Online Time-Windows TSP with Predictions

Shuchi Chawla
University of Texas at Austin, United States

Dimitris Christou
University of Texas at Austin, United States

Abstract

In the *Time-Windows TSP* (TW-TSP) we are given requests at different locations on a network; each request is endowed with a reward and an interval of time; the goal is to find a tour that visits as much reward as possible during the corresponding time window. For the online version of this problem, where each request is revealed at the start of its time window, no finite competitive ratio can be obtained. We consider a version of the problem where the algorithm is presented with predictions of where and when the online requests will appear, without any knowledge of the quality of this side information.

Vehicle routing problems such as the TW-TSP can be very sensitive to errors or changes in the input due to the hard time-window constraints, and it is unclear whether imperfect predictions can be used to obtain a finite competitive ratio. We show that good performance can be achieved by explicitly building slack into the solution. Our main result is an online algorithm that achieves a competitive ratio logarithmic in the diameter of the underlying network, matching the performance of the best offline algorithm to within factors that depend on the quality of the provided predictions. The competitive ratio degrades smoothly as a function of the quality and we show that this dependence is tight within constant factors.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Travelling Salesman Problem, Predictions, Learning-Augmented Algorithms, Approximation

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2024.2

Category APPROX

Related Version Due to space limitations, some proofs are deferred to the full version of this paper.
Full Version: `https://arxiv.org/abs/2304.01958` [13]

Funding *Shuchi Chawla*: This work was funded in part by NSF award CCF-2217069.

1 Introduction

Many optimization problems exhibit a large gap in how well they can be optimized offline versus when their input arrives in online fashion. In order to obtain meaningful algorithmic results in the online setting, a natural direction of investigation is to consider "beyond worst case" models that either limit the power of the adversary or increase the power of the algorithm. A recent line of work in this direction has considered the use of *predictions* in bridging the offline versus online gap. Predictions in this context are simply side information about the input that an online algorithm can use for its decision making; the true input is still adversarially chosen and arrives online. The goal is to show that on the one hand, if the predictions are aligned with the input, the algorithm performs nearly as well as in the offline setting (a property known as *consistency*); and on the other hand, if the predictions are completely unrelated to the input, the algorithm nevertheless performs nearly as well as the best online algorithm (a.k.a. *robustness*). *Put simply, good predictions should help, but bad predictions should not hurt, and ideally we should reap the benefits without any upfront knowledge about the quality of the predictions.*

Predictions have been shown to effectively bypass lower bounds for a variety of different online decision-making problems including, for example, caching [30, 31, 25], scheduling [12, 24, 3], online graph algorithms [4], load balancing [27, 28], online set cover [6], matching problems [17], k-means [19], secretary problems [1, 18], network design [20, 33, 9] and more.[1]

In this paper, we consider a problem whose objective function value is highly sensitive to changes in the input, presenting a significant challenge for the predictions setting. In the Traveling Salesman Problem with Time Windows (TW-TSP for short), we are given a sequence of service requests at different locations on a weighted undirected graph. Each request is endowed with a reward as well as a time window within which it should be serviced. The goal of the algorithm is to produce a path that maximizes the total reward of the requests visited within their respective time windows. In the online setting, the requests arrive one at a time at the start of their respective time windows, and the algorithm must construct a path incrementally without knowing the locations or time windows of future requests.

Vehicle routing problems such as the TW-TSP that involve hard constraints on the lengths of subpaths (e.g. the time at which a location is visited) are generally more challenging than their length-minimization counterparts. In particular, a small bad decision at the beginning of the algorithm, such as taking a slightly suboptimal path to the first request, can completely obliterate the performance of the algorithm by forcing it to miss out on all future reward. In the offline setting, this means that the approximation algorithm has to carefully counter any routing inefficiency in some segments by intentionally skipping reachable value in other segments. In the online setting, this means that no sublinear competitive ratio is possible.

Given the sensitivity of the TW-TSP objective to small routing inefficiencies, is it possible to design meaningful online algorithms for this problem using imperfect predictions?

We consider a model where the algorithm is provided with a predicted sequence of requests at the beginning, each equipped with a predicted location and a predicted time window. The true sequence of requests is revealed over time as before. Of course if the predicted sequence is identical to the true request sequence, the algorithm can match the performance of the best offline algorithm. But what if the predictions are slightly off? Could these small errors cause large losses for the online algorithm? Can the algorithm tolerate large deviations?

Our main result is an online algorithm for the TW-TSP based on predictions whose performance degrades smoothly as a function of the errors in prediction. We obtain this result by explicitly building slack into our solution and benchmark. In a slight departure from previous work on TW-TSP, we require the server to spend one unit of idle “service time” at each served request. We show that this is necessary to obtain a sublinear approximation even with predictions (Theorem 9). (However, in the absence of predictions, the setting with service times continues to admit a linear lower bound on the competitive ratio; See Theorem 8.) We then use service times judiciously in planning a route and accounting for delays caused by prediction errors.

There are two primary sources of error in predictions: (1) the predicted locations of requests may be far from the true locations; and, (2) the predicted time windows may be different from the true time windows. The competitive ratio of our algorithm depends linearly on each of these components, taking the maximum error over each predicted request and normalizing appropriately.[2] This dependence is tight to within constant factors. Besides this

[1] A comprehensive compendium of literature on the topic can be found at [29].
[2] Formally, the competitive ratio depends linearly on the ratio of the maximum location error of any predicted request to the minimum service time, as well as the ratio of the maximum time window error to the minimum time window length.

dependence on the prediction error, the competitive ratio depends logarithmically on the diameter of the underlying network, matching the performance of the best known offline algorithm for TW-TSP.

Although our competitive ratio is stated in terms of the maximum location or time window errors, where the maximum is taken over all requests in the instance, our algorithm performs well even when some of the errors are large and most errors are small. In particular, our algorithm's performance is *simultaneously* competitive against the maximum achievable reward over *any* subset of requests, scaled down by the maximum prediction error over that subset. (See "Extensions" in Section 3 for a formal statement.) In this respect, our guarantees fall within the framework of *metric error with outliers* proposed by [4]. On the other hand, when all or most requests are predicted poorly, our algorithm also inevitably performs poorly as it inherits lower bounds from fully online instances.

Importantly, our algorithm requires little to no information about how the predictions match up against the true requests. For the purpose of analysis, we measure the error in predictions with respect to some underlying matching between the predicted and true requests – the error parameters are then defined in terms of the maximum mismatch between any predicted request and its matched true request. This matching is never revealed to the algorithm and in fact the performance of the algorithm depends on the quality of the *best* possible matching between the predicted and true requests. The only information the algorithm requires about the quality of the predictions is the location error – the maximum distance between any prediction and its matched true request. Even for this parameter, an upper bound suffices (and at a small further loss, a guess suffices).

Our overall approach has several components. The first of these is to construct an instance of the TW-TSP over predicted requests that requires the server to spend some idle time at each request as a "service delay". We then extend offline TW-TSP algorithms to this service delay setting, obtaining a logarithmic in diameter approximation. We then follow and adapt this offline solution in the online setting. Every time the offline solution services a predicted request, we match this request to a previously revealed true request, take a detour from the computed path to visit and service the true request, and then resume the precomputed path. Altogether this provides the desired competitive ratio.

Our results further generalize to a setting where predictions are coarse in that each single predicted location captures multiple potential true requests that are nearby. We show that with prediction errors defined appropriately, we can again achieve a competitive ratio for this "many to one matching" setting that is logarithmic in the diameter of the graph and polynomial in the prediction error.

Finally, our algorithm and analysis incorporates error in estimating rewards of requests in a straightforward manner achieving the optimal dependence on this third source of error.

Further related work

Using predictions in the context of online algorithm design was first proposed by [30] for the well-studied caching problem. Since that work, the literature on online algorithm design with predictions has grown rapidly. We point the interested reader to a compendium at [29] for further references.

Metric error with outliers

Azar et al. [4] initiated the study of predictions in the context of online graph optimization problems, and proposed a framework for quantifying errors in predictions, called *metric error with outliers*, that we adapt. The idea behind this framework is to capture two sources of error: (1) Some true requests may not be captured by predictions and some predictions may not correspond to any true requests; (2) For requests that are captured by predictions, the predictions may not be fully faithful or accurate. The key observation is that it is possible to design algorithms with performance that depends on these two sources of error *without explicit upfront knowledge of the (partial) correspondence between predicted and true requests.*

In this work, we focus mostly on the second source of error, which we further subdivided into three kinds of error in order to obtain a finer understanding of the relationship between the competitive ratio and different kinds of error. As in the work of Azar et al. [4], we assume that the correspondence between predicted and true requests is never explicitly revealed to the algorithm. The performance of the algorithm nevertheless depends on the error of the *best* matching between predicted and true requests. In Section 3 we describe how the first source of error in Azar et al.'s framework can also be incorporated into our bounds.

