Car-Sharing between Two Locations: Online Scheduling with Two Servers

Kelin Luo[1]
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 → Design and analysis of algorithms → 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. 2016M592811), and the China Scholarship Council (Grant No. 201706280058).

43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018).
Editors: Igor Potapov, Paul Spirakis, and James Worrell; Article No. 50; pp. 50:1–50:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

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 $0 \le c < r$, the lower bound is 2, and if $c \ge r$, 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 0 and 1) and two servers (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 $r_i = (\tilde{t}_{r_i}, t_{r_i}, p_{r_i})$ and is specified by the *booking time* or *release time* $\tilde{t}_{r_i}$, 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 $t_{r_i} - \tilde{t}_{r_i}$ is equal to a fixed value a for all requests $r_i \in R$, and we assume that $a \ge t$ 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 $\dot{p}_{r_i} = 1 - p_{r_i}$, the *drop-off location* of the request, at time $\dot{t}_{r_i} = t_{r_i} + t$, 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 time 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, $0 \le c \le r$, 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 $r'_{l,j}$. Suppose $r'_{l,j}$ ($j \in \{1, 2\}$) is r'_i. We say that request r'_i is accepted *without cost* if $l = 1$ and $p_{r'_{l,j}} = 0$ or if $l > 1$ and $p_{r'_{l,j}} = \dot{p}_{r'_{l-1,j}}$. 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 $\tilde{t}_{r_i}$. For any request sequence R, let P_{R^A} denote the objective value produced by an on-line algorithm A, and P_{R^*} 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 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. A value β 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 β 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.

▶ **Theorem 1.** *For $0 \leq c < r$, no deterministic on-line algorithm for 2S2L can achieve competitive ratio smaller than* 2.

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

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 $P_{R^*}/P_{R'} \geq 2$.

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 $P_{R^*}/P_{R'} \geq 2$.

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 $P_{R^*}/P_{R'} = \infty$. ◀

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. $R_i' \cup \{r_{i,s_j'}\}$ denotes the union of R_i' and $\{r_{i,s_j'}\}$, where $r_{i,s_j'}$ represents the request r_i assigned to server s_j', $j \in \{1, 2\}$, without conflict.
- $r_{i,j}^n$ denotes the latest request which was assigned to s_j', $j \in \{1, 2\}$, before r_i is released. (If there is no such request, take $r_{i,j}^n$ to be a dummy request with drop-off location 0 and drop-off time 0.)
- r_i is *acceptable* if and only if $\exists j \in \{1, 2\} : t_{r_i} - \dot{t}_{r_{i,j}^n} \geq t$ if $p_{r_i} = p_{r_{i,j}^n}$, and $t_{r_i} - \dot{t}_{r_{i,j}^n} \geq 0$ if $p_{r_i} \neq p_{r_{i,j}^n}$.
- r_i^n is the latest request that was accepted before r_i is released. Note that $r_i^n = r_{i,j}^n$ with $j = \arg\max\{t_{r_{i,j}^n} \mid j = 1, 2\}$. Note that $\dot{t}_{r_1^n} = 0$.

- If an accepted request is acceptable by both servers, it is assigned to the *most economical* server, which is the server s'_j with $j = \arg\max\{P_{R'_i \cup \{r_{i,s'_j}\}} \mid j = 1, 2\}$. If $P_{R'_i \cup \{r_{i,s'_1}\}} = P_{R'_i \cup \{r_{i,s'_2}\}}$, s'_j is chosen as the server which has accepted r_i^n (or arbitrarily in case r_i^n does not exist).

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_i and assign it to the most economical server s'_j if r_i is acceptable and $P_{R'_i \cup \{r_{i,s'_j}\}} - P_{R'_i} = r$ ($j \in \{1, 2\}$), or if r_i is acceptable, $P_{R'_i \cup \{r_{i,s'_j}\}} - P_{R'_i} > 0$ ($j \in \{1, 2\}$) and $t_{r_i} - \dot{t}_{r_i^n} \geq t$;

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 $t_{r_i} \leq t_{r_{i+1}}$ for $1 \leq i < n$ because $t_{r_i} - \tilde{t}_{r_i} = a$ 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 $R'^0 = \{r'^0_1, r'^0_2, \ldots r'^0_{k_0}\}$ and the requests accepted by SG which start at location 1 by $R'^1 = \{r'^1_1, r'^1_2, \ldots r'^1_{k_1}\}$. Denote the requests accepted by OPT which start at location 0 by $R^{*0} = \{r^{*0}_1, r^{*0}_2, \ldots r^{*0}_{k^*_0}\}$ and the requests accepted by OPT which start at location 1 by $R^{*1} = \{r^{*1}_1, r^{*1}_2, \ldots r^{*1}_{k^*_1}\}$. Note that $R'^0 \bigcup R'^1 = R'$ and $R^{*0} \bigcup R^{*1} = R^*$.

