Online Traveling Salesman Problems with Rejection Options[*]

Patrick Jaillet[‡] Xin Lu[§]

March 2013; revised December 2013; accepted July 2014

Abstract

In this paper we consider online versions of the Traveling Salesman Problem (TSP) on metric spaces for which requests to visit points are not mandatory. Associated with each request is a penalty (if rejected). Requests are revealed over time (at their release dates) to a server who must decide which requests to accept and serve in order to minimize a linear combination of the time to serve all accepted requests and the total penalties of all rejected requests. In the *basic* online version of the problem, a request can be accepted any time after its release date. In the *real-time* online version, a request must be accepted or rejected at the time of its release date.

For the basic version, we provide a best possible 2-competitive online algorithm for the problem on a general metric space. For the real-time version, we first consider special metric spaces: on the non-negative real line, we provide a best possible 2.5-competitive polynomial time online algorithm; on the real line, we prove a lower bound of 2.64 on any competitive ratios and give a 3-competitive online algorithm. We then consider the case of a general metric space and prove a $\Omega(\sqrt{\ln n})$ lower bound on the competitive ratio of any online algorithms. Finally, among the restricted class of online algorithms with prior knowledge about the total number of requests n, we propose an asymptotically best possible $O(\sqrt{\ln n})$-competitive algorithm.

key words: online, competitive analysis, traveling salesman problem

1 Introduction

In the classical Traveling Salesman Problem (TSP) in a metric space, we are given an origin, a set of points in that space, and the task is to find a tour of minimum total length, beginning and ending at the origin, that visits each point at least once. If one introduces a "time" aspect to the problem by considering a server visiting these points with a given constant speed, the objective can equivalently be stated as to minimize the time required to complete a tour. When requests to visit points include release dates, i.e., when a point can only be visited on or after its release date, we obtain the so-called "TSP with release dates". Removing the need to visit all requests, one can associate a penalty with each request to visit a point. The server can then decide which points to serve and the objective is to minimize a linear combination of the time to go through all accepted requests plus the penalties of the rejected ones.

[*]Research funded in part by ONR, grants N00014-09-1-0326 and N00014-12-1-0033, and by AFOSR, grant FA9550-10-1-0437

[‡]Laboratory for Information and Decision Systems, Department of Electrical Engineering and Computer Science, and Operations Research Center, MIT, Cambridge, MA, 02139; jaillet@mit.edu

[§]Operations Research Center, MIT, Cambridge, MA, 02139; luxin@mit.edu

This paper is a follow-up companion to our recently published article [10], where we consider online versions of the TSP with release dates and rejection penalty. In that article, the decisions to accept or reject requests could be done any time after their release dates. In the current paper, we also consider the case when the decisions must be made immediately at the release dates. To distinguish these two versions, the first model is called the *basic* version, and the second one is called the *real-time* version.

1.1 Formal definitions of the problems

Online TSP with Rejection Options

Instance: A metric space $\mathcal{M}$ with a given origin o and a distance metric $d(\cdot,\cdot)$. A series of n requests represented by triples $(l_i, r_i, p_i)_{1\leq i\leq n}$, where $l_i \in \mathcal{M}$ is the location (point in metric space) of request i, $r_i \in \mathbb{R}_+$ is its release date (first time after which it can be served), and $p_i \in \mathbb{R}_+$ is its penalty (for not being served). The problem begins at time 0; the server is initially idle at the origin (initial state), can travel at unit speed (when not idle), and eventually must be back and idle at the origin (final state). The earliest time the server reaches this final state is called the makespan.

Offline context: The number of requests n is known to the offline server. All requests are revealed to the offline server at time 0.

Online context: The number of requests n is not known to the online server. Requests are revealed to the online server at their release dates $r_i \geq 0$; assume $r_1 \leq r_2 \cdots \leq r_n$. There are two online versions:

Basic: The online server can accept or reject a request any time after the request's release date.

Real-time: The online server must accept or reject a request immediately at the time of the request's release date. Decisions are then final.

Objective: In all cases, minimize {the makespan to serve all accepted requests plus the total penalties of all rejected requests} among all feasible solutions.

The offline problem is thus a TSP with release dates and penalty, and the two online versions of the problem differ as to when decisions to accept or reject a request can be done.

1.2 Our contributions

For the basic online version of the TSP, we provide a best possible 2-competitive online algorithm in a general metric space, improving the 2.28-competitive ratio of [10]. A slightly modified version of our algorithm can be applied to the prize-collecting TSP as defined in [3]. Our modified algorithm remains 2-competitive for any variations of that problem, improving the 7/3-competitive ratio of [3].

For the real-time version, we provide a best possible 2.5-competitive polynomial time online algorithm on the non-negative real line. On the real line, we prove a lower bound of 2.64 on any competitive ratios and give a 3-competitive online algorithm. Finally, we consider the case of a general metric space and prove a $\Omega(\sqrt{\ln n})$ lower bound on the competitive ratio of any online algorithms. Among the restricted class of online algorithms with prior knowledge about the total number of requests n, we also propose an asymptotically best possible $O(\sqrt{\ln n})$-competitive one.

1.3 Literature review

The Traveling Salesman Problem (TSP), along with its many variations, has received wide attention from different communities. Several books, including those of Lawler et al. [16] and Korte and Vygen [14] offer comprehensive coverage of results about the classical variants. The problems for which points do not need to be all visited have comparatively received less attention, and most of the work originated in the mid-80s, with a paper by Balas [6] on the prize collecting traveling salesman problem. A good survey about various models and solution strategies about these variants up to 2005 can be found in [8].

With respect to the literature on online versions of the TSP as considered in this paper, it started with the paper by Ausiello et al. [5], in which the authors introduce and study the online version of the TSP with release dates. We refer the readers to our previous companion paper [10] for a more detailed account of the key literature on online TSP and its variants since then, including [2, 17, 7, 11, 21, 1, 4, 3, 12, 18]. Most related to this current paper, are those dealing with online routing problems which do not require the server to visit every revealed request ([4, 3, 10]).

Let us conclude this review by noting that other online models of the TSP have been proposed in the literature, including some recent papers. In one model, closely related to graph exploration, new requests are revealed locally during the traversal of a tour (i.e., an arrival at a node reveals any adjacent nodes that must also be visited). In that case, constant-competitive algorithms are also known for many cases ([13, 19, 9]). In another model, the nodes of an unknown graph with metric edge cost appear one by one and must be connected in such a way that the resulting tour has low cost. Without additional power, no online algorithms can guarantee constant-competitive ratio. In [20], the authors investigate under what conditions, allowing some recourse (e.g., limited edge rearrangement per iteration) would lead to constant-competitive algorithms.

Outline: The remainder of the paper is as follows: after introducing some key notations and assumptions in Section 2, we first present our main result on the basic online version for the TSP in a general metric space in Section 3. We then concentrate on the real-time version of the problem in Section 4, presenting results on the non-negative real line, on the real line, and on a general metric space. We offer few concluding remarks in Section 5.

2 Notations

The formal definitions of the problems considered in this paper have been given in Section 1.1. We assume that we have at our disposal an exact algorithm (a black box) that solves any instance of the corresponding offline problems.

An instance I consisting of n requests is gradually revealed. The online server observes a series of partial instances $\{I_k\}_{1\leq k\leq n}$, where I_k is the instance consisting of the first k requests. For the instance I_k, let $C_{\text{opt}}(k)$ be the objective value of an optimal offline solution, τ_k be the corresponding optimal route (tour or path), T_k be the corresponding makespan, and S_k be the set of accepted requests by the black box. Given a route τ, we also use τ as a function $\tau(t) : \mathbb{R}^+ \to \mathcal{M}$, where $\tau(t)$ is the position of the server who follows τ at t. $L(\tau)$ represents the length of the route, i.e. the shortest time to travel through it, ignoring release dates of requests.

For a given online algorithm A, we let $C_A(k)$ be the total cost incurred by the online server on the instance I_k. We measure the quality of this online algorithm via its competitive ratio, i.e., the smallest upper bound on $C_A(n)/C_{\text{opt}}(n)$ for any instances of any size n. If there exists such a finite upper-bound c, then we say that A is c-competitive, and, in case no other online algorithms have smaller competitive ratios, we say that A is best possible. If a finite upper bound does not exist, then one can characterize the asymptotic behavior of the competitive ratios as a function of n (n representing the problem size) by providing functions f and g such that an $\Omega(f(n))$ and $O(g(n))$ are asymptotic lower and upper bounds on the competitive ratios for the problem.

3 Basic Version of the Online Problem

We focus here on the basic version of the problem. In Jaillet and Lu [10], we show a lower bound of 2 on the competitive ratio of any online algorithms for this problem, even under the simple case of the non-negative real-line:

Theorem ([10])**.** *Any c-competitive online algorithm on $\mathbb{R}_+$ must have $c \geq 2$.*

In the remainder of the section, we first propose an online algorithm whose competitive ratio matches this lower bound in any general metric spaces. We then consider the design of a polynomial time online algorithm for this problem and, we finally address a slight generalization of the problem, involving both penalty for rejection of a request and prize for collection, if a request is accepted and served.

