Car-sharing between two locations

Citation for published version (APA):

DOI:
10.4230/LIPIcs.MFCS.2018.50

Document status and date:
Published: 01/08/2018

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.
Car-Sharing between Two Locations: Online Scheduling with Two Servers

Kelin Luo
School of Management, Xi’an Jiaotong University
Xianning West Road, Xi’an, China
luokelin@xjtu.edu.cn
https://orcid.org/0000-0003-2006-0601

Thomas Erlebach
Department of Informatics, University of Leicester
Leicester, United Kingdom
te17@leicester.ac.uk
https://orcid.org/0000-0002-4470-5868

Yinfeng Xu
School of Management, Xi’an Jiaotong University
Xianning West Road, Xi’an, China
yfxu@xjtu.edu.cn

Abstract

In this paper, we consider an on-line scheduling problem that is motivated by applications such as car sharing, in which users submit ride requests, and the scheduler aims to accept requests of maximum total profit using two servers (cars). Each ride request specifies the pick-up time and the pick-up location (among two locations, with the other location being the destination). The length of the time interval between the submission of a request (booking time) and the pick-up time is fixed. The scheduler has to decide whether or not to accept a request immediately at the time when the request is submitted. We present lower bounds on the competitive ratio for this problem and propose a smart greedy algorithm that achieves the best possible competitive ratio.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Car-sharing system, Competitive analysis, On-line scheduling

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.50

1 Introduction

In a car-sharing system, a company offers cars to customers for a period of time. Customers can pick up a car in one location, drive it to another location, and return it there. Car booking requests arrive on-line, and the goal is to maximize the profit obtained from satisfied requests. We consider a setting where all driving routes go between two fixed locations, but can be in either direction. For example, the two locations could be a residential area and a nearby shopping mall or central business district. Other applications that provide motivation for the problems we study include car rental, taxi dispatching and boat rental for river crossings.

1 This work was partially supported by the China Postdoctoral Science Foundation (Grant No. 2016M592881), and the China Scholarship Council (Grant No. 201706280058).
In a real setting, customer requests for car bookings arrive over time, and the decision about each request must be made immediately, without knowledge of future requests. This gives rise to an on-line problem that bears some resemblance to interval scheduling, but in which additionally the pick-up and drop-off locations play an important role: The server that serves a request must be at the pick-up location at the start time of the request and will be located at the drop-off location at the end time of the request. A server can serve two consecutive requests 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' that allow a server to 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.

We assume that every request is associated with a profit $r > 0$ that is obtained if the request is accepted. When a server moves while not serving a request, a certain cost $c$, $0 \leq c \leq r$, is incurred. The goal is to maximize the total profit, which is the sum of the profits of the accepted requests minus the costs incurred for moving servers while not serving a request. We refer to this problem as the car-sharing problem. The time interval between the submission of a request (booking time) and the pick-up time is called the booking interval. In this paper, we focus on the special case of two servers and assume that the booking interval for each request is a fixed value $a$ that is the same for all requests. We assume that $a \geq t$, where $t$ is the time to move a server from one location to the other.

In [8], the authors studied the car-sharing problem for the special case of a single server, considering both the case of fixed booking intervals and the case of flexible booking intervals, and presented tight results for the competitive ratio. The optimal competitive ratio was shown to be $2r/(r - c)$ for fixed booking intervals and $(3r - c)/(r - c)$ for flexible booking intervals if $0 \leq c < r$, and 1 for fixed booking intervals and proportional to the length of the booking horizon (the range of allowed booking intervals) for flexible booking intervals if $c = r$.

The car-sharing problem belongs to the class of dynamic pickup and delivery problems surveyed by Berbeglia et al. [2]. The problem that is closest to our setting is the on-line dial-a-ride problem (OLDARP) that has been widely studied in the literature. 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. Known results for OLDARP include on-line algorithms for minimizing the makespan [1, 3] or the maximum flow time [7]. Work on versions of OLDARP where not all requests can be served includes competitive algorithms for requests with deadlines where each request must be served before its deadline or rejected [9], and for settings with a given time limit where the goal is to maximize the revenue from requests served before the time limit [6]. In contrast to existing work on OLDARP, in this paper we consider requests that need to be served at a specific time that is specified by the request when it is released. Another related problem is the $k$-server problem [5, Ch. 10], but in that problem all requests must be served and requests are served at a specific location.

Off-line versions of car-sharing problems are studied by Böhmová et al. [4]. They show that if all customer requests for car bookings are known in advance, the problem of maximizing the number of accepted requests can be solved in polynomial time using a minimum-cost network flow algorithm. Furthermore, they consider 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 prove that this variant is NP-hard and APX-hard. In contrast to their work, we consider the on-line version of the problem with two servers.
In Section 2, we define the problem, introduce terminology, and present lower bounds on the competitive ratio. If $0 \leq c < r$, the lower bound is 2, and if $c \geq r$, the lower bound is 1. In Section 3, we propose a smart greedy algorithm that achieves the best possible competitive ratio. Section 4 concludes the paper.

## 2 Problem Formulation and Preliminary Results

### 2.1 Definitions and Problem Formulation

We consider a setting with only two locations (denoted by $s_1$ and $s_2$). The travel time from 0 to 1 is the same as the travel time from 1 to 0 and is denoted by $t$. Let $R$ denote a sequence of requests that are released over time. The $i$-th request is denoted by $r_i = (l_{r_i}, t_{r_i}, p_{r_i})$ and is specified by the booking time or release time $l_{r_i}$, the start time (or pick-up time) $t_{r_i}$, and the pick-up location $p_{r_i} \in \{0, 1\}$. We assume that the booking interval $t_{r_i} - l_{r_i}$ is equal to a fixed value $a$ for all requests $r_i \in R$, and we assume that $a \geq t$ so that an available server always has enough time to travel to the pick-location of a request. If $r_i$ is accepted, the server must pick up the customer at $p_{r_i}$ at time $t_{r_i}$ and drop off the customer at location $\hat{p}_{r_i} = 1 - p_{r_i}$, the drop-off location of the request, at time $t_{r_i} = l_{r_i} + t$, the end time (or drop-off time) of the request. We say that the request $r_i$ starts at time $t_{r_i}$. For an interval $[b, d)$, we say that $r_i$ starts in the interval if $t_{r_i} \in [b, d)$. Each server can only serve one request at a time. Serving a request yields profit $r > 0$. The two servers are initially located at location 0. If the pick-up location $p_{r_i}$ of a request $r_i$ is different from the current location of a server and if at least $t$ time units remain before the start of $r_i$, the server can move from its current location to $p_{r_i}$. We refer to such moves (which do not serve a request) as empty moves. An empty move takes time $t$ and incurs a cost of $c$, $0 \leq c \leq r$, and we say that $r_i$ is accepted with cost in this case. If the server is already located at $p_{r_i}$, we say that $r_i$ is accepted without cost. If two requests are such that they cannot both be served by one server, we say that the requests are in conflict. We do not require that the algorithm assigns an accepted request to a server immediately, provided that it ensures that one of the two servers will serve the request. In our setting with fixed booking intervals, however, it is not necessary for an algorithm to use this flexibility.

We denote the requests accepted by an algorithm by $R'$, and the $i$-th request in $R'$, in order of request start times, is denoted by $r'_i$. The $l$-th request which is assigned to $s_j$ ($j \in \{1, 2\}$) in $R'$, in order of request start times, is denoted by $r'_{i,j}$. Suppose $r'_{i,j}$ ($j \in \{1, 2\}$) is $r'_i$. We say that request $r'_i$ is accepted without cost if $l = 1$ and $p_{r'_i,j} = 0$ or if $l > 1$ and $p_{r'_{i,j}} = \hat{p}_{r'_{i-1,j}}$. Otherwise, $r'_i$ is accepted with cost. We denote the profit of serving the requests in $R'$ by $P_{R'}$. If $R'_c$ denotes the subset of $R'$ consisting of the requests that are accepted with cost, we have $P_{R'} = r \cdot |R'| - c \cdot |R'_c|$. The goal of the car-sharing problem is to accept a set of requests $R'$ that maximizes the profit $P_{R'}$. The problem for two servers and two locations is called the 2S2L problem.