▶ **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^{*0}_j and r^{*1}_j are both assigned to the same server for $j < i$ and r^{*0}_i and r^{*1}_i are assigned to different servers (without loss of generality, suppose r^{*0}_i is assigned to s^*_1 and r^{*1}_i is assigned to s^*_2), we reassign r^{*1}_i to server s^*_1, reassign all requests in $R^* \backslash (\{r^{*0}_1, r^{*0}_2, \ldots, r^{*0}_{i+1}\} \bigcup \{r^{*1}_1, r^{*1}_2, \ldots, r^{*1}_i\})$ that are assigned to s^*_1 (denote the set of these requests by $\Re_1$) to server s^*_2, and reassign all requests in $R^* \backslash (\{r^{*0}_1, r^{*0}_2, \ldots, r^{*0}_{i+1}\} \bigcup \{r^{*1}_1, r^{*1}_2, \ldots, r^{*1}_i\})$ that are assigned to s^*_2 (denote them by $\Re_2$) 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 $\dot{t}_{r'^0_i} \leq t_{r'^1_i}$ (for all $i \leq k'_1$) and $\dot{t}_{r^{*0}_i} \leq t_{r^{*1}_i}$ (for all $i \leq k^*_1$). That means for $i \leq k^*_1$, r^{*0}_i and r^{*1}_i are not in conflict, and hence reassigning r^{*1}_i to server s^*_1 is valid. Observe that s^*_2 must serve a request which has pick-up location 0 and starts during interval $[t_{r^{*0}_i}, t_{r^{*1}_i} - t]$ and that request is r^{*0}_{i+1}. Because $t_{r^{*0}_{i+1}} \leq t_{r^{*1}_i} - t$ and the first request in $\Re_1$, denoted by r_o, has pick-up location 1 and starts after $t_{r^{*1}_i}$, r_o and r^{*0}_{i+1} are not in conflict. As any two consecutive requests in $\Re_1$ are not in conflict, reassigning all requests of $\Re_1$ to server s^*_2 is valid. Note that $t_{r^{*0}_{i+2}} \geq \dot{t}_{r^{*1}_i}$ as OPT accepts at most two requests which start during interval $[t_{r^{*0}_i}, t_{r^{*1}_i}]$ (during interval $[0, t_{r^{*1}_i}]$ if $i = 1$) and have pick-up location 0. Because the first request (r_l) in $\Re_2$ starts at 0 and starts after $\dot{t}_{r_i^{*1}}$, r_l and r_i^{*1} are not in conflict. As any two consecutive requests in $\Re_2$ are not in conflict, reassigning all requests of $\Re_2$ 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_i^{*0} and r_i^{*1} are assigned to the same server for $i \leq k_1^*$, and $r_i'^0$ and $r_i'^1$ are assigned to the same server for $i \leq k_1'$. 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_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 1 (r_δ is the first request in R that starts at location 1 and starts after $\dot{t}_{r_\gamma}$), it is clear that $t_{r_1'^0} \leq t_{r_1^{*0}}$ and $t_{r_1'^1} \leq t_{r_1^{*1}}$. If $r_1'^0 \neq r_1^{*0}$, we can replace r_1^{*0} by $r_1'^0$ in R^{*0}, and if $r_1'^1 \neq r_1^{*1}$, we can replace r_1^{*1} by $r_1'^1$ in R^{*1}.

Now assume, that R' and R^* are identical with respect to $2i$ requests (i requests in R^{*0} and i requests R'^0, and i requests in R^{*1} and i requests in R'^1, where $1 \leq i \leq k_1^*$), and both requests r_j^{*0} and r_j^{*1} are assigned to the same server for $1 \leq j \leq i$.

Without loss of generality, suppose $r_i'^1$ is assigned to s_1^* by OPT and $r_i'^1$ is assigned to s_1' by SG. Observe that s_1^* and s_1' are at location 0 at time $\dot{t}_{r_i'^1}$. We claim that s_2^* (resp. s_2') is at location 0 at time $\dot{t}_{r_{i-1}'^1}$ and $\dot{t}_{r_{i-1}'^1} \leq t_{r_{i+1}^{*0}}$. If $r_{i-1}'^1$ is assigned to s_2^* (resp. s_2'), s_2^* (resp. s_2') is at location 0 at time $\dot{t}_{r_{i-1}'^1}$ and $\dot{t}_{r_{i-1}'^1} = \min\{\dot{t}_{r_{i-1}'^1}, \dot{t}_{r_i'^1}\} \leq t_{r_{i+1}^{*0}}$. If $r_{i-1}'^1$ is assigned to s_1^* (resp. s_1'), we have $\dot{t}_{r_{i-1}'^1} \leq t_{r_i'^0} \leq t_{r_{i+1}^{*0}}$. Observe that OPT does not accept any request which starts in period $(t_{r_{i-1}'^1}, \dot{t}_{r_{i-1}'^1})$. 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_o (where r_o is the last request which starts at or before $t_{r_{i-1}'^1}$ 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 $t_{r_{i-1}'^1}$. Hence s_2^* (resp. s_2') is at location 0 at time $\dot{t}_{r_o}$ ($\leq \dot{t}_{r_{i-1}'^1}$), or at time 0 if r_o does not exist, and stays at that location until time $\dot{t}_{r_{i-1}'^1}$.

If there are two requests r_{i+1}^{*0} and r_{i+1}^{*1}, as s_2' is at location 0 at $\dot{t}_{r_{i-1}'^1}$ and $\dot{t}_{r_{i-1}'^1} \leq t_{r_{i+1}^{*0}}$, there must also be two requests $r_{i+1}'^0$ and $r_{i+1}'^1$ with $t_{r_{i+1}'^0} \leq t_{r_{i+1}^{*0}}$ and $t_{r_{i+1}'^1} \leq t_{r_{i+1}^{*1}}$, as SG could accept r_{i+1}^{*0} and r_{i+1}^{*1} by s_2'. We can replace r_{i+1}^{*0} and r_{i+1}^{*1} by $r_{i+1}'^0$ and $r_{i+1}'^1$ in R^{*0} and R^{*1}. If $k_0^* = k_1^*$, the claim thus follows by induction.