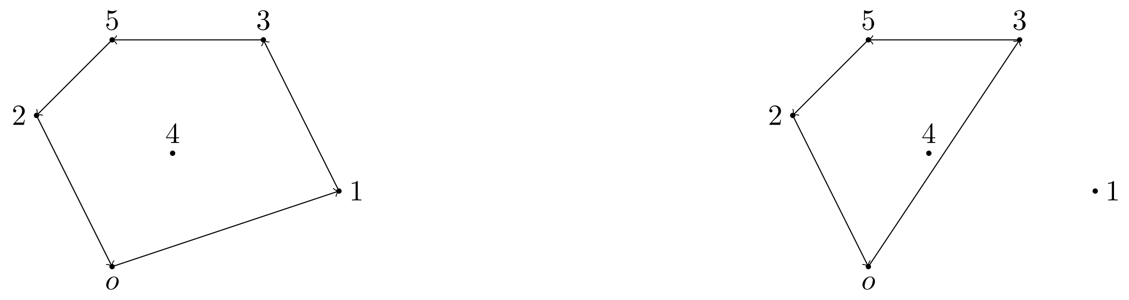
3.1 Best possible 2-competitive online algorithm

In the proposed algorithm, the online server makes use of the offline black box only when at the origin, and, any time at the origin, waits an appropriate amount of time (to be defined) before it starts on a new route. While engaged on a route, the server ignores all new requests. We label this algorithm WOGI for “Wait, Optimize, Go, and Ignore”. In the sense that this algorithm ignores all additional requests while en route, this "stubborn" behavior is shared with algorithms which have been proposed for other online routing problems, such as the online traveling repairman problem (see in particular the INTERVAL algorithm of [15], and the BREAK algorithm of [11]).

Before presenting the algorithm in details, let us first define some notations. We use i as a counter indicating how many times the online server has left the origin. We let u_i be the number of requests that have been released so far when the online server leaves the origin for the i^{th} time. We let P_i be the set of all requests among the first u_i requests that have not been served by the online server when it returns to the origin for the i^{th} time. We let s_i be the first request, not among the first u_{i-1} requests, which the online server visits on route τ_{u_i} (so $s_i > u_{i-1}$).

Our algorithm is designed in such a way that when the online server leaves the origin for the i^{th} time, it has two candidate routes to follow. It either follows the route τ_{u_i} exactly, or it uses a WOGI shortcut $\tau_{u_i,u_{i-1}}$, defined as follows: it skips the first requests on τ_{u_i} whose indices are no greater than u_{i-1}, goes directly to request s_i, and then follows the remaining part of τ_{u_i}.

Figure 1 below provides an illustration of a WOGI shortcut. In this example, five requests



route τ_5 by black box

WOGI shortcut $\tau_{5,2}$

Figure 1: WOGI Shortcuts

are released sequentially. The route τ_5 on the left, computed by the black box, passes through requests $1, 5, 3$, and 2. According to the definitions above, $s_5 = 3$. As showed on the right, the WOGI shortcut $\tau_{5,2}$ skips request 1 and goes directly to request 3. Note that no request is skipped afterwards. Assume that the server have returned to the origin once and request 1 has not been served, then $P_2 = \{1, 4\}$. Even if some requests are released shortly after the server leaves the origin to follow $\tau_{5,2}$, they are not included in P_2.

We are now ready to provide a full description of Algorithm WOGI:

Algorithm 1 (WOGI)**.**

0. Initialization: counter $i = 0$, $u_0 = 0$, *and* $P_0 = \emptyset$.

1. Assume k *requests have been released. If* $S_k \subset \{1, 2, ..., u_i\}$, *wait for the next released request, and go to Step 1; otherwise, go to Step 2.*

2. Assume k *requests have been released, the WOGI server waits until* $\max\{C_{\text{opt}}(k), t_{k,u_i}\}$, *where* $t_{k,u_i} \doteq 2C_{\text{opt}}(k) - L(\tau_{k,u_i}) - \sum_{j>u_i, j\notin S_k} p_j - \sum_{j\in P_i} p_j$ *is the latest time to leave the origin* *to follow τ_{k,u_i} and maintain a competitive ratio of 2. If a new request is released during the waiting time, go to Step 1; otherwise, update $u_{i+1} = k, i = i+1$, and go to Step 3.*

3. *The WOGI server takes one of the two routes and ignores all new requests before reaching back the origin:*

   3a. *If $t_{u_i,u_{i-1}} \geq C_{\text{opt}}(u_i)$, he follows the WOGI shortcut $\tau_{u_i,u_{i-1}}$. After finishing the route, go to Step 4.*

   3b. *If $t_{u_i,u_{i-1}} < C_{\text{opt}}(u_i)$, he follows τ_{u_i}. After finishing the route, go to Step 4.*

4. *Update P_i. Go to Step 1.*

Let us first look at some properties of WOGI:

Lemma 1. *$r_{s_{i+1}} \geq C_{\text{opt}}(u_i)$.*

Proof. If $i = 0$, $C_{\text{opt}}(0) = 0$. Thus, the lemma is trivially true. If $i > 0$, let t be the time when the WOGI server leaves the origin for the i^{th} time. According to Step 2, $t \geq C_{\text{opt}}(u_i)$. On the other hand, at time t, only u_i requests are released. Thus, $\forall j > u_i$, $r_j \geq t \geq C_{\text{opt}}(u_i)$. In particular, it is true for $j = s_{i+1} > u_i$. $\square$

Lemma 2. *$t_{u_{i+1},u_i} \geq 2C_{\text{opt}}(u_i) - \sum_{j\in P_i} p_j$.*

Proof. If $i = 0$, $C_{\text{opt}}(0) = 0$. The lemma is trivially true. If $i > 0$, the offline server cannot visit request s_{i+1} before its release time $r_{s_{i+1}}$. Thus, $T_{u_{i+1}} \geq r_{s_{i+1}} + L(\tau_{u_{i+1},u_i}) - |l_{s_{i+1}}|$. Therefore,

$$\begin{aligned}
t_{u_{i+1},u_i} &= 2C_{\text{opt}}(u_{i+1}) - L(\tau_{u_{i+1},u_i}) - \textstyle\sum_{j>u_i, j\notin S_{u_{i+1}}} p_j - \sum_{j\in P_i} p_j \\
&= 2T_{u_{i+1}} + 2\textstyle\sum_{1\leq j\leq u_{i+1}, j\notin S_{u_{i+1}}} p_j - L(\tau_{u_{i+1},u_i}) - \sum_{j>u_i, j\notin S_{u_{i+1}}} p_j - \sum_{j\in P_i} p_j \\
&\geq 2r_{s_{i+1}} + 2L(\tau_{u_{i+1},u_i}) - 2|l_{s_{i+1}}| + 2\textstyle\sum_{1\leq j\leq u_{i+1}, j\notin S_{u_{i+1}}} p_j - L(\tau_{u_{i+1},u_i}) \\
&\quad - \textstyle\sum_{j>u_i, j\notin S_{u_{i+1}}} p_j - \sum_{j\in P_i} p_j \\
&\geq 2r_{s_{i+1}} + L(\tau_{u_{i+1},u_i}) - 2|l_{s_{i+1}}| - \textstyle\sum_{j\in P_i} p_j \\
&\geq 2r_{s_{i+1}} - \textstyle\sum_{j\in P_i} p_j.
\end{aligned}$$

According to Lemma 1, $r_{s_{i+1}} \geq C_{\text{opt}}(u_i)$, we conclude $t_{u_{i+1},u_i} \geq 2C_{\text{opt}}(u_i) - \sum_{j\in P_i} p_j$. $\square$

Lemma 3. *The WOGI server returns to the origin for the i^{th} time before $2C_{\text{opt}}(u_i) - \sum_{j\in P_i} p_j$.*

Proof. We use induction on i to prove this lemma. It is trivially true for $i = 0$.

Consider i. By induction, the server finishes his $(i-1)^{\text{th}}$ trip before $2C_{\text{opt}}(u_{i-1}) - \sum_{j\in P_{i-1}} p_j$. According to Lemma 2, $2C_{\text{opt}}(u_{i-1}) - \sum_{j\in P_{i-1}} p_j \leq t_{u_i,u_{i-1}}$. Thus, the WOGI is at the origin at $\max\{t_{u_i,u_{i-1}}, C_{\text{opt}}(u_i)\}$. According to Step 2, he leaves the origin for the i^{th} time at exactly $\max\{t_{u_i,u_{i-1}}, C_{\text{opt}}(u_i)\}$. Based on which of the two is larger, we have two cases:

1. If $t_{u_i,u_{i-1}} \geq C_{\text{opt}}(u_i)$, then the WOGI server will take the shortcut τ_{u_{i+1},u_i}, and arrive at the origin at time $t_{u_i,u_{i-1}} + L(\tau_{u_{i+1},u_i}) = 2C_{\text{opt}}(u_i) - \sum_{u_{i-1}<j\leq u_i, j\notin S_{u_i}} p_j - \sum_{j\in P_{i-1}} p_j \leq 2C_{\text{opt}}(u_i) - \sum_{j\in P_i} p_j$. The last inequality is due to $P_i \subset P_{i-1} \cup \{j : u_{i-1} < j \leq u_i, j \notin S_{u_i}\}$.

2. If $t_{u_i,u_{i-1}} < C_{\text{opt}}(u_i)$, then the server will follow τ_{u_i}, and arrive at the origin at time $C_{\text{opt}}(u_i) + L(\tau_{u_i}) \leq C_{\text{opt}}(u_i) + T_{u_i} = 2C_{\text{opt}}(u_i) - \sum_{j\notin S_{u_i}} p_j \leq 2C_{\text{opt}}(u_i) - \sum_{j\in P_i} p_j$. The last inequality is due to $P_i \subset S^c_{u_i}$, because τ_{u_i} pass through all requests in S_{u_i}.