### 2.2 Online Optimization and Competitive Analysis

From an online perspective, the requests in $R$ and the number of requests in $R$ are unknown, and request $r_i$ only becomes known at time $l_{r_i}$. For any request sequence $R$, let $P_{R,A}$ denote the objective value produced by an on-line algorithm $A$, and $P_{R,OPT}$ that obtained by an optimal scheduler $OPT$ that has full information about the request sequence in advance.
The performance of an online algorithm for 2S2L is measured using competitive analysis (see [5]). The competitive ratio of an algorithm $A$ is defined as $\rho_A = \sup_R \frac{\text{cost}_A(R)}{\text{cost}_O(R)}$. We say that $A$ is $\rho$-competitive if $\rho_A \leq \rho \cdot \rho_{OA}$ for all request sequences $R$. Let $ON$ be the set of all on-line algorithms for a problem. A value $\beta$ is a lower bound on the best possible competitive ratio if $\rho_A \geq \beta$ for all $A$ in $ON$. We say that an algorithm $A$ is optimal if there is a lower bound $\beta$ with $\rho_A = \beta$.

2.3 Lower Bounds

In this subsection, we present the lower bounds for the 2S2L problem. We use $ALG$ to denote any on-line algorithm and $OPT$ to denote an optimal scheduler. We refer to the servers of $ALG$ as $s'_1$ and $s'_2$, and the servers of $OPT$ as $s_1^*$ and $s_2^*$, respectively. The set of requests accepted by $ALG$ is referred to as $R'$, and the set of requests accepted by $OPT$ as $R^*$. For the case $c \geq r$, a lower bound of 1 on the competitive ratio of any algorithm holds trivially.

\begin{itemize}
    \item \textbf{Theorem 1.} For $0 \leq c < r$, no deterministic on-line algorithm for 2S2L can achieve competitive ratio smaller than 2.
\end{itemize}

\textbf{Proof.} Initially, the adversary releases $r_1$ and $r_2$ with $r_1 = r_2 = (t, t + a, 1)$. We distinguish three cases.

\begin{itemize}
    \item \textbf{Case 1:} $ALG$ accepts $r_1$ and $r_2$ (with cost). Note that $r_1$ and $r_2$ are assigned to different servers as they are in conflict. The adversary releases requests $r_3$ and $r_4$ with $r_3 = r_4 = (\varepsilon + t, a + \varepsilon + t, 0)$ and $r_5$ and $r_6$ with $r_5 = r_6 = (\varepsilon + 2t, a + \varepsilon + 2t, 1)$, where $0 < \varepsilon < t$. $OPT$ accepts $r_3$, $r_4$, $r_5$ and $r_6$ without cost, but $ALG$ cannot accept any of these requests as they are in conflict with $r_1$ and $r_2$. We have $P_{R'} = 4r$ and $P_{R^*} \leq 2(r - c)$, and hence $P_{R^*}/P_{R'} \geq 2$.
    \item \textbf{Case 2:} $ALG$ accepts either $r_1$ or $r_2$. The adversary accepts $r_1$ and $r_2$. We have $P_{R^*} = 2(r - c)$ and $P_{R'} \leq r - c$, and hence $P_{R^*}/P_{R'} \geq 2$.
    \item \textbf{Case 3:} $ALG$ does not accept request $r_1$ and $r_2$. In this case, $OPT$ accepts $r_1$ and $r_2$ and we have $P_{R^*} = 2(r - c)$ and $P_{R'} = 0$, and hence $P_{R^*}/P_{R'} = \infty$.
\end{itemize}

3 Upper Bound

In this section, we propose a Smart Greedy Algorithm (SG) for the 2S2L problem, shown in Algorithm 1. Intuitively, if a request is acceptable, the algorithm always accepts it if this increases the profit by $r$, and it accepts the request only if it starts at least $t$ time units later than the end time of the latest previously accepted request if the profit increase is positive but less than $r$. The algorithm uses the following notation:

- $R'_i$ is the set of requests accepted by SG before $r_i$ is released, together with the server to which each request is assigned. $R'_i \cup \{r_{i,s'}\}$ denotes the union of $R'_i$ and $\{r_{i,s'}\}$, where $r_{i,s'}$ represents the request $r_i$ assigned to server $s'$, $j \in \{1, 2\}$, without conflict.
- $r_{i,j}$ denotes the latest request which was assigned to $s'$, $j \in \{1, 2\}$, before $r_i$ is released. (If there is no such request, take $r_{i,j}$ to be a dummy request with drop-off location 0 and drop-off time 0.)
- $r_i$ is acceptable if and only if $\exists j \in \{1, 2\}: t_{r_i} - \hat{t}_{r_{i,j}} \leq t$ if $p_{r_i} = p_{r_{i,j}}$, and $t_{r_i} - \hat{t}_{r_{i,j}} \geq 0$ if $p_{r_i} \neq p_{r_{i,j}}$.
- $r_n^*$ is the latest request that was accepted before $r_i$ is released. Note that $r_n^* = r_{i,j}$ with $j = \arg \max \{t_{r_{i,j}} \mid j = 1, 2\}$. Note that $\hat{t}_{r_n^*} = 0$. 


Algorithm 1 Smart Greedy Algorithm (SG).

Input: two servers, requests arrive over time with fixed booking interval $a$.

Step: When request $r_i$ arrives, accept $r_j$ and assign it to the most economical server $s_j'$ if $r_i$ is acceptable and $P_{R'_i \cup \{r_{s_j'}\}} - P_{R'_i} = r (j \in \{1, 2\})$, or if $r_i$ is acceptable, $P_{R'_i \cup \{r_{s_j'}\}} - P_{R'_i} > 0$ ($j \in \{1, 2\}$) and $t_{r_i} - t_{r_{s_j'}} \geq t$;

If an accepted request is acceptable by both servers, it is assigned to the most economical server, which is the server $s_j'$ with $j = \arg \max \{P_{R'_i \cup \{r_{s_j'}\}} \mid j \in \{1, 2\}\}$. If $P_{R'_i \cup \{r_{s_j'}\}} = P_{R'_i \cup \{r_{s_j'}\}}$, $s_j'$ is chosen as the server which has accepted $r_i^0$ (or arbitrarily in case $r_i^0$ does not exist).

We use OPT to denote an optimal scheduler. We refer to the servers of SG as $s_1'$ and $s_2'$, and the servers of OPT as $s_1^*$ and $s_2^*$, respectively. For an arbitrary request sequence $R = (r_1, r_2, r_3, \ldots, r_n)$, note that we have $t_{r_i} \leq t_{r_{s_i}}$ for $1 \leq i \leq n$ because $t_{r_i} - t_{r_{s_i}} = a$ is fixed. Denote the requests accepted by $OPT$ by $R^* = \{r_1^*, r_2^*, \ldots, r_k^*\}$ and the requests accepted by SG by $R' = \{r_1', r_2', \ldots, r_k'\}$, indexed in order of non-decreasing start times. Denote the requests accepted by SG which start at location 0 by $R^0 = \{r_1^0, r_2^0, \ldots, r_k^0\}$ and the requests accepted by SG which start at location 1 by $R^1 = \{r_1^1, r_2^1, \ldots, r_k^1\}$. Denote the requests accepted by OPT which start at location 0 by $R^0 = \{r_1^0, r_2^0, \ldots, r_k^0\}$ and the requests accepted by OPT which start at location 1 by $R^1 = \{r_1^1, r_2^1, \ldots, r_k^1\}$. Note that $R^0 \cup R^1 = R'$ and $R^0 \cup R^1 = R^*$.

