Online Traveling Salesman Problems with Service Flexibility[*]

Patrick Jaillet[‡] Xin Lu[§]

January 2009; accepted February 2010; published August 2011

Abstract

The Traveling Salesman Problem (TSP) is a well-known combinatorial optimization problem. We are concerned here with online versions of this problem defined on metric spaces. One novel aspect in the paper is the introduction of a sound theoretical model to incorporate "yes-no" decisions on which requests to serve, together with an online strategy to visit the accepted requests.

In order to do so, we assume that there is a penalty for not serving a request. Requests for visit of points in the metric space are revealed over time to a server, initially at a given origin, who must decide in an online fashion which requests to serve in order to minimize the time to serve all accepted requests plus the sum of the penalties associated with the rejected requests.

We first look at the special case of the non-negative real line. After providing a polynomial time algorithm for the offline version of the problem, we propose and prove the optimality of a 2-competitive polynomial time online algorithm based on re-optimization approaches. We also consider the impact of advanced information (lookahead) on this optimal competitive ratio. We then consider the generalizations of these results to the case of the real line. We show that the previous algorithm can be extended to an optimal 2-competitive online algorithm. Finally we consider the case of a general metric space and propose an original c-competitive online algorithm, where $c = \frac{\sqrt{17}+5}{4} \approx 2.28$. We also give a polynomial-time $(1.5\rho + 1)$-competitive online algorithm which uses a polynomial-time ρ-approximation for the offline problem.

Key words: online algorithms; traveling salesman; prize-collecting

[*]Appeared in *Networks*, 58, 137-146, 2011. Research funded in part by ONR, grant N00014-09-1-0326

[‡]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

1 Introduction

The context of this paper deals with routing and scheduling problems under incomplete and uncertain data. More generally, we are interested on the fundamental aspects of decision making under uncertain data sets, dynamically revealed over time in an online fashion, and on how to design and evaluate corresponding intelligent algorithms.

Here we want to consider these questions around the most basic combinatorial optimization problem in routing and scheduling - the Traveling Salesman Problem (TSP). In one of its simplest forms, we are given a metric space with an origin and a set of points in the space. The task is to find a tour of minimum total length, beginning and ending at the origin, that visits each point at least once. One can introduce a “time” aspect to the problem by considering a server moving along, and visiting these points. Assuming a constant speed for the server, the TSP objective is then to minimize the time required to complete a tour. By incorporating release dates with points, where a point can only be visited on or after its release date, we obtain the so-called “TSP with Release Dates”.

Finally one can associate a penalty to each point and remove the requirement that the server needs to visit all points. Rather the server can decide which points to serve and its objective is to minimize the time to go through all accepted points plus the sum of the penalties associated with the rejected points. This latter problem, which we will call hereafter the “TSP with Flexible Service” will be our main fundamental offline canonical problem of interest.

1.1 Context and Definitions for Online and Offline Versions

The assumption that problem instances are completely known a priori is unrealistic in many applications. One answer is to consider an *online* model in which requests (i.e. points to visit) are revealed over time, while the server is “en route” serving previously released requests. Such considerations lead to the formal definition of the online TSP, which, together with several of its variants, has recently received a detailed treatment from a competitive analysis point-of-view (see Subsection 1.3).

In this paper we consider a new online problem - the Online TSP with Flexible Service. Besides providing a solid building block for many real applications in routing, scheduling, robotics, etc., this problem’s main theoretical attraction is to provide a basic canonical enrichment of the online TSP by adding “yes-no” decisions on which points/requests to serve. The formal definition of the online and offline version of the problem is as follows:

The TSP with Flexible Service:

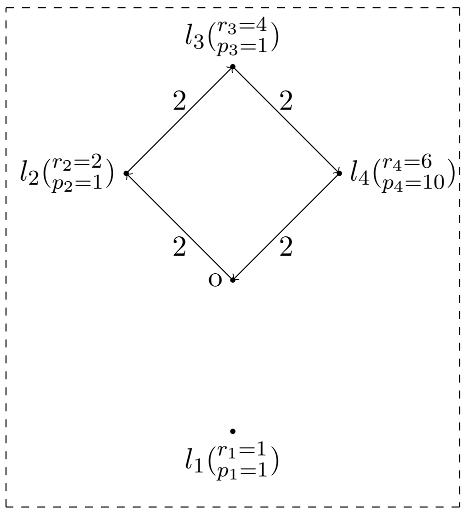
> **Instance:** A metric space $\mathcal{M}$ with a given origin o and a distance metric $d(\cdot, \cdot)$. A series of n requests $(l_i, r_i, p_i)_{1 \le i \le n}$ where $l_i \in \mathcal{M}$ is the location (point in metric space), $r_i \in \mathbb{R}_+$ the release date (first time after which service can be done), and $p_i \in \mathbb{R}_+$ the penalty (for not serving) of request i. 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 needs to be idle at the origin at the end (final state). The earliest time the server reaches this final state will be called the makespan.
>
> **Offline version:** The number of requests n is known to the offline server. All requests $(l_i, r_i, p_i)_{1 \le i \le n}$ are revealed to the offline server at time 0.
>
> **Online version:** The number of requests n is not known to the online server. Requests are revealed to the online server at their release dates $r_i \ge 0$; assume $r_1 \le r_2 \cdots \le r_n$.
>
> **Objective:** In both cases, minimize {the makespan to serve all accepted requests + the sum of the penalties of all rejected requests}.

A natural approach for solving the online version of this problem is to consider re-optimisation: at the time of a new request, immediately recompute an optimal solution through all revealed but not yet served requests. This motivates the introduction of the following path problem with an arbitrary starting location x, and with no consideration of release dates:

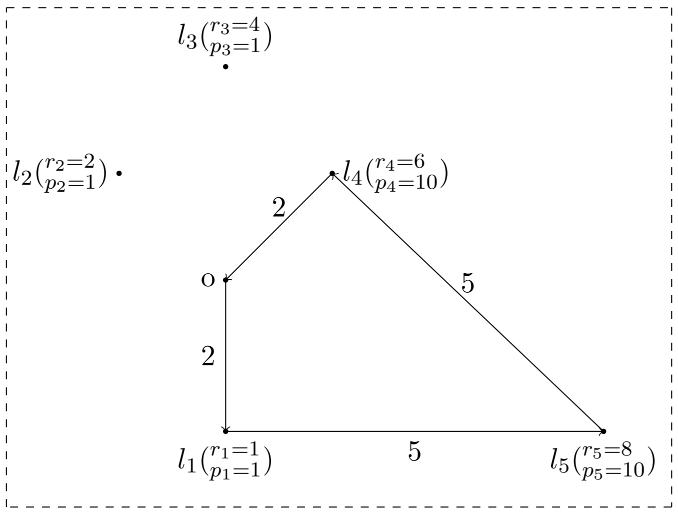
Flexible Path Problem:

> **Instance:** A metric space $\mathcal{M}$ with a given origin o and a distance metric $d(\cdot, \cdot)$. A total of n requests $(l_i, p_i)_{1 \le i \le n}$ where $l_i \in \mathcal{M}$ is the location (point in metric space) and $p_i \in \mathbb{R}_+$ the penalty (for not serving) of request i. The problem begins at time 0; the server is initially idle at a point $x \in \mathcal{M}$ (initial state), can travel at unit speed (when not idle), and eventually needs to be idle at the origin at the end (final state). The earliest time the server reaches this final state will be called the makespan.
>
> **Objective:** minimize {the makespan to serve all accepted requests + the sum of the penalties of all rejected requests}.