□

We now can prove our main result:

Theorem 1. *Algorithm WOGI is 2-competitive and best possible.*

Proof. Assume there are a total of m requests and the WOGI server leaves and returns to the origin i times. According to Lemma 3, after the i^{th} trip, the server returns to the origin before time $2C_{\text{opt}}(u_i) - \sum_{j \in P_i} p_j$ and never leaves again. Therefore, total cost $C_{WOGI} \leq 2C_{\text{opt}}(u_i) - \sum_{j \in P_i} p_j + \sum_{j \in P_i} p_j + \sum_{j=u_i+1}^{m} p_j = 2C_{\text{opt}}(u_i) + \sum_{j=u_i+1}^{m} p_j$. Since the WOGI server does not leaves the origin afterwards, $\forall u_i + 1 \leq j \leq m, j \notin S_m$. Thus, $C_{WOGI} \leq 2C_{\text{opt}}(u_i) + \sum_{j=u_i+1}^{m} p_j = 2C_{\text{opt}}(m) - \sum_{j=u_i+1}^{m} p_j \leq 2C_{\text{opt}}(m)$. □

3.2 Polynomial-time algorithms

WOGI repeatedly calls a black box that provides optimal solutions to corresponding offline problems. However, because these offline problems are NP-hard, WOGI can't be considered to be a polynomial-time algorithm, and this makes WOGI impractical for very large size problems. To address the complexity issue, we propose here a polynomial time algorithm, WOGI-apx, at the expense of an increase in the competitive ratio. WOGI-apx is simply the analog of WOGI with an approximation black box. Instead of solving offline TSPs optimally, WOGI-apx uses a ρ-approximation black box algorithm. Noting that, other than optimality, few properties of offline solutions are used in proving 2-competitiveness of WOGI, we expect the analysis in Section 3.1 to carry through for WOGI-apx.

Before presenting and analyzing WOGI-apx, let us first define some notations, which are analogs of the ones used for WOGI. Given the instance I_k, the offline approximation algorithm provides a solution that has cost $\tilde{C}_{\text{apx}}(k) \leq \rho C_{\text{opt}}(k)$. Let $\tilde{T}_k$ be the makespan of the approximation solution, $\tilde{\tau}_k$ be the corresponding route, and $\tilde{S}_k$ be the set of requests served by the approximation solution. u_i is the number of released requests when the online server leaves the origin for the i^{th} time. The rejection set P_i is the set of requests that have not been served when the online server returns to the origin for the i^{th} time. Request s_i is the first request on the route $\tilde{\tau}_{u_i}$ whose index is strictly greater than u_{i-1}. When the online server leaves the origin for the i^{th} time, it has two candidate routes to follow. It either follows $\tilde{\tau}_{u_i}$, computed by the approximation solver, or a WOGI shortcut $\tilde{\tau}_{u_i, u_{i-1}}$. The WOGI shortcut $\tilde{\tau}_{u_i, u_{i-1}}$ skips the first few requests on $\tilde{\tau}_{u_i}$ whose indices are no greater than u_{i-1}, goes directly to request s_{i-1}, and then follows the remaining fraction of $\tilde{\tau}_{u_i}$.

Now we can present WOGI-apx:

Algorithm 2 (WOGI-apx)**.**

0. *Initialization: counter $i = 0$, $u_0 = 0$, and $P_0 = \emptyset$.*

1. *Assume k requests have been released. If $\tilde{S}_k \subset \{1, .., u_i\}$, wait for the next released request, and go to Step 1; otherwise, go to Step 2.*

2. *Assume k requests have been released, the WOGI server waits until* $\max\{\tilde{C}_{\text{apx}}(k), t_{k,u_i}\}$*, where* $t_{k,u_i} \doteq 2\tilde{C}_{\text{apx}}(k) - L(\tilde{\tau}_{k,u_i}) - \sum_{j>u_i, j\notin\tilde{S}_k} p_j - \sum_{j\in P_i} p_j$*. If a new request is released during the waiting time, go to Step 1; otherwise, update* $u_{i+1} = k, i = i+1$*, and go to Step 3.*

3. *The WOGI server takes one of the two routes and ignores all new requests before reaching back the origin:*

   3a. *If* $t_{u_i,u_{i-1}} \geq \tilde{C}_{\text{apx}}(u_i)$*, he follows the WOGI shortcut* $\tilde{\tau}_{u_i,u_{i-1}}$*. After finishing the route, go to Step 4.*

   3b. *If* $t_{u_i,u_{i-1}} < \tilde{C}_{\text{apx}}(u_i)$*, he follows* $\tilde{\tau}_{u_i}$*. After finishing the route, go to Step 4.*

4. *Update* P_{i+1}*.* $i = i+1$*. Go to Step 1.*

As Lemma 1 through 3 use no property of the offline solutions from the black box, their analogs for the approximated version are also valid.

Lemma 4. $r_{s_{i+1}} \geq \tilde{C}_{\text{apx}}(u_i)$.

Lemma 5. $t_{u_{i+1},u_i} \geq 2\tilde{C}_{\text{apx}}(u_i) - \sum_{j\in P_i} p_j$.

Lemma 6. *The WOGI-apx server finishes his* i^{th} *trip before* $2\tilde{C}_{\text{apx}}(u_i) - \sum_{j\in P_i} p_j$.

We omit the proofs because they are the same as the ones for Lemma 1 through 3. We are now ready to prove our main result. The proof is different from the one of Theorem 1, because unlike $C_{\text{opt}}(k)$, $\tilde{C}_{\text{apx}}(k)$ is not necessarily non-decreasing in k.

Theorem 2. *Algorithm WOGI-apx is* 2ρ*-competitive.*

Proof. Assume there are a total of m requests, and the WOGI-apx server leaves and returns to the origin i times. According to Lemma 6, after the i^{th} trip, the server returns to the origin before time $2\tilde{C}_{\text{apx}}(u_i) - \sum_{l\in P_i} p_l$ and never leaves again. Therefore, total cost

$$\begin{array}{rcl} C_{\text{WOGI-apx}} & \leq & 2\tilde{C}_{\text{apx}}(u_i) - \sum_{j\in P_i} p_j + \sum_{j\in P_i} p_j + \sum_{j=u_i+1}^{m} p_j \\ & = & 2\tilde{C}_{\text{apx}}(u_i) + \sum_{j=u_i+1}^{m} p_j. \end{array}$$

If $\forall u_i + 1 \leq j \leq m, j \notin S_m$, i.e. none of these requests are served in the optimal offline solution, then $C_{\text{opt}}(m) = C_{\text{opt}}(u_i) + \sum_{j=u_i+1}^{m} p_j$. Therefore,

$$\begin{array}{rcl} C_{\text{WOGI-apx}} & \leq & 2\tilde{C}_{\text{apx}}(u_i) + \sum_{j=u_i+1}^{m} p_j \\ & \leq & 2\rho C_{\text{opt}}(u_i) + \sum_{j=u_i+1}^{m} p_j \leq 2\rho C_{\text{opt}}(m). \end{array}$$

If $\exists u_i + 1 \leq j \leq m$, such that $j \in S_m$, then $C_{\text{opt}}(m) \geq r_j \geq \tilde{C}_{\text{apx}}(u_i)$, where the last inequality is due to Step 2 in the WOGI-apx. Because of Step 1 of the algorithm, $\tilde{C}_{\text{apx}}(m) = \tilde{C}_{\text{apx}}(u_i) + \sum_{j=u_i+1}^{m} p_j \geq C_{\text{opt}}(u_i) + \sum_{j=u_i+1}^{m} p_j$, we have $\rho C_{\text{opt}}(m) \geq \tilde{C}_{\text{apx}}(m) \geq C_{\text{opt}}(u_i) + \sum_{j=u_i+1}^{m} p_j$. Therefore,

$$\begin{array}{rcl} C_{\text{WOGI-apx}} & \leq & 2\tilde{C}_{\text{apx}}(u_i) + \sum_{j=u_i+1}^{m} p_j \\ & \leq & (2 - 1/\rho)\tilde{C}_{\text{apx}}(u_i) + C_{\text{opt}}(u_i) + \sum_{j=u_i+1}^{m} p_j \\ & \leq & (2 - 1/\rho)C_{\text{opt}}(m) + \rho C_{\text{opt}}(m) \\ & \leq & \rho C_{\text{opt}}(m) + \rho C_{\text{opt}}(m) = 2\rho C_{\text{opt}}(m). \end{array}$$

From the discussion above, we can conclude that WOGI-apx is 2ρ-competitive.

□

3.3 A prize-collecting generalization

The prize collecting TSP (PCTSP) is a generalization of the TSP (see [6, 8]), where associated with each request is a penalty (if rejected) and a prize (if accepted and served). The server must collect enough prizes exceeding a given quota while minimizing the makespan needed to collect the prizes plus the total penalty of rejected requests. We consider the online version of this problem.

<u>**Online PCTSP**</u>

Instance: A metric space $\mathcal{M}$ with a given origin o and a distance metric $d(\cdot,\cdot)$. A series of n requests represented by quadruples $(l_i, r_i, p_i, w_i)_{1\leq i\leq n}$, where $l_i \in \mathcal{M}$ is the location (point in metric space) of request i, $r_i \in \mathbb{R}_+$ its release date (first time after which it can be served), $p_i \in \mathbb{R}_+$ its penalty (for not being served), and $w_i \in \mathbb{R}_+$ its prize (collected if served). A parameter $W_{min} \in \mathbb{R}_+$ (a quota for prizes to be collected). The problem begins at time 0; the server is initially idle at the origin (initial state), can travel at unit speed (when not idle), and eventually must be back and idle at the origin (final state). The earliest time the server reaches this final state is called the makespan.

Feasible solution: Any subset $\mathcal{S} \subset \{1,\ldots,n\}$ of requests to be served and a feasible TSP tour with release dates $\tau(\mathcal{S})$ through $\mathcal{S}$ so that $\sum_{i\in\mathcal{S}} w_i \geq W_{min}$.

Offline context: The number of requests n is known to the offline server. All requests are revealed to the offline server at time 0.

Online context: The number of requests n is not known to the online server. Requests are revealed to the online server at their release dates $r_i \geq 0$; assume $r_1 \leq r_2 \cdots \leq r_n$. The online server can accept or reject a request any time after the request's release date.

Objective: In all cases, minimize {the makespan to serve all accepted requests plus the total penalties of all rejected requests} among all feasible solutions.