Theorem 2. Algorithm SG is 1-competitive for 2S2L if $c = r$.

Proof. If $c = r$, accepting a request with cost yields profit $r - c = 0$. Without loss of generality, we can therefore assume that both SG and OPT only accept requests without cost. Observe that this means that both the SG servers ($s_1'$ and $s_2'$) and the OPT servers ($s_1^*$ and $s_2^*$) accept requests with alternating pick-up location, starting with a request with pick-up location 0. Therefore each server can accept at most one more request which starts at location 0 over the requests which start at location 1. That means when OPT accepts $w$ requests which start at location 1, OPT at least accepts $w$ requests which start at location 0, and accepts at most $w + 2$ requests which start at location 0 ($k_1^* \leq k_0^* \leq k_1^* + 2$).

Considering the condition that requests $r_i^0$ and $r_i^1$ are both assigned to the same server for $j < i$ and $r_i^0$ and $r_i^1$ are assigned to different servers (without loss of generality, suppose $r_i^0$ is assigned to $s_1^*$ and $r_i^1$ is assigned to $s_2^*$), we reassign $r_i^1$ to server $s_1^*$, reassign all requests in $R^* \setminus \{\{r_i^0, r_i^0, \ldots, r_{i+1}^0\} \cup \{r_i^1, r_i^1, \ldots, r_{i+1}^1\}\}$ that are assigned to $s_1^*$ (denote the set of these requests by $R_1$) to server $s_2^*$, and reassign all requests in $R^* \setminus \{\{r_i^0, r_i^0, \ldots, r_{i+1}^0\} \cup \{r_i^1, r_i^1, \ldots, r_{i+1}^1\}\}$ that are assigned to $s_2^*$ (denote them by $R_2$) to server $s_1^*$. As each server accepts requests with alternating pick-up location, starting with a request with pick-up location 0, we have $t_{r_i^0} \leq t_{r_i^1}$ (for all $i \leq k_1^*$) and $t_{r_i^0} \leq t_{r_i^1}$ (for all $i \leq k_1^*$). That means for $i \leq k_1^*$, $r_i^0$ and $r_i^1$ are not in conflict, and hence reassigning $r_i^1$ to server $s_1^*$ is valid. Observe that $s_2^*$ must serve a request which has pick-up location 0 and starts during interval $[t_{r_i^0}, t_{r_i^1} - t]$ and that request is $r_{i+1}^0$. Because $t_{r_i^1} \leq t_{r_i^1} + t$ and the first request in $R_1$, denoted by $r_o$, has pick-up location 1 and starts after $t_{r_i^1}$, $r_o$ and $r_{i+1}^0$ are not in conflict. As any two consecutive requests in $R_1$ are not in conflict, reassigning all requests of $R_1$ to server $s_2^*$ is valid. Note that $t_{r_{i+2}^0} \geq t_{r_{i+1}^1}$, as OPT accepts at most two requests which start during interval $[t_{r_i^0}, t_{r_i^1}]$ (during interval $[0, t_{r_i^1}]$ if $i = 1$) and have pick-up location 0. Because the first request $(r_1)$ in $R_2$ starts at 0 and starts after $t_{r_{i+1}^1}$, $r_l$ and
r^1 are not in conflict. As any two consecutive requests in R_2 are not in conflict, reassigning all requests of R_2 to server s^1 is valid. From this it follows that R' is still a valid solution with the same profit after the reassignment. For simplification of the analysis, we reassign the requests in R^* and R' based on the above process until both request r^0 and r^1 are assigned to the same server for i ≤ k^*_0, and r^0 and r^1 are assigned to the same server for i ≤ k^*_1. Note that this reassignment does not affect the validity of R^* and R', and P_{R^*} and P_{R'} do not change.

We claim that R^* can be transformed into R' without reducing its profit, thus showing that P_{R^*} = P_{R'}. As SG accepts the request r^i which is the first acceptable request that starts at location 0 and the request r^j which is the first acceptable request that starts at location 1 (r_0 is the first request in R that starts at location 1 and starts after t_i.), it is clear that t_r_0 ≤ t_r_0 and t_r_1 ≤ t_r_1. If r^0 ≠ r^0, we can replace r^0 by r^0 in R^0, and if r^1 ≠ r^1, we can replace r^1 by r^1 in R^1.

Now assume, that R' and R^* are identical with respect to 2i requests (i requests in R^0 and i requests in R^1) and with i requests in R^0 and i requests in R^1, where 1 ≤ i ≤ k^*_1, and both requests r^0 and r^1 are assigned to the same server for 1 ≤ j ≤ i.

Without loss of generality, suppose r^1 is assigned to s^1 by OPT and r^1 is assigned to s^1 by SG. Observe that s^1 and s^1 are at location 0 at time t_i.. We claim that s^2 (resp. s^2) is at location 0 at time t_i, and t_r_i ≤ t_r_i. If r^1 is assigned to s^2 (resp. s^2), s^2 (resp. s^2) is at location 0 at time t_r_i and t_r_i = min{t_r_i, t_r_i}. If r^1 is assigned to s^1 (resp. s^1), we have t_r_i ≤ t_r_i ≤ t_r_i. Observe that OPT does not accept any request which starts in period (t_r_i, t_r_i). As both SG servers, s^1 and s^2, and OPT servers, s^1 and s^2, accept requests with alternating pick-up location and starting with a request with pick-up location 0, either the pick-up location of the request r (where r is the last request which starts at or before t_r_i and is assigned to s^2 (resp. s^2)) is 1, or s^2 (resp. s^2) does not accept any request which starts before t_r_i. Hence s^2 (resp. s^2) is at location 0 at time t_r_i (≤ t_r_i), or at time 0 if r does not exist, and stays at that location until time t_r_i.

If there are two requests r^0 and r^1, as s^2 is at location 0 at time t_r_i and t_r_i ≤ t_r_i, there must also be two requests r^0 and r^1 with t_r_i ≤ t_r_i and t_r_i ≤ t_r_i, as SG could accept r^0 and r^1 by s^2. We can replace r^0 and r^1 by r^0 and r^1 in R^0 and R^1. If k^* = k^*, the claim thus follows by induction.

If k^* ≠ k^* (k^* - k^* = 1 or k^* - k^* = 2), then R^* is already identical to R^1, and the first k_i requests of R^0 are already identical to the first k_i requests of R^0 by the argument above. If k^* - k^* = 1, there is a request r^0, as s^2 is at location 0 at time t_r_i, and t_r_i ≤ t_r_i, there must also be one request r^0, with t_r_i ≤ t_r_i, as SG could accept r^0 by s^2. We can replace r^0 by r^0 in R^0, making R^0 identical to R^0. If k^* - k^* = 2, there are two requests r^0 and r^0. Note that r^0 and r^0 must be assigned to different servers by OPT as k^* - k^* = 2. Recall that s^1 is at location 0 at time t_r_i, and s^2 is at location 0 at time t_r_i. Hence t_r_i ≥ t_r_i and t_r_i ≥ t_r_i. As s^1 is at location 0 at time t_r_i, and s^2 is at location 0 at time t_r_i, there must also be two requests r^0 and r^0 with t_r_i ≤ t_r_i and t_r_i ≤ t_r_i, as SG could accept r^0 by s^2, and accept r^0 by s^2. We can replace r^0 and r^0 by r^0 and r^0 in R^0, making R^0 identical to R^0. As R^1 is already identical to R^1, R^* is identical to R' because R^* = R^0 ∪ R^1 and R' = R^0 ∪ R^1.