TSP with predictions

Recently a few papers [10, 23, 22] have considered the online TSP and related routing problems with predictions. The input to the online TSP is similar to ours: requests arrive over time in a graph, and a tour must visit each request after its arrival time. However, the objective is different. In our setting, requests also have deadlines, and the algorithm cannot necessarily visit all requests. The goal therefore is to visit as many as possible. In the online TSP, there are no deadlines, and so the objective is to visit *all* requests as quickly as possible, or in other words to minimize the makespan. This makespan minimization objective is typically much easier than the deadline setting, as evidenced by constant factor approximations for it in the offline, online, and predictions settings, as opposed to logarithmic or worse approximations for the latter problem.

The algorithmic idea of precomputing an offline path based on the predictions and then adapting it to the online input has also been used in [10, 23]. The main challenge in the setting that our works considers, is that due to the existence of deadlines, our algorithm needs to be careful on how it adapts its path, as taking a large detour could result in entirely missing the time-windows of future (unrevealed) requests. We circumvent this issue by introducing appropriately large idle times on the predicted requests that our offline solution visits.

TW-TSP *without* predictions

The (offline) TW-TSP problem has a rich literature and has been studied for over 20 years. The problem is known to be NP-hard even for special cases, e.g. on the line [32], and when all requests have the same release times and deadlines (a.k.a. Orienteering) [11]. Orienteering admits constant factor approximations [11, 7, 14], and even a PTAS when requests lie in a fixed dimensional Euclidean space [2, 16]. For general time windows, constant factor approximations are only known for certain special cases: e.g. constant number of distinct time windows [15]; and on line graphs [32, 26, 8, 21]. For general graphs and time-windows, the best approximations known are logarithmic in input parameters [7, 14].

To the best of our knowledge, the online setting for TW-TSP has only been considered by Azar and Vardi [5]. Azar and Vardi assume that service times are non-zero and present competitive algorithms under the assumption that the smallest time window length $L_{\min}$ is comparable to the diameter D of the graph, as no sublinear competitive ratio can be achieved if $L_{\min} < D/2$. We are able to beat this lower bound by relying on predictions.

Organization of the paper

We formally define the problem and our error model in Section 2. Section 3 describes our results and provides an outline for our analysis. All of our main results are covered in that section. Proofs of these results can be found in subsequent sections. In particular, Sections 4, 5, and 6 fill in the details of our upper bound, and Section 7 proves the stated lower bounds. Proofs omitted from the main body of this paper can be found in the appendix of the full version [13].

2 Definitions

2.1 The Traveling Salesman Problem with Time-Windows

An instance of the TW-TSP consists of a network G and a (finite) sequence of service requests I. Here, $G = (V, E, \ell)$ is an undirected network with edge lengths $\{\ell_e\}_{e \in E}$. Extending the notion of distance to all vertex pairs in G, we define $\ell(u, v)$ for $u, v \in V$ to be the length of the shortest path from u to v. We assume without loss of generality that G is connected and that the edge lengths ℓ_e are integers. A service request $\sigma = (v_\sigma, r_\sigma, d_\sigma, \pi_\sigma)$ consists of a vertex $v_\sigma \in V$ at which the request arrives, a release time $r_\sigma \in \mathbb{Z}^+$, a deadline $d_\sigma \in \mathbb{Z}^+$ with $d_\sigma > r_\sigma$, and a reward $\pi_\sigma \in \mathbb{Z}^+$. We use $\Sigma \subseteq V \times \mathbb{Z}^+ \times \mathbb{Z}^+ \times \mathbb{Z}^+$ to denote the set of all possible client requests and $I \subset \Sigma$ to denote the set of requests received by the algorithm.

The solution to TW-TSP is a continuous *walk* on G that is allowed to remain idle on the vertices of the graph.[3] Formally, the walk starts from some vertex at time $t = 0$; at every time-step that it occupies a vertex $u \in V$, it can either remain idle on u for some number of time-steps *or* it can move to some $v \in V$ by spending time $t = \ell_{(u,v)}$; we comment that while the path is mid-transition, no more decisions can be made. Notice that this creates a notion of a discrete time-horizon that will be important towards formalizing the online variant of the problem.

We use $\mathcal{W}(G)$ to denote the set of all walks on G. Given a request $\sigma \in \Sigma$, we say that a walk W *covers* it if W remains idle on vertex v_σ for at least one time-step,[4] starting on some step $\tau \in [r_\sigma, d_\sigma - 1]$. For a sequence of requests $I \subset \Sigma$, we use $\mathrm{Cov}(W, I) \subseteq I$ to denote the set of requests in I that are covered by W. Then, the reward obtained by walk W is denoted by $\mathrm{Rew}(W, I) := \sum_{\sigma \in \mathrm{Cov}(W,I)} \pi_\sigma$. The objective of TW-TSP is to compute a walk $W \in \mathcal{W}(G)$ of maximum reward. We denote this by $\mathrm{OPT}(G, I) := \max_{W \in \mathcal{W}(G)} [\mathrm{Rew}(W, I)]$.

2.2 The offline, online, and predictions settings

We assume that the network G is known to the algorithm upfront in all of the settings we consider. In the **offline** version of the problem, the sequence of requests I is given to the algorithm in advance. In the **online** version, requests $\sigma \in I$ arrive in an online fashion; specifically, each request $\sigma \in I$ is revealed to the algorithm at its release time r_σ.

In the **predictions** setting, the true sequence of requests I arrives online, as in the online setting. However, the algorithm is also provided with a predicted sequence $I' \subset \Sigma$ in advance, where every request $\sigma' \in I'$ is endowed with a location, a time window, and a reward. The

[3] To keep our exposition simple, we do not specify a starting location for the walk. However, all of our algorithms can be adapted without loss to the case where a starting location is fixed, as described towards the end of Section 3.

[4] As we mentioned in the introduction, this requirement of a minimal one-unit service time is necessary in order to achieve any sublinear approximation for the online TW-TSP even with predictions. See Theorem 9 in Section 7.

quality of predictions is expressed in terms of their closeness to true requests. To this end, we define three notions of mismatch or error. For a true request σ and predicted request σ', the location error, time windows error, and reward error are defined as:

$$\begin{aligned}\text{LocErr}(\sigma,\sigma') &:= \ell(v_\sigma, v_{\sigma'})\\ \text{TWErr}(\sigma,\sigma') &:= \max\{|r_\sigma - r_{\sigma'}|, |d_\sigma - d_{\sigma'}|\}\\ \text{RewErr}(\sigma,\sigma') &:= \max\{\pi_\sigma/\pi_{\sigma'}, \pi_{\sigma'}/\pi_\sigma\}\end{aligned}$$

We extend these definitions to the entire sequences I and I' through an underlying (but unknown to the algorithm) matching between the requests in the two lists:

▶ **Definition 1.** *Given two request sequences $I, I' \subset \Sigma$ with $|I| = |I'|$ and a perfect matching $M : I \mapsto I'$, we define the location, time window, and reward errors for the matching M as:*

$$\begin{aligned}\Lambda_M &:= \max_{\sigma\in I} \text{LocErr}(\sigma, M(\sigma))\\ \tau_M &:= \max_{\sigma\in I} \text{TWErr}(\sigma, M(\sigma))\\ \rho_M &:= \max_{\sigma\in I} \text{RewErr}(\sigma, M(\sigma))\end{aligned}$$

We use $n = |V|$ to denote the number of vertices in G, D to denote the diameter of the graph, and $L_{\min}$ and $L_{\max}$ to denote the size of the smallest and largest time windows respectively (of a true or predicted request) in the given instance; that is, we denote $L_{\min} = \min_{\sigma\in I\cup I'} |d_\sigma - r_\sigma|$ and $L_{\max} = \max_{\sigma\in I\cup I'} |d_\sigma - r_\sigma|$. The competitive ratios of the algorithms we develop depend on these parameters.

Knowns and unknowns

We denote an instance of the TW-TSP with predictions by (G, I, I', M). All components of the instance are chosen adversarially. As mentioned earlier, the network G and the predicted sequence I' are provided to the algorithm at the start. The sequence I arrives online. We assume that the algorithm receives no direct information about the matching M, but is provided with an upper bound on the error Λ_M. We will also assume that the algorithm knows the parameter $L_{\min}$, although this is without loss of generality as the parameter can be inferred within constant factor accuracy from the predictions.[5]

2.3 The TW-TSP with service times

At a high level our algorithm has two components: an offline component that computes a high-reward walk over the predicted locations of requests, and an online component that largely follows this walk but takes "detours" to cover the arriving true sequence of requests. In particular, as the algorithm follows the offline walk, for each predicted location it visits where a "close by" true request is available, the algorithm takes a "detour" to this true request, returns back to the predicted location, and resumes the remainder of the walk. In order to incorporate the time spent taking these detours in our computation of the offline walk, we require the walk to spend some "service time" at each predicted location it covers. Accordingly, we define a generalization of the TW-TSP:

[5] In particular, assuming $\tau_M \le L_{\min}/2$, which is necessary for our results to hold, the time window of any true request can be no shorter than half the smallest time window of any predicted request.