Assume there is a black box that provides the optimal offline solution for PCTSP. For the simplicity of the algorithm, let us assume that if there is no feasible solution for the offline problem, no request will be accepted. Let us replace the black box in WOGI by the black box for PCTSP; and the resulting algorithm is WOGI-PC. Since in the proofs of Lemma 1, 2, 3 and Theorem 1, no property other than the optimality of the solution provided by the black box is used, these results are also valid for WOGI-PC. In particular,

Theorem 3. *Algorithm WOGI-PC is 2-competitive for PCTSP.*

Similarly, assume there is a black box that provides ρ-approximation offline solution for PCTSP. Replacing the black box in WOGI-apx by an approximation black box for PCTSP results in algorithm WOGI-PC-apx. Similar to the argument above, we have:

Theorem 4. *Algorithm WOGI-PC-apx is 2ρ-competitive for PCTSP.*

4 Real-time Version of the Online Problem

In this section, we consider the real-time version of our online problem. The decision to accept or reject a given request must be made immediately upon its arrival.

4.1 The case of the non-negative real-line $\mathbb{R}_+$

We first study the problem when the locations of the requests are all on the non-negative real line, equipped with the traditional Euclidean distance. In that case, the notation for the location of a request i, l_i, will also represent the distance from the origin to the point.

4.1.1 Lower bound on competitive ratios

Theorem 5. *Any c-competitive online algorithm on $\mathbb{R}_+$ has $c \geq 2.5$.*

Before proving the theorem, let us first present how we construct instances for Theorem 5. The same idea is used for Theorem 7 later. A series of requests with small penalties with the same location and almost the same release date are released until the online algorithm accept one such request. The online algorithm then faces a dilemma on whether to accept or reject such requests. On the one hand, accepting one such request too early may not be beneficial, because the offline solution only pays a small amount of penalties while the online solution must spend some time to visit the accepted request; on the other hand, accepting one too late may not be beneficial either, because the offline solution can simply accept and visit all these requests while the online solution must visit the accepted request and pay a large amount of penalties for rejected ones. Such a dilemma leads to large competitive ratios. In the proofs of Theorem 5 and Theorem 7, penalties, locations and release dates are chosen carefully to take advantage of the dilemma.

Proof of Theorem 5. Assume that an online server follows a c-competitive online algorithm, where c is a finite constant. Let $c_0 = 2.5$ and ϵ be a small positive number. For an arbitrary integer n, consider a series of up to $n+1$ requests as follows: $(l_i, r_i, p_i) = (1, 1+i\epsilon, 3/c_0^i)$ for $1 \leq i \leq n$, and $(l_{n+1}, r_{n+1}, p_{n+1}) = (1, 1+(n+1)\epsilon, \infty)$.

Let t_0 be the time when the online server begins to move away from the origin for the first time. If $0 \leq t_0 < r_1$, no request would be presented. As a result, $C_A > 0$ and $C_{\text{opt}} = 0$, which contradicts with the assumption that algorithm A is finite competitive. Thus, $t_0 \geq r_1$. Since the last possible request has an infinite penalty, any finite competitive online algorithm cannot reject all requests. Let request $m (1 \leq m \leq n+1)$ be the first request that is accepted by A. After the online server accepts request m, no more request is presented. We now consider two cases to compute the competitive ratio:

1. If $m \leq n$, then the optimal solution is to reject all m requests. In this case, $C_{\text{opt}}(m) = \sum_{i=1}^{m} p_i = \frac{3(c_0^m - 1)}{(c_0-1)c_0^m}$, and $C_A(m) \geq r_1 + 2l_m + \sum_{i=1}^{m-1} p_i \geq \frac{3(c_0^m-1)}{(c_0-1)c_0^{m-1}}$. Thus, $c \geq \frac{C_A(m)}{C_{\text{opt}}(m)} = c_0 = 2.5$.
2. If $m = n+1$, then the optimal solution is to accept all $n+1$ requests. In this case, $C_{\text{opt}}(n+1) = r_{n+1} + l_{n+1} = 2 + (n+1)\epsilon$, and $C_A(n+1) \geq r_1 + 2l_{n+1} + \sum_{i=1}^{n} p_i = \epsilon + \frac{3(c_0^{n+1}-1)}{(c_0-1)c_0^n}$. Using the

fact that $c_0 = 2.5$, $\epsilon + \frac{3(c_0^{n+1}-1)}{(c_0-1)c_0^n} = \epsilon + 5 - 2/c_0^n$. Thus, $c \geq \frac{C_A(n+1)}{C_{\text{opt}}(n+1)} \geq \frac{5-2/c_0^n+\epsilon}{2+(n+1)\epsilon}$. By letting $\epsilon = 1/(n+1)^2$, and $n \to +\infty$, we have $\frac{5-2/c_0^n+\epsilon}{2+(n+1)\epsilon} \to 2.5$. Thus, $c \geq 2.5$.

□

4.1.2 An optimal 2.5-competitive online algorithm

The algorithm we propose here is an extension of the "move right if necessary" (MRIN) algorithm introduced in Blom et al. [7] for the online TSP without rejection options. The acceptance/rejection decisions are based on the offline optimal solutions. Let us call this algorithm the "estimate and move right if necessary" (EMRIN) algorithm:

Algorithm 3 (EMRIN).

1. *Whenever a new request m comes, if $m \in S_m$, accept it; otherwise reject it.*

2. *If there is an accepted and unserved request on the right side of the server, move toward it.*

3. *If there are no accepted and unserved requests on the right side of the server, move back toward the origin. Upon reaching the origin, become idle.*

First, let us show that the total penalty of rejected requests is not large:

Lemma 7. *$\forall m$, $C_{\text{opt}}(m) \geq \sum_{i \leq m, i \notin S_i} p_i$.*

Proof. We use induction on m. When $m = 1$, if $1 \in S_1$, $C_{\text{opt}}(1) \geq 0 = \sum_{i \leq 1, i \notin S_i} p_i$; otherwise, $C_{\text{opt}}(1) = p_1 = \sum_{i \leq 1, i \notin S_i} p_i$. Therefore, the assertion is true. Assume now that the assertion holds for $m-1$, and let us consider m. If $m \in S_m$, $C_{\text{opt}}(m) \geq C_{\text{opt}}(m-1) \geq \sum_{i \leq m-1, i \notin S_i} p_i = \sum_{i \leq m, i \notin S_i} p_i$; otherwise, $C_{\text{opt}}(m) = C_{\text{opt}}(m-1) + p_m \geq \sum_{i \leq m-1, i \notin S_i} p_i + p_m = \sum_{i \leq m, i \notin S_i} p_i$. □

Then, let us prove our main result:

Theorem 6. *Algorithm 3 (EMRIN) is 2.5-competitive, and thus best possible.*

Proof. We will use induction on the number of released requests m. When $m = 1$, if $1 \notin S_1$, $C_A(1) = C_{\text{opt}}(1)$; otherwise, $C_A(1) = r_1 + 2l_1$, $C_{\text{opt}}(1) = \max\{r_1 + l_1, 2l_1\}$, and thus $\frac{C_A(1)}{C_{\text{opt}}(1)} \leq 1.5$. Assume now that the assertion holds for $m-1$, and let us consider m:

1. If $m \notin S_m$, $C_A(m) = C_A(m-1) + p_m \leq 2.5C_{\text{opt}}(m-1) + p_m \leq 2.5(C_{\text{opt}}(m-1) + p_m) = 2.5C_{\text{opt}}(m)$. Thus, $c \leq 2.5$.
2. If $m \in S_m$, assume request k is the rightmost request that is accepted but not be served at time r_m; assume x is the position of the online server at r_m.
   - 2a. If $x \leq l_k$, because request k is an accepted and unserved request, the online server has been moving right since time r_k. Hence, the online server will return to the origin no later than $r_k + 2l_k$. Therefore, $C_A(m) \leq r_k + 2l_k + \sum_{i \leq m, i \notin S_i} p_i$. Because $k \in S_k$, $C_{\text{opt}}(m) \geq C_{\text{opt}}(k) \geq \max\{r_k + l_k, 2l_k\}$. As a result, we conclude $\frac{C_A(m)}{C_{\text{opt}}(m)} \leq \frac{r_k+l_k}{C_{\text{opt}}(k)} + \frac{l_k}{C_{\text{opt}}(k)} + \frac{\sum_{i \leq m, i \notin S_i} p_i}{C_{\text{opt}}(m)} \leq 1 + 0.5 + 1 = 2.5$.
   - 2b. If $x \geq l_k$, no extra time is needed to serve request k, because it can be served on the online's way back to the origin. Thus, $C_A(m) = C_A(m-1) \leq 2.5C_{\text{opt}}(m-1) \leq 2.5C_{\text{opt}}(m)$.

□

4.2 The case of the real-line $\mathbb{R}$

In this section, we study the problem when the locations of the requests are on the real line, equipped with the traditional Euclidean distance. On the positive ("right") side of the line, the notation for the location of a request i, l_i, will also represent the distance from the origin to the point. On the negative ("left") side of the line, the location of a request i, l_i will be given by a negative number, and the distance from the origin to the point will be its absolute value $|l_i|$.

4.2.1 Lower bounds on competitive ratios

General lower bound

Theorem 7. *Any c-competitive online algorithm on $\mathbb{R}$ has $c \geq \frac{17+\sqrt{17}}{8} \approx 2.64$.*

Proof. Assume that an online server A follows a given c-competitive online algorithm. For any given $\varepsilon > 0$, there exists $N \in \mathbb{N}$, such that $N\varepsilon > c(4+2\varepsilon)$, and there also exists $M \in \mathbb{N}$, such that $15c - 8 < \varepsilon c^{M-1}$.

Again we use the idea mentioned in Section 4.1.1 to construct an example. In this example, three series of requests are presented.