Below is an illustration of the TSP with Flexible Service problem which shows how successive optimal offline solutions can vary drastically:


$l_3\binom{r_3=4}{p_3=1}$
2
2
$l_2\binom{r_2=2}{p_2=1}$
$l_4\binom{r_4=6}{p_4=10}$
2
2
o
$l_1\binom{r_1=1}{p_1=1}$

1a: initial optimal solution


$l_3\binom{r_3=4}{p_3=1}$
$l_2\binom{r_2=2}{p_2=1}$
$l_4\binom{r_4=6}{p_4=10}$
2
o
5
2
$l_1\binom{r_1=1}{p_1=1}$
5
$l_5\binom{r_5=8}{p_5=10}$

1b: after one more request (l_5) is released

Figure 1: An illustration of the TSP with Flexible Service. As shown, the addition of request l_5 in the input cause dramatic changes in the optimal offline solution.

1.2 Our Contributions

Besides introducing a solid framework for looking at the additional impacts of including "yes-no" decisions on the online TSP problem, this paper presents many new results generalizing and/or improving existing known results. We first look at the special case of the non-negative real line. After providing a polynomial time algorithm for the offline version of the problem, we propose and prove the optimality of a 2-competitive polynomial time online algorithm based on re-optimization approaches. One of the key results here is the lower bound (for the optimality result). We also consider the impact of advanced information (lookahead) on this optimal competitive ratio, and generalize similar results that were obtained on the online TSP in [15]. We then consider the generalizations of these results to the case of the real line. We show that the previous algorithm can be extended to an optimal 2-competitive online algorithm. Finally we consider the case of a general metric space and propose an original c-competitive online algorithm, where $c = \frac{\sqrt{17}+5}{4} \approx 2.28$. We also give a polynomial-time $(1.5\rho + 1)$-competitive online algorithm which uses a polynomial-time ρ-approximation for the offline problem.

1.3 Literature Review

The literature for the TSP is vast. The interested reader is referred to the books by Lawler et al. [20] and Korte and Vygen [19] for comprehensive coverage of results concerning the classical TSP.

A systematic study of online algorithms is given by Sleator and Tarjan [22], who suggest comparing an online algorithm with an optimal offline algorithm. Karlin et al. [18] introduce the notion of a *competitive ratio*. Online algorithms have been used to analyze paging in computer memory systems, distributed data management, navigation problems in robotics, multiprocessor scheduling, etc. (e.g. see the survey paper of Albers [1] and the books of Borodin and El-Yaniv [11] and Fiat and Woeginger [13] for more details and references.)

Research concerning online versions of the TSP is more recent but has been growing steadily. Kalyanasundaram and Pruhs [17] examine a unique version where new cities are revealed locally during the traversal of a tour (i.e., an arrival at a city reveals any adjacent cities that must also be visited). Angelelli et al. [3, 4] study related online routing problems in a multi-period setting. Bent and Van Hentenryck [9] have looked at online stochastic optimization techniques (e.g. scenario-based approaches) for addressing dynamic online routing problems; see their book [14] for more references and applications of these approaches.

More closely related to our paper is the stream of works which started with the paper by Ausiello et al. [8]. In this paper, the authors study the online version of the TSP with release dates (but with no service flexibility); they analyze the problem on the real line and on general metric spaces, developing online algorithms for both cases and achieving an optimal online algorithm for general metric spaces, with a competitive ratio of 2. They also provide a polynomial-time online algorithm, for general metric spaces, which is 3-competitive. Subsequently, the paper by Ascheuer et al. [5] implies the existence of a polynomial-time algorithm, for general metric spaces, which is 2.65-competitive as well as a $(2 + \epsilon)$-competitive $(\epsilon > 0)$ algorithm for Euclidean spaces. Lipmann [21] develops an optimal online algorithm for the real line, with a competitive ratio of 1.64. Blom et al. [10] give an optimal online algorithm for the non-negative real line, with a competitive ratio of $\frac{3}{2}$, and also consider different adversarial algorithms in the definition of the competitive ratio. Jaillet and Wagner [15] introduce the notion of a disclosure date, which is a form of advanced notice for the online salesman, and quantify the improvement in competitive ratios as a function of the advanced notice. A similar approach was taken by Allulli et al. [2]

in the form of a *lookahead*.

There has also been work on generalizing the basic online TSP framework. The paper by Feuerstein and Stougie [12] considers the online Dial-a-Ride problem, where each city is replaced by an origin-destination pair. The authors consider both the uncapacitated case, giving a best-possible 2-competitive algorithm, and the capacitated case, giving a 2.5-competitive algorithm. The previously cited paper by Ascheuer et al. [5] also gives a 2-competitive online algorithm and a $(1 + \sqrt{1 + 8\rho})/2$-competitive polynomial-time online algorithm for the uncapacitated online Dial-a-Ride problem (ρ being the approximation ratio of a simpler but related offline problem). Their algorithm is generalizable to the case where there are multiple servers with capacities; this generalization is also 2-competitive. Jaillet and Wagner [16] consider the (1) online TSP with precedence and capacity constraints and the (2) online TSP with m salesmen. For both problems they propose 2-competitive online algorithms (optimal in case of the m-salesmen problem), consider polynomial-time online algorithms, and then consider resource augmentation, where the online servers are given additional resources to offset the powerful offline adversary advantage. Finally, they study online algorithms from an asymptotic point of view, and show that, under general stochastic structures for the problem data, *unknown and unused by the online player*, the online algorithms are almost surely asymptotically optimal.

Finally there has been other recent work dealing with online routing problems which do not require the server to visit every revealed request. Ausiello et al. [7] analyze the online Quota TSP, where each city to be visited has a weight associated with it and the server is given the task to find the shortest sub-tour through cities in such a way to collect a given quota of weights by visiting the chosen cities. They present an optimal 2-competitive algorithm for a general metric space. Wagner [23] considers the case where whenever a new request arrives, the online algorithm has the option of rejecting the request and pay a given penalty. In the model considered, there are however no release dates; rather the online algorithm is presented a series of request, must decide to accept or reject each request, before the next one is revealed. The overall objective is to minimize the tour length through the accepted requests plus the sum of the penalties associated with the rejected requests. The paper presents an optimal 2-competitive algorithm for this problem on the non-negative half-line. Ausiello et al. [6] provide a competitive analysis of the "prize-collecting TSP", a generalization of the quota problem where penalties for not visiting cities are also included, beyond meeting a given quota. They provide a 7/3-competitive algorithm and a lower bound on any competitive ratios of 2 for a general metric space, and refer to a 2-competitive algorithm and a lower bound of 1.89 on the non-negative real line. More generally, assuming a ρ-approximation algorithm for the offline problem, they show that their online algorithm is a $(2\rho+\frac{\rho}{1+2/\rho})$-competitive polynomial time algorithm. Our main results, when comparable, provide improved competitive ratios on all these problems. The "prize-collecting TSP" is a slight generalization of the problem we consider here; we feel confident that our stronger (competitive ratio) results would in fact carry through to this generalization, but this is beyond the scope of this current paper.