▶ **Definition 2.** *The* TW-TSP with Service Times *(TW-TSP-S) takes as input a network G, a sequence of service requests I, and a service time $S \in \mathbb{Z}^+$, and returns a walk $W \in \mathcal{W}(G)$. We say that W covers a request $\sigma \in I$, denoted $\sigma \in \mathrm{Cov}(W, I, S)$, if it remains idle on vertex v_σ for at least S time steps, starting at some step $t \in [r_\sigma, d_\sigma - S]$. We define the reward of W as $\mathrm{Rew}(W, I, S) := \sum_{\sigma \in \mathrm{Cov}(W,I,S)} \pi_\sigma$. The optimal value of the instance is given by:*

$$\mathrm{OPT}(G, I, S) := \max_{W \in \mathcal{W}(G)} [\mathrm{Rew}(W, I, S)].$$

Note that the original version of TW-TSP as defined previously simply corresponds to the special case of TW-TSP-S with service time $S = 1$, and in particular, we have $\mathrm{Rew}(W, I) = \mathrm{Rew}(W, I, 1)$, and $\mathrm{OPT}(G, I) = \mathrm{OPT}(G, I, 1)$.

3 Our results and an outline of our approach

Our main result is as follows.

▶ **Theorem 3.** *Given any instance (G, I, I', M) of the TW-TSP with predictions whose errors satisfy $\tau_M \le L_{\min}/2$ and $\Lambda_M \le (L_{\min} - 1)/4$, there exists a polynomial-time online algorithm that takes the tuple (G, I', Λ_M) as offline input and I as online input, and constructs a walk $W \in \mathcal{W}(G)$ such that*

$$\mathbb{E}[\mathrm{Rew}(W, I)] \ge \frac{1}{O(\Lambda_M \cdot \rho_M^2 \cdot \log \min(D, L_{\max}))} \cdot \mathrm{OPT}(G, I).$$

As mentioned previously, our algorithm consists of two components. The offline component constructs a potential walk in the network with the help of the predicted requests. Then an online component adapts this walk to cover true requests that arrive one at a time. We break up the design and analysis of our algorithm into four steps. The first two steps relate the offline instance we solve to the hindsight optimal solution for the online instance. The third step then applies an offline approximation to the predicted instance with appropriate service times. The final step deals with the online adaptation of the walk to the arriving requests.

The following four lemmas capture the four steps. First, we show (Section 4) that introducing a service time of S hurts the optimal value by at most a factor of $2S - 1$. As we prove in Lemma 14 of Section 4, this dependency on S is tight. Observe that we require $S \le L_{\min}$, as for any tour to feasibly cover a request, the service time for that request must fit within its time window.

▶ **Lemma 4.** *For any instance (G, I) of the TW-TSP with service times, and any integer $S \le L_{\min}$, we have*

$$\mathrm{OPT}(G, I, S) \ge \frac{1}{2S - 1} \cdot \mathrm{OPT}(G, I, 1).$$

Our second step (also in Section 4) relates the value of the optimal solution over the true requests I to the optimum over the predicted sequence I'. In both cases, we impose some service time requirements. Note that this argument needs to account for the discrepancy in locations, time windows, as well as the rewards of the true and predicted requests.

▶ **Lemma 5.** *Let (G, I, I', M) be an instance of the TW-TSP with predictions, where Λ_M, ρ_M, and τ_M denote the maximum location, reward, and time window errors of the instance respectively. Define $S := 4\Lambda_M + 1$ and $S' := 2\Lambda_M + 1$. Then, if $\tau_M \le L_{\min}/2$ and $\Lambda_M \le (L_{\min} - 1)/4$, we have*

$$\mathrm{OPT}(G, I', S') \ge \frac{1}{3\rho_M} \cdot \mathrm{OPT}(G, I, S).$$

Our third step (Section 5) captures the offline component of our algorithm: computing an approximately optimal walk over the predicted requests with the specified service times. For this we leverage previous work on the TW-TSP without service times and show how to adapt it to capture the service time requirement.

▶ **Lemma 6.** *Given any instance (G, I', S') of the TW-TSP with service times, there exists a polynomial time algorithm that returns a walk $W \in \mathcal{W}(G)$ with reward*

$$\mathrm{Rew}(W, I', S') = \frac{1}{\mathcal{O}\left(\log \min(D, L_{\max})\right)} \cdot \mathrm{OPT}(G, I', S').$$

Finally, the fourth component (Section 6) addresses the online part of our algorithm. Given a walk computed over the predicted request sequence, it solves an appropriate online matching problem to construct detours to capture true requests. As in Lemma 5, this part again needs to account for the discrepancy in locations, time windows, as well as the rewards of the true and predicted requests.

▶ **Lemma 7.** *Given an instance (G, I, I', M) of the TW-TSP with predictions satisfying $\tau_M \le L_{\min}/2$ and $\Lambda_M \le (L_{\min} - 1)/4$; a walk $W' \in \mathcal{W}(G)$; and any integer $S' \ge 2\Lambda_M + 1$, there exists an online algorithm (Algorithm 1) that returns a walk $W \in \mathcal{W}(G)$ with expected reward*

$$\mathbb{E}\left[\mathrm{Rew}(W, I, 1)\right] \ge \frac{1}{6\rho_M} \cdot \mathrm{Rew}(W', I', S').$$

Theorem 3 follows immediately by putting Lemmas 4, 5, 6 and 7 together.

Lower bounds and tightness of our results

We show that the online TW-TSP does not admit sublinear competitive algorithms in the absence of predictions if $L_{\min} < D$, even with non-zero service times. Furthermore, if the service times are all 0, no sublinear competitive ratio is possible even using predictions that are accurate in all respects except the request location. Therefore, in order to achieve a nontrivial competitive ratio, it is necessary to use predictions as well as to impose non-zero service times on the optimum. The proofs are presented in Section 7.

▶ **Theorem 8.** *The competitive ratio of any randomized online algorithm for Online TW-TSP on instances with $L_{\min} \le D$ and all service times equal to 1 is at most $1/n$.*

▶ **Theorem 9.** *For any $S > 0$, there exists an instance (G, I, I', M) of the TW-TSP with predictions and service times 0, satisfying $\tau_M = 0$, $\rho_M = 1$, and $\Lambda_M = S$, such that any randomized online algorithm taking the tuple (G, I', Λ_M) as offline input and I as online input achieves a reward no larger than $O(1/n) \cdot \mathrm{OPT}(G, I, 0)$. Here n is the number of vertices in G.*

As mentioned earlier, the best known approximation factor for the offline TW-TSP is $O(\log L_{\max})$ (which we show can be improved slightly to $O(\log \min(D, L_{\max}))$). We inherit this logarithmic dependence on D and $L_{\max}$ in the predictions setting. Furthermore, any improvements to the offline approximation would immediately carry through into our competitive ratio as well. In particular, given an offline TW-TSP algorithm that achieves a competitive ratio of $\alpha(D, L_{\max})$, we obtain an online algorithm that achieves a competitive ratio of $O(\Lambda_M \cdot \rho_M^2 \cdot \alpha(D, L_{\max}))$.

The dependence of our bound on ρ_M can easily be seen to be tight – consider a star graph with requests on leaves, and edge lengths and time windows defined in such a manner that any feasible walk can cover at most one request. Then an uncertainty of a factor of

ρ_M in the predicted rewards can force any online algorithm to obtain an $\Omega(\rho_M^2)$ competitive ratio even if the predictions are otherwise perfect. Finally, we show in Section 7.2 that the dependence of our competitive ratio on Λ_M is also tight:

▶ **Theorem 10.** *For any $S > 0$, there exists an instance (G, I, I', M) of the TW-TSP with predictions satisfying $\tau_M = 0$, $\rho_M = 1$, and $\Lambda_M = S$ such that the competitive ratio of any randomized online algorithm taking the tuple (G, I', Λ_M) as offline input and I as online input asymptotically approaches $1/(S + 1)$.*

Extensions and generalizations

We now describe some ways in which we can weaken the assumptions in Theorem 3 while maintaining its competitive ratio guarantee:

- **Lack of knowledge of $\boldsymbol{\Lambda_M}$.** Our algorithm continues to work as intended if it is provided with an upper bound on Λ_M rather than the exact value of the parameter, with the performance of the algorithm degrading linearly with the upper bound, as in the theorem above. One such upper bound is simply $L_{\min}/4$. Moreover, by guessing Λ_M within a factor of 2 in the range $[0, L_{\min}/4]$, we can obtain the claimed approximation with a further loss of $\mathcal{O}(\log L_{\min})$. Thus, our algorithm can achieve non-trivial guarantees that scale with the location error even in settings where no information is given about any of the prediction errors $\Lambda_M, \tau_M, \rho_M$.