If $k_0^* \neq k_1^*$ ($k_0^* - k_1^* = 1$ or $k_0^* - k_1^* = 2$), then R^{*1} is already identical to R'^1, and the first k_1^* requests of R^{*0} are already identical to the first k_1^* requests of R'^0 by the argument above. If $k_0^* - k_1^* = 1$, there is a request $r_{k_1^*+1}^{*0}$. As s_2' is at location 0 at $\dot{t}_{r_{k_1^*-1}'^1}$ and $\dot{t}_{r_{k_1^*-1}'^1} \leq t_{r_{k_1^*+1}^{*0}}$, there must also be one request $r_{k_1^*+1}'^0$ with $t_{r_{k_1^*+1}'^0} \leq t_{r_{k_1^*+1}^{*0}}$, as SG could accept $r_{k_1^*+1}^{*0}$ by s_2'. We can replace $r_{k_1^*+1}^{*0}$ by $r_{k_1^*+1}'^0$ in R^{*0}, making R^{*0} identical to R'^0. If $k_0^* - k_1^* = 2$, there are two requests $r_{k_1^*+1}^{*0}$ and $r_{k_1^*+2}^{*0}$. Note that $r_{k_1^*+1}^{*0}$ and $r_{k_1^*+2}^{*0}$ must be assigned to different servers by OPT as $k_0^* - k_1^* = 2$. Recall that s_1^* is at location 0 at $\dot{t}_{r_{k_1^*}'^1}$, and s_2^* is at location 0 at $\dot{t}_{r_{k_1^*-1}'^1}$. Hence $t_{r_{k_1^*+1}^{*0}} \geq \dot{t}_{r_{k_1^*-1}'^1}$ and $t_{r_{k_1^*+2}^{*0}} \geq \dot{t}_{r_{k_1^*}'^1}$. As s_1' is at location 0 at $\dot{t}_{r_{k_1^*}'^1}$ and s_2' is at location 0 at $\dot{t}_{r_{k_1^*-1}'^1}$, there must also be two requests $r_{k_1^*+1}'^0$ and $r_{k_1^*+2}'^0$ with $t_{r_{k_1^*+1}'^0} \leq t_{r_{k_1^*+1}^{*0}}$ and $t_{r_{k_1^*+2}'^0} \leq t_{r_{k_1^*+2}^{*0}}$, as SG could accept $r_{k_1^*+1}^{*0}$ by s_2', and accept $r_{k_1^*+2}^{*0}$ by s_1'. We can replace $r_{k_1^*+1}^{*0}$ and $r_{k_1^*+2}^{*0}$ by $r_{k_1^*+1}'^0$ and $r_{k_1^*+2}'^0$ 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 $R^* = R^{*0} \bigcup R^{*1}$ and $R' = R'^0 \bigcup R'^1$. ◀

▶ **Theorem 3.** *Algorithm SG is 2-competitive for 2S2L if $c = 0$.*

Proof. We partition the time horizon $[0, \infty)$ into intervals (periods) that can be analyzed independently. Period i, for $1 < i < k$, is the interval $[\max\{\dot{t}_{r'_{i-1}}, t_{r'_i}\}, \max\{\dot{t}_{r'_i}, t_{r'_{i+1}}\})$. Period 1 is $[0, \max\{\dot{t}_{r'_1}, t_{r'_2}\})$, and period k is $[\max\{\dot{t}_{r'_{k-1}}, t_{r'_k}\}, \infty)$. (If $k = 1$, there is only a single period $[0, \infty)$.) Set $t_{r'_{k+1}} = \infty$ and $\dot{t}_{r'_0} = 0$. Let R^*_i denote the set of requests accepted by OPT that start in period i, for $1 \le i \le k$. For all $1 \le i \le k$, if $\max\{\dot{t}_{r'_{i-1}}, t_{r'_i}\} \ge \max\{\dot{t}_{r'_i}, t_{r'_{i+1}}\}$, $R^*_i = \emptyset$, and hence $P_{R^*_i} = 0$. Denote $R'_i = \{r'_i\}$ for $1 \le i \le k$.

For $1 < i \le k$, r'_i starts at time $t_{r'_i}$ and the first request of R^*_i starts during the interval $[\max\{\dot{t}_{r'_{i-1}}, t_{r'_i}\}, \max\{\dot{t}_{r'_i}, t_{r'_{i+1}}\})$ (or the interval $[\max\{\dot{t}_{r'_{k-1}}, t_{r'_k}\}, \infty)$ if $i = k$). Furthermore, r'_1 is the first acceptable request in R, and so the first request of R^*_1 cannot start before $t_{r'_1}$. Hence, for all $1 \le i \le k$, the first request in R^*_i cannot start before $t_{r'_i}$.

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

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_o. As s'_2 is at location 0 at time 0, r_o would be acceptable to SG by s'_2. Hence, there cannot be such a request r_o that starts in period $[0, \infty)$. As we have shown that OPT can accept at most one request (r'_1), we get that $\frac{P_{R^*}}{P_{R'}} \le \frac{r}{r} < 2$.

Case 2: $k > 1$. For all $1 \le i \le k$, 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_o 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'_i is assigned to s'_1.

Case 2.1: $\dot{t}_{r'_i} > t_{r'_{i+1}}$ (Fig. 1.a shows an example). If $i > 1$, the period i, which is the period $[\max\{\dot{t}_{r'_{i-1}}, t_{r'_i}\}, \max\{\dot{t}_{r'_i}, t_{r'_{i+1}}\}) = [\max\{\dot{t}_{r'_{i-1}}, t_{r'_i}\}, \dot{t}_{r'_i})$, has length less than t. If $i = 1$, note that the period $[t_{r'_1}, \max\{\dot{t}_{r'_1}, t_{r'_2}\}) = [t_{r'_1}, \dot{t}_{r'_1})$ has length less than t and no request of R^*_1 can start before $t_{r'_1}$ during period 1, $[0, \max\{\dot{t}_{r'_1}, t_{r'_2}\})$. Therefore, each server can accept at most one request that starts during period i, and hence R^*_i contains at most two requests.