Outline: The remainder of the paper is as follows: we present first our results on the Online TSP with Service Flexibility for the non-negative half-line in Section 2; we then consider extensions of these results to the real line in Section 3. We then treat the general metric space in Section 4 and then offer few concluding remarks in Section 5.

2 Online TSP with Service Flexibility on $\mathbb{R}_+$

In this section, we study the problem when the locations of the requests are all on the non-negative real line. In that case, for sake of notational simplicity, we will use l_i as both the location of the request i as well as its distance to the origin (an abuse of notation, but circumventing the need to use $d(o, l_i)$). We begin with an offline analysis.

2.1 Polynomial Time Offline Algorithm for the TSP with Flexible Service

The offline server is given n requests $(l_i, r_i, p_i)_{1\leq i\leq n}$ all at once. If the server accepts request i, then the makespan has to exceed $max\{r_i + l_i, 2l_i\}$. Let $a_i \doteq max\{r_i + l_i, 2l_i\}$. Reorder all the requests in non-decreasing order of a_i, i.e., $a_{k_1} \leq a_{k_2} \leq \cdots \leq a_{k_n}$. Let C_{opt} be the optimal offline cost, then we have the following lower bound:

Lemma 1. $C_{opt} \geq \min\{\sum_{i=1}^n p_i, \min_{1\leq m\leq n}\{a_{k_m} + \sum_{i=m+1}^n p_{k_i}\}\}$

Proof. If the server does not accept any requests, then $C_{opt} = \sum_{i=1}^n p_i$. Otherwise, $C_{opt} \geq a_{k_m} + \sum_{i=m+1}^n p_{k_i}$, where k_m is the largest index of accepted requests. □

This result leads to the following algorithm:

Offline Algorithm on $\mathbb{R}^+$:

1. If $\sum_{i=1}^n p_i \leq \min_{1\leq m\leq n}\{a_{k_m} + \sum_{i=m+1}^n p_{k_i}\}$, the server remains idle (at the origin) and rejects all the requests.

2. Otherwise, let $m^* = \max\{\text{argmin}_{1 \le m \le n}\{a_{k_m} + \sum_{i=m+1}^{n} p_{k_i}\}\}$:

   2a. Find the furthest point that needs to be visited, $l_{max} = \max\limits_{1 \le m \le m^*} l_{k_m}$, and go directly to l_{max};

   2b. Remain idle at point l_{max} for $\max\limits_{1 \le m \le m^*} \{\max\{0, r_{k_m} - 2l_{max} + l_{k_m}\}\}$ units of time;

   2c. Go back to the origin and serve all requests ready to be served along the way.

It is easy to see that this algorithm is an $O(n \log n)$ polynomial-time exact algorithm for the TSP with Flexible Service on $\mathbb{R}_+$.

2.2 Lower Bound on Competitive Ratios

Theorem 1. *Any c-competitive online algorithm on $\mathbb{R}_+$ has $c \ge 2$.*

Proof. Assume that an online server A follows a given c-competitive online algorithm, where c is a finite constant. With n an arbitrary number to be defined by the adversary, consider a series of up to $n+1$ requests as follows: $(l_i, r_i, p_i) = (2^{1-i}, 2 - 2^{1-i}, 2^{1-i})$ for $1 \le i \le n$, and $(l_{n+1}, r_{n+1}, p_{n+1}) = (2^{-n}, 2 - 2^{-n}, \infty)$. Let $C_A(k)$ and $C_{opt}(k)$ be the costs incurred by the online server A and the offline adversary, respectively, when the instance is drawn from the first k requests of the series: $(l_i, r_i, p_i)_{1 \le i \le n+1}$. Let t_0 be the time the online server begins to move from the origin for the first time.

First note that $r_1 \le t_0 \le \infty$. Indeed if $0 \le t_0 < r_1$, then no request would be presented, leading to $C_A > 0$, $C_{opt} = 0$, and thus an infinite c; also t_0 obviously needs to be finite in order to have a finite c. We can now consider two cases:

1. If $t_0 < r_{n+1}$, let $m \in \{1, 2, \dots, n\}$, be such that $r_m \le t_0 < r_{m+1}$. Then only the first m requests are presented, and the offline server rejects them all. Then $C_{opt}(m) = \sum_{i=1}^{m} p_i = 2 - 2^{1-m}$. If A rejects all requests, then $C_A(m) \ge r_m + \sum_{i=1}^{m} p_i = 4 - 2^{2-m}$. Otherwise, let k, $k \le m$, be the accepted request whose location is furthest away from the origin. We have $C_A(m) \ge r_m + 2l_k + \sum_{i=1}^{k-1} p_i = 4 - 2^{1-m} \ge 4 - 2^{2-m}$. We then get $c \ge (4 - 2^{2-m})/(2 - 2^{1-m}) = 2$.

2. If $t_0 \ge r_{n+1}$, then all $n+1$ requests are presented. The offline adversary serves them all, and we have $C_{opt}(n+1) = 2l_1 = 2$. On the other hand, A will have to serve at least the last request $n+1$. If it serves only this last request, then $C_A(n+1) \ge r_{n+1} + 2l_{n+1} + \sum_{i=1}^{n} p_i = 2 - 2^{-n} + 2 \cdot 2^{-n} + 2 - 2^{1-n} = 4 - 2^{-n}$. Otherwise,

let k, $k \leq n$, be the accepted request whose location is furthest away from the origin. We have $C_A(n+1) \geq r_{n+1} + 2l_k + \sum_{i=1}^{k-1} p_i = 2 - 2^{-n} + 2 \cdot 2^{1-k} + 2 - 2^{2-k} = 4 - 2^{-n}$. Therefore, either way, we have $c \geq 2 - 2^{-n-1}$. By letting $n \to +\infty$, then $c \geq 2$.

□

2.3 A 2-Competitive Algorithm

Let us first consider the version of the offline optimization problem without release dates, the Flexible Path Problem as defined in Section 1.1. Assume that the current initial position of the server is a given point $x > 0$, and that we have n requests $(l_i, p_i)_{1 \leq i \leq n}$. Let C_{opt}^x be the optimal objective value. Reorder the requests in non-decreasing order of l_i, i.e., $l_{k_1} \leq \cdots \leq l_{k_n}$. For simplicity, define $l_{k_0} = 0$, and $l_{k_{n+1}}$ as an arbitrary large number strictly greater than $\max\{x, \max_{1 \leq i \leq n} l_i\}$. First note that if there exists $1 \leq m \leq n$ such that $l_{k_m} < x \leq l_{k_{m+1}}$, then irrespective of the other acceptance decisions, the server can accept all requests $k_1, \ldots, k_m$ at no extra costs. If there is not such an m, then simply define $m = 0$. Consider now the following cases:

1. If the server does not accept any other requests, then $C_{opt}^x \geq x + \sum_{i=m+1}^{n} p_{k_i}$.
2. Otherwise, let k_r $(r > m)$ be the visited request whose location is furthest away from the origin. Then $C_{opt}^x \geq 2l_{k_r} - x + \sum_{i=r+1}^{n} p_{k_i}$.