- **Assumptions on $\boldsymbol{\tau_M}$ and $\boldsymbol{\Lambda_M}$.** It is easy to see that it is necessary to assume $\Lambda_M \leq L_{\min}$ to obtain a nontrivial competitive ratio, as predictions with a location error larger than the time window size are of no value to the online algorithm. On the other hand, assuming $\tau_M \leq L_{\min}$ is not necessary. We can accommodate larger time window errors by following one out of roughly $\tau_M/L_{\min}$ different time shifts of the offline walk. This worsens our approximation factor by an additional factor of $\tau_M/L_{\min}$. In particular, this algorithm achieves a competitive ratio of $O(\Lambda_M \cdot \rho_M^2 \cdot \tau_M/L_{\min} \cdot \log \min(D, L_{\max}))$.

- **Random rewards.** Our results also hold in the case of random rewards. Specifically, consider a setting where the rewards $\{\pi_\sigma\}_{\sigma \in I}$ are drawn from some joint (not necessarily product) distribution D over $\mathbb{R}_+^I$. In that case, we define $\mathrm{Rew}(W, I) := \sum_{\sigma \in \mathrm{Cov}(W,I)} \mathrm{E}[\pi_\sigma]$, and $\mathrm{OPT}(G, I)$ as the maximum reward obtained by any walk $W \in \mathcal{W}(G)$.[6] Finally, we define $\mathrm{RewErr}(\sigma, \sigma')$ as the mismatch between π'_σ and $\mathrm{E}[\pi_\sigma]$. Our analysis provides the same approximation as before in this setting. See the full version for a formal proof.

  ▶ **Corollary 11.** *Given an instance (G, I, I', M) of the TW-TSP with predictions where requests have randomly drawn rewards, and predictions errors satisfy that $\tau_M \leq L_{\min}/2$ and also $\Lambda_M \leq (L_{\min} - 1)/4$, there exists a polynomial-time online algorithm that takes the tuple (G, I', Λ_M) as offline input and I as online input, and constructs a walk $W \in \mathcal{W}(G)$ such that*

$$\mathbb{E}[\mathrm{Rew}(W, I)] \geq \frac{1}{O(\Lambda_M \cdot \rho_M^2 \cdot \log \min(D, L_{\max}))} \cdot \mathrm{OPT}(G, I)$$

[6] Note that we do not allow the optimal walk to adapt to instantiations of rewards. Adaptive walks cannot be competed against in an online setting even with predictions.

- **Rooted instances.** Next, we consider the case where a starting vertex v_0 is also specified, and the solution space $\mathcal{W}(G)$ includes all walks on G that *start* on vertex v_0 at $t = 0$. We can easily see that this setting is essentially equivalent to its unrooted counterpart, under the extra assumption that each request $\sigma = (v_\sigma, r_\sigma, d_\sigma, \pi_\sigma)$ satisfies the conditions $\ell(v_0, v_\sigma) \le r_\sigma$. This is a reasonable assumption as no algorithm can visit a request σ before time $\ell(v_0, v_\sigma)$ anyway. Clearly, for any rooted instance (G, I, v_0), the unrooted optimal $\mathrm{OPT}(G, I)$ is an upper bound on the rooted optimal $\mathrm{OPT}(G, I, v_0)$. On the other hand, the unrooted path computed by our algorithm can be transformed to a path of same reward rooted at v_0 by going directly from v_0 to the predicted request serviced first, as this distance is at most equal to the request's release time.
- **Partial matching.** Next we consider the case where not all true requests are captured by the predicted requests and, on the flip side, where some predicted requests do not correspond to true requests at all. Following the framework of [4], we consider partial matchings between I and I', and define Δ_1^M to be the total reward of all true requests that are unmatched, and Δ_2^M to be the total predicted reward of predicted requests that are unmatched. Then, it is easy to see that our analysis goes through for the subsets of I and I' that are matched to each other, costing us an additive amount of no more than $\Delta_1^M + \Delta_2^M$. See the full version for a formal proof.

  ▶ **Corollary 12.** *Given an instance (G, I, I') of the TW-TSP with predictions, let M be any (incomplete) matching between I and I', and let the error parameters $\Lambda_M, \rho_M, \tau_M, \Delta_1^M$, and Δ_2^M be defined as above. Then, there exists an online algorithm that takes (G, I', Λ_M) as offline input and I as online input, and returns a walk $W \in \mathcal{W}(G)$ such that*

  $$\mathbb{E}\left[\mathrm{Rew}(W, I)\right] \ge \Omega\left(\frac{1}{\Lambda_M \cdot \rho_M^2 \cdot \log\min(D, L_{\max})}\right) \cdot \left(\mathrm{OPT}(G, I) - \Delta_1^M\right) - \frac{\Delta_2^M}{\rho_M}.$$

- **Many to one matching.** Consider a setting where predictions are coarse in that each single predicted location captures multiple potential true requests. We can model such a setting within our predictions framework and obtain almost the same guarantee as in Theorem 3. In particular, for this setting, let M be a many-to-one matching from I to I'. We define the location error of a predicted request $\sigma' \in I'$ as the length of the shortest path that starts at σ', visits all of the locations of the true requests that are preimages of σ' in M, spending one unit of time at each, and returns back to σ'. Observe that this location error is the length of the optimal solution to an orienteering problem rooted at σ'. Correspondingly, we want the reward associated with σ' to capture the total reward of all the true requests matched to σ', and define its reward error accordingly. Finally, the time window error is defined as before, as a maximum over all pairs σ and σ' that are matched to each other. Our algorithm for the setting of Theorem 3 constructs a matching between I' and I in an online fashion. For this one to many setting, we solve instances of the orienteering problem rooted at each predicted request we visit. The performance of the algorithm accordingly worsens by a small constant factor and we achieve a competitive ratio of $O(\Lambda_M \rho_M^2 \log\min(D, L_{\max}))$ as before. Due to space limitations, the details of the proof are omitted from this version. See Section 8 of the full version [13] for further details.

  ▶ **Theorem 13.** *Given an instance (G, I, I', M) of the TW-TSP with predictions where M is a many-to-one matching with errors as defined above, and satisfying $\tau_M \le L_{\min}/2$ and $\Lambda_M \le L_{\min}/2$, there exists a polynomial-time online algorithm that takes the tuple (G, I', Λ_M) as offline input and I as online input, and constructs a walk $W \in \mathcal{W}(G)$ such that*

  $$\mathbb{E}[\mathrm{Rew}(W, I)] \ge \frac{1}{O(\Lambda_M \cdot \rho_M^2 \cdot \log\min(D, L_{\max}))} \cdot \mathrm{OPT}(G, I).$$

4 Relating the Optima

In this section we provide the proofs of Lemmas 4 and 5 that relate the optima over the true and the predicted request sequences, using service times as a mechanism to capture the prediction errors. We begin by proving that a service time of S can hurt the optimal by at most a factor of $2S-1$.

▶ **Lemma 4.** *For any instance (G, I) of the TW-TSP with service times, and any integer $S \leq L_{\min}$, we have*

$$\mathrm{OPT}(G, I, S) \geq \frac{1}{2S-1} \cdot \mathrm{OPT}(G, I, 1).$$

Proof. Let $W \in \mathcal{W}(G)$ be the walk that achieves the optimum $\mathrm{OPT}(G, I, 1)$, and let the requests in I that are covered by W be denoted as $\sigma_i = (v_i, r_i, d_i, \pi_i)$ and ordered in the sequence in which they are covered by W. The lemma follows directly from the simple observation that if we don't service the $(S-1)$-requests *prior* and *after* some request σ_i, then we can save enough time to service σ_i for S time-steps within its time window.

Formally, if $t_i \in [r_i, d_i - 1]$ is the step at which W begins servicing request σ_i, then by skipping the idle times on the $(S-1)$-previous and next requests we can remain idle on v_i from step $t_i - (S-1)$ until step $t_i + S$ (since W already remained idle on v_i for 1 step) while still being able to keep up with walk W. Since $S \leq L_{min}$, it is easy to verify that at least S of these time-steps are going to fall in the time-window $[r_i, d_i]$.