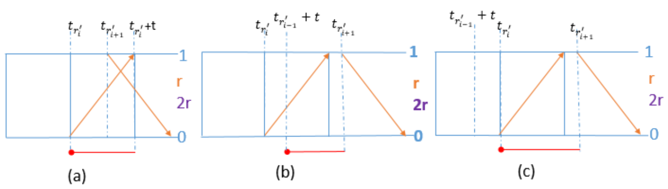

Figure 1 $c = 0$, $|R'| = k > 1$, $1 \le i \le k$

Case 2.2: $\dot{t}_{r'_i} \le t_{r'_{i+1}}$ and $\dot{t}_{r'_{i-1}} > t_{r'_i}$ (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 $\dot{p}_{r'_{i-1}}$ at $\dot{t}_{r'_{i-1}}$. As the first request in R^*_i does not start before $\dot{t}_{r'_{i-1}}$, we have $t_{r_o} \ge \dot{t}_{r'_{i-1}} + t$. This means that r_o would be acceptable to s'_2. Therefore, SG accepts either r_o or another request starting before t_{r_o}, and that request becomes r'_{i+1}. Hence, there cannot be such a request r_o that starts in period i.

Case 2.3: $\dot{t}_{r'_i} \le t_{r'_{i+1}}$ and $\dot{t}_{r'_{i-1}} \le t_{r'_i}$ (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_l (where

r_l 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_l be a dummy request with $\dot{t}_{r_l} = 0$ and $\dot{p}_{r_l} = 0$) at $\dot{t}_{r_l}$ and $\dot{t}_{r_l} \leq \dot{t}_{r'_{i-1}} \leq t_{r'_i}$. Observe that s'_2 does not accept any request which starts during period $[t_{r'_i}, \dot{t}_{r'_i})$, s'_2 does not start to move before $t_{r'_i}$ for serving the next request, and hence s'_2 is at $\dot{p}_{r_l}$ (0 or 1) at $t_{r'_i}$. As the first request in R^*_i does not start before $t_{r'_i}$, we have $t_{r_o} \geq t_{r'_i} + t$. This means that r_o would be acceptable to s'_2. Therefore, SG accepts either r_o or another request starting before t_{r_o}, and that request becomes r'_{i+1}. Hence, there cannot be such a request r_o 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. ◀

▶ **Lemma 4.** *When $0 < c < r$, for all $1 < i \leq k$, one server of SG is at $p_{r'_i}$ at $t_{r'_i}$ and the other server of SG is at 0 or 1 at $\max\{\dot{t}_{r'_{i-1}}, t_{r'_i}\}$.*

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

If $\dot{t}_{r'_{i-1}} > t_{r'_i}$, then r'_{i-1} must be assigned to s'_2, and hence s'_2 is at $\dot{p}_{r'_{i-1}}$ (0 or 1) at $\dot{t}_{r'_{i-1}}$ $(= \max\{\dot{t}_{r'_{i-1}}, t_{r'_i}\}$).

If $\dot{t}_{r'_{i-1}} \leq t_{r'_i}$, then s'_2 is at 0 or 1 at the drop-off time t' ($t' \leq \dot{t}_{r'_{i-1}}$) of the latest request which is assigned to s'_2 and starts at or before $t_{r'_i}$. (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 $t_{r'_i}$ and is served by s'_2. If r_f does not exist, then s'_2 does not move after $\dot{t}_{r'_{i-1}}$, and s'_2 is at 0 or 1 at $\max\{\dot{t}_{r'_{i-1}}, t_{r'_i}\}$ ($\dot{t}_{r'_{i-1}} \leq t_{r'_i}$). If r_f exists and r_f is accepted with cost, then $t_{r_f} - \dot{t}_{r'_i} \geq t$ ($\dot{t}_{r'_i} \leq \dot{t}_{r^n_f}$) because SG accepts a request r_j with cost only if the condition $t_{r_j} - \dot{t}_{r^n_j} \geq t$ is satisfied. That means s'_2 starts an empty move at or after $\dot{t}_{r'_i}$. If r_f exists and r_f is accepted without cost, then s'_2 starts to move at or after $t_{r'_i}$ ($t_{r_f} \geq t_{r'_i}$). Therefore s'_2 is at 0 or 1 at $t_{r'_i}$ $(= \max\{\dot{t}_{r'_{i-1}}, t_{r'_i}\})$. ◀

▶ **Lemma 5.** *When $0 < c < r$, for all $1 \leq i \leq k$, if r'_i is accepted with cost, then one server of SG is at $p_{r'_i}$ at $t_{r'_i}$ and the other server of SG is at $\dot{p}_{r'_i}$ at $t_{r'_i}$.*

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

If $i = 1$, then $p_{r'_1} = 1$ and s'_2 is at $\dot{p}_{r'_1}$ (location 0) at time 0. Suppose r_o is the first request which is assigned to s'_2. If $p_{r_o} = 0$, then s'_2 starts to move at t_{r_o} $(\geq t_{r'_1})$, and hence s'_2 is at 0 at $t_{r'_i}$. If $p_{r_o} = 1$, then $t_{r_o} \geq \dot{t}_{r'_1} + t$ because by definition SG accepts a request r_j with cost only if the condition $t_{r_j} - \dot{t}_{r^n_j} \geq t$ is satisfied. Observe that s'_2 starts to move at $t_{r_o} - t$ $(\geq \dot{t}_{r'_1})$, and hence s'_2 is at 0 at $t_{r'_i}$.