So $C_{opt}^x = \min\{\min_{m+1 \leq r \leq n}\{2l_{k_r} - x + \sum_{i=r+1}^{n} p_{k_i}\}, x + \sum_{i=m+1}^{n} p_{k_i}\}$, and this leads to the following optimal algorithm for the Flexible Path Problem, which we label OFP$(x \neq 0)$ (for "Optimal Flexible Path starting at a location x away from the origin"):

OFP$(x \neq 0)$:

0. Reorder the requests in non-decreasing order of l_i, $l_{k_1} \leq \cdots \leq l_{k_n}$. Let $1 \leq m \leq n$ be such that $l_{k_m} < x \leq l_{k_{m+1}}$ if it exists, $m = 0$ otherwise. Go to Step 1.
1. If $x + \sum_{i=m+1}^{n} p_{k_i} \leq \min_{m+1 \leq r \leq n}\{2l_{k_r} - x + \sum_{i=r+1}^{n} p_{k_i}\}$, the server goes directly to the origin and serves all requests along the way.
2. Otherwise, let $r^* = \max\{\text{argmin}_{m+1 \leq r \leq n}\{2l_{k_r} - x + \sum_{i=r+1}^{n} p_{k_i}\}\}$:

    2a. go directly to $l_{k_{r^*}}$;

    2b. go back to the origin and serve all requests along the way.

It is easy to see that OFP($x \neq 0$) is an $O(n \log n)$ polynomial-time exact algorithm for the Flexible Path Problem on $\mathbb{R}_+$, when $x > 0$. When $x = 0$, this algorithm needs to be adapted in order to take care of a special case of interest for the online version of the TSP with Flexible Service. For this case, the amount of time the server has been idle at the origin, say w, is a critical factor. The new algorithm, OFP(0), is as follow:

```
**OFP**(0):
  0. Let $w$ be the amount of time the server has been at the origin prior to
     facing the $n$ requests. Reorder the requests in non-decreasing order of $l_i$,
     $l_{k_1} \leq \cdots \leq l_{k_n}$. Go to Step 1.
  1. If $-w + \sum_{i=1}^{n} p_{k_i} \leq \min_{1 \leq r \leq n} \{2l_{k_r} + \sum_{i=r+1}^{n} p_{k_i}\}$, the server remains at the
     origin.
  2. Otherwise, let $r^* = \max\{\text{argmin}_{1 \leq r \leq n}\{2l_{k_r} + \sum_{i=r+1}^{n} p_{k_i}\}\}$:
       2a. go directly to $l_{k_{r^*}}$;
       2b. go back to the origin and serve all requests along the way.
```

We can now present the main online algorithm of this section: ReOpt.

Online Algorithm ReOpt:

0. Initial server location: $x = 0$, initial time $t = 0$, iteration counter $i = 0$. Go to Step 1.
1. If no new request, Stop: any previously unserved requests are rejected and the final makespan is t. Otherwise, $i \doteq i + 1$, the server remains idle at the origin until the time r_i of the next request. Go to Step 2.
2. If $x > 0$ the server calls OFP($x \neq 0$), otherwise it calls OFP(0) with $w = r_i - t$ [in each case, on the instance of requests which have been revealed but not yet served], and then follows the corresponding new solution. Go to Step 3.
3. If no new requests come before the time t_r the server finishes its current route, set $x = 0$, $t = t_r$ and go to Step 1. Otherwise, increment $i \doteq i + 1$, update x as the server's current location at time of this next request and t as the release date of the next request. Go to Step 2.

Theorem 2. *ReOpt is 2-competitive, and thus optimal.*

Proof. At the time of the last request, say request n, ReOpt is faced with an optimization problem containing all known information to the offline server, and this is thus a critical time to consider. Let C_{ReOpt} and C_{opt} be the objective value obtained by the online and offline server, on the overall instance. Also let C_i be the total cost that the online server would have obtained if only the first i requests were presented. So we have $C_{ReOpt} = C_n$. Consider now the following cases, depending on what the offline server has done on the overall instance:

1. The offline server does not accept any requests. Obviously, $C_{i+1} \leq C_i + p_{i+1}$, $C_1 \leq p_1$. So $C_{ReOpt} = C_n \leq \sum_{i=1}^{n} p_i = C_{opt}$. Hence $C_{ReOpt}/C_{opt} = 1$.
2. The offline server visits the last request n. Let S be the set of requests accepted by the offline server. Let $m = argmax_{i \in S}\{l_i\}$, and x be the position of the online server at time r_n:

   2a. If $x \leq l_m$, then it won't be worse if the online server moves to l_m and then go back to the origin (serving requests along the way). In this case, the online server will have served all requests in S. So$C_{ReOpt} \leq r_n + 2l_m + \sum_{i \notin S} p_i$, while $C_{opt} \geq \max\{r_n, 2l_m\} + \sum_{i \notin S} p_i$. So $C_{ReOpt}/C_{opt} \leq 2$.

2b. If $x > l_m$, then assume that the furthest location in his current route is L ($x \leq L$ and this location may or may not have been visited yet). Let us show that L cannot be too large. Let S_1 denote the set of requests not accepted by the offline server, but visited by ReOpt between his last leaving l_m (to the right) to his next arriving to l_m. We will show that $2(L - l_m) \leq \sum_{i \in S_1} p_i$. Assume that after every previous iteration of the algorithm, the furthest request that the online server is going to serve is $l_{q_1}, l_{q_2}, \cdots, l_{q_{n-1}}$ (if he is going to reject all requests, let $q_i = 0$ for all i and $l_0 = 0$). We can find $j_1 < j_2 < \cdots < j_k$, so that $l_m = l_{q_{j_0}} < l_{q_{j_1}} < \cdots < l_{q_{j_k}} = L$ and , for all i such that $j_{u-1} < i < j_u$ and $1 \leq u \leq k$, we have $l_{q_i} \leq l_{q_{j_{u-1}}}$. For $1 \leq u \leq k$, let $\bar{S}_u$ denote the set of locations that ReOpt serves between its first and second crossing of $l_{q_{j_{u-1}}}$. We then have $2(l_{q_{j_u}} - l_{q_{j_{u-1}}}) \leq \sum_{i \in \bar{S}_u} p_i$ (otherwise, at the time of the first crossing of $l_{q_{j_{u-1}}}$, the online server could have saved $\sum_{i \in \bar{S}_u} p_i - 2(l_{q_{j_u}} - l_{q_{j_{u-1}}})$ from its cost by not going to $l_{q_{j_u}}$, contradicting the optimality of the ReOpt algorithm). Since $S_1 = \cup_{u=1}^{k} \bar{S}_u$, $2(L - l_m) \leq \sum_{i \in S_1} p_i$. Now we can conclude with Case 2b. $C_{ReOpt} \leq r_n + 2L + \sum_{i \notin S} p_i - \sum_{i \in S_1} p_i \leq r_n + 2l_m + \sum_{i \notin S} p_i$, and $C_{opt} \geq \max\{r_n, 2l_m\} + \sum_{i \notin S} p_i$. Therefore, $C_{ReOpt}/C_{opt} \leq 2$.