We now partition the requests $\sigma_i = (v_i, r_i, d_i, \pi_i)$ into $2S-1$ sub-sequences, each of which starts at some request $i \in [S]$, and covers the requests $\sigma_i, \sigma_{i+(2S-1)}, \sigma_{i+2(2S-1)}$, and so forth. Each such sequence can be covered with a walk, with idle times built in as above, so as to be feasible for the instance (G, I, S). Clearly, one of these walks obtains a reward of at least $\mathrm{OPT}(G, I, 1)/(2S-1)$, completing the proof. ◀

In the appendix of the full version, we show that the above lemma obtains a tight gap between the optima at different service times.

▶ **Lemma 14.** *For any pair of integers (L, S) such that $L \geq 2S-2 \geq 1$, there exists a rooted instance (G, I) of the TW-TSP with service costs such that $L_{min} = L$ and*

$$\mathrm{OPT}(G, I, S) = \frac{1}{2S-1} \cdot \mathrm{OPT}(G, I, 1).$$

Next, we provide the proof of Lemma 5 that relates the optima between the predicted and true request sequences, by appropriately addressing all three possible types of prediction errors.

▶ **Lemma 5.** *Let (G, I, I', M) be an instance of the TW-TSP with predictions, where Λ_M, ρ_M, and τ_M denote the maximum location, reward, and time window errors of the instance respectively. Define $S := 4\Lambda_M + 1$ and $S' := 2\Lambda_M + 1$. Then, if $\tau_M \leq L_{\min}/2$ and $\Lambda_M \leq (L_{\min} - 1)/4$, we have*

$$\mathrm{OPT}(G, I', S') \geq \frac{1}{3\rho_M} \cdot \mathrm{OPT}(G, I, S).$$

Proof. Let W be the walk that achieves the optimum $\mathrm{OPT}(G, I, S)$, and let the requests in I covered by W be denoted as $\sigma_i = (v_i, r_i, d_i, \pi_i)$ and ordered in the sequence in which they are visited by W. Let $\sigma'_i = (v'_i, r'_i, d'_i, \pi'_i)$ denote the predicted request matched to σ_i, that is, $\sigma'_i = M(\sigma_i)$. Observe that the total reward of all requests $\{\sigma'_i\}$ corresponding to $\sigma_i \in \mathrm{Cov}(W, I, S)$ is at least $\mathrm{Rew}(W, I, S)/\rho_M$.

We will consider a walk W' in G defined as follows. The walk W' follows W, visiting the requests σ_i in sequence. As soon as W starts servicing σ_i, W' takes a detour to visit σ'_i; remains idle at σ'_i for S' time steps; returns back to σ_i; remains idle at σ_i for $S - 2\ell(v_i, v'_i) - S' \geq 0$ time steps; and then resumes the walk W. Observe that W' is identical to W outside of the detours it takes to visit the σ'_i's.

Our goal is to feasibly capture all of the reward contained in the σ'_is. The problem is that the walk W' may miss some of this reward due to the mismatch in the time windows of the true and predicted requests. To this end, we will consider two variations of the walk W'. Let $K := L_{\min}/2 \geq \tau_M$. The walk W'_1 is identical to W' except that it starts K steps after W' starts, and accordingly visits every location exactly K steps after W' visits it. The walk W'_2 is identical to W' except that it starts K steps *before* W' starts,[7] and accordingly visits every location exactly K steps before W' visits it.

Now consider some σ'_i corresponding to a request σ_i covered by W in the instance (G, I, S). We claim that at least one of the walks W', W'_1, and W'_2 covers σ'_i in (G, I', S'). Let t be the time at which W' arrives at v'_i; recall that W' remains at the node until at least $t + S'$. Note that $t \geq r_i$ and $t + S' \leq d_i$ due to $\sigma_i \in \mathrm{Cov}(W, I, S)$.

First, suppose that $r'_i \leq t$ and $d'_i \geq t + S'$, then σ' is covered by W' in (G, I', S'). Next suppose that $r'_i > t$. Then, W'_1 arrives at v'_i at time $t + K \geq r_i + K \geq r_i + \tau_M \geq r'_i$. On the other hand, it remains at v'_i until time $t + K + S' < r'_i + K + S' \leq r'_i + L_{\min} \leq d'_i$. Therefore, σ'_i is covered by W'_1. Finally, suppose that $d'_i < t + S'$. Then, W'_2 arrives at v'_i at time $t - K > d'_i - S' - K \geq d'_i - L_{\min} \geq r'_i$. On the other hand, it remains at v'_i until time $t - K + S' \leq d_i - K \leq d_i - \tau_M \leq d'_i$. Therefore, σ'_i is covered by W'_2.

We get that at least one of W', W'_1, or W'_2 obtains at least a $1/3\rho_M$ fraction of $\mathrm{OPT}(G, I, S)$, where the factor of ρ_M is lost due to the mismatch in the predicted rewards. The lemma follows directly from this. ◀

5 The offline approximation

In this section, we design an $O(\log \min(D, L_{\max}))$ deterministic and polynomial-time approximation algorithm for the TW-TSP with service times, providing the proof of Lemma 6. Our proof relies on a series of reductions between different offline problems, applications of existing algorithms as well as the design of novel algorithmic components. We break up our argument into a series of lemmas. Due to space limitations, all the proofs are moved to the appendix of the full version [13].

1. First, we designing an $O(\log \min(D, L_{\max}))$ approximation algorithm for TW-TSP (without service times). Since the work of [14] already provides a $O(\log L_{max})$ approximation for the setting with integer time-windows (see Lemma 5.3 of [14]), it suffices to prove the following:

   ▶ **Lemma 15.** *Given an instance of the TW-TSP (without service times) with $L_{min} \geq 4D$, there exists a polynomial time algorithm that achieves an $O(1)$ approximation.*

   Our proof relies on the observation that when time-windows are sufficiently large compared to the diameter of the graph, the problem essentially reduces to an instance of the well-studied Orienteering problem, for which constant approximation algorithms are

[7] To be precise, this walk starts at the location where W' is at at step K.

known. We comment that similar ideas have been used in [5]. Then, it is straightforward to combine this algorithm together with the algorithm of [14] to acquire an $O(\log\min(L_{\max}, D))$ approximation of TW-TSP.

▶ **Lemma 16.** *There exists an* $O(\log\min(D, L_{\max}))$ *approximation algorithm for the TW-TSP problem.*

2. Next, we design a simple approximation-preserving reduction from TW-TSP with service times to TW-TSP (without service times). The main idea behind this reduction is to treat service times as edge lengths in an augmented graph whose diameter is roughly $D + S$. For instances with $S \leq D$, this increase becomes negligible and thus by combining our reduction with Lemma 16, we immediately get the following:

   ▶ **Lemma 17.** *Given an instance* (G, I, S) *of the TW-TSP with service times such that* $S \leq D$*, there exists a polynomial time algorithm that achieves an* $O(\log\min(D, L_{\max}))$ *approximation.*

3. Finally, we handle the case of large service times, specifically $S \geq D$. In that case, it turns out that we can reduce the instance to one over a uniform complete graph. Then, the TW-TSP-S essentially becomes equivalent to the well-studied *Job Scheduling* problem, for which constant approximations are known.

   ▶ **Lemma 18.** *Given an instance* (G, I, S) *of the TW-TSP with service costs such that* $S \leq D$*, there exists a polynomial time algorithm that achieves an* $O(1)$ *approximation.*

The proof of Lemma 6 follows immediately from Lemma 17 and Lemma 18. We comment that any improvement in the best known approximation algorithm for TW-TSP will immediately imply an improvement for all the results that this work presents. Lemma 18 essentially enables us to assume that in all instances of interest, $S \leq D$. Under this assumption, our reduction used in the proof of Lemma 17 essentially states that TW-TSP-S becomes equivalent to TW-TSP in graphs of diameter $\mathcal{O}(D)$ and maximum window size $\mathcal{O}(L_{\max})$. As an immediate corollary, given an offline TW-TSP algorithm that achieves a competitive ratio of $\alpha(D, L_{\max})$, we immediately obtain an offline $O(\alpha(D, L_{\max}))$ approximation for TW-TSP-S, that can be used in order to substitute Lemma 6 in our analysis and improve the competitive ratio of Theorem 3.

6 The online algorithm

In this section we present an online algorithm that takes as input a pre-computed walk over the predicted request sequence and solves an appropriate online matching problem in order to construct detours that capture true requests, while taking into account the possible errors in the predictions. The formal guarantee of our algorithm is given in Lemma 7, which we restate for the reader's convenience:

▶ **Lemma 7.** *Given an instance* (G, I, I', M) *of the TW-TSP with predictions satisfying* $\tau_M \leq L_{\min}/2$ *and* $\Lambda_M \leq (L_{\min} - 1)/4$*; a walk* $W' \in \mathcal{W}(G)$*; and any integer* $S' \geq 2\Lambda_M + 1$*, there exists an online algorithm (Algorithm 1) that returns a walk* $W \in \mathcal{W}(G)$ *with expected reward*

$$\mathbb{E}\left[\mathrm{Rew}(W, I, 1)\right] \geq \frac{1}{6\rho_M} \cdot \mathrm{Rew}(W', I', S').$$

We begin by establishing some notation. Let $W' \in \mathcal{W}(G)$ be any walk that services some predicted requests in I' with a service time of S'. We use $\sigma'_i = (v'_i, r'_i, d'_i, \pi'_i)$ to denote the predicted requests in I' that are covered by W', ordered in the sequence in which they are visited by W'. Likewise, we use $\sigma_i = (v_i, r_i, d_i, \pi_i) \in I$ to denote the true request matched to the prediction σ'_i, that is, $\sigma'_i = M(\sigma_i)$.

