_j) \le \frac{2ki}{\ln m}$, then break;
  else $i = i+1$ and the adversary releases the requests in phase i;
$\gamma = i$;
(1) Phase i ($1 < i \le m$) consists of k copies of the request $(\tilde{t}_{r_1}, t_{r_1}, i, 0)$.

We make four observations.

- **(a)** The requests in any two different phases are in conflict.
- **(b)** For all $i < \gamma$, $\sum_{j=1}^{i}(j \cdot k_j) > \frac{2ki}{\ln m}$;
- **(c)** If $\gamma > 1$, $k_1 > \frac{2k}{\ln m}$;
- **(d)** For all $i < \gamma$, $\sum_{j=1}^{i} k_j > \frac{2k}{\ln m} \cdot \sum_{j=1}^{i} \frac{1}{j}$. (This follows from (b) and (c).)

If $\gamma < m$, the adversary has stopped releasing requests because $\sum_{j=1}^{\gamma}(j \cdot k_j) \le \frac{2k\gamma}{\ln m}$. In this case, OPT accepts all requests in phase γ, and we have $P_{R^*} = \gamma k r$. Since $P_{R'} = r \cdot \sum_{j=1}^{\gamma}(j \cdot k_j) \le \frac{2kr\gamma}{\ln m}$, $P_{R^*}/P_{R'} \ge \frac{1}{2} \cdot \ln m$.

Now assume $\gamma = m$. Using (d) for $i = m-1$, we have $\sum_{j=1}^{m-1} k_j > \frac{2k}{\ln m} \cdot \sum_{j=1}^{m-1} \frac{1}{j} > \frac{2k}{\ln m} \cdot \ln m = 2k$, a contradiction because the algorithm has only k servers and by (a) no server can serve requests from different phases. Therefore, the case $\gamma = m$ cannot occur. ◀

The Classified Greedy Algorithm

We use a deterministic version of the "Classify and Randomly Select" paradigm, which has been widely used in on-line interval scheduling and many other problems [2, 8], to design a classified greedy algorithm (CGA). We partition the requests into classes based on their travel time, and we assign a number of servers to each class of the requests. Given k servers and a star of $m+1$ locations whose edge lengths satisfy $1 \leq d(0,i) \leq L$ for all $1 \leq i \leq m$, we use $\lambda = \lceil \log L \rceil$ groups of servers. We require that $k \geq \lambda$, and for ease of presentation we assume that k is an integer multiple of λ. Group j, $1 \leq j \leq \lceil \log L \rceil$, contains k/λ servers that only serve requests whose travel time is between $2^{j-1}t$ and $2^j t$ (we say that those requests are in class j).

The classified greedy algorithm (CGA) can now be stated in a simple way: When a request r_i arrives, let $j = \lceil \log d(p_{r_i}, \dot{p}_{r_i}) \rceil$ be the class of r_i (if $d(p_{r_i}, \dot{p}_{r_i}) = 1$, set $j = 1$). If r_i is acceptable to any server from group j, accept r_i with that server. Otherwise reject it.

To simplify the analysis, we suppose that for each j, $1 \leq j \leq \lceil \log L \rceil$, OPT can use k separate servers to serve the requests whose travel times are between $2^{j-1}t$ and $2^j t$. This simplification does not decrease the profit gained by OPT. In this way we can analyse the requests in different classes independently. In the following analysis, we focus on an arbitrary class j. For class j, let OPT_j be the requests of class j that are accepted by OPT, and let CGA_j be the requests of class j accepted by CGA. It is clear that $P_{R^*}/P_{R'} = O(\log L)$ follows if we can show that $\frac{|OPT_j|}{|CGA_j|} = O(\log L)$ for each j.

▶ **Lemma 9.** *For each j, $\frac{|OPT_j|}{|CGA_j|} = O(\log L)$.*

Proof. For the purpose of the analysis, partition the set of k servers of OPT_j into k/λ sets of size λ arbitrarily, where $\lambda = \lceil \log L \rceil$ as above. Each of these sets is assigned to a distinct server s'_i among the k/λ servers of CGA_j. For $1 \leq i \leq k/\lambda$, let A_i be the set of λ servers of OPT_j that is assigned to s'_i, and let $R'(i)$ denote the set of requests accepted by s'_i.

For each OPT_j server $s^*_e \in A_i$, let $R^*(e)$ be the set of requests accepted by s^*_e and $\bar{R}^*(e)$ be the set of requests accepted by s^*_e that are not accepted by CGA_j. Let $\bar{R}^*(A_i) = \bigcup_{s^*_e \in A_i} \bar{R}^*(e)$. We claim that $|\bar{R}^*(e)| \leq \alpha |R'(j)|$ for some constant α. If this claim holds, we get that $|OPT_j| \leq |CGA_j| + \sum_i |\bar{R}^*(A_i)| \leq |CGA_j| + \sum_i \lambda\alpha|R'(i)| = (1+\lambda\alpha)|CGA_j| = O(\lambda) \cdot |CGA_j|$, proving the lemma.