**Theorem 3.** Algorithm SG is 2-competitive for 2S2L if c = 0.
Case 2.3:

Case 2.2:

Case 2:

\[ k > \]

Hence, for all \( \max \{0, \infty\} \), we partition the time horizon into intervals (periods) that can be analyzed independently. Period \( i \), for \( 1 < i < k \), is the interval \( \{\max \{t_{r_{i-1}}, t_{r_{i}}\}, \max \{t_{r_{i}}, t_{r_{i+1}}\}\} \). Period 1 is \( \{0, \max \{t_{r_{0}}, t_{r_{1}}\}\} \), and period \( k \) is \( \{\max \{t_{r_{k-1}}, t_{r_{k}}\}, \infty\} \). (If \( k = 1 \), there is only a single period \( [0, \infty) \).) Set \( t_{r_{i+1}} = \infty \) and \( t_{r_{0}} = 0 \). Let \( R_{i}^{*} \) denote the set of requests accepted by \( \text{OPT} \) that start in period \( i \), for \( 1 \leq i \leq k \). For all \( 1 \leq i \leq k \), if \( \max \{t_{r_{i-1}}, t_{r_{i}}\} \geq \max \{t_{r_{i}}, t_{r_{i+1}}\} \), \( R_{i}^{*} = \emptyset \), and hence \( P_{R_{i}^{*}} = 0 \). Denote \( R_{i}^{*} = \{r_{i}\} \) for \( 1 \leq i \leq k \).

For \( 1 \leq i \leq k \), \( r_{i} \) starts at time \( t_{r_{i}} \) and the first request of \( R_{i}^{*} \) starts during the interval \( \{\max \{t_{r_{i-1}}, t_{r_{i}}\}, \max \{t_{r_{i}}, t_{r_{i+1}}\}\} \) (or the interval \( \{\max \{t_{r_{i-1}}, t_{r_{i}}\}, \infty\} \) if \( i = k \)). Furthermore, \( r_{i} \) is the first acceptable request in \( R \), and so the first request of \( R_{i}^{*} \) cannot start before \( t_{r_{i}} \). Hence, for all \( 1 \leq i \leq k \), the first request in \( R_{i}^{*} \) cannot start before \( t_{r_{i}} \).

We bound the competitive ratio of \( \text{SG} \) by analyzing each period independently. As \( R' = \bigcup_{i} R_{i}^{*} \) and \( R^{*} = \bigcup_{i} R_{i}^{*} \), it is clear that \( P_{R^{*}} / P_{R'} \leq \alpha \) follows if we can show that \( P_{R^{*}} / P_{R'} \leq \alpha \) for all \( 1 \leq i \leq k \). For \( 1 \leq i \leq k \) we distinguish the following cases in order to bound \( P_{R^{*}} / P_{R'} \). As \( R_{i}^{*} = \{r_{i}\} \), \( P_{R_{i}^{*}} = r \) (because \( c = 0 \)). We need to show \( P_{R^{*}} / P_{R_{i}^{*}} \leq 2r \).

**Case 1:** \( k = 1 \). Without loss of generality, suppose \( r_{1} \) is assigned to \( s_{1} \). We claim \( R^{*} \) contains at most one request \( (r_{1}^{*}) \). Assume that \( R^{*} \) contains at least two requests and the second request is \( r_{a} \). As \( s_{2}^{*} \) is at location 0 at time 0, \( r_{a} \) would be acceptable to \( \text{SG} \) by \( s_{2}^{*} \). Hence, there cannot be such a request \( r_{a} \) that starts in period \( [0, \infty) \). As we have shown that \( \text{OPT} \) can accept at most one request \( (r_{1}^{*}) \), we get that \( \frac{P_{R^{*}}}{P_{R_{1}^{*}}} \leq \frac{2}{2} < 2 \).

**Case 2:** \( k > 1 \). For all \( 1 \leq i \leq k \), we claim that \( R_{i}^{*} \) contains at most two requests (each server accepts at most one request). Assume that \( s_{q}^{*} \) \((q \in \{1, 2\})\) accepts at least two requests. Let \( r_{a} \) be the second request (in order of start time) which is assigned to \( s_{q}^{*} \) in \( R_{i}^{*} \). We distinguish three sub-cases. Without loss of generality, suppose \( r_{1}^{*} \) is assigned to \( s_{1}^{*} \).

**Case 2.1:** \( \hat{t}_{r_{i}} > t_{r_{i+1}} \) (Fig. 1.a shows an example). If \( i > 1 \), the period \( i \), which is the period \( \{\max \{t_{r_{i-1}}, t_{r_{i}}\}, \max \{t_{r_{i}}, t_{r_{i+1}}\}\} \) has length less than \( t \). If \( i = 1 \), note that the period \( \{t_{r_{i}}, \max \{t_{r_{i}}, t_{r_{i+1}}\}\} \) has length less than \( t \) and no request of \( R_{i}^{*} \) can start before \( t_{r_{i}} \) during period 1, \( [0, \max \{t_{r_{i}}, t_{r_{i+1}}\}] \). Therefore, each server can accept at most one request that starts during period \( i \), and hence \( R_{i}^{*} \) contains at most two requests.

**Case 2.2:** \( t_{r_{i}} < t_{r_{i+1}} \) and \( t_{r_{i+1}} > t_{r_{i}} \) (Fig. 1.b shows an example). Observe that \( s_{1}^{*} \) is at \( p_{r_{i}}^{*} \) at \( t_{r_{i}} \). As the drop-off time of \( r_{i-1}^{*} \) is later than the pick-up time of \( r_{i}^{*} \), \( r_{i-1}^{*} \) must be assigned to \( s_{2}^{*} \) and we have that \( s_{2}^{*} \) is at \( p_{r_{i-1}}^{*} \) at \( t_{r_{i-1}}^{*} \). As the first request in \( R_{i}^{*} \) does not start before \( \hat{t}_{r_{i-1}}^{*} \), we have \( t_{r_{a}} \geq \hat{t}_{r_{i-1}}^{*} + t \). This means that \( r_{a} \) would be acceptable to \( s_{2}^{*} \). Therefore, \( \text{SG} \) accepts either \( r_{a} \) or another request starting before \( t_{r_{a}} \), and that request becomes \( r_{i+1}^{*} \). Hence, there cannot be such a request \( r_{a} \) that starts in period \( i \).

**Case 2.3:** \( t_{r_{i}} \leq t_{r_{i+1}} \) and \( t_{r_{i-1}} \leq t_{r_{i}} \) (Fig. 1.c shows an example). As the drop-off time of \( r_{i-1}^{*} \) is earlier than the pick-up time of \( r_{i}^{*} \), \( s_{2}^{*} \) is at the drop-off location of the request \( r_{i} \) (where \( r_{i} \) denotes the latest request that starts at or before \( t_{r_{i}} \) and is assigned to \( s_{2}^{*} \).
if there is no such request, let \( r_1 \) be a dummy request with \( i_{r_1} = 0 \) and \( \hat{p}_{r_1} = 0 \) at \( i_{r_1} \) and \( t_{r_1} \leq t_{i_{r_1}-1} \leq t_{r_1} \). Observe that \( s'_2 \) does not accept any request which starts during period \([t_{r_1}', t_{i_{r_1}}')\). \( s'_2 \) does not start to move before \( t_{r_1}' \) for serving the next request, and hence \( s'_2 \) is at \( \hat{p}_{r_1} \) (0 or 1) at \( t_{r_1}' \). As the first request in \( R'_i \) does not start before \( t_{r_1}' \), we have \( t_{r_0} \geq t_{r_1}' + t \). This means that \( r_0 \) would be acceptable to \( s'_2 \). Therefore, SG accepts either \( r_0 \) or another request starting before \( t_{r_1}' \), and that request becomes \( r'_{i+1} \). Hence, there cannot be such a request \( r_0 \) that starts in period \( i \).