First, consider a series of up to N requests as follows: $(l_i, r_i, p_i) = (1, 1 + \frac{i\varepsilon}{N}, \varepsilon)$ for $1 \leq i \leq N$. Note that A cannot reject them all. Otherwise, the cost will be $N\varepsilon > c(4+2\varepsilon)$, while the optimal cost is at most $2+\varepsilon$, which is a contradiction. Assume the first request accepted by A is n_1. Truncate the first series: only the first n_1 requests are presented.

Consider a second series of up to N requests as follows: $(l_{n_1+i}, r_{n_1+i}, p_{n_1+i}) = (-1, 1+\varepsilon+\frac{i\varepsilon}{N}, \varepsilon)$, for $1 \leq i \leq N$. Note that A cannot reject them all. Otherwise, the cost will be at least $N\varepsilon > c(4+2\varepsilon)$, while the optimal cost is at most $4+2\varepsilon$, which is a contradiction. Assume the first request in this second series accepted by A is n_1+n_2. Truncate the second series: only the first n_2 requests in the second series are presented. Let $P_1 = (n_1-1)\varepsilon$, $P_2 = (n_2-1)\varepsilon$, $a_1 = \min\{2+2\varepsilon, P_1+\varepsilon\}$, and $a_2 = \min\{2+2\varepsilon, P_2+\varepsilon\}$.

At time $1+\varepsilon+\frac{n_2\varepsilon}{N}$, A has two requests at ± 1 to visit. Without loss of generality, assume A is visiting 1 before visiting -1. Assume t_0 is the first time when A is at the origin after visiting 1. Note that the optimal cost is at most a_1+a_2. In order to be c-competitive, we have $3 \leq t_0 \leq (c-1)(a_1+a_2)-2 \leq 5c-2$.

Consider a third series of up to M requests as follows: $l_{n_1+n_2+i} = t_0 - 2$, $r_{n_1+n_2+i} = t_0 + \frac{i\varepsilon}{M+1}$, $p_{n_1+n_2+i} = \max\{0, \frac{3t_0-2-(c-1)(P_1+P_2)-(2c+1)\varepsilon}{c^i}\}$, for $1 \leq i \leq M$. We claim that A has to reject all these M requests. Otherwise, assume the first one accepted is n_1+n_2+m; then truncate the third series so that only the first m requests in the third series are presented. The online server's cost is at least $3t_0 - 2 + P_1 + P_2 + \sum_{i=1}^{m-1} p_{n_1+n_2+i} \geq c(a_1+a_2+\sum_{i=1}^{m} p_{n_1+n_2+i}) + \varepsilon$, while optimal cost is at most $a_1+a_2+\sum_{i=1}^{m} p_{n_1+n_2+i}$, which is a contradiction.

Then, we present the last request $(l_{n_1+n_2+M+1}, r_{M+1+n_1+n_2}, p_{M+1+n_1+n_2}) = (t_0-2, t_0+\varepsilon, \infty)$. Because of its infinite penalty, A has to accept this request. Thus, online server's cost is at least $3t_0 - 2 + P_1 + P_2 + \sum_{i=1}^{M} p_{i+n_1+n_2}$. Noting that $\sum_{i=1}^{M} p_{i+n_1+n_2} = p_{1+n_1+n_2}\sum_{i=1}^{M}\frac{1}{c^{i-1}} = \frac{c}{c-1}p_{1+n_1+n_2} - \frac{p_{1+n_1+n_2}}{c^M - c^{M-1}}$, $p_{1+n_1+n_2} \leq 3t_0 - 2 \leq 15c-8, c^M - c^{M-1} > c^{M-1}$, and $15c - 8 < c^{M-1}\varepsilon$, we then have $\geq 3t_0 - 2 + P_1 + P_2 + \frac{3t_0-2-(2c+1)\varepsilon}{c-1} - (P_1+P_2) - \varepsilon = \frac{c(3t_0-2)}{c-1} - \frac{3c\varepsilon}{c-1}$. The optimal cost is at most $2t_0 - 2 + \varepsilon$. Consequently, $c \cdot (2t_0 - 2 + \varepsilon) \geq \frac{c(3t_0-2)}{c-1} - \frac{3c\varepsilon}{c-1} \Rightarrow 2c - 4 - (c+2)\varepsilon \leq (2c-5)t_0 \leq (2c-5)((c-1)(a_1+a_2)-2) \leq (2c-5)(4c+(4c_4)\varepsilon) - 6$. By letting $\varepsilon \to 0$, we conclude $2c - 4 \leq (2c-5)(4c-6) \Rightarrow c \geq \frac{17+\sqrt{17}}{8}$. $\square$

4.2.2 A 3-competitive online algorithm

This algorithm uses a different offline subroutine, hereafter called black box 2, that solves a variant of the offline problem where the server starts initially at a point x that may be different from the origin.

Algorithm 4 (ReOpt)**.**

1. *Whenever a new request m comes, if $m \in S_m$, then accept it; otherwise, reject it.*
2. *At any time when a new request is accepted, reoptimize (using black box 2) and follow the corresponding new optimal route to serve all accepted and unserved requests.*

First, let us show that the total penalty of rejected requests is not very large. Consider if only the first k requests are released, let $L_k = \min_{1 \le i \le k, i \in S_k} \{l_i, 0\}$ be the leftmost accepted request and $R_k = \max_{1 \le i \le k, i \in S_k} \{l_i, 0\}$ be the rightmost accepted request. Then,

Lemma 8. *If $k \in S_k$ and $l_k < 0$, then $\forall l \in (l_k, 0]$, $\sum_{i:l_i<l} p_i \ge l - l_k$.*

Proof. Consider the route τ_k that the offline server follows if only the first k requests are released. Let $\tau_k(t)$ be the server's position at time t. Assume that request k is served at time $t_0(\ge r_k)$. Let $t_1 = \max\{t : \tau_k(t) = l, t < t_0\}$ and $t_2 = \min\{t : \tau_k(t) = l, t > t_0\}$. Consider another feasible solution:

$$\tau'_k(t) = \begin{cases} \tau_k(t), & t \le t_1 \\ l, & t_1 < t \le t_0 \\ \tau_k(t - (t_2 - t_0)), & t \ge t_0 \end{cases}.$$

Since both solutions have the same motion before t_1, all requests served by τ_k before t_1 are also served by the new solution. Furthermore, because the interval covered by τ_k after t_2 is also covered by τ'_k after t_0 and all the first k requests are released before t_0, every request that is served by τ_k after t_2 will be served by τ'_k. Therefore, all requests served by τ_k but not by τ'_k are served by τ_k between t_1 and t_2. Because of the definition of t_1 and t_2, all those requests are located beyond l. Thus, τ_k saves at most $\sum_{i:l_i<l} p_i$ on penalties and spends $t_2 - t_0$ more units of time on traveling. From τ_k, we have $\sum_{i:l_i<l} p_i \ge t_2 - t_0 \ge l - l_k$, where the last inequality is due to the unit speed of the offline server. □

By symmetry, we also have:

Lemma 9. *If $k \in S_k$ and $l_k > 0$, then $\forall l \in [0, l_k)$, $\sum_{i:l_i>l} p_i \ge l_k - l$.*

Then we are ready to prove the competitive ratio of ReOpt:

Theorem 8. *Algorithm 4 (ReOpt) is 3-competitive.*

Proof. Assume n requests are released:

1. If $n \in S_n$. Both the online and offline servers accept request n. Let $L_{\text{on}} = \min\{0, l_i : i \in S_i\}$ be the leftmost request accepted by the online server, $R_{\text{on}} = \max\{0, l_i : i \in S_i\}$ be the rightmost request accepted by the online server, $L_{\text{off}} = \min\{0, l_i : i \in S_n\}$ be the leftmost request accepted by the offline server, and $R_{\text{off}} = \max\{0, l_i : i \in S_n\}$ be the rightmost request accepted by the offline server. From the description of ReOpt, the online server never

moves beyond interval $[L_{on}, R_{on}]$. In Lemma 9, let $k = \text{argmax}_i\{0, l_i : i \in S_i\}$, we have $R_{on} \leq R_{\text{off}} + \sum_{i:l_i > R_{\text{off}}} p_i$. Similarly, $-L_{on} \leq -L_{\text{off}} + \sum_{i:l_i < L_{\text{off}}} p_i$. Therefore, the online server serves all accepted requests and returns to the origin no later than $r_n - 2L_{\text{on}} + 2R_{\text{on}}$. Thus,

$$\begin{aligned}
C_{\text{ReOpt}}(n) &\leq r_n - 2L_{\text{on}} + 2R_{\text{on}} + \textstyle\sum_{i \leq n, i \notin S_i} p_i \\
&\leq r_n + (-2L_{\text{off}} + 2\textstyle\sum_{i:l_i < L_{\text{off}}} p_i) + (2R_{\text{off}} + 2\textstyle\sum_{i:l_i > R_{\text{off}}} p_i) + \textstyle\sum_{i \leq n, i \notin S_i} p_i \\
&\leq (r_n + \textstyle\sum_{i \notin S_n} p_i) + (2R_{\text{off}} - 2L_{\text{off}} + \textstyle\sum_{i \notin S_n} p_i) + \textstyle\sum_{i \leq n, i \notin S_i} p_i \\
&\leq 3C_{\text{opt}}(n).
\end{aligned}$$

2. If $\forall i \in \{1, 2, \cdots, n\}, i \notin S_i$, then $C_{\text{ReOpt}}(n) = \sum_{i=1}^n p_i = C_{\text{opt}}(n)$.