It remains to prove the claim. Consider any request r_h in $\bar{R}^*(e)$. As s'_i did not accept r_h, s'_i must have accepted another request r_c with start time in $(t_{r_h} - 3 \cdot 2^j t, t_{r_h}]$; otherwise, the $3 \cdot 2^j t$ time units would have been sufficient for s'_i to serve the previous request and make an empty move to the pick-up location of r_h. We charge r_h to r_c. In this way, every request in $\bar{R}^*(e)$ is charged to a request in $R'(i)$.

We bound the number of requests that can be charged to a single request r_c in $R'(i)$. By the above charging scheme, every request that was accepted by s^*_e and charges r_c has a start time in $[t_{r_c}, t_{r_c} + 3 \cdot 2^j t)$. As all requests in class j have travel time at least $2^{j-1}t$, the start times of consecutive requests accepted by s^*_e differ by at least $2^{j-1}t$. A half-open interval of length $3 \cdot 2^j t$ can therefore contain at most $\frac{3 \cdot 2^j t}{2^{j-1} t} = 6$ request start times, and hence r_c is charged by at most 6 requests from $\bar{R}^*(e)$. This establishes the claim, with $\alpha = 6$. ◀

▶ **Theorem 10.** *CGA is $O(\log L)$-competitive for kSmL-A if $a \geq 2Lt$ and the number of servers is at least $\lceil \log L \rceil$.*

5 Conclusion

We have studied an on-line problem with k servers and $m+1$ locations in a star network that is motivated by applications such as car sharing between an airport and hotels. In particular, we have analyzed the effects that different constraints on the booking time of requests have on the competitive ratio that can be achieved. We have given matching lower and upper bounds on the competitive ratio. It would be interesting to extend our results to the case of other networks.

References

1 Norbert Ascheuer, Sven Oliver Krumke, and Jörg Rambau. Online Dial-a-Ride Problems: Minimizing the Completion Time. In *Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science (STACS '00)*, volume 1770 of *LNCS*, pages 639–650. Springer, 2000. doi:10.1007/3-540-46541-3_53.

2 Baruch Awerbuch, Yair Bartal, Amos Fiat, and Adi Rosén. Competitive Non-Preemptive Call Control. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '94)*, pages 312–320. ACM/SIAM, 1994. URL: http://dl.acm.org/citation.cfm?id=314464.314510.

3 Antje Bjelde, Yann Disser, Jan Hackfeld, Christoph Hansknecht, Maarten Lipmann, Julie Meißner, Kevin Schewior, Miriam Schlöter, and Leen Stougie. Tight Bounds for Online TSP on the Line. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '17)*, pages 994–1005. SIAM, 2017. doi:10.1137/1.9781611974782.63.

4 Katerina Böhmová, Yann Disser, Matús Mihalák, and Rastislav Srámek. Scheduling Transfers of Resources over Time: Towards Car-Sharing with Flexible Drop-Offs. In *Proceedings of the 12th Latin American Symposium on Theoretical Informatics (LATIN'16)*, volume 9644 of *LNCS*, pages 220–234. Springer, 2016. doi:10.1007/978-3-662-49529-2_17.

5 Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis.* Cambridge University Press, 1998.

6 Ananya Christman, William Forcier, and Aayam Poudel. From theory to practice: Maximizing revenues for on-line dial-a-ride. *J. Comb. Optim.*, 35(2):512–529, 2018. doi:10.1007/s10878-017-0188-z.

7 Sven Oliver Krumke, Willem de Paepe, Diana Poensgen, Maarten Lipmann, Alberto Marchetti-Spaccamela, and Leen Stougie. On Minimizing the Maximum Flow Time in the Online Dial-a-Ride Problem. In *Proceedings of the 3rd International Workshop on Approximation and Online Algorithms (WAOA 2005)*, volume 3879 of *LNCS*, pages 258–269. Springer, 2006. doi:10.1007/11671411_20.

8 Richard J. Lipton and Andrew Tomkins. Online Interval Scheduling. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '94)*, pages 302–311. ACM/SIAM, 1994. URL: http://dl.acm.org/citation.cfm?id=314464.314506.

9 Kelin Luo, Thomas Erlebach, and Yinfeng Xu. Car-Sharing Between Two Locations: Online Scheduling with Flexible Advance Bookings. In *Proceedings of the 24th International Conference on Computing and Combinatorics (COCOON 2018)*, volume 10976 of *LNCS*, pages 242–254. Springer, 2018. doi:10.1007/978-3-319-94776-1_21.

10 Kelin Luo, Thomas Erlebach, and Yinfeng Xu. Car-Sharing between Two Locations: Online Scheduling with Two Servers. In *Proceedings of the 43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*, volume 117 of *LIPIcs*, pages 50:1–50:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.MFCS.2018.50.

11 Kelin Luo, Thomas Erlebach, and Yinfeng Xu. Online Scheduling of Car-Sharing Requests between Two Locations with Many Cars and Flexible Advance Bookings. In *Proceedings of the 29th International Symposium on Algorithms and Computation (ISAAC 2018)*, volume 123 of *LIPIcs*, pages 64:1–64:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ISAAC.2018.64.

12 Fanglei Yi and Lei Tian. On the Online Dial-A-Ride Problem with Time-Windows. In *Proceedings of the 1st International Conference on Algorithmic Applications in Management (AAIM '05)*, volume 3521 of *LNCS*, pages 85–94. Springer, 2005. doi:10.1007/11496199_11.