3. The last accepted request by the offline server is request m, $m < n$. As $C_{i+1} \leq C_i + p_{i+1}$, then $C_{ReOpt} = C_n \leq C_m + \sum_{i=m+1}^{n} p_i$. Also, it is easy to check that the offline server's behavior will remain the same if only the first m requests are ever presented. Let $C_{opt}(m)$ be the optimal offline solution through the first m requests. We then have $C_{opt} = C_{opt}(m) + \sum_{i=m+1}^{n} p_i$. According to Case 2, $C_m/C_{opt}(m) \leq 2$. Therefore $c = C_{ReOpt}/C_{opt} \leq (C_m + \sum_{i=m+1}^{n} p_i)/(C_{opt}(m) + \sum_{i=m+1}^{n} p_i) \leq 2$.

$\square$

2.4 Competitive Ratios under Advanced Information

In this subsection we consider the situation where the online server receives advanced notice for each request in a problem instance. In other words, let $q_i \in \mathbb{R}_+$ be the i^{th} request's disclosure date: at time q_i, the server learns about request i and its corresponding values l_i, r_i, and p_i. We will be concerned here with the case where the quantity of advanced information is uniform over all requests: More specifically we assume that there exists a constant $a \in [0, r_{max}]$ such that $q_i = (r_i - a)^+$, $\forall i$, where $(x)^+ = \max\{x, 0\}$. Let $L = \max_i l_i$ and $\alpha = \frac{a}{L}$.

Lemma 2. *If $\alpha < 1$, any c-competitive online algorithm on $\mathbb{R}_+$ has $c \geq 2 - \frac{\alpha}{2}$.*

Proof. A generalization of the proof of Theorem 1. Assume that an online server A follows a given c-competitive online algorithm, where c is a finite constant. Let $y = \frac{1}{2-\alpha}$. Since $\alpha < 1$, we have $y < 1$. With n an arbitrary number to be defined by the adversary, consider a series of up to $n+1$ requests as follows: $(l_i, r_i, p_i) = \big(y^{i-1}L, (2 - y^{i-1})L, 2(y^{i-1} - y^i)L\big)$ for $1 \leq i \leq n$, and $(l_{n+1}, r_{n+1}, p_{n+1}) = (y^n L, (2-y^n)L, \infty)$. Let $C_A(k)$ and $C_{opt}(k)$ be the costs incurred by the online server A and the offline adversary, respectively, when the instance is drawn from the first k requests of the series. Let t_0 be the time the online server begins to move from the origin for the first time. Again note that $q_1 \leq t_0 \leq \infty$. Indeed if $0 \leq t_0 < q_1$, then no request would be presented, leading to $C_A > 0$, $C_{opt} = 0$, and thus an infinite c; also t_0 obviously needs to be finite in order to have a finite c. We can now consider two cases:

1. If $t_0 < q_{n+1}$, let $m \in \{1, 2, \ldots, n\}$, be such that $q_m \leq t_0 < q_{m+1}$. Then only the first m requests are presented, and the offline server rejects them all. Then $C_{opt}(m) = \sum_{i=1}^{m} p_i = (2 - 2y^m)L$. If A rejects all requests, then $C_A(m) \geq q_m + \sum_{i=1}^{m} p_i = (4 - y^{m-1} - 2y^m - \alpha)L$. Otherwise, let k, $k \leq m$, be the accepted request whose location is furthest away from the origin. We have $C_A(m) \geq q_m + 2l_k + \sum_{i=1}^{k-1} p_i = (4 - y^{m-1} - \alpha)L$. We then get $c \geq \frac{4 - y^{m-1} - 2y^m - \alpha}{2 - 2y^m} = 2 - \frac{\alpha}{2}$.

2. If $t_0 \geq q_{n+1}$, then all $n+1$ requests are presented. The offline server accepts them all, and we have $C_{opt}(n+1) = 2l_1 = 2L$. On the other hand, A will have to serve at least the last request $n+1$. If it serves only this last request, then $C_A(n+1) \geq q_{n+1} + 2l_{n+1} + \sum_{i=1}^{n} p_i = 2 - 2^{-n} + 2\dot{2}^{-n} + 2 - 2^{n-1} = (4 - y^n - \alpha)L$. Otherwise, let k, $k \leq n$, be the accepted request whose location is furthest away from the origin. We have $C_A(n+1) \geq q_{n+1} + 2l_k + \sum_{i=1}^{k-1} p_i = (4 - y^n - \alpha)L$. Therefore, either way, we have $c \geq 2 - \frac{\alpha}{2} - \frac{y^n}{2}$. By letting $n \to +\infty$, then $c \geq 2 - \frac{\alpha}{2}$.

$\square$

Let ReOpt(α) be the online algorithm ReOpt when decision times associated with the requests are made at their disclosure dates rather than their release dates.

Theorem 3. *If $0 < \alpha \leq 2$, Algorithm ReOpt(α) is $2 - \frac{\alpha}{2}$-competitive.*

Proof. A generalization of the proof of Theorem 2. At the disclosure date of the last request, say q_n, the online server faces an optimization problem with all known information and this is again a critical time to consider. Let $C_{ReOpt(\alpha)}$ and C_{opt} be the objective value obtained by the online and offline server, respectively on the overall instance. Also let C_i be the total cost that the online server would have obtained if only the first i requests where presented. So we have $C_{ReOpt(\alpha)} = C_n$. Consider now the following cases, depending on what the offline server has done on the overall instance:

1. The offline server does not accept any requests. Obviously, $C_{i+1} \leq C_i + p_{i+1}$, $C_1 \leq p_1$. So $C_{ReOpt(\alpha)} = C_n \leq \sum_{i=1}^{n} p_i = C_{opt}$. Hence $C_{ReOpt(\alpha)}/C_{opt} = 1$.

2. The offline server visits the last request n. Let S be the set of requests accepted by the offline server. Assume $m = argmax_{i \in S}\{l_i\}$. Let T be its makespan. Obviously, $T \geq \max\{r_n, 2l_m\}$. Let x be the position of the online server at time q_n:

   2a. If $x \leq l_m$, then it won't be worse if the online server moves to l_m, waits until $t_0 = \max\{T - l_m, q_n + l_m\}$ and then goes back to the origin (serving requests along the way). In this case, since for any $i \in S$ request i is ready to be served at time $T - l_i$, the online server will have served all requests in S on his last return. So $C_{ReOpt(\alpha)} \leq t_0 + l_m + \sum_{i \notin S} p_i \leq \max\{T, q_n + 2l_m\} + \sum_{i \notin S} p_i$. If $T \geq q_n + 2l_m, C_{ReOpt(\alpha)} \leq T + \sum_{i \notin S} p_i = C_{opt}$. Otherwise, $C_{ReOpt(\alpha)} \leq q_n + 2l_m + \sum_{i \notin S} p_i = r_n + 2l_m - a + \sum_{i \notin S} p_i$, while $C_{opt}(n) = T + \sum_{i \notin S} p_i \geq \max\{r_n, 2l_m\} + \sum_{i \notin S} p_i$. So $c \leq \frac{r_n + 2l_m - a + \sum_{i \notin S} p_i}{\max\{r_n, 2l_m\} + \sum_{i \notin S} p_i} \leq \max\{\frac{r_n + 2l_m - a}{\max\{r_n, 2l_m\}}, 1\}$. If $r_n \leq 2L$, $\frac{r_n + 2l_m - a}{\max\{r_n, 2l_m\}} = \frac{r_n}{\max\{r_n, 2l_m\}} + \frac{2l_m}{\max\{r_n, 2l_m\}} - \frac{a}{\max\{r_n, 2l_m\}} \leq 2 - \frac{a}{2L}$. If $r_n > 2L$, $\frac{r_n + 2l_m - a}{\max\{r_n, 2l_m\}} = \frac{r_n + 2l_m - a}{r_n} = 1 + \frac{2l_m - a}{r_n} \leq 1 + \max\{\frac{2l_m - a}{2L}, 0\} \leq 2 - \frac{a}{2L}$. Therefore, $c \leq 2 - \frac{a}{2L} = 2 - \frac{\alpha}{2}$.