As we have shown that \( R'_i \) contains at most two requests, we get that \( P_{R'_i} \leq 2r \). Since \( P_{R'_i} = r \), we have \( P_{R'_i} / P_{R'_i} \leq 2r/r = 2 \). The theorem follows.

\( \blacklozenge \)

\textbf{Lemma 4.} When \( 0 < c < r \), for all \( 1 < i \leq k \), one server of \( SG \) is at \( p_{r_1} \) at \( t_{r_1}' \) and the other server of \( SG \) is at 0 or 1 at \( \max\{t_{r_1}-1, t_{r_1}'\} \).

\textbf{Proof.} For \( 1 < i \leq k \), without loss of generality, suppose \( r'_1 \) is assigned to \( s'_1 \). Observe that \( s'_1 \) is at \( p_{r_1} \) at \( t_{r_1}' \).

- If \( t_{r_1}-1 > t_{r_1}' \), then \( s'_2 \) at 0 or 1 at \( t_{r_1}-1 \) (= max\{\( t_{r_1}-1, t_{r_1}' \)\}).

- If \( t_{r_1} \leq t_{r_1}' \), then \( s'_2 \) at 0 or 1 at the drop-off time \( t' \) (\( t'_r \leq t_{r_1}-1 \)) of the latest request which is assigned to \( s'_2 \) and starts at or before \( t_{r_1}' \). (If no such request exists, \( s'_2 \) is at 0 at \( t' = 0 \).)

Suppose \( r_f \) is the first request that starts at or after \( t_{r_1}' \) and is served by \( s'_2 \). If \( r_f \) does not exist, then \( s'_2 \) does not exist after \( t_{r_1}' \), and \( s'_2 \) at 0 or 1 at \( \max\{t_{r_1}-1, t_{r_1}' \} \) (\( t_{r_1}-1 \leq t_{r_1}' \)).

If \( r_f \) exists and \( r_f \) is accepted with cost, then \( t_{r_f} - t_{r_1}' \geq t \) (\( t_{r_f} \leq t_{r_1}' \)) because \( SG \) accepts a request \( r_f \) with cost only if the condition \( t_{r_f} - t_{r_1}' \geq t \) is satisfied. That means \( s'_2 \) starts an empty move at or after \( t_{r_1}' \). If \( r_f \) exists and \( r_f \) is accepted without cost, then \( s'_2 \) starts to move at or after \( t_{r_1}' \). Therefore \( s'_2 \) at 0 or 1 at \( t_{r_1}' \) (= max\{\( t_{r_1}-1, t_{r_1}' \)\}).

\( \blacklozenge \)

\textbf{Lemma 5.} When \( 0 < c < r \), for all \( 1 \leq i \leq k \), if \( r'_1 \) is accepted with cost, then one server of \( SG \) is at \( p_{r_1} \) at \( t_{r_1}' \) and the other server of \( SG \) is at \( p_{r_1} \) at \( t_{r_1}' \).

\textbf{Proof.} For \( 1 \leq i \leq k \), without loss of generality, suppose \( r'_1 \) is assigned to \( s'_1 \). Observe that \( s'_1 \) is at \( p_{r_1} \) at \( t_{r_1}' \).

- If \( i = 1 \), then \( p_{r_1} = 1 \) and \( s'_2 \) is at \( p_{r_1} \) (location 0) at time 0. Suppose \( r_0 \) is the first request which is assigned to \( s'_2 \). If \( p_{r_0} = 0 \), then \( s'_2 \) starts to move at \( t_{r_0} \) (\( \geq t_{r_1}' \)), and hence \( s'_2 \) at 0 at \( t_{r_0} \). If \( p_{r_0} = 1 \), then \( t_{r_0} \geq t_{r_1}' + t \) because by definition \( SG \) accepts a request \( r_f \) with cost only if the condition \( t_{r_f} - t_{r_1}' \geq t \) is satisfied. Observe that \( s'_2 \) starts to move at \( t_{r_1}' \) and hence \( s'_2 \) at 0 at \( t_{r_1}' \).

- If \( 1 < i \leq k \), \( s'_2 \) at 0 or 1 at \( \max\{t_{r_1}-1, t_{r_1}' \} \) according to Lemma 4. As \( r'_1 \) is accepted with cost, \( t_{r_1}' - t_{r_1}-1 \geq t \) because \( SG \) accepts a request \( r_f \) with cost only if the condition \( t_{r_f} - t_{r_1}' \geq t \) is satisfied, and hence \( \max\{t_{r_1}-1, t_{r_1}' \} = t_{r_1}' \). We prove this lemma by contradiction. Assume that \( s'_2 \) is at \( p_{r_1} \) at \( t_{r_1}' \). Note that \( r'_1 \) is acceptable to \( SG \) by \( s'_2 \) without cost, and hence \( SG \) assigns \( r'_1 \) to \( s'_2 \) because \( SG \) always assigns a request to the most economical server (Recall Algorithm 1). This contradicts our initial assumption that \( r'_1 \) is assigned to \( s'_1 \).

\( \blacklozenge \)

\textbf{Lemma 6.} When \( 0 < c < r \), for all \( 1 < i < k \), if \( r'_1 \) is accepted without cost and \( r'_{i+1} \) is accepted with cost, then one server of \( SG \) is at \( p_{r_1} \) at \( t_{r_1}' \), and the other server of \( SG \) is at \( p_{r_1} \) at \( \max\{t_{r_1}-1, t_{r_1}' \} \).

\textbf{Proof.} For \( 1 < i < k \), without loss of generality, suppose \( r'_1 \) is assigned to \( s'_1 \). Observe that \( s'_1 \) is at \( p_{r_1} \) at \( t_{r_1}' \). According to Lemma 4, \( s'_2 \) at 0 or 1 at \( \max\{t_{r_1}-1, t_{r_1}' \} \). As \( r'_{i+1} \) is accepted with cost, \( t_{r_1}-1 \geq t \) because \( SG \) accepts a request \( r_j \) with cost only if the
condition \( t_{r_j} - t_{r_j}^* \geq t \) is satisfied. Note that \( p_{r_{i+1}} = p_{r_i^*} \), otherwise \( r_i^* + 1 \) is acceptable to SG by \( s_i^* \) without cost.

We prove this lemma by contradiction. Assume that \( s_i^* \) is at \( p_{r_i^*} \) at \( \max\{t_{r_i^*}, t_{r_{i+1}}\} \). Suppose \( r_j = r_i^* + 1 \). Observe that \( p_{r_j} = p_{r_i^*} \) and \( t_{r_j} \geq t_{r_i^*} + t \geq \max\{t_{r_i^*}, t_{r_{i+1}}\} \). From this it follows that \( r_j \) is acceptable to SG by \( s_i^* \) without cost, and hence SG assigns \( r_j \) to \( s_i^* \) because SG always assigns a request to the most economical server (Recall Algorithm 1). This contradicts the statement that \( r_i^* + 1 \) is accepted with cost. ▶

**Lemma 7.** When \( 0 < c < r \), for all \( 1 < i < k \), if \( r_i^* \) is accepted without cost and \( r_i^* + 1 \) is accepted with cost, then \( r_i^* - 1 \) must be accepted without cost.