3. Assume m is the last request such that $m \in S_m$. According to Case 1, $C_{\text{ReOpt}}(m) \leq 3C_{\text{opt}}(m)$. Since, $i \notin S_i$ for all $i \in [m+1, n]$, we have $C_{ReOpt}(i) = C_{ReOpt}(i-1) + p_i$, which is due to the optimality of $C_{ReOpt}(i)$ and $C_{ReOpt}(i-1)$. Therefore, $C_{ReOpt}(n) = C_{ReOpt}(m) + \sum_{i=m+1}^n p_i \leq 3C_{\text{opt}}(m) + \sum_{i=m+1}^n p_i \leq 3(C_{\text{opt}}(m) + \sum_{i=m+1}^n p_i) = 3C_{\text{opt}}(n)$.

□

Note that 3 is a tight competitive ratio for ReOpt as the following example illustrates. Let ϵ be an arbitrarily small positive number, k be an arbitrary large integer, and let the instance consist of the following $2k+3$ requests: $(l_1, r_1, p_1) = (1, 0, 2 - 1/k)$, $(l_2, r_2, p_2) = (-2/k, \epsilon/k, \infty)$, $(l_3, r_3, p_3) = (1, 1/k, \infty)$, $(l_i, r_i, p_i) = (-2/k, (i-1)/k, \infty)$, $4 \leq i \leq 2k+3$. It is easy to check that $C_{ReOpt} = 6 + 4/k$ and $C_{\text{opt}} = 2 + 4/k$. By letting $k \to \infty$, we have $c \geq 3$.

4.3 The case of general metric spaces

In this subsection, we first construct a series of special metric spaces, for which we prove that there are no online algorithms with a constant competitive ratio, and show a $\Omega(\sqrt{\ln n})$ lower bound on any competitive ratios (where n is the number of requests in the given instance of the problem). Then, among the restricted class of online algorithms with prior knowledge about the total number of requests n, we propose one which is $O(\sqrt{\ln n})$-competitive; hence, asymptotically best possible among that class.

4.3.1 Lower bound on competitive ratios

Splitting operation: Such an operation on an edge AB of length l consists in splitting it into countably infinite many copies, each represented by a middle points $\{C_i\}_{i\in\mathbb{N}}$ such that: each C_i satisfies: $AC_i = BC_i = l/2$; and any path from one middle point C_{i_1} to another middle point C_{i_2} must pass through either A or B.

The metric spaces $\{\mathcal{M}_j\}_{j\in\mathbb{N}}$: The spaces $\{\mathcal{M}_j\}_{j\in\mathbb{N}}$ are created iteratively by the splitting operation described above. Given one space $\mathcal{M}_j$, we split each of its edge into countably infinite many copies to create $\mathcal{M}_{j+1}$:

- $\mathcal{M}_0$: A line segment A_0A_1 of length 1.

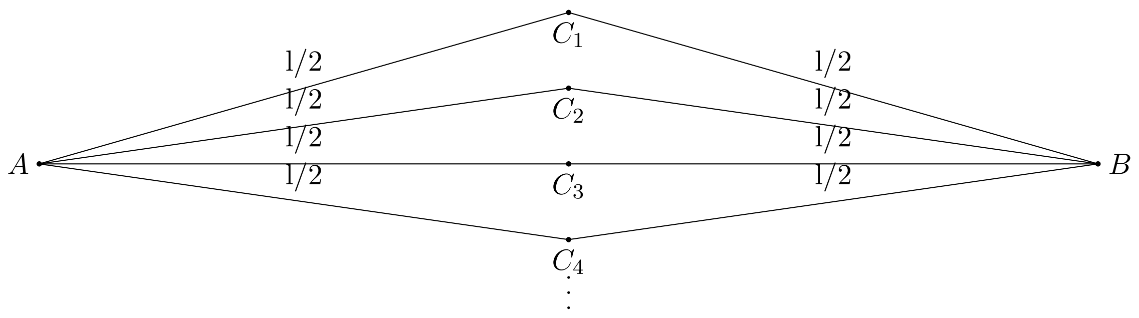


Figure 2: Splitting Operation

- $\mathcal{M}_1$: Split A_0A_1 into copies with middle points $A_{1/2,i_1}$.
- $\mathcal{M}_2$:
  - For every i_1, split $A_0A_{1/2,i_1}$ into copies with middle points $A_{1/4,i_1i_2}$ (Here the part before the comma indicates the point's distance from A_0. The part after the comma indicates its location, e.g. $A_{1/4,11}$ is a middle point of $A_0A_{1/2,1}$, but $d(A_{1/4,11}, A_{1/2,2}) \neq 1/4$.);
  - For every i_1, split $A_{1/2,i_1}A_1$ into copies with middle points $A_{3/4,i_1i_2}$;
- etc, ...

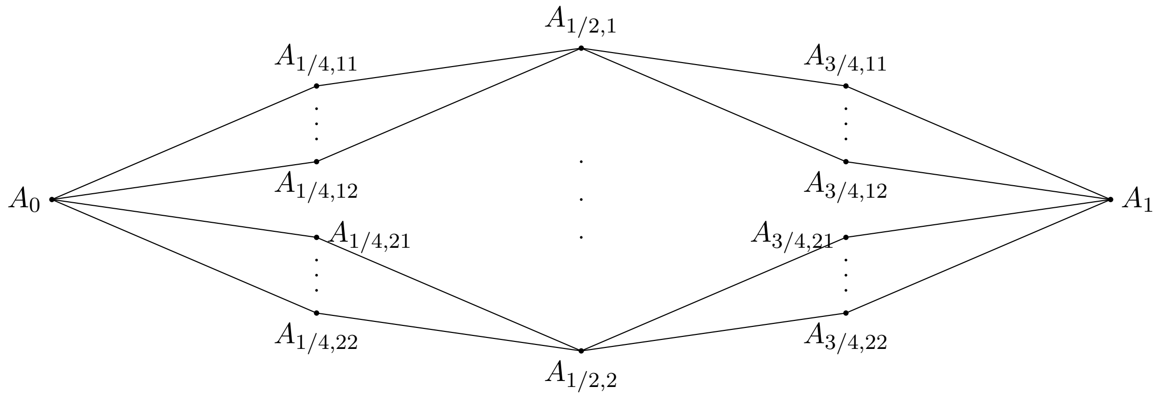


Figure 3: An illustration of $\mathcal{M}_2$

A point is called α-point if its distance from A_0 is α. For instance, $A_{1/2,i_1}$ are all 1/2-points and $A_{3/4,i_1i_2}$ are all 3/4-points. Let $V_0 = \{A_0, A_1\}$ and V_j be the set of middle points created when creating the space $\mathcal{M}_j$. For example, $V_2 = \{A_{1/4,i_1i_2}, A_{3/4,i_1i_2} | \forall i_1, i_2\}$. By the construction of the spaces and the splitting operations, the distance between two nearest points in V_j is $1/2^{j-1}$, and the distance between V_j and V_{j-1} is $1/2^j$.

Proof of unbounded competitive ratio: In order to present requests one by one to the online server, the release dates of all successive requests should be different. However, for simplicity in the exposition of our proofs, all release dates are set to be 0. Simply assume that the online server is given each request one by one, and has to make an acceptance/rejection decision before the next one is revealed. (Formally one could instead assume that for all $i \geq 1$, $r_i = i\epsilon$, where ϵ is an arbitrarily small positive number.)

Theorem 9. *For any $m \in \mathcal{N}$, there is no algorithm with a competitive ratio less than $m-1$.*

Proof. Consider the following instance defined on the metric space $\mathcal{M}_{2m^2}$. The online server is at A_0 initially. First, two requests with infinite penalties are released at A_0 and A_1. Then requests with penalties 1/m are released one by one at 1/2-points $\{A_{1/2,i_1}\}$ until the online server rejects one of them. We will later show that this is well defined, i.e. the online server must reject one such request. Assume he accepts $A_{1/2,1}, \cdots, A_{1/2,a_1-1}$ and rejects $A_{1/2,a_1}$. Then, release requests with penalties $1/(2m)$ at 1/4-th points $A_{1/4,a_1 i_2}$ until the online server rejects one (again our formal proof shows that this is a well defined stopping criteria), and at 3/4-th point $A_{3/4,a_1 i_2}$ until the online server rejects one. Repeat this procedure at $1/8, 3/8, 5/8, 7/8, ..., 1/2^{2m^2}, ..., (2^{2m^2}-1)/2^{2m^2}$-points. The penalty of a request at a point in V_j is $1/(2^{j-1}m)$. First, let us show that requests accepted by the online server are not close to each other:

Lemma 10. *Consider any two requests that are accepted by the online server. Assume that one request is at a point in V_{j_1} and the other is at a point to V_{j_2} (j_1 and j_2 may or may not be different). Then, the distance between those two requests is at least $1/2^{j_1} + 1/2^{j_2}$.*

Proof. Because of the way the instance and the metric spaces are constructed, any path from one request to the other must pass through a point in set $V_{\min\{j_1,j_2\}-1}$. Since the distance between V_{j_1} and $V_{\min\{j_1,j_2\}-1}$ is $1/2^{j_1}$ and the distance between V_{j_2} and $V_{\min\{j_1,j_2\}-1}$ is $1/2^{j_2}$, the distance between these two requests is at least $1/2^{j_1} + 1/2^{j_2}$. □

Lemma 10, combined with the fact that any request at a point in V_j has a penalty $1/(2^{j-1}m)$, indicates that the distance between two requests accepted by the online server is at least $m/2$ times the sum of the penalties of the two requests. Therefore, if the total penalty of requests accepted by the online server is P, the online server must travel at least mP units of time to serve all of them. Then, let us show that the instance is well defined, i.e. the online server cannot accept all requests:

Lemma 11. *Assume the online algorithm is $(m-1)$-competitive. For any $1 \leq j \leq 2m^2$, let k_j be the total number of requests at points in V_j that are accepted by the online server. Then, $k_j \leq m^2 2^j$.*

Proof. Assume there exists j such that $k_j > m^2 2^j$. Consider the instance in which no request at points in $\bigcup_{i=j+1}^{2m^2} V_i$ is presented. We will show that the algorithm is worse than $(m-1)$-competitive for the instance.