At a high level, our algorithm follows the walk W', but when it reaches a predicted request σ'_i, it considers taking a detour to service a true request that is available at that point of time. To this end, we define the set of "reachable" true requests as follows.

▶ **Definition 19.** *Given a partial walk W that is at request $\sigma'_i \in I'$ at time t, we define the set of reachable requests $R_i(W, t)$ to be the set of all $\sigma \in I$ such that:*

1. $r_\sigma \le t \le d_\sigma - \ell(v'_i, v_\sigma) - 1$, *and*
2. $2\ell(v'_i, v_\sigma) + 1 \le S'$.

Our algorithm considers all of the reachable requests that have not been covered by the walk as yet, chooses the one with the highest reward, and takes a detour to visit and cover the request, before returning to σ'_i and resuming the walk. In order to deal with time window errors, our algorithm starts the walk a little early, or on time, or a little late, as in the proof of Lemma 5. The algorithm is described below formally.

Algorithm 1 Online algorithm for TSP-TW with predictions.

Offline input: Graph G, predicted requests I', walk $W' \in \mathcal{W}(G)$, service times S'.
Online input: True requests I.
Output: Walk $W \in \mathcal{W}(G)$.

```
1: Let $K = L_{\min}/2$. Select $\epsilon$ uniformly at random from $\{-1, 0, 1\}$.
2: Define the set of covered requests $C = \emptyset$.
3: for $i \leftarrow 1$ to $|\mathrm{Cov}(W', I', S')|$ do
4:     Let $t'_i$ denote the time at which $W'$ visits $\sigma'_i$.
5:     Set $t_i \leftarrow t'_i + \epsilon K$.
6:     Visit $v'_i$ at time $t_i$.
7:     Construct the set $R_i(W, t_i)$ of requests in $I$ reachable at time $t_i$.
8:     if $R_i(W, t_i) \setminus C = \emptyset$ then
9:         Do nothing.
10:    else
11:        Let $\hat{\sigma}$ be the highest reward request in $R_i(W, t_i) \setminus C$.
12:        Visit $v_{\hat{\sigma}}$; spend one unit of idle time at $v_{\hat{\sigma}}$; return to $v'_i$.
13:        Set $C \leftarrow C \cup \{\hat{\sigma}\}$.
```

We begin our analysis by noting that the walk W constructed by the algorithm is always able to visit the vertices v'_i corresponding to requests $\sigma'_i \in \mathrm{Cov}(W', I', S')$ feasibly at the desired times t_i. This is because, by construction, the length of the detours that the walk W takes in Step 12 is always at most S' – the amount of idle time W' spends at v'_i – by virtue of the fact that $\hat{\sigma} \in R_i(W, t_i)$ and therefore, $2\ell(v'_i, v_{\hat{\sigma}}) + 1 \le S'$. Therefore, all of the requests $\hat{\sigma}$ visited in Step 12 are indeed visited by the walk W.

We now relate the total reward covered by W to the reward contained in the true requests σ_i corresponding to $\sigma'_i \in \mathrm{Cov}(W', I', S')$. To do so, we first note that with constant probability each such request is reachable by W.

▷ Claim 20. For each i, $\sigma_i \in R_i(W, t_i)$ with probability at least $1/3$.

Proof. Recall that by definition we have $\sigma'_i = M(\sigma_i)$ and so, $2\ell(v'_i, v_i) + 1 \leq 2\Lambda_M + 1 \leq S'$. So the request σ always satisfies the second requirement in the definition of the reachable set $R_i(W, t_i)$. Let us now consider the first requirement and recall that $t_i = t'_i + \epsilon K$ where $\epsilon \in \{-1, 0, 1\}$. We will now argue that $t_i \in [r_i, d_i - 1 - \ell(v'_i, v_i)]$ for at least one of the three choices of ϵ. The claim then follows from the uniformly random choice of ϵ.

1. If $t'_i \in [r_i, d_i - 1 - \ell(v'_i, v_i)]$, then the claim holds for $\epsilon = 0$ and $t_i = t'_i$.
2. Suppose that $t'_i < r_i$. Then, for $\epsilon = 1$ we have that $t_i = t'_i + K \geq r'_i + \tau_M \geq r_i$, and also $t_i = t'_i + K < r_i + K < d_i - L_{min} + L_{min}/2$ and thus $t_i = t'_i + K \leq d_i - 1 - \ell(v'_i, v_i)$ since $\ell(v'_i, v_i) \leq \Lambda_M \leq L_{min}/2$. Thus, in this case we have $t_i = t'_i + K \in [r_i, d_i - 1 - \ell(v'_i, v_i)]$ with the choice of $\epsilon = 1$.
3. Finally, suppose that $t'_i > d_i - 1 - \ell(v'_i, v_i)$. Then, for $\epsilon = -1$ we have that $t_i = t'_i - K \leq d'_i - S' - \tau_M \leq d_i - S'$ and thus $t'_i - K \leq d_i - 1 - \ell(v'_i, v_i)$ since $S' \geq 2\Lambda_M + 1 \geq \ell(v'_i, v_i) + 1$. Also, $t_i = t'_i - K > d_i - 1 - \ell(v_i, v'_i) - L_{min}/2 > r_i + L_{min}/2 - \ell(v_i, v'_i) - 1$ and thus $t'_i - K \geq r_i$, since $\ell(v_i, v'_i) \leq \Lambda_M \leq L_{min}/2$. Thus, in this case we have obtained that $t_i = t'_i - K \in [r_i, d_i - 1 - \ell(v'_i, v_i)]$ with the choice of $\epsilon = -1$. ◁

We are now ready to prove Lemma 7 via a matching-type argument. To account for the reward covered by the walk W constructed by the algorithm, we will employ a standard charging scheme. Every time the algorithm takes a detour to cover some true request $\hat{\sigma}$ from a predicted request σ'_i in Step 12, we will credit half of the earned reward $\pi_{\hat{\sigma}}$ to $\hat{\sigma}$ itself, and half of the reward to the request σ_i. Formally, let $\text{Cr}(\sigma)$ denote the total credit received by $\sigma \in I$. Then during Step 12 we will increment both $\text{Cr}(\hat{\sigma})$ and $\text{Cr}(\sigma_i)$ by $\pi_{\hat{\sigma}}/2$.

Now consider some $\sigma_i \in I$ corresponding to $\sigma'_i \in \text{Cov}(W', I', S')$. By Claim 20, this request is in $R_i(W, t_i)$ with probability at least 1/3. If at time t_i, the request has already been covered by W, then we get $\text{Cr}(\sigma_i) \geq \pi_i/2$. Otherwise, we pick a $\hat{\sigma} \in R_i(W, t_i)$ with $\pi_{\hat{\sigma}} \geq \pi_i$, and therefore, once again we get $\text{Cr}(\sigma_i) \geq \pi_i/2$.

Putting everything together, we get

$$\begin{aligned}
\text{E}[\text{Rew}(W, I, S)] = \text{E}\left[\sum_{\sigma \in I} \text{Cr}(\sigma)\right] &\geq \sum_{i:\sigma'_i \in \text{Cov}(W', I', S')} \text{E}\left[\frac{\pi_i}{2} \mathbb{1}[\sigma_i \in R_i(W, t_i)]\right] \\
&\geq \frac{1}{3} \cdot \sum_{i:\sigma'_i \in \text{Cov}(W', I', S')} \frac{\pi_i}{2} \\
&\geq \frac{1}{6} \cdot \frac{1}{\rho_M} \cdot \sum_{i:\sigma'_i \in \text{Cov}(W', I', S')} \pi'_i \\
&= \frac{1}{6\rho_M} \cdot \text{Rew}(W', I', S')
\end{aligned}$$

This completes the proof of the lemma.

7 Lower bounds

In this section, we present lower bounds that complement our results. First, we will motivate the need for predictions in Section 7.1. Then, in Section 7.2 we will show that the competitive ratio of TW-TSP with predictions must scale linearly with the error in locations. Finally, in Section 7.3 we argue the need for non-zero service times in the definition of TW-TSP with predictions.