   2b. If $x > l_m$, then assume that the furthest location in his current route is L_{max}. Let S_1 denote the set of cities being visited by the online server between the time of his last passing l_m and the time of his next passing l_m (l_m is not included). So $2(L_{max} - l_m) \leq \sum_{i \in S_1} p_i$ (see proof of Theorem 2). Let the server go to L_{max} and return to l_m, and then wait at l_m until $\max\{T - l_m, q_n + 2L_{max} - x - l_m\}$, and then return to the origin. So the server visits all the cities belonging to $S \cup S_1$. $C_{ReOpt(\alpha)} \leq \max\{T - l_m, q_n + 2L_{max} - x - l_m\} + l_m + \sum_{i \notin S \cup S_1} p_i$. If $T \geq q_n + 2L_{max}, C_{ReOpt(\alpha)} \leq T + \sum_{i \notin S \cup S_1} p_i \leq C_{opt}$. Otherwise, $C_{ReOpt(\alpha)} \leq q_n + 2L + \sum_{i \notin S \cup S_1} p_i \leq r_n - a + 2l_m + \sum_{i \notin S} p_i$. On the other hand, $C_{opt}(n) \geq \max\{r_n, 2l_m\} + \sum_{i \notin S} p_i$. Therefore, $c \leq \max\{1, \frac{r_n - a + 2l_m + \sum_{i \notin S} p_i}{\max\{r_n, 2l_m\} + \sum_{i \notin S} p_i}\} \leq \max\{1, 2 - \frac{a}{2L}\} = 2 - \frac{\alpha}{2}$.

3. The last accepted request by the offline server is request m, $m < n$. Then, $C_{ReOpt(\alpha)} \leq C_m + \sum_{i=m+1}^{n} p_i$. If only the first m requests are ever presented, let C'_{opt} be the optimal offline solution. We have $C_{opt} = C'_{opt} + \sum_{i=m+1}^{n} p_i$. According to Case 2, $C_m/C'_{opt} \leq 2 - \frac{\alpha}{2}$. Therefore $c = C_{ReOpt}/C_{opt} \leq (C_m + \sum_{i=m+1}^{n} p_i)/(C'_{opt} + \sum_{i=m+1}^{n} p_i) \leq 2 - \frac{\alpha}{2}$.

□

From Lemma 2 and Theorem 3 we have

Corollary 1. *Algorithm ReOpt(α) is optimal when $\alpha < 1$.*

3 Online TSP with Service Flexibility on $\mathbb{R}$

In this section, we study the problem when the locations of the requests are on the real line. In order to distinguish the requests on the negative side from those on the non-negative side, we will use $\bar{l}_i$ to indicate that request i's location is on the negative side. We will continue to use the same notation for both the location of a request i and its distance to the origin.

From Theorem 1 we know that the lower bound on the competitive ratio of any online algorithm is at least 2. We will prove here that ReOpt is in fact 2-competitive and thus optimal on the line.

First let us look at the offline version of the problem. Suppose there are r requests on the right side (including the origin; i.e., the non-negative side) and l on the negative side (for a total of $n = r + l$ requests). Reorder all requests as follows: $-\bar{l}_{j_l} \leq -\bar{l}_{j_{l-1}} \leq \cdots \leq -\bar{l}_{j_1} < 0 \leq l_{j_1} \leq \cdots \leq l_{j_r}$. Define $a_h = \max\{2l_h, l_h + r_h\}$ and $\bar{a}_h = \max\{2\bar{l}_h, \bar{l}_h + r_h\}$. Reorder as follows: $a_{i_1} \leq a_{i_2} \leq \cdots \leq a_{i_r}$ and $\bar{a}_{i_1} \leq \bar{a}_{i_2} \leq \cdots \leq \bar{a}_{i_l}$. If the server visits the right side requests first, the optimal solution should be in the following form: let t_1 be the time when the server arrives at the origin after visiting the requests on the right side, and t_2 be the time when he arrives at the origin after visiting the requests on the left side. Then, request h on the right side is visited if and only if $a_h \leq t_1$ and request h on the left side is visited if and only if $\bar{a}_h \leq t_2, 2\bar{l}_h \leq t_2 - t_1$. So $S = \{h | a_h \leq t_1\} \cup \{h | \bar{a}_h \leq t_2, 2\bar{l}_h \leq t_2 - t_1\}$ is the set of visited requests. Consequently, the server's cost should be $t_2 + \sum_{h \notin S} p_h$. In order to be optimal, $t_1 \in \{a_{i_1}, a_{i_2}, \cdots, a_{i_r}, 0\}$, $t_2 \in \{\bar{a}_{i_1}, \bar{a}_{i_2}, \cdots, \bar{a}_{i_l}, t_1 + 2\bar{l}_{i_1}, t_1 + 2\bar{l}_{i_2}, \cdots, t_1 + 2\bar{l}_{i_l}, t_1\}$ and $t_1 \leq t_2$. It is very similar if the server visits the left side requests first. Find the smallest value among these and move accordingly, then we get the optimal offline algorithm. The idea is also very similar when one considers solving the Flexible Path Problem on the line.

Theorem 4. *Algorithm ReOpt on $\mathbb{R}$ is 2-competitive, and thus optimal.*

Proof. The proof is quite similar to the proof of Theorem 2. Let C_{ReOpt} and C_{opt} be the objective value obtained by the online and offline server, on the overall instance. Let T_{opt} be the makespan of the optimal offline solution. Also let C_i be the total cost that the online server would have obtained if only the first i requests where presented. So we have $C_{ReOpt} = C_n$. Consider now the following cases, depending on what the offline server has done on the overall instance:

1. The offline server does not accept any requests. Obviously, $C_{i+1} \le C_i + p_{i+1}$, $C_1 \le p_1$. So $C_{ReOpt} = C_n \le \sum_{i=1}^{n} p_i = C_{opt}$. Hence $C_{ReOpt}/C_{opt} = 1$.