**Proof.** For \( 1 < i < k \), without loss of generality, suppose \( r_i^* - 1 \) is assigned to \( s_i^* \). Observe that \( s_i^* \) is at \( p_{r_i^* - 1} \) at \( t_{r_i^* - 1} \) (and is at \( p_{r_i^* - 1} \) at \( t_{r_i^* - 1} \)). We prove this lemma by contradiction. Assume \( r_i^* - 1 \) is accepted with cost. According to Lemma 5, \( s_i^* \) is at \( p_{r_i^* - 1} \) at \( t_{r_i^* - 1} \). As \( r_i^* \) is accepted without cost, the pick-up location of \( r_i^* \) is \( p_{r_i^* - 1} \). Suppose \( r_j = r_i^* + 1 \). Observe that \( t_{r_j} \geq t - t + t \) (because \( r_j \) is accepted with cost) and \( p_{r_j} = p_{r_i^*} = p_{r_i^* - 1} \) (otherwise, the server that has served \( r_j \) could accept \( r_j \) without cost).

If \( r_i^* \) is assigned to \( s_i^* \), then \( s_i^* \) does not accept any request which starts in period \( \max\{t_{r_i^* - 1}, t_{r_i^* - 1}\}, t_{r_j}\) and hence \( s_i^* \) is at \( p_{r_i^* - 1} \) in period \( \max\{t_{r_i^* - 1}, t_{r_i^* - 1}\}, t_{r_j} \). If \( r_i^* \) is assigned to \( s_i^* \), then \( s_i^* \) does not accept any request which starts in period \( \max\{t_{r_i^* - 1}, t_{r_i^* - 1}\}) \), and hence \( s_i^* \) is at \( p_{r_i^* - 1} \) in period \( \max\{t_{r_i^* - 1}, t_{r_i^* - 1}\}, t_{r_j} - t \). As \( r_j \) is released and \( t_{r_j} = t_{r_j} \), then \( s_i^* \) is at \( p_{r_i^* - 1} \) and does not plan to move, hence \( r_j \) is acceptable to SG by \( s_i^* \) without cost. From this it follows that \( r_j \) will be accepted by SG without cost because SG always assigns a request to the most economical server. This contradicts the statement that \( r_i^* + 1 \) is accepted with cost. ▶

For simplicity of the analysis, we suppose that the OPT servers make an empty movement only if they do so in order to serve a request \( r_i \) such that the pick-up location of \( r_i \) is the pick-up location of the previous request which is assigned to the same server, or the pick-up location is 1 if \( r_i \) is the first request which is assigned to a server \( s_i^* (q \in \{1,2\}) \), and we suppose that for all such requests \( r_i (r_i \in R^*) \), the OPT server serving \( r_i \) makes an empty movement between \( t_{r_i} - t \) and \( t_{r_i} \). This simplification does not affect the validity of \( R^* \), and does not decrease \( P_{R^*} \).

**Theorem 8.** Algorithm SG is 2-competitive for 2S2L if \( 0 < c < r \).

**Proof.** Assume that SG accepts \( k \) (\( k = |R'| \)) requests. We partition the time horizon \([0, \infty)\) into \( k' \) (\( 1 \leq k' \leq k \)) intervals (periods) that can be analyzed independently. We partition the time horizon based on Algorithm 2, in such a way that all requests in the first period are accepted with cost (if \( r_i^* \) is accepted with cost), and exactly one request of each period (except the first period if \( r_i^* \) is accepted with cost), the first request of each period, is accepted without cost. Denote the request number in \( R' \) (in order of starting time) of the first request of period \( j (1 \leq k' \leq k) \) by \( b_j \). For \( 1 < j < k' \), SG j period is \([t_{r_{j-1}}, t_{r_{j+1}}]\). SG 1 period is \([0, t_{r_0}) \) and SG \( k' \) period is \([t_{r_{k'}} \infty) \) (If \( k' = 1 \), there is only a single period \([0, \infty) \). We set \( l_{k'+1} = k + 1, t_0 = 0 \) and \( t_{k'+1} = \infty \). Let \( R_j' \) (\( 1 \leq j \leq k' \)) denote the set of requests accepted by SG that start in SG \( j \) period. For \( 1 < j < k' \), if \( t_{r_{j-1}} = t_{r_{j+1}} \), let \( R_{j-1}' = \{t_{r_{j-1}}\} \) and \( R_j' = \{t_{r_{j-1}} \} \), \( \ldots \} \). Note that there are exactly \( l_{j+1} - l_j (l_{j+1} - l_j \geq 1) \) requests in \( R_j' \) (\( 1 \leq j \leq k' \)), and \( R_j' = \{t_{r_{j-1}} \} \), \( \ldots \} \).

For all \( 1 < j < k' \), we have the following property: if \( |R_j'| = 1 \), then \( r_i^* \) is accepted without cost; if \( |R_j'| > 1 \), then \( r_i^* \) is accepted without cost, the remaining requests in \( R_j' \)
Car-Sharing between Two Locations: Online Scheduling with Two Servers

Algorithm 2 Partition Rule (PR).

Initialization: \( k = |R'|, k' = 1, j = 1, t_j = j \) for all \( 1 \leq j \leq k \).

For \( i = 2 \) to \( k \)

if \( r_i' \) is accepted without cost then

\[ j = j + 1, t_j = i; \]

else \( k' = j, t_{k' + 1} = k + 1 \).

Figure 2 An example of \( t_j \).

are accepted with cost. For \( j = 1 \), if \( r_1'(= r_1^*) \) is accepted with cost, all requests in \( R'_1 \) are accepted with cost; if \( r_1' \) is accepted without cost, then except \( r_1' \) all requests in \( R'_1 \) are accepted with cost.

Definition 9. For \( 1 < j \leq k' \), \( t_j \) is defined as follows: \( t_j = t_{r_j'} \) if \( r_{j - 1}' \) is accepted with cost, \( r_j' \) is accepted without cost, \( t_{r_{j - 1}'} > t_{r_j'} \) and \( p_{r_{j - 1}'} = p_{r_j'} \) (Fig. 2 shows an example). Otherwise, \( t_j = \max\{t_{r_{j - 1}'} , t_{r_j'} \} \), \( t_{k' + 1} = t_{r_{k' + 1}'} = \infty \).

For \( 1 < j \leq k' \), \( t_{j + 1} = t_{r_{j + 1}'} \) or \( t_{j + 1} = \max\{t_{r_{j + 1}'} , t_{r_{j + 1}'} \} \). Because \( t_{r_{j - 1}'} \leq t_{r_{j + 1}'} \) and \( t_{r_{j - 1}'} \leq t_{r_{j + 1}'} \), if \( r_{j - 1}' \) and \( r_j' \) are assigned to different servers, then \( t_{r_{j - 1}'} \leq t_{r_{j + 1}'} \); and if \( r_j' \) is assigned to different servers, then \( t_{j + 1} = \max\{t_{r_{j - 1}'} , t_{r_{j + 1}'} \} \). As \( t_j \leq \max\{t_{r_{j - 1}'} , t_{r_{j + 1}'} \} \), we have that \( t_j \leq t_{j + 1} \) always holds. For \( 1 < j \leq k' \), OPT period \( j \) is defined as \( [t_j, t_{j + 1}) \). OPT period 1 is defined as \( [0, t_2) \) (If \( k' = 1 \), there is only a single period \( [0, \infty) \)). Let \( R_j^* \) denote the set of requests accepted by OPT that start in OPT period \( j \), and \( R_j^* = \emptyset \) if \( t_j = t_{j + 1} \).

For all \( 1 < j \leq k' \), \( r_j' \) starts at time \( t_{r_j'} \) and the first request of \( R_j^* \) starts during the interval \([t_j, t_{j + 1})\) where \( t_j = t_{r_j'} \) or \( t_j = \max\{t_{r_{j - 1}'} , t_{r_{j + 1}'} \} \) (recall the definition of \( t_j \)). Furthermore, \( r_1' \) is the first acceptable request in \( R \), and so the first request of \( R_1^* \) cannot start before \( r_1' \). Hence, for all \( 1 \leq j \leq k' \), the first request in \( R_j^* \) cannot start before \( t_{r_j'} \).