7.1 Lower bounds for online TW-TSP without predictions

We argue that Online TW-TSP does not admit any reasonable competitive ratio in the absence of predictions. In the case of deterministic algorithms where their entire behavior is predictable, simple instances with only 2 vertices and appropriately small time-windows suffice to argue that no bounded guarantee for the approximation ratio is achievable.

▶ **Lemma 21.** *The competitive ratio of any deterministic online algorithm for Online TW-TSP on instances with $L_{\min} \leq D$ is unbounded.*

Proof. Let DET be any deterministic algorithm and let G be the line graph with just two vertices v_1, v_2 connected via an edge of length D. Since DET is deterministic, we can assume knowledge of its position at any step t as soon as we have specified all requests with release time $\leq t$. We will now construct a request sequence I that uses the information.

For the first D time-steps, we don't release any request. Then, at $t = D$, let $v_D \in \{v_1, v_2\}$ be the position that DET's walk is currently at and likewise let v'_D be the other vertex of G. We construct the first request to be $\sigma = (v'_D, D, D + L, 1)$ for any $L \leq D$. Clearly, DET cannot service this request as even for $L = D$ it arrives on v'_D at deadline and cannot service it for one step. Then, we don't release any new request for the next $2D$ steps, and at $t = 3D$ we repeat the same process, by requesting the vertex that DET doesn't currently occupy. Likewise, we repeat the same process at $t = 5D$, $t = 7D$ etc. Independently of the size of our request sequence, the total reward collected by DET is 0.

On the other hand, it is not hard to see that if our request sequence I has N requests in total, then $\text{OPT}(G, I, 1) = N$. This is due to the fact that requests are spaced $2D$-steps from each other, and thus an optimal offline algorithm that had knowledge of the entire sequence in advance would always be able to arrive at each request on time, servicing it within its respective time-window. ◀

For randomized algorithms, a slight improvement can be achieved. In particular, the randomized algorithm that picks a vertex uniformly at random and then remains idle on it for the entire sequence achieves a competitive ratio of $1/n$. It turns out that this is actually the best possible ratio that a randomized algorithm can achieve on Online TW-TSP:

▶ **Theorem 8.** *The competitive ratio of any randomized online algorithm for Online TW-TSP on instances with $L_{\min} \leq D$ is at most $1/n$.*

Proof. Let $G(V, E, \ell)$ be the uniform complete graph of $n = |V|$ vertices, where all edges have a length of D. Fix any integer N and let vertices $v_1, v_2, \ldots, v_N \in V$ be drawn independently and uniformly at random. Next, we fix any window length $L \leq D$ and consider the (random) request sequence on these vertices $I = \{\sigma_i\}_{i=1}^N$ for $\sigma_i = (v_i, (2i-1)D, (2i-1)D + L, 1)$. From Yao's mininmax principle, a lower bound on the (expected) competitive ratio of deterministic algorithms on this randomized instance will imply the same lower bound for randomized algorithms.

Since the time-windows are spaced $2D$-away from each other, it is not hard to see that for any realization of I, $\text{OPT}(G, I, 1) = N$. On the other hand, since $D \geq L$, we get that the only way to service a request is to be on its corresponding vertex on release time. Since the vertices are random, for any deterministic algorithm this happens with probability precisely $1/n$, and thus the expected reward of any deterministic algorithm on this instance is N/n, proving the claim. ◀

7.2 Tight dependence on location error

In this section we show that a linear dependency on the location error is unavoidable for any randomized online algorithm for the TW-TSP with predictions, even assuming exponential computational power. In other words, we formally prove Theorem 10, which we re-state for the reader's convenience:

▶ **Theorem 10.** *For any $S > 0$, there exists an instance (G, I, I', M) of the TW-TSP with predictions satisfying $\tau_M = 0$, $\rho_M = 1$, and $\Lambda_M = S$ such that the competitive ratio of any randomized online algorithm taking the tuple (G, I', Λ_M) as offline input and I as online input asymptotically approaches $1/(S + 1)$.*

Proof. Fix any $S > 0$ and let K, C and N be integer parameters that will be specified later. We construct a graph $G = \cup_{i=0}^{N-1} G_i$ that consists of N copies $G_0, \ldots, G_{N-1}$ of the complete graph on C vertices with all edge lengths equal to S, arranged in a way so that each vertex in G_i connects to each vertex in G_{i+1} with an edge of length KS. A pictorial example for small values of C and N is shown in Figure 1.

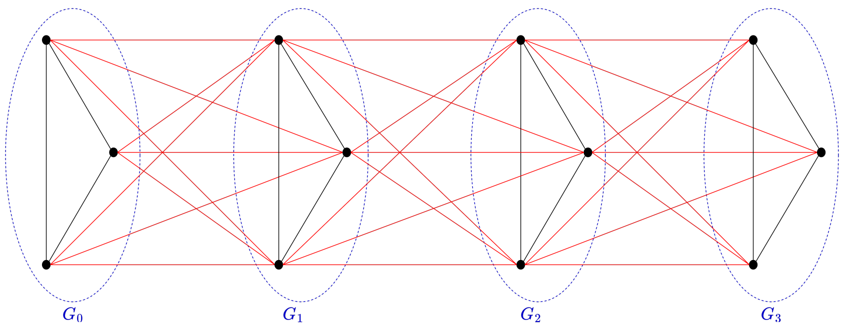

Figure 1 An example of G for $C = 3$ and $N = 4$. Black edges have a length of S and red edges have a length of KS.

Next, we select independently and uniformly at random one vertex v_i from each sub-graph G_i and construct the (randomized) request sequence $I = \{\sigma_i\}_{i=0}^{N-1}$ where we denote $\sigma_i = (v_i, r_i, r_i + KS, 1)$ for $r_i = i \cdot (KS + 1)$. As for the predictions, we simply construct a second instance I' in the same manner and offer it as an offline prediction for I. Observe that for all possible constructions of I and I' it holds that there exists a matching M between them with $\tau_M = 0$, $\rho_M = 1$ and $\Lambda_M = S$, namely the matching that pairs together vertices from the same sub-graphs. This prediction provides no information to the algorithm other than the fact that the (true) instance I was constructed via the above randomized approach. Also, notice that the location error Λ_M can be arbitrarily small compared to the window length $L = KS$ by choosing appropriately large K.

It is easy to see that for all possible realizations of I it holds that $\text{OPT}(G, I, 1) = N$. Indeed, consider the walk that starts from the (random) vertex v_0, remains idle for 1 step and then visits vertex v_1, remains idle for one step, visits v_2, etc. Such a walk would visit each vertex v_i at step $t = i \cdot KS + i = r_i$ and thus would service all the requests in I, achieving a total reward of N. Next, we will show that the expected reward of any deterministic algorithm on the random sequence I approaches N/S. Using Yao's minimax principle, this will immediately translate to a lower bound that approaches $1/S$ for randomized algorithms, completing the proof of the theorem.

Fix any deterministic algorithm for TW-TSP with predictions on instance (G, I, I', M). Note that since the predictions I' supply zero information, it suffices to analyze the algorithm as a deterministic online algorithm for Online TW-TSP on instance I. The key observation is that due to the fact that both the time windows of the requests and the edges that connect different sub-graphs have a length of KS, it is impossible for the algorithm to know on what vertex v_i of subgraph G_i the request is going to arrive before visiting some possibly different vertex of the same subgraph. In particular, if the algorithm is in subgraph G_j for $j < i$ at time r_i, then it cannot reach vertex v_i before the time window of request i ends. Thus, in order for any deterministic algorithm to serve the request on sub-graph G_i, it first has to visit some vertex v_i' of G_i. If it so happens that $v_i' = v_i$ then it can immediately service the request, otherwise it has to travel a distance of S in order to reach v_i.

We partition the set of requests into two sets N_+ and N_- based on whether the deterministic algorithm happens to arrive on the correct vertex of the sub-graph or not. All the requests in N_+ can be serviced without any extra delay, exactly as done by the optimal walk. On the other hand, servicing a request in N_- requires the algorithm to spend an extra time of S in order to transition to the right vertex. Since the time-windows have a length of KS, this can be done at most K times before the algorithm runs out of slack to spare. When this happens, the algorithm would have to skip the next S requests in order to recover enough slack to fix its next mistake.