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

▶ **Lemma 6.** *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 one server of SG is at $p_{r'_i}$ at $t_{r'_i}$, and the other server of SG is at $\dot{p}_{r'_i}$ at $\max\{\dot{t}_{r'_{i-1}}, t_{r'_i}\}$.*

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

We prove this lemma by contradiction. Assume that s'_2 is at $p_{r'_i}$ at $\max\{\dot{t}_{r'_{i-1}}, t_{r'_i}\}$. Suppose $r_f = r'_{i+1}$. Observe that $p_{r'_f} = p_{r'_i}$ and $t_{r'_f} \geq \dot{t}_{r'_i} + t \geq \max\{\dot{t}_{r'_{i-1}}, t_{r'_i}\}$. From this it follows that r'_f is acceptable to SG by s'_2 without cost, and hence *SG* assigns r'_f to s'_2 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'_1. Observe that s'_1 is at $p_{r'_{i-1}}$ at $t_{r'_{i-1}}$ (and is at $\dot{p}_{r'_{i-1}}$ at $\dot{t}_{r'_{i-1}}$). We prove this lemma by contradiction. Assume r'_{i-1} is accepted with cost. According to Lemma 5, s'_2 is at $\dot{p}_{r'_{i-1}}$ at $t_{r'_{i-1}}$. As r'_i is accepted without cost, the pick-up location of r'_i is $\dot{p}_{r'_{i-1}}$. Suppose $r_f = r'_{i+1}$. Observe that $t_{r_f} \geq \dot{t}_{r'_i} + t$ (because r_f is accepted with cost) and $p_{r_f} = p_{r'_i} = \dot{p}_{r'_{i-1}}$ (otherwise, the server that has served r'_i could accept r_f without cost).

If r'_i is assigned to s'_1, then s'_2 does not accept any request which starts in period $[\max\{\dot{t}_{r'_{i-2}}, t_{r'_{i-1}}\}, t_{r_f})$, and hence s'_2 is at $\dot{p}_{r'_{i-1}}$ in period $[\max\{\dot{t}_{r'_{i-2}}, t_{r'_{i-1}}\}, t_{r_f} - t)$. If r'_i is assigned to s'_2, then s'_1 does not accept any request which starts in period $[\dot{t}_{r'_{i-1}}, t_{r_f})$, and hence s'_1 is at $\dot{p}_{r'_{i-1}}$ in period $[\dot{t}_{r'_{i-1}}, t_{r_f} - t)$. As r_f is released and $\tilde{t}_{r_f} = t_{r_f} - a \leq t_{r_f} - t$, server s'_q (for a $q \in \{1, 2\}$) is at $\dot{p}_{r'_{i-1}}$ and does not plan to move, hence r_f is acceptable to SG by s'_q without cost. From this it follows that r_f 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 simplification 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^*_q ($q \in \{1, 2\}$), 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'_1 is accepted with cost), and exactly one request of each period (except the first period if r'_1 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 ($1 \leq j \leq k'$) by l_j. For $1 < j < k'$, SG j period is $[t_{r'_{l_j}}, t_{r'_{l_{j+1}}})$. SG 1 period is $[0, t_{r'_{l_2}})$ and SG k' period is $[t_{r'_{l_{k'}}}, \infty)$ (If $k' = 1$, there is only a single period $[0, \infty)$). We set $l_{k'+1} = k + 1$, $t_{r'_0} = 0$ and $t_{r'_{k+1}} = \infty$. Let R'_j ($1 \leq j \leq k'$) denote the set of requests accepted by *SG* that start in *SG* j period. For $1 < j \leq k'$, if $t_{r'_{l_{j-1}}} = t_{r'_{l_j}}$, let $R'_{j-1} = \{r'_{l_{j-1}}\}$ and $R'_j = \{r'_{l_j}, r'_{l_j+1} ..., r'_{l_{j+1}-1}\}$. 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 $R'_j = \{r'_{l_j}, r'_{l_j+1} ..., r'_{l_{j+1}-1}\}$.

Algorithm 2 Partition Rule (PR)

Initialization: $k = |R'|$, $k' = 1$, $j = 1$, $l_j = j$ for all $1 \leq j \leq k$.
For $i = 2$ to k
if r'_i is accepted without cost then
$j = j + 1$, $l_j = i$;
$k' = j$, $l_{k'+1} = k + 1$.

For all $1 < j \leq k'$, we have the following property: if $|R'_j| = 1$, then r'_{l_j} is accepted without cost; if $|R'_j| > 1$, then r'_{l_j} is accepted without cost, the remaining requests in R'_j are accepted with cost. For $j = 1$, if $r'_1 (= r'_{l_1})$ 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: $t_j = t_{r'_{l_j}}$ if $r'_{l_j - 1}$ is accepted with cost, r'_{l_j} is accepted without cost, $\dot{t}_{r'_{l_j-1}} > t_{r'_{l_j}}$ and $\dot{p}_{r'_{l_j-1}} = p_{r'_{l_j}}$ (Fig. 2 shows an example). Otherwise, $t_j = \max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\}$. $t_{k'+1} = t_{r'_{k+1}} = \infty$.

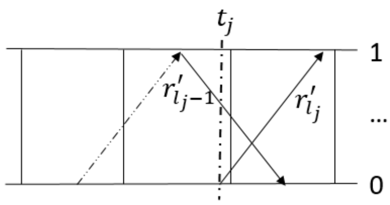

Figure 2 An example of t_j

For $1 < j \leq k'$, $t_{j+1} = t_{r'_{l_{j+1}}}$ or $t_{j+1} = \max\{\dot{t}_{r'_{l_{j+1}-1}}, t_{r'_{l_{j+1}}}\}$, and $t_j = t_{r'_{l_j}}$ or $t_j = \max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\}$. Because $t_{r'_{l_j}} \leq t_{r'_{l_{j+1}}}$ and $\dot{t}_{r'_{l_j-1}} \leq t_{r'_{l_{j+1}}}$ (if r'_{l_j-1} and r'_{l_j} are assigned to the same server, then $\dot{t}_{r'_{l_j-1}} \leq t_{r'_{l_j}}$; and if r'_{l_j-1} and r'_{l_j} are assigned to different servers, then $\dot{t}_{r'_{l_j-1}} \leq t_{r'_{l_{j+1}}}$), $t_j \leq t_{j+1}$ if $t_j = \max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\}$ and $t_{j+1} = t_{r'_{l_{j+1}}}$. As $t_j \leq \max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\}$ and $t_{j+1} \geq t_{r'_{l_{j+1}}}$, 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 $R^*_i = \emptyset$ if $t_j = t_{j+1}$.