We bound the competitive ratio of SG by analyzing each period independently. As \( R' = \bigcup_{j=1}^{k'} R_j' \) and \( R^* = \bigcup_{j=1}^{k'} R_j^* \), it is clear that \( P_{R_1^*} / P_{R_1'} \leq \alpha \) follows if we can show that \( P_{R_i^*} / P_{R_i'} \leq \alpha \) for all \( 1 \leq j \leq k' \). For \( 1 \leq j \leq k' \), if \( t_j = t_{j + 1} \), then \( R_j^* = \emptyset \) and hence \( P_{R_j^*} = 0 \). Otherwise, for \( 1 \leq j \leq k' \) we distinguish the following cases in order to bound \( P_{R_j^*} / P_{R_i'} \).

Case 1: \( j = 1 \). The first request of SG period 1 is \( r_1' \). Without loss of generality, suppose \( r_1' \) is assigned to \( s_1' \).

Case 1.1: \( r_1' \) is accepted with cost. Note that all requests in \( R_1' \) are accepted with cost and \( P_{R_1} = (l_2 - l_1)(r - c) \) (if \( k' = 1 \), then \( P_{R_1} = k(r - c) \)). Observe that \( p_{r_i'} = 1 \) (\( 1 \leq i \leq l_2 \)) and all requests in \( R_1' \) are assigned to \( s_1' \) by the definition of Algorithm 1. As \( r_{l_2 - 1}' \) is accepted with cost, one server is at \( p_{r_{l_2 - 1}'} + 1 \) at \( t_{r_{l_2 - 1}'} \) (and this server is at \( p_{r_{l_2 - 1}'} + 1 \) at \( t_{r_{l_2 - 1}'} \)).
and the other server is at $\hat{t}_{r_{1'2}}$ at $t_{r_{1'2}}$ (by Lemma 5). As $r_{1'2}$ is accepted without cost, we have $p_{r_{1'2}} = p_{r_{1'2}}$. If $k' = 1$, $t_2 = \infty$. If $k' > 1$, then $t_2 = t_{r_{1'2}}$; if $t_{r_{1'2}} = t_{r_{1'2}}$, $t_2 = t_{r_{1'2}}$ because $p_{r_{1'2}} = p_{r_{1'2}}$, $r_{1'2}$ is accepted with cost and $r_{1'2}$ is accepted without cost; if $t_{r_{1'2}} \leq t_{r_{1'2}}$, $t_2 = \max\{t_{r_{1'2}}, t_{r_{1'2}}\} = t_{r_{1'2}}$. As $s_{2'}$ does not accept any request which starts before $t_{r_{1'2}}$, $s_{2'}$ would not accept any request with cost which starts in period $[t_{r_{1'2}}, t_{r_{1'2}}]$

(Recall that Algorithm 1 accepts a request $r_j$ with cost only if $t_{r_j} - t_{r_j} \geq t$ is satisfied.), $s_{2'}$ is at 0 in period $[0, t_{r_{1'2}}]$. We claim that $R_{j'}$ only contains requests which start at 1. Otherwise, the request is acceptable to SG by $s_{2'}$ without cost. Assume that $R_{j'}$ contains a request $r_o$ which start at location 0. As $t_{r_o} \leq t = t_{r_{1'2}}$, $r_o$ is acceptable to SG by $s_{2'}$ without cost. Therefore, SG accepts either $r_o$ or another request starting before $t_{r_o}$, and that request becomes $r_{j'}$. Hence, there cannot be such a request $r_o$ in $R_{j'}$.

Note that each server of OPT does not accept any request which starts in period $[0, t_{r_{1'2}}]$.

For all $l_1 \leq i \leq l_2 - 2$, we claim that each server of OPT can accept at most one request which starts during period $[t_{r_{1'2}}, t_{r_{1'2}+1}]$ ($l_1 \leq i \leq l_2 - 2$), or period $[t_{r_{1'2}+1}, t^*]$ (if $k' > 1$, $t^* = t_{r_{1'2}}$; if $k' = 1, t^* = t_{r_{1'2}} + 2t$). Assume that $s_{*}^q (q \in \{1, 2\})$ accepts at least two requests in one of those periods. Let $r_o$ be the second request (in order of start time) which is assigned to $s_{*}^q$ and starts during period $[t_{r_{1'2}}, t_{r_{1'2}+1}]$ ($l_1 \leq i \leq l_2 - 2$) or period $[t_{r_{1'2}+1}, t_{r_{1'2}+1}]$. As the request does not start before $t_{r_{1'2}} (l_1 \leq i \leq l_2 - 1)$, we have $t_{r_o} \geq t_{r_{1'2}} + 2t$. $r_o$ is acceptable to SG with cost. Therefore, SG accepts either $r_o$ or another request starting before $t_{r_o}$, and that request becomes $r_{j'+1} (l_1 \leq i < l_2)$. Hence, there cannot be such a request $r_o$ that starts during period $[t_{r_{1'2}, t_{r_{1'2}+1}}]$ ($l_1 \leq i \leq l_2 - 2$) or period $[t_{r_{1'2}+1}, t^*]$. Therefore OPT can accept at most $2(l_2 - l_1) (= 2(l_2 - 2 - l_1 + 1 + 1))$ requests that start during period $[t_{r_{1'2}}, t^*]$.

When $k' = 1$, we claim that OPT does not accept any request which starts in period $[t^*, \infty)$. Without loss of generality we assume that OPT accepts at least one request. Let $r_o$ be the request in $R_{j'}$ that starts during period $[t^*, \infty)$. As $t_{r_o} \geq t_{r_{1'2}} + 2t$. $r_o$ is acceptable to SG. Therefore, SG accepts either $r_o$ or another request starting before $t_{r_o}$, and that request becomes $r_{j'+1}$. Hence, there cannot be such a request $r_o$ that starts in period $[t^*, \infty)$.

As we have shown that $R_{j'}$ contains at most $2(l_2 - l_1)$ requests and the pick-up locations of them are the same (location 1), we get that $P_{R_{j'}} \leq 2(l_2 - l_1)(r - c)$. Since $P_{R_{j'}} = (l_2 - l_1)(r - c)$, we have $P_{R_{j'}}/P_{R_{j'}} \leq 2(l_2 - l_1)(r - c)/((l_2 - l_1)(r - c)) = 2$.

**Case 1.2**: $r_{1'2}$ is accepted without cost. If $k = 1$, then $k' = 1$, $s_{2'}$ is at 0 in period $[0, \infty)$. If $k > 1$, we claim that $r_{1'2}$ is also accepted without cost. Assume that $r_{1'2}$ is accepted with cost, we have $t_{r_{1'2}} - t_{r_{1'2}} > t$ because Algorithm 1 accepts a request $r_{1'2}$ with cost only if $t_{r_{1'2}} - t_{r_{1'2}} \geq t$ is satisfied. If $p_{r_{1'2}} = 0$, $r_{1'2}$ is acceptable to SG by $s_{1'2}$ without cost; if $p_{r_{1'2}} = 1$, $r_{1'2}$ is acceptable to SG by $s_{1'2}$ without cost. Therefore $s_{2'}$ must be accepted by SG without cost because by definition (see Algorithm 1) SG always assigns a request to the most economical server. This contradicts the assumption that $r_{1'2}$ is accepted with cost. Observe that $t_2 = \max\{t_{r_{1'2}}, t_{r_{1'2}}\}$ (Recall from the definition of $t_2$ that $t_2 = t_{r_{1'2}}$ only if $r_{1'2}$ is accepted with cost), $|R_{j'}| = 1$ and hence $P_{R_{j'}} = r$. As $s_{2'}$ does not accept any request which starts before $t_{r_{1'2}}$, $s_{2'}$ would not accept any request with cost which starts in period $[t_{r_{1'2}}, t_{r_{1'2}}]$ (Recall that Algorithm 1 accepts a request $r_{1'2}$ with cost only if $t_{r_{1'2}} - t_{r_{1'2}} \geq t$ is satisfied.). $s_{2'}$ is at 0 in period $[0, t_{r_{1'2}}]$.