Formally, consider the last request in N_- that the algorithm feasibly serves, call it l. Let A be the number of requests in N_- the algorithm serves through extra delay prior to l, and let B be the number of requests the algorithm skips in N_+ or N_- prior to l. Then in order for the algorithm to have reached l before its time window ends, it must be the case that the total extra delay incurred by the algorithm, namely $AS - B$, is no more than KS. Rearranging we get:

$$A(S+1) - (A+B) \leq KS, \qquad \text{or,} \qquad A \leq \frac{A+B}{S+1} + \frac{KS}{S+1} < \frac{N}{S+1} + K$$

Thus, we get that the total expected reward gathered by the algorithm is at most

$$\mathbb{E}[|N_+|] + A + 1 \leq \mathbb{E}[|N_+|] + \frac{N}{S+1} + K + 1$$

Finally, using the fact that $\mathbb{E}[|N_+|] = N/C$, we get that the competitive ratio of any deterministic algorithm on the random instance I is at most

$$\frac{1}{C} + \frac{K+1}{N} + \frac{1}{S+1}$$

Choosing $N >> K$ and C sufficiently large, we can make this competitive ratio asymptotically approach $1/(S+1)$ as desired. ◀

7.3 Comparing the optimal with and without service times

As we saw in Lemma 4, the gap between $\mathrm{OPT}(G, I, 1)$ and $\mathrm{OPT}(G, I, S)$ depends linearly on the service time S. In this section, we study the gap between $\mathrm{OPT}(G, I, 1)$ and $\mathrm{OPT}(G, I, 0)$, showing that there exists a much sharper separation between them.

▶ **Theorem 22.** *For any integers L and D, $L \leq D$, there exists an offline instance (G, I) of the TW-TSP where D is the diameter of the network and every request has a time window length equal to L, such that* $\mathrm{OPT}(G, I, 1) \leq \frac{L}{D+1} \cdot \mathrm{OPT}(G, I, 0)$.

Proof. Consider the line graph G with vertices v_0 through v_D connected sequentially via edges of length 1. Clearly, the diameter of G is D. Next, consider the sequence of $(D+1)$-requests $I = \{\sigma_i\}_{i=0}^{D}$ where $\sigma_i = (v_i, i, L+i, 1)$. Observe that $\mathrm{OPT}(G, I, 0) = D + 1$ as simply following the walk from v_0 to v_D will cover all the requests if the service costs are 0.

On the other hand, it is not very hard to see that $\mathrm{OPT}(G, I, 1) = L$. First, observe that without loss we can assume that the optimal walk starts on v_0. If not, let σ_i be the *first* request served by the optimal. Since σ_i cannot be served prior to step $r_i = i$, we could instead start at v_0 and take i steps in order to reach v_i and then follow the original walk, achieving precisely the same reward.

Our argument is completed by the simple observation that any walk that starts from v_0 and has served x requests with service cost 1 *cannot* visit vertex v_j prior to step $j + x$. Thus, as soon as x becomes L all the future time-windows will be missed. ◀

Theorem 22 states that $\mathrm{OPT}(G, I, 0)$ is a much stronger benchmark than $\mathrm{OPT}(G, I, 1)$. However, it doesn't exclude the possibility of designing an algorithm that is competitive against this stronger benchmark $\mathrm{OPT}(G, I, 0)$. We will next show that no randomized online algorithm with predictions can obtain a better than linear competitive ratio against this benchmark. Intuitively, requiring the algorithm to spend non-zero time at each request also allows the algorithm to recover from possible mistakes due to the prediction errors by skipping requests that the optimum services with a delay of 1. A similar approach is not possible in the 0 service time setting.

▶ **Theorem 9.** *For any $S > 0$, there exists an instance (G, I, I', M) of the TW-TSP with predictions and service times 0, satisfying $\tau_M = 0$, $\rho_M = 1$, and $\Lambda_M = S$, such that any randomized online algorithm taking the tuple (G, I', Λ_M) as offline input and I as online input achieves a reward no larger than $O(1/n) \cdot \mathrm{OPT}(G, I, 0)$. Here n is the number of vertices in G.*

Proof. We construct the same graph $G = \cup_{i=0}^{N-1} G_i$ as in Theorem 10, that consists of N copies of the complete graph with edge lengths S, connected sequentially with edges of length KS (see Figure 1). As for the request sequence, we once again select independently and uniformly at random one vertex v_i from each sub-graph G_i and construct the (randomized) request sequence $I = \{\sigma_i\}_{i=0}^{N-1}$ where $\sigma_i = (v_i, iKS, (i+1)KS - 1, 1)$; notice that release times are slightly different from Theorem 10 to account for the absence of service costs.

For the predictions, we simply construct a second instance I' in the same manner and offer it as an offline prediction for I. By matching the requests on the same sub-graph together, we get $\tau_M = 0$, $\rho_M = 1$ and $\Lambda_M = S$. As it clearly holds that $\mathrm{OPT}(G, I, 0) = N$ for any realization of I, to prove our theorem it suffices to argue that any deterministic algorithm for TW-TSP with predictions on instance (G, I, I', M) gets an expected reward of $\mathcal{O}(N/n)$ and apply *Yao's minimax principle.* Furthermore, since the prediction I' does not provide any information on I, it suffices to bound the reward of deterministic algorithms for Online TW-TSP on (online) instance I.

Fix any deterministic algorithm for Online TW-TSP on instance I. Observe that since edges between different sub-graphs have a length of KS and time-windows have a length of $KS - 1$, it is impossible for the algorithm to service some request σ_i unless at $t = r_i$ it is already in some vertex of sub-graph G_i. For the same reason, it is not possible to service any request σ_j after servicing some other request σ_i with $i > j$. Finally, since all requests can be reached by their release time if the algorithm starts from a vertex in G_0, we can assume without loss that this is indeed the case.

We partition our set of N requests into two sets N_+ and N_- based on whether the deterministic algorithm *happens* to arrive on the correct vertex of the sub-graph before the request's release time or not. From the random construction of our instance, we have that $\mathbb{E}[|N_+|] \leq N/C$. For requests in N_-, if the algorithm wishes to service them it has to take a detour of length S in order to reach the correct vertex. Our proof relies on the fact that after servicing K requests in N_-, the algorithm can no longer service any other request. To see this, let v_j be the K-th request in N_- that was serviced by the deterministic algorithm. As we can already established, any request v_i with $i < j$ can no longer be serviced. On the other hand, since without loss the algorithm starts at a vertex of G_0, just reaching a vertex in G_i while taking K detours of length S requires at least $iKS + K > d_i$ time-steps. Putting everything together, we get that the expected reward of the algorithm is at most $N/C + K = \mathcal{O}(N/n)$ by setting $K = \mathcal{O}(1)$ and $N = 2K$, since $n = NC$. ◀

References

1 Antonios Antoniadis, Themis Gouleakis, Pieter Kleer, and Pavel Kolev. Secretary and online matching problems with machine learned advice. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

2 Esther M. Arkin, Joseph B. M. Mitchell, and Giri Narasimhan. Resource-constrained geometric network optimization. In *Symposium on Computational Geometry (SoCG)*, 1998.

3 Yossi Azar, Stefano Leonardi, and Noam Touitou. Flow time scheduling with uncertain processing time. In *Symposium on the Theory of Computing (STOC)*, 2021.

4 Yossi Azar, Debmalya Panigrahi, and Noam Touitou. Online graph algorithms with predictions. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2021.

5 Yossi Azar and Adi Vardi. TSP with time windows and service time. *CoRR*, abs/1501.06158, 2015. `arXiv:1501.06158`.

6 Étienne Bamas, Andreas Maggiori, and Ola Svensson. The primal-dual method for learning augmented algorithms. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

7 Nikhil Bansal, Avrim Blum, Shuchi Chawla, and Adam Meyerson. Approximation algorithms for deadline-TSP and vehicle routing with time-windows. In *Symposium on the Theory of Computing (STOC)*, 2004.

8 Reuven Bar-Yehuda, Guy Even, and Shimon Shahar. On approximating a geometric prize-collecting traveling salesman problem with time windows. *J. Algorithms*, 55:76–92, 2005.

9 Magnus Berg, Joan Boyar, Lene M. Favrholdt, and Kim S. Larsen. Online minimum spanning trees with weight predictions. In *Workshop on Algorithms and Data Structures*, 2023.

10 Giulia Bernardini, Alexander Lindermayr, Alberto Marchetti-Spaccamela, Nicole Megow, Leen Stougie, and Michelle Sweering. A universal error measure for input predictions applied to online graph problems. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

11 Avrim Blum, Shuchi Chawla, David Karger, Terran Lane, and Maria Minkoff. Approximation algorithms for orienteering and discounted-reward TSP. In *Foundations of Computer Science (FOCS)*, 2003.

12 Niv Buchbinder, Yaron Fairstein, Konstantina Mellou, Ishai Menache, and Joseph (Seffi) Naor. Online virtual machine allocation with lifetime and load predictions. In *International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2021.

13 Shuchi Chawla and Dimitris Christou. Online TSP with predictions. *CoRR*, abs/2304.01958, 2024. `arXiv:2304.01958`.

14 Chandra Chekuri, Nitish Korula, and Martin Pál. Improved algorithms for orienteering and related problems. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2008.

15 Chandra Chekuri and