For all $1 < j \leq k'$, r'_{l_j} starts at time $t_{r'_{l_j}}$ and the first request of R^*_j starts during the interval $[t_j, t_{j+1})$ where $t_j = t_{r'_{l_j}}$ or $t_j = \max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\}$ (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 $t_{r'_{l_j}}$.

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 $P_{R^*}/P_{R'} \leq \alpha$ follows if we can show that $P_{R^*_j}/P_{R'_j} \leq \alpha$ for all $1 \leq j \leq k'$. For $1 \leq j \leq k'$, if $t_j = t_{j+1}$, then $R^*_i = \emptyset$ and hence $P_{R^*_i} = 0$. Otherwise, for $1 \leq j \leq k'$ we distinguish the following cases in order to bound $P_{R^*_j}/P_{R'_j}$.

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'_j are accepted with cost and $P_{R'_1} = (l_2 - l_1)(r - c)$ (if $k' = 1$, then $P_{R'} = k(r - c)$). Observe that $p_{r'_i} = 1$ $(1 \leq i < l_2)$ 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 $p_{r'_{l_2-1}}$ at $t_{r'_{l_2-1}}$ (and this server is at $\dot{p}_{r'_{l_2-1}}$ at $\dot{t}_{r'_{l_2}-1}$), and the other server is at $\dot{p}_{r'_{l_2-1}}$ at $t_{r'_{l_2-1}}$ (by Lemma 5). As r'_{l_2} is accepted without cost, we have $\dot{p}_{r'_{l_2-1}} = p_{r'_{l_2}}$. If $k' = 1$, $t_2 = \infty$. If $k' > 1$, then $t_2 = t_{r'_{l_2}}$: if $\dot{t}_{r'_{l_2-1}} > t_{r'_{l_2}}$, $t_2 = t_{r'_{l_2}}$ because $p_{r'_{l_2}} = \dot{p}_{r'_{l_2-1}}$, r'_{l_2-1} is accepted with cost and r'_{l_2} is accepted without cost; if $\dot{t}_{r'_{l_2-1}} \leq t_{r'_{l_2}}$, $t_2 = \max\{\dot{t}_{r'_{l_2-1}}, t_{r'_{l_2}}\} = t_{r'_{l_2}}$. As s'_2 does not accept any request which starts before $t_{r'_{l_2}}$ and s'_2 would not accept any request with cost which starts in period $[t_{r'_{l_2}}, \dot{t}_{r'_{l_2}})$ (Recall that Algorithm 1 accepts a request r_j with cost only if $t_{r_j} - \dot{t}_{r^n_j} \geq t$ is satisfied.), s'_2 is at 0 in period $[0, t_{r'_{l_2}}]$. 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 $t_{r_o} \leq t_2 = t_{r'_{l_2}}$, 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'_{l_2}. 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 $[0, t_{r'_1})$. 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 $[t_{r'_i}, t_{r'_{i+1}})$ $(l_1 \leq i \leq l_2 - 2)$, or period $[t_{r'_{l_2-1}}, t^*)$ (if $k' > 1$, $t^* = t_{r'_{l_2}}$; if $k' = 1$, $t^* = t_{r'_k} + 2t$). 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 $[t_{r'_i}, t_{r'_{i+1}})$ $(l_1 \leq i \leq l_2 - 2)$ or period $[t_{r'_{l_2-1}}, t^*)$. As the request does not start before $t_{r'_i}$ $(l_1 \leq i \leq l_2 - 1)$, we have $t_{r_o} \geq t_{r'_i} + 2t$. 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'_{i+1} $(l_1 \leq i < l_2)$. Hence, there cannot be such a request r_o that starts during period $[t_{r'_i}, t_{r'_{i+1}})$ $(l_1 \leq i \leq l_2 - 2)$ or period $[t_{r'_{l_2-1}}, t^*)$. Therefore OPT can accept at most $2(l_2 - l_1)$ $(= 2(l_2 - 2 - l_1 + 1 + 1))$ requests that start during period $[t_{r'_1}, t^*)$.

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^*_1 that starts during period $[t^*, \infty)$. As $t_{r_o} \geq t_{r'_k} + 2t$. 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'_{k+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 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'_2 is also accepted without cost. Assume that r'_2 is accepted with cost, we have $t_{r'_2} - \dot{t}_{r'_1} > t$ because Algorithm 1 accepts a request r_j with cost only if $t_{r_j} - \dot{t}_{r^n_j} \geq t$ is satisfied. If $p_{r'_2} = 0$, r'_2 is acceptable to SG by s'_2 without cost; if $p_{r'_2} = 1$, r'_2 is acceptable to SG by s'_1 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'_2 is accepted with cost. Observe that $t_2 = \max\{\dot{t}_{r'_1}, t_{r'_2}\}$ (Recall from the definition of t_2 that $t_2 = t_{r'_2}$ only if r'_1 is accepted with cost), $|R'_1| = 1$ and hence $P_{R'_1} = r$. As s'_2 does not accept any request which starts before $t_{r'_2}$ and s'_2 would not accept any request with cost which starts in period $[t_{r'_2}, \dot{t}_{r'_2})$ (Recall that Algorithm 1 accepts a request r_j with cost only if $t_{r_j} - \dot{t}_{r^n_j} \geq t$ is satisfied.), s'_2 is at 0 in period $[0, t_{r'_2}]$.