2. The offline server visits the last request n. Let S be the set of requests accepted by the offline server, and $[-L, R]$ be the interval covered. Let $P(t)$ be the position of the online server at any time t, and consider its position at time r_n, $x = P(r_n)$:

   2a. If $-L \le x \le R$. Without loss of generality, let $-L \le x \le 0$. Assume that at time T_1, T_2, the offline server is at point $-L$, R, respectively. Define $T_1^s = \max\{t < T_1 | P(t) = x\}, T_1^e = \min\{t > T_1 | P(t) = 0\}, T_2^s = \max\{t < T_2 | P(t) = 0\}, T_2^e = \min\{t > T_2 | P(t) = 0\}$. Let τ_i denote the portion of the route followed by the offline server from time T_i^s to T_i^e $(i = 1, 2)$. Obviously, τ_1 and τ_2 will not overlap. Let ReOpt start τ_1 at time r_n, and then start route τ_2 after arriving at the origin. By doing so, ReOpt can serve all the cities that the offline server serves. So $C_{ReOpt} \le r_n + T(\tau_1) + T(\tau_2) + \sum_{i \notin S} p_i \le 2T_{opt} + \sum_{i \notin S} p_i$, while $C_{opt} = T_{opt} + \sum_{i \notin S} p_i$. Therefore, $c \le 2$.

   2b. If $x \notin [-L, R]$. Without loss of generality, let $x < -L$. Assume that at time T_1, T_2, the offline server is at point $-L$, R, respectively. Define $T_1^s = T_1, T_1^e = \min\{t > T_1 | P(t) = 0\}, T_2^s = \max\{t < T_2 | P(t) = 0\}, T_2^e = \min\{t > T_2 | P(t) = 0\}$. Let τ_i denote the portion of the route followed by the offline server from time T_i^s to T_i^e $(i = 1, 2)$. Obviously, τ_1 and τ_2 will not overlap. Let S_0 denote the cities that ReOpt visits from last leaving $-L$ to next arriving to $-L$. The length of such a tour is L_0. Then, $L_0 \le \sum_{i \in S_0} p_i$. Let ReOpt ignore request n until his next arriving to $-L$, then follow route τ_1, and then follow route τ_2. In that case, $C_{ReOpt} \le r_n + L_0 + T_{opt} + \sum_{i \in S \setminus S_0} p_i \le T_{opt} + \sum_{i \in S_0} p_i + T_{opt} + \sum_{i \in S \setminus S_0} p_i \le 2T_{opt} + \sum_{i \in S} p_i$, while $C_{opt} = T_{opt} + \sum_{i \in S} p_i$. Therefore, $c \le 2$.

3. The last accepted request by the offline server is request m, $m < n$. As $C_{i+1} \leq C_i + p_{i+1}$, then $C_{ReOpt} = C_n \leq C_m + \sum_{i=m+1}^{n} p_i$. Also, it is easy to check that the offline server's behavior will remain the same if only the first m requests are ever presented. Let $C_{opt}(m)$ be the optimal offline solution through the first m requests. We then have $C_{opt} = C_{opt}(m) + \sum_{i=m+1}^{n} p_i$. According to Case 2, $C_m / C_{opt}(m) \leq 2$. Therefore $c = C_{ReOpt}/C_{opt} \leq (C_m + \sum_{i=m+1}^{n} p_i)/(C_{opt}(m) + \sum_{i=m+1}^{n} p_i) \leq 2$.

□

4 Online TSP with Service Flexibility on a General Metric Space

4.1 A 2.28-Competitive Algorithm

We assume here that the online algorithm has access to an optimal offline algorithm (a black box) for solving any given subproblem of the TSP with Flexible Service on those requests which have been revealed to the online server so far (hence ignoring the release dates of these requests while solving the TSP with Flexible Service subproblems).

For any $k \geq 1$, let C_k^* be the objective value of such an optimal offline solution (given by the black box) through the first k revealed requests, τ_k be its corresponding optimal route, and T_k be the corresponding makespan. Finally let S_k be the set of accepted cities among the first k requests.

Our proposed algorithm is such that the online server will follow the solution given by the re-optimizing black box only when at the origin, and only after waiting a proper amount of time there. Once started on a given new route - if a new request comes, say m, it will use the black box only to estimate the likelihood that this new request should eventually be served, and it will ask the question: is $m \in S_m$? If the answer is no, the server will continue on its current route; otherwise it will go back directly to the origin along the shortest path. This is thus a "plan at home" type of algorithm with the incorporation of (i) waiting time decisions at the origin, and (ii) service estimation decisions upon the arrival of any new requests. We will label this algorithm WOE for "Wait, Optimize, Estimate". A formal description of WOE is as follows. For simplicity, let c denote $\frac{\sqrt{17}+5}{4}$. So $2 + \frac{1}{2c-1} = c$. If n is the last request in a given instance, define $r_{n+1} \doteq +\infty$.

Algorithm WOE

```
Whenever a new request $m$ comes (at time $r_m$):
    1. If the server is at the origin, let $k$ be the largest index such that $k \in S_k$,
        i. If every request in $S_k$ has been visited, the server remains idle.
        ii. If at least one of them has not been visited, the server remains idle
            until $\max\{r_m, (c-1)C^*_m\}$, and then follows $\tau_k$.
    2. If the server is not at the origin and is currently completing a route,
        i. If $m \in S_m$, the server interrupts his route, and goes back to the
            origin along the shortest path. Go to Step 3.
        ii. If $m \notin S_m$, the server continues on his current route.
    3. If the server is not at the origin and has interrupted his route, he
        continues to go back to the origin. Let $t$ be the time he reaches the
        origin, $\bar{m}$ be the total number of requests at that time, and $\bar{k}$ be the
        largest index such that $\bar{k} \in S_{\bar{k}}$. Then the server remains idle at the
        origin until $\max\{t, (c-1)C^*_{\bar{m}}\}$, and then follows $\tau_{\bar{k}}$.
```

Lemma 3. *Assume there are n requests, then for any $l \in S_l$, the server is at the origin at time $(c-1)C^*_l$.*

Proof. When $l = 1$, it is very easy to check the server is at the origin at time $(c-1)C^*_1$. Assume the assertion holds for all $1 \le l < m, (m < n)$, let us consider the situation when $l = m$. We only need to prove the case when $m \in S_m$ and the server is at $x (\neq 0)$ other than the origin at time r_m. If the server is not following any route τ_i, then he must be on his way back to the origin as a result of decision associated with a previous request k, $(k < m)$. In that case, according to our induction, he will be at the origin before time $(c-1)C^*_k \le (c-1)C^*_l$. Otherwise, the server must follow a route τ_k, $k < m$, $k \in S_k$, and will arrive at the origin at time $t \doteq r_m + d(x, o)$. Assume $r_m = (c-1)C^*_k + \alpha T_k$, which means the server left the origin αT_k units of time ago and will finish its current route in $(1-\alpha)T_k$ units of time. Notice that $r_m = (c-1)C^*_k + \alpha T_k \ge (c-1)T_k + \alpha T_k = (c-1+\alpha)T_k$. If $0 < \alpha \le 0.5$, then $t = r_m + d(x, o) \le r_m + \alpha T_k \le r_m + \frac{\alpha}{c-1+\alpha} r_m \le r_m + \frac{0.5}{c-1+0.5} T_m \le (1 + \frac{1}{2c-1})C^*_m$. If $0.5 \le \alpha < 1$, then $t = r_m + d(x, o) \le r_m + (1-\alpha)T_k \le r_m + \frac{1-\alpha}{c-1+\alpha} r_m \le r_m + \frac{1-0.5}{c-1+0.5} r_m \le (1 + \frac{1}{2c-1})C^*_m$. Therefore, $t \le (1 + \frac{1}{2c-1})C^*_m = (c-1)C^*_m$ □

Lemma 4. *If $m \notin S_m$, then $C^*_m = C^*_{m-1} + p_m$*

Proof. First, when considering the instance drawn from the first m requests, let the server follow route τ_{m-1} and reject all requests which are not in τ_{m-1} (including request m). In that case we then have $C^*_m \le T_{m-1} + \sum_{i \notin S_{m-1}} p_i + p_m = C^*_{m-1} + p_m$.