First, let us consider the cost of the online server. Since the total penalty of accepted requests is $\sum_{i=1}^{j} k_i/(2^{i-1}m)$, from the discussion above, the online server must spend at least $\sum_{i=1}^{j} k_i/2^i$ units of time serving these requests. Because for each $1 \leq i \leq j$ the total penalty of rejected requests at points in V_i is $1/m$, the total penalty of rejected requests is j/m. Thus, the online cost is $\sum_{i=1}^{j} k_i/2^{i-1} + j/m$.

Then, let us consider a feasible solution B and its cost. Requests rejected by the online server are accepted and requests accepted by the online server are rejected. Noting a carefully chosen shortest path from A_0 to A_1 passes through all requests rejected by the online server, the new feasible solution spends 2 units of time to visit these rejected requests. Thus, $C_B = \sum_{i=1}^{j} k_i/(2^{i-1}m) + 2$.

Since the online algorithm is $(m-1)$-competitive, $C_{online} \leq (m-1)C_{OPT} \leq (m-1)C_B$. Thus, $k_j/(2^{j-1}m) \leq \sum_{i=1}^{j} k_i/(2^{i-1}m) + j/m \leq 2(m-1) < 2m$, which implies $k_j \leq m^2 2^j$. Contradict with our assumption. □

We are now ready to finish the prove Theorem 9. According to the proof of Lemma 11, the online cost is at least $\sum_{i=1}^{2m^2} k_j/2^{i-1} + 2m$; there exists a feasible solution B whose cost C_B is $\sum_{i=1}^{2m^2} k_i/(2^{i-1}m) + 2$. However, $C_{online} \geq mC_B \geq mC_{OPT}$. Thus, the online algorithm is not $(m-1)$-competitive. □

Asymptotic lower bound: From the detailed proof of Theorem 9, at most $m^2 2^j$ requests are presented at points in V_j. At most $n \leq m^2 2^{2m^2}$ requests make any online algorithm worse than $(m-1)$-competitive. In other words, we have showed:

Theorem 10. *Any c-competitive online algorithm on an instance with n requests must have $c \geq \Omega(\sqrt{\ln n})$, even assuming n is given in advance.*

4.3.2 Best possible $O(\sqrt{\ln n})$-competitive algorithm when n is known

The algorithm proposed in this section requires a priori knowledge on the total number of requests. It is not clear that there exists an algorithm that achieves the same competitive ratio without such a knowledge.

For the simplicity of the algorithm and its analysis, we would like to consider problems without release dates. All requests arrive in sequence. The online algorithm must accept or reject a request before seeing the next. After making all the decisions, the online algorithm then decides how to serve all accepted requests. We argue that removing release dates (but preserving the order) only changes competitive ratios by at most 2:

Lemma 12. *Given an online algorithm A that is designed to solve the instances with all release dates 0(but still have to make decisions sequentially before knowing future requests), it can be transformed to another online algorithm A', such that for any instance I, $\frac{C_{A'}(I)}{C_{OPT}(I)} \leq \frac{C_A(I')}{C_{OPT}(I')}+2$, where I' is almost the same instance as I, the only difference is all release dates are 0 in I', but the order of requests remains.*

Proof. Consider the following algorithm A':

> Whenever a new request m comes, the server applies algorithm A on the instance consisting of first m requests with release dates all zeros, to make an accept/reject decision for the new request. If he accepts the new one, he goes back to the origin and then follows the newly computed route; otherwise, he just continues his current route.

Assume for instance I, $\rho = \frac{C_{A'}(I)}{C_{OPT}(I)}$ and the last request accepted by offline server is request m. According to the construction, the online server will go back to the origin at time r_m and then follows his last route. Note that at time r_m, the server is at most r_m units of distance away from the origin, thus, $C_{A'}(I) \leq r_m + r_m + C_A(I') \leq 2C_{OPT}(I) + \rho C_{OPT}(I') \leq (2+\rho)C_{OPT}(I)$. □

As mentioned in Section 4.1.1, accepting a request at a faraway location with a small penalty may not be beneficial; however, if many requests with small penalties are close to each other, it may be beneficial to accept them. Let us first define the concept of distance for a group of requests:

Definition 1. *Given a set U of requests and a route τ, define $T(U,\tau)$ as the shortest time to visit all requests in U along τ if τ passes all requests in U; otherwise, define as ∞.*

According to the definition, a server can begin at any request in U, visit all the other requests in U, and then go back to its initial position within $2T(U,\tau)$ units of time. A group of requests that are close and have a large overall penalty may be beneficial to visit. More formally,

Definition 2. *If τ passes all requests in U, given any nonempty subset of U: $\{i_1, i_2, ..., i_k\}$, there exists a division of $U = \bigcup_{l=1}^{k} I_l$, such that $i_{k_l} \in I_l, \forall l = 1, ..., k$, and $\sum_{l=1}^{k} T(I_l, \tau) \leq \sqrt{\ln n}(\sum_{i\in U} p_i - \sum_{l=1}^{k} p_{i_l})$, then we call τ a good route to visit U.*

We are now ready to present our algorithm. It consists of two stages: decision making and route traversing. In the decision making stage, decisions to accept or reject requests are made one by one. At the end of step k, $P_k \subset \{1, ..., k\}$ is the set of requests that have not been selected for a visit so far, and $V_k = \{1, ..., k\}\backslash Q$ is the set of requests to be visited (i.e. those who have been accepted, and those who have been rejected but will be served anyway). Every request i receives a label b_i during the decision making stage, which will be used for construct a route later. $i > b_i$ indicates request i is accepted by the online algorithm; $b_i < i < \infty$ indicates request i is rejected but visited; $b_i = \infty$ indicates request i is rejected and not visited.

DECISION MAKING:

0. Initialization. $P_0 = \emptyset$, $V_0 = \{0\}$, and $k = 1$.
1. Request k is accepted if and only if there exists a subset $Q_k \subset P_{k-1}$ and a good route μ_k to visit $Q_k \cup \{k\}$, such that $T(Q_k \cup \{k\}, \mu_k) + d(Q_k \cup \{k\}, V_{k-1}) \leq (\sum_{i\in Q_k} p_i + p_k)\sqrt{\ln n}$.
2. If request k is accepted, assume $j \in V_{k-1}$ satisfies $d(Q_k \cup \{k\}, j) = d(Q_k \cup \{k\}, V_{k-1})$. Update $P_k = P_{k-1}\backslash Q_k$, $V_k = V_{k-1} \cup \{k\} \cup Q_k$ update labels: $b_k = j$ and $b_i = k$ for all $i \in Q_k$.
3. If request k is rejected, update $P_k = P_{k-1} \cup \{k\}$, $V_k = V_{k-1}$, and $b_k = \infty$.
4. $k = k + 1$. Go to Step 1.

Let S_a be the subset of requests accepted by the online algorithm. After making accept/reject decisions, a route is constructed iteratively, based on the information gathered in the DECISION

MAKING stage. Traveling along the route, the online server visits requests in S_a and in V_n:

ROUTE CONSTRUCTION:

0. Initialization. $\tilde{\tau}_0 = \tau_n$, and $k = 1$.

1. If $k \in S_a$ and $\tilde{\tau}_{k-1}$ covers at least one request in $Q_k \cup \{k\}$, let requests $c_{i_1}, c_{i_2}, \cdots, c_{i_w}$ be all the requests in $Q_k \cup \{k\}$ that are covered by $\tilde{\tau}_{k-1}$. Find the division of $Q_k \cup \{k\} = \bigcup_{t=1}^{w} I_t$ that satisfies the condition in the definition of good routes. Construct $\tilde{\tau}_k$ as follows, it is the same as $\tilde{\tau}_{k-1}$ except for w detours: when arriving at request $c_{i_t}, (1 \leq t \leq w)$, follow the shortest route to visit I_t, go back to c_{i_t}, and then continue to follow $\tilde{\tau}_{k-1}$.

2. If $k \in S_a$ and τ does not cover any request in $Q_k \cup \{k\}$, construct $\tilde{\tau}_k$ as follows: it is the same as $\tilde{\tau}_{k-1}$ except for a detour: when arriving at request b_k, follow the shortest route to visit all the requests in Q_k, go back to b_k, and then continue to follow $\tilde{\tau}_{k-1}$.

3. If $k \notin S_a$, $\tilde{\tau}_k = \tilde{\tau}_{k-1}$.

4. $k = k + 1$. Go to Step 1.

The cost of the resulting solution comes from two parts: the penalties of rejected requests and the time to visit requests. We will provide upper bounds for both parts. First, let us consider the penalty part. Let W be the set of requests that are accepted by the optimal offline solution, but rejected by the online solution. Following is a upper bound for penalties of requests in W:

Lemma 13. $\sum_{k \in W} p_k \leq (2\sqrt{\ln n} + 1) C_{\text{opt}}(n)$.

Proof. We divide requests in W into some subsets by the following algorithm:

0. Initialization: $W_1 = W, k = 1$.

1. Divide all requests in W_k into m_k subsets, such that each subset consists of one or several successive requests, in every subset $A_k^j (1 \leq j \leq m_k)$, the largest index is smaller than the smallest label, and τ_n is a good route to visit A_k^j. The division has the fewest subsets among all possible divisions.

2. If $m_k \leq 1$, terminate; otherwise, go to Step 3.

3. Let $B_k = \{j | \sum_{i \in A_k^j} p_i \geq \max\{\sum_{i \in A_k^{j-1}} p_i, \sum_{i \in A_k^{j+1}} p_i\}\}$. Let $W_{k+1} = \bigcup_{j \in B_k} A_k^j$.