We claim that R^*_1 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^*_1. As the first request in R^*_1 does not start before $t_{r'_1}$, we have $t_{r_o} \geq t_{r'_1} + t$. If $p_{r_o} = \dot{p}_{r_1}$, r_o is acceptable to SG by s'_1 without cost; if $p_{r_o} = p_{r_1}$, 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 $P_{R^*_1} \leq 2r$, and hence $P_{R^*_1}/P_{R'_1} \leq 2r/r = 2$.

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

CASE 2.1: $|R'_j| = 1$. Note that r'_{l_j} is accepted without cost. We distinguish two sub-cases.

(1) $\dot{t}_{r'_{l_j}} > t_{r'_{l_{j+1}}}$. Because r'_{l_j} $(= r'_{l_{j+1}-1})$ is accepted without cost, $t_{j+1} = \max\{\dot{t}_{r'_{l_j}}, t_{r'_{l_{j+1}}}\} = \dot{t}_{r'_{l_j}}$ (Recall that $t_{j+1} = t_{r'_{l_{j+1}}}$ only if $r'_{l_{j+1}-1}$ is accepted with cost by the definition of t_{j+1}). As OPT period j $[t_j, t_{j+1})$ has length less than t ($t_j = \max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\}$ or $t_j = t_{r'_{l_j}}$), each server of OPT can accept at most one request in R^*_j, and hence R^*_j contains at most two requests.

(2) $\dot{t}_{r'_{l_j}} \leq t_{r'_{l_{j+1}}}$ ($t_{r'_{l_{j+1}}} = \infty$ if $j = k'$). Note that $t_{j+1} = t_{r'_{l_{j+1}}}$. There are two sub-cases based on the position of s'_2 at $\max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\}$ (recall that by Lemma 4, s'_2 is at $p_{r'_{l_j}}$ or $\dot{p}_{r'_{l_j}}$ at time $\max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\}$).

The first sub-case is that s'_2 is at $p_{r'_{l_j}}$ at $\max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\}$. 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 requests in R^*_j do not start before $t_{r'_{l_j}}$, we have $t_{r_o} \geq t_{r'_{l_j}} + t$. If $p_{r_o} = \dot{p}_{r'_{l_j}}$, r_o is acceptable to SG by s'_1 without cost; if $p_{r_o} = p_{r'_{l_j}}$, 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'_{l_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 $\dot{p}_{r'_{l_j}}$ at $\max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\}$. Note that $t_j = \max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\}$ (Recall from the definition of t_j that $t_j = t_{r'_{l_j}}$ only if $\dot{t}_{r'_{l_j-1}} > t_{r'_{l_j}}$ and $\dot{p}_{r'_{l_j-1}} = p_{r'_{l_j}}$ are satisfied. From this it follows that r'_{l_j-1} must be assigned to s'_2, that means s'_2 is at $\dot{p}_{r'_{l_j-1}}$ $(= p_{r'_{l_j}})$ at $\dot{t}_{r'_{l_j-1}}$ $(= \max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\})$. This contradicts the initial assumption that s'_2 is at $\dot{p}_{r'_{l_j}}$ at $\max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\}$.). 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 $p_{r'_{l_j}}$. Assume that R^*_j contains a request r_i which starts at $\dot{p}_{r'_{l_j}}$. As the requests in R^*_j cannot start before t_j ($t_j = \max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\}$), r_i is acceptable to s'_2 (without cost) as s'_2 is at $\dot{p}_{r'_{l_j}}$ at $\max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\}$. 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 $t_{r'_{l_j}}$ and the pick-up location of r_i and r_o both are $p_{r'_{l_j}}$, we have $t_{r_o} \geq t_{r'_{l_j}} + 2t$. If $p_{r_o} = \dot{p}_{r'_{l_j}}$, r_o is acceptable to SG by s'_1 without cost; if $p_{r_o} = p_{r'_{l_j}}$, 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 $r'_{l_{j+1}}$ (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'_{l_j} is accepted without cost and r'_{l_j+1} is accepted with cost. We have that s'_2 is at $\dot{p}_{r'_{l_j}}$ at $\max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\}$ by Lemma 6, and that r'_{l_j-1} is accepted without cost by Lemma 7. Hence, $t_j = \max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\}$ (recall from the definition of t_j that $t_j = t_{r'_{l_j}}$ only if r'_{l_j-1} is accepted with cost). As $r'_{l_{j+1}-1}$ is accepted with cost, one server is at $p_{r'_{l_{j+1}-1}}$ at $t_{l_{j+1}-1}$ (and this server is at $\dot{p}_{r'_{l_{j+1}-1}}$ at $\dot{t}_{r'_{l_{j+1}-1}}$), and the other server is at $\dot{p}_{r'_{l_{j+1}-1}}$ at $t_{r'_{l_{j+1}-1}}$ (Recall Lemma 5). As $r'_{l_{j+1}}$ is accepted without cost, we have $\dot{p}_{r'_{l_{j+1}-1}} = p_{r'_{l_{j+1}}}$. If $\dot{t}_{r'_{l_{j+1}-1}} > t_{r'_{l_{j+1}}}$ $(1 \le j < k')$, $t_{j+1} = t_{r'_{l_{j+1}}}$ according to the definition of t_s $(1 \le s \le k')$. If $\dot{t}_{r'_{l_{j+1}-1}} \le t_{r'_{l_{j+1}}}$ $(1 \le j < k')$, $t_{j+1} = \max\{\dot{t}_{r'_{l_{j+1}-1}}, t_{r'_{l_{j+1}}}\} = t_{r'_{l_{j+1}}}$. Hence, $t_{j+1} = t_{r'_{l_{j+1}}}$ $(1 \le j < k')$. Observe that if $j = k'$, $t_{j+1} = t_{r'_{l_{j+1}}} = \infty$.