Second, when considering the instance drawn from the first $m-1$ request, let the server follow route τ_m and reject all requests which are not in τ_m. Because request m is not among the first $m-1$ requests, its penalty should not be counted. In that case we then have $C^*_{m-1} \le T_m + \sum_{i \notin S_m} p_i - p_m = C^*_m - p_m$.

Consequently, $C^*_m = C^*_{m-1} + p_m$. □

Theorem 5. *WOE is c-competitive, where $c = \frac{\sqrt{17}+5}{4}$.*

Proof. Assume there are n requests. If for any $1 \le m \le n$, $m \notin S_m$, then according to Lemma 4, $C^*_n = C^*_{n-1} + p_n = \cdots = \sum_{i=1}^n p_i$. While the server will also rejects all the requests which makes $C_{WOE} = \sum_{i=1}^n p_i = C^*_n$. Otherwise, let m be the largest request such that $m \in S_m$. Then, according to Lemma 3, the server will be at the origin at time $(c-1)C^*_m$, then, he will start to follow his last tour τ_m. So $C_{WOE} \le (c-1)C^*_m + T_m + \sum_{i \notin S_m} p_i + \sum_{i=m+1}^n p_i = cC^*_m + \sum_{i=m+1}^n p_i \le cC^*_m + c\sum_{i=m+1}^n p_i = cC^*_n$. □

4.2 A Polynomial-Time Algorithm

Suppose we have a $\rho-$approximate polynomial-time offline algorithm A for the offline optimization subproblems considered in the previous section (i.e. for any sequence of requests $\mathcal{R}$, $C_A(\mathcal{R}) \le \rho C_{opt}(\mathcal{R})$). Let $x(t)$ denote the position of the online server at any time t, and $C(t)$ denote the cost of online algorithm if no more requests are presented after time t. Let C^*_i, T_i, and S_i be the optimal offline cost, time, and set of accepted requests by the offline optimal black box, and $\bar{C}^*_i$, $\bar{T}_i$, $\bar{S}_i$ be the corresponding quantity given by A, through the first i revealed requests.

The online algorithm is a simpler version of the WOE algorithm (without the waiting time at the origin, and with a different estimation condition for interrupting or not a given route). We will label this algorithm AE for "Approximation and Estimation"

Algorithm AE

When a new request m is presented, there are two options:

1. If $C(r_m) + p_m \ge r_m + d(x(r_m), o) + \bar{C}^*_m$, go back to the origin along the shortest way and then follow $\bar{\tau}_m$.

2. If $C(r_m) + p_m < r_m + d(x(r_m), o) + \bar{C}^*_m$, ignore the new request.

Theorem 6. *Algorithm AE is* $(1.5\rho + 1)$*-competitive.*

Proof. We will use induction to prove this theorem. It is easy to check that it holds if there is only 1 request. Assume it holds with at most $m - 1$ requests. Consider the case with m requests.

1. If $m \in S_m$, then $C^*_m \geq r_m$. Assume that the previous route that the online server follows is $\bar{\tau}_k$, $k < m$, then $d(x(r_m), o) \leq 0.5\bar{T}_k \leq 0.5\bar{C}^*_k \leq 0.5\rho C^*_k \leq 0.5\rho C^*_m$. So the total cost $\bar{C} = \min\{C(r_m) + p_m, r_m + d(x(r_m), o) + \bar{C}^*_m\} \leq r_m + d(x(r_m), o) + \bar{C}^*_m \leq C^*_m + 0.5\rho C^*_m + \rho C^*_m = (1.5\rho + 1)C^*_m$.
2. If $m \notin S_m$, then according to Lemma 4, $C^*_m = C^*_{m-1} + p_m$, and according to the induction, $C(r_m) \leq (1.5\rho + 1)C^*_{m-1}$. So the total cost $\bar{C} = \min\{C(r_m) + p_m, r_m + d(x(r_m), o) + \bar{C}^*_m\} \leq C(r_m) + p_m \leq (1.5\rho + 1)C^*_{m-1} + p_m \leq (1.5\rho + 1)C^*_m$.

□

5 Conclusions

In this paper we have considered a new online problem - the Online TSP with Flexible Service. Besides providing a solid building block for many real applications in routing, scheduling, robotics, etc., it also introduces a basic canonical enrichment of the online TSP by adding "yes-no" decisions on which points/requests to serve.

For some special metric spaces (the non-negative real line and the real line) we have been able to provide optimal 2-competitive polynomial time online algorithms based on re-optimization approaches. We have also quantified the impact of advanced information (lookahead) on this optimal competitive ratio. In a general metric space we have proposed an original c-competitive online algorithm, with $c = \frac{\sqrt{17}+5}{4} \approx 2.28$.

There are a number of interesting future research questions. First for the case of the general metric space, it would be interesting to close the gap between the current upper bound of 2.28 and the lower bound of 2, and to find the optimal ratio. Then there are a number of generalizations of this problem that would merit a similar analysis. We have indicated the online prize-collecting TSP of [6]. In fact it would be interesting to consider an online analysis of a most general form of a TSP with profits, for which requests have release dates, penalties for not being served, and profits for being served, and for which the goal is to bring enough profit (exceeding a given threshold), while minimizing the time to serve all accepted requests plus the sum of the penalties associated with the rejected requests.

Acknowledgements

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

References

[1] S. Albers. Competitive online algorithms. *OPTIMA: Mathematical Programming Society Newsletter*, 54:1–8, June 1997.

[2] 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.

[3] E. Angelelli, M. Savelsbergh, and M. Speranza. Competitive analysis for dynamic multi-period uncapacitated routing problems. *Networks*, 49:308–317, 2007.

[4] E. Angelelli, M. Savelsbergh, and M. Speranza. Competitive analysis of a dispatch policy for a dynamic multi-period routing problem. *Operations Research Letters*, 35:713–721, 2007.

[5] 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.

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

[7] 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.

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

[9] R. Bent and P. Van Hentenryck. Scenario based planning for partially dynamic vehicle routing problems with stochastic customers. *Operations Research*, 52(6):977–987, 2004.

[10] 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.

[11] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, first edition, 1998.

[12] E. Feuerstein and L. Stougie. On-line single-server dial-a-ride problems. *Theoretical Computer Science*, 268(1):91–105, 2001.

[13] A. Fiat and G. Woeginger. *Online Algorithms: The State of the Art*. Springer Verlag LNCS State of the Art Survey, 1998.

[14] P. Van Hentenryck and R. Bent. *Online Stochastic Combinatorial Optimization*. MIT Press, first edition, 2006.

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

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

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

[18] A. Karlin, M. Manasse, L. Rudolph, and D. Sleator. Competitive snoopy caching. *Algorithmica*, 3:79–119, 1988.

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

[20] 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.

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

[22] D. Sleator and R. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[23] M. Wagner. Online tsp with rejections. Short note, California State University East Bay, October 2006.