4. Update $k = k + 1$. Go to step 1.

We first show the existence of divisions that satisfy the requirements in Step 1: a trivial division consists of $|W_k|$ singletons. Thus, this algorithm is well defined. Noting that $\bigcup_{j\in B_k} A_k^j$ is also a feasible division of W_{k+1}, from the minimum number of subsets, we have $m_{k+1} \le |B_k| \le m_k/2$. Furthermore, because $m_1 \le |W| \le n$, the algorithm terminates after at most $\ln n$ iterations. Let K be the number of iterations before termination.

We then consider any two adjacent subset A_k^j and A_k^{j+1} in iteration $k < K$. We will show that $T(A_k^j \cup A_k^{j+1}, \tau_n) \ge \sqrt{\ln n}\min\{\sum_{i\in A_k^j} p_i, \sum_{i\in A_k^{j+1}} p_i\}$. Without loss of generality, let us assume $\max_{i\in A_k^j} i < \max_{i\in A_k^{j+1}} i$. Consider $\min_{i\in A_k^j} b_i$:

1. If $\min_{i\in A_k^j} b_i < \max_{i\in A_k^{j+1}} i$, let $z = \arg\min_{i\in A_k^j} b_i$. When request $\max_{i\in A_k^{j+1}} i$ is released, all requests in A_k^{j+1} are released and not covered by τ, and z is covered by τ. Noting that τ_n is a good route to visit A_k^{j+1}, we have $\sum_{i\in A_k^{j+1}} p_i\sqrt{\ln n} < T(A_k^{j+1}, \tau_n) + d(A_k^{j+1}, z)$; or request $\max_{i\in A_k^{j+1}} i$ will be accepted. Therefore, $T(A_k^j \cup A_k^{j+1}, \tau_n) \ge T(A_k^{j+1}, \tau_n) + d(A_k^{j+1}, z) > \sum_{i\in A_k^{j+1}} p_i\sqrt{\ln n}$.

2. If $\min_{i\in A_k^j} b_i < \max_{i\in A_k^{j+1}} i$, then τ is not a good route to visit $A_k^j \cup A_k^{j+1}$; otherwise, the two subsets can be merged to one. According to the definition of good routes ,we can show that $T(A_k^j \cup A_k^{j+1}, \tau_n) \ge \min\{\sum_{i\in A_k^j} p_i\sqrt{\ln n}, \sum_{i\in A_k^{j+1}} p_i\sqrt{\ln n}\}$.

From the above inequality, we have

$$\begin{array}{rcl}2C_{\text{opt}}(n) & \ge & \sum_{j=1}^{m_k} T(A_k^j \cup A_k^{j+1}, \tau_n) \\ & \ge & \sum_{j=1}^{m_k} \min\{\sum_{i\in A_k^j} p_i\sqrt{\ln n}, \sum_{i\in A_k^{j+1}} p_i\sqrt{\ln n}\} \\ & \ge & \sum_{j\notin B_k} \sum_{i\in A_k^j} p_i\sqrt{\ln n} \\ & = & \sum_{i\in W_k\setminus W_{k+1}} p_i\sqrt{\ln n}\end{array}$$

In the last iteration, from the requirement of divisions, $\min_{i\in W_K} b_i \ge \max_{i\in W_K} i$. Hence when request $\max_{i\in W_K} i$ is revealed, all requests in W_K are still in $P_{\max_{i\in W_K} i-1}$. On the other hand, request $\max_{i\in W_K} i$ is not accepted. Combined with the fact that τ_n is a good route to visit W_K, we have $C_{\text{opt}}(n) \ge \sum_{i\in W_K} p_i\sqrt{\ln n}$.

By summing these inequalities up, we conclude $\sum_{k\in W} p_k \le (2\sqrt{\ln n}+1)C_{\text{opt}}(n)$. □

Consider now the route $\tilde{\tau}_n$ that visits all requests in V_n. We have a corresponding lemma:

Lemma 14. $L(\tilde{\tau}_n) \le (2\sqrt{\ln n}+1)C_{\text{opt}}(n)$.

Proof. Let us establish the upper bound $L(\tilde{\tau}_n)$ by considering $L(\tilde{\tau}_k) - L(\tilde{\tau}_{k-1})$ for all $1 \le k \le n$:

1. If $k \in S_a$ and $\tilde{\tau}_{k-1}$ passes at least one request in $Q_k \cup \{k\}$, then according to the definition of good route, $L(\tilde{\tau}_k) - L(\tilde{\tau}_{k-1}) \le 2\sqrt{\ln n}\sum_{t\in Q_k\cup\{k\}\setminus S_n} p_t$.

2. If $k \in S_a$ and $\tilde{\tau}_{k-1}$ does not cover any request in $Q_k \cup \{k\}$, then $L(\tilde{\tau}_k) - L(\tilde{\tau}_{k-1}) \le 2T(Q_k \cup \{k\}, \mu_k) + 2d(Q_k \cup \{k\}, R_{k-1}) \le 2\sqrt{\ln n}\sum_{t\in Q_k\cup\{k\}} p_t \le 2\sqrt{\ln n}\sum_{t\in Q_k\cup\{k\}\setminus S_n} p_t$.

3. If $k \notin S_a$, then $L(\tilde{\tau}_k) = L(\tilde{\tau}_{k-1})$.

Since every request in R_n belongs to at most one of Q_i, by summing these inequalities up, we have $L(\tilde{\tau}_n) \le L(\tau_n) + 2\sqrt{\ln n}\sum_{t\in R_n} p_t \le (2\sqrt{\ln n}+1)C_{\text{opt}}(n)$. □

After providing upper bounds on both parts, we can now conclude:

Theorem 11. *The algorithm is $O(\sqrt{\ln n})$-competitive.*

Proof. Since the requests rejected by the online server are in $R_n \cup W$,

$$\begin{array}{rcl} C_{on}(n) & \leq & \sum_{i \in R_n} p_i + \sum_{i \in W} p_i + L(\tilde{\tau}_n) \\ & \leq & C_{\text{opt}}(n) + (2\sqrt{\ln n} + 1)C_{\text{opt}}(n) + (2\sqrt{\ln n} + 1)C_{\text{opt}}(n) \\ & = & (4\sqrt{\ln n} + 3)C_{\text{opt}}(n). \end{array}$$

Therefore, the algorithm is $O(\sqrt{\ln n})$-competitive. □

5 Conclusions

From our results it is clear that, from a competitive analysis perspective, the *basic* versions of our online problem are easier to tackle than their *real-time* versions - the corresponding competitive ratios are much smaller. There are some key questions left open in this paper:

We have not presented results in this paper on the version for which the server doesn't have to return home to a specific location at the end. It turns out that this version is more difficult to solve for an online server - the lack of certainty about where to end proves to be yet another source of difficulty in an adversarial situation.

Also, it would be of interest to know if the introduction of quota for prizes to collect fundamentally changes the nature of the results for the real-time online version of our problems.

Acknowledgements

We thank the anonymous referees for their thoughtful and detailed comments, which improved the quality and clarity of our paper.

References

[1] L. Allulli, G. Ausiello, and L. Laura. On the power of lookahead in on-line vehicle routing problems. In *Proceedings of the Eleventh International Computing and Combinatorics Conference, Lecture Notes in Computer Science*, volume 3595, pages 728–736, 2005.

[2] N. Ascheuer, S. Krumke, and J. Rambau. Online dial-a-ride problems: Minimizing the completion time. In *Proceedings of the 17th International Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science*, volume 1770, pages 639–650, 2000.

[3] G. Ausiello, V. Bonifaci, and L. Laura. The on-line prize-collecting traveling salesman problem. *Information Processing Letters*, 107(6):199–204, 2008.

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

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

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

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

[8] D. Feillet, P. Dejax, and M. Gendreau. Traveling salesman problems with profits. *Transportation Science*, 39:188–205, 2005.

[9] Y. Higashikawa, N. Katoh, S. Langerman, and S. Tanigawa. Online graph exploration algorithms for cycles and trees by multiple searchers. *Journal of Combinatorial Optimization*, pages 1–16, 2012.

[10] P. Jaillet and X. Lu. Online traveling salesman problems with service flexibility. *Networks*, 58:137–146, 2011.

[11] P. Jaillet and M. Wagner. Online routing problems: value of advanced information as improved competitive ratios. *Transportation Science*, 40(2):200–210, 2006.

[12] P. Jaillet and M. Wagner. Generalized online routing: New competitive ratios, resource augmentation and asymptotic analyses. *Operations Research*, 56:745–757, 2008.

[13] B. Kalyanasundaram and K.R. Pruhs. Constructing competitive tours from local information. *Theoretical Computer Science*, 130(1):125–138, 1994.

[14] B. Korte and J. Vygen. *Combinatorial Optimization, Theory and Algorithms.* Springer, second edition, 2002.

[15] S. Krumke, W. de Paepe, D. Poensgen, and L. Stougie. News from the online traveling repairman. *Theoretical Computer Science*, 295:279–294, 2003.

[16] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. *The Traveling Salesman Problem, A Guided Tour of Combinatorial Optimization.* John Wiley & Sons Ltd., 1985.

[17] M. Lipmann. *On-line Routing.* PhD thesis, Technische Universiteit Eindhoven, 2003.

[18] X. Lu. *Online Optimization Problems.* PhD thesis, Operations Research Center, MIT, June 2013.

[19] N. Megow, K. Mehlhorn, and P. Schweitzer. Online graph exploration: New results on old and new algorithms. In *Automata, Languages and Programming*, volume 6756 of *Lecture Notes in Computer Science*, pages 478–489. Springer Berlin Heidelberg, 2011.

[20] N. Megow, M. Skutella, J. Verschae, and A. Wiese. The power of recourse for online mst and tsp. In *Automata, Languages, and Programming*, volume 7391 of *Lecture Notes in Computer Science*, pages 689–700. Springer Berlin Heidelberg, 2012.

[21] M. Wagner. *Online Optimization in Routing and Scheduling.* PhD thesis, Operations Research Center, MIT, June 2006.