We claim that R^*_j only contains requests which start at $p_{r'_{l_j}}$. Assume that R^*_j contains a request r_i which starts at $\dot{p}_{r'_{l_j}}$. As the first request in R^*_j cannot start before t_j, we have $t_{r_i} \ge t_j = \max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\}$. As s'_2 is at $\dot{p}_{r'_{l_j}}$ at $\max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\}$ and s'_2 does not accept any request which starts in period $[\max\{\dot{t}_{r'_{l_j-1}}, t_{r'_{l_j}}\}, t_{r_i})$, and hence r_i is acceptable to SG by s'_2 without cost. This contradicts the property of R'_j that except r'_{l_j} all requests in R'_j are accepted with cost. Hence, there cannot be such a request r_i that starts in OPT period j.

We claim that each server of OPT can accept at most one request which starts in period $[t_j, t_{r'_{l_j+1}})$, or period $[t_{r'_i}, t_{r'_{i+1}})$ $(l_j + 1 \le i \le l_{j+1} - 2)$, or period $[t_{r'_{l_{j+1}-1}}, t^*)$ (if $1 \le j < k'$, $t^* = t_{r'_{l_{j+1}}}$; if $j = k'$, $t^* = t_{r'_k} + 2t$). Assume that s^*_q $(q \in \{1, 2\})$ accepts at least two requests in one of these periods. Let r_o be the second request (in order of start time) which is assigned to s^*_q and starts in one of these periods. As the requests in R^*_j that start in one of these periods do not start before the corresponding $t_{r'_i}$ $(l_j \le i \le l_{j+1} - 1)$ and have the same pick-up location $p_{r'_{l_j}}$, we have $t_{r_o} \ge t_{r'_{l_j}} + 2t$. r_o is acceptable to SG with cost. Therefore, SG accepts either r_o or another request starting before t_{r_o}, that request becomes r'_{i+1} $(l_j \le i \le l_{j+1} - 2)$, or we get a contradiction to $r'_{l_{j+1}-1}$ being the last request that is accepted with cost and starts in period $[t_{r'_{l_{j+1}-1}}, t^*)$ $(i = l_{j+1} - 1)$. Hence, there cannot be such a request r_o that starts in period $[t_j, t_{r'_{l_j+1}})$, or period $[t_{r'_i}, t_{r'_{i+1}})$ $(l_j + 1 \le i \le l_{j+1} - 1)$. Therefore OPT can accept at most $2(l_{j+1} - l_j)$ $(= 2(l_{j+1} - 2 - (l_j + 1) + 1 + 2))$ requests that start in period $[t_{r'_{l_{j+1}}}, t^*)$.

When $j = k'$, 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 which starts during period $[t^*, \infty)$. As $t_{r_o} \ge t_{r'_k} + 2t$, 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'_{k+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 $2(l_{j+1} - l_j)$ requests and the pick-up locations of them are the same $(p_{r'_{l_j}})$, we get that $P_{R^*_j} \le 2r + 2(l_{j+1} - l_j - 1)(r - c)$. Since $P_{R'_j} = r + (l_{j+1} - l_j - 1)(r - c)$, we have $P_{R^*_j}/P_{R'_j} \le (2r + 2(l_{j+1} - l_j - 1)(r - c))/(r + (l_{j+1} - l_j - 1)(r - c)) = 2$.

Because $P_{R^*_j}/P_{R'_j} \le 2$ holds for all $1 \le j \le k'$, we have $P_{R^*}/P_{R'} \le 2$. This proves the theorem. ◀

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 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

1. Norbert Ascheuer, Sven Oliver Krumke, and Jörg Rambau. Online dial-a-ride problems: Minimizing the completion time. In Horst Reichel and Sophie Tison, editors, *STACS 2000, 17th Annual Symposium on Theoretical Aspects of Computer Science, Lille, France, February 2000, Proceedings*, volume 1770 of *LNCS*, pages 639–650. Springer, 2000. URL: https://doi.org/10.1007/3-540-46541-3_53, doi:10.1007/3-540-46541-3_53.
2. Gerardo Berbeglia, Jean-François Cordeau, and Gilbert Laporte. Dynamic pickup and delivery problems. *European Journal of Operational Research*, 202(1):8–15, 2010. URL: https://doi.org/10.1016/j.ejor.2009.04.024, doi:10.1016/j.ejor.2009.04.024.
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 Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 994–1005. SIAM, 2017. URL: https://doi.org/10.1137/1.9781611974782.63, 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 Evangelos Kranakis, Gonzalo Navarro, and Edgar Chávez, editors, *12th Latin American Symposium on Theoretical Informatics (LATIN 2016)*, volume 9644 of *LNCS*, pages 220–234. Springer, 2016. URL: https://doi.org/10.1007/978-3-662-49529-2_17, 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. URL: https://doi.org/10.1007/s10878-017-0188-z, 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 Thomas Erlebach and Giuseppe Persiano, editors, *Approximation and Online Algorithms, Third International Workshop, WAOA 2005, Palma de Mallorca, Spain, October 6-7, 2005, Revised Papers*, volume 3879 of *LNCS*, pages 258–269. Springer, 2006. URL: https://doi.org/10.1007/11671411_20, doi:10.1007/11671411_20.
8. Kelin Luo, Thomas Erlebach, and Yinfeng Xu. Car-sharing between two locations: Online scheduling with flexible advance bookings. In *Proceedings of the 24th International Computing and Combinatorics Conference, COCOON 2018*, LNCS. Springer, 2018. To appear.
9. Fanglei Yi and Lei Tian. On the online dial-a-ride problem with time-windows. In Nimrod Megiddo, Yinfeng Xu, and Binhai Zhu, editors, *Algorithmic Applications in Management, First International Conference, AAIM 2005, Xian, China, June 22-25, 2005, Proceedings*, volume 3521 of *LNCS*, pages 85–94. Springer, 2005. URL: https://doi.org/10.1007/11496199_11, doi:10.1007/11496199_11.