We claim that $R_{j'}$ contains at most two requests (each server serves at most one request). Assume that $s_{*}^q (q \in \{1, 2\})$ accepts at least two requests. Let $r_o$ be the second request (in order of start time) which is assigned to $s_{*}^q$ in $R_{j'}$. As the first request in $R_{j'}$ does not
start before \( t_{r_1} \), we have \( t_{r_o} \geq t_{r_1} + t \). If \( p_{r_o} = \tilde{p}_{r_1} \), \( r_o \) is acceptable to SG by \( s'_1 \) without cost; if \( p_{r_o} = \tilde{p}_{r_1} \), \( r_o \) is acceptable to SG by \( s'_2 \) without cost. Hence, there cannot be such a request in \( R'_1 \). Since \( P_{R'_1} = r \), we have \( P_{R'_1}/P_{R'_1} \leq 2r/r = 2 \).

Case 2: \( j > 1 \) (1 \(< j \leq k' \)). The first request of SG period \( j \) is \( r'_j \). Without loss of generality, suppose \( r'_j \) is assigned to \( s'_1 \). We distinguish the following cases based on \( |R'_j| \).

Case 2.1: \( |R'_j| = 1 \). Note that \( r'_j \) is accepted without cost. We distinguish two sub-cases.

1. \( t_{r'_j} > t_{r'_j+1} \). Because \( r'_j = (r'_j+1) \) is accepted without cost, \( t_j = \max \{ t_{r'_j} \}, \) \( t'_{r'_j+1} \) \( = \).

2. \( t_{r'_j} \leq t_{r'_j+1} \). \( (r'_j+1 = \infty \) if \( j = k' \)). Note that \( t_{r'_j+1} = t_{r'_j+1} \). There are two sub-cases based on the position of \( s'_2 \) at \( \max \{ t_{r'_j-1}, t_{r'_j} \} \) (recall that by Lemma 4, \( s'_2 \) is at \( p_{r'_j} \) or \( \tilde{p}_{r'_j} \) at time \( \max \{ t_{r'_j-1}, t_{r'_j} \} \).

The first sub-case is that \( s'_2 \) is at \( p_{r'_j} \) at \( \max \{ t_{r'_j-1}, t_{r'_j} \} \). We claim that \( R'_j \) contains at most two requests (each server serves at most one request). Assume that \( s'_2 \) \( q \in \{1, 2\} \) accepts at least two requests. Let \( r_o \) be the second request (in order of start time) which is assigned to \( s'_q \) in \( R'_j \). As the requests in \( R'_j \) do not start before \( t_{r'_j} \), we have \( t_{r_o} \geq t_{r'_j} + t \).

If \( p_{r_o} = \tilde{p}_{r'_j} \), \( r_o \) is acceptable to SG by \( s'_1 \) without cost; if \( p_{r_o} = \tilde{p}_{r'_j} \), \( r_o \) is acceptable to SG by \( s'_2 \) without cost. Therefore, SG accepts either \( r_o \) or another request starting before \( t_{r_o} \), and that request becomes \( r'_{j+1} \). Hence, there cannot be such a request \( r_o \) that starts in \( OPT \) period \( j \).

The second sub-case is that \( s'_2 \) is at \( \tilde{p}_{r'_j} \) at \( \max \{ t_{r'_j-1}, t_{r'_j} \} \). (Recall from the definition of \( t_j \) that \( t_j = t_{r'_j} \) if \( t_{r'_j} > t_{r'_j} \) and \( \tilde{p}_{r'_j} \) are satisfied. From this it follows that \( r'_{j-1} \) must be assigned to \( s'_2 \), that means \( s'_2 \) is at \( \tilde{p}_{r'_j} \) at \( t_{r'_j-1} \). \( (\in \max \{ t_{r'_j-1}, t_{r'_j} \}) \)). This contradicts the initial assumption that \( s'_2 \) is at \( \tilde{p}_{r'_j} \) at \( \max \{ t_{r'_j-1}, t_{r'_j} \} \)). We claim that \( R'_j \) contains at most two requests (each server serves at most one request) and the pick-up locations of these two requests are \( p_{r'_j} \). Assume that \( R'_j \) contains a request \( r_i \) which starts at \( \tilde{p}_{r'_j} \). As the requests in \( R'_j \) cannot start before \( t_j \) \( (t_j = \max \{ t_{r'_j-1}, t_{r'_j} \}) \), \( r_i \) is acceptable to \( s'_2 \) without cost as \( s'_2 \) is at \( \tilde{p}_{r'_j} \) at \( \max \{ t_{r'_j-1}, t_{r'_j} \} \). Hence, there cannot be such a request \( r_i \) that starts in \( OPT \) period \( j \). Next assume that \( s'_q \) \( q \in \{1, 2\} \) accepts at least two requests. Let \( r_i \) and \( r_o \) be the first and second request (in order of start time) which is assigned to \( s'_q \) in \( R'_j \). As the requests in \( R'_j \) do not start before \( t_{r'_j} \) and the pick-up location of \( r_i \) and \( r_o \) both are \( \tilde{p}_{r'_j} \), we have \( t_{r_o} \geq t_{r'_j} + 2t \). If \( p_{r_o} = \tilde{p}_{r'_j} \), \( r_o \) is acceptable to SG by \( s'_1 \) without cost; if \( p_{r_o} = \tilde{p}_{r'_j} \), \( r_o \) is acceptable to SG by \( s'_2 \) with cost. Therefore, SG accepts either \( r_o \) or another request starting before \( t_{r_o} \), and that request becomes \( r'_{j+1} \) (if it is accepted without cost) or gets added to \( R'_j \) (if it is accepted with cost). Hence, there cannot be such a request \( r_o \) that starts in \( OPT \) period \( j \).

As we have shown that \( R'_j \) contains at most two requests, we get that \( P_{R'_j} \leq 2r \). Since \( P_{R'_j} = r \), we have \( P_{R'_j}/P_{R'_j} \leq 2r/r = 2 \).

Case 2.2: \( |R'_j| > 1 \). Note that \( r'_j \) is accepted without cost and \( r'_{j+1} \) is accepted with cost. We have that \( s'_2 \) is at \( \tilde{p}_{r'_j} \) at \( \max \{ t_{r'_j-1}, t_{r'_j} \} \) by Lemma 6, and that \( r'_{j+1} \) is accepted without
We have studied an on-line problem with two servers and two locations that is motivated by applications such as car sharing and taxi dispatching. The upper bounds for the 2S2L problem are all achieved by the smart greedy algorithm. A number of directions for future work are all achieved by the smart greedy algorithm.

4 Conclusion

We have studied an on-line problem with two servers and two locations that is motivated by applications such as car sharing and taxi dispatching. The upper bounds for the 2S2L problem are all achieved by the smart greedy algorithm. A number of directions for future work are all achieved by the smart greedy algorithm.

MFCS 2018
work arise from this work. If there are $k$ servers, does a kind of greedy algorithm still work well? Furthermore, it would be interesting to extend our results to the case of more than two locations. It would be interesting to determine how the constraints on the servers affect the competitive ratio for the general car-sharing problem with $k$ servers and $m$ locations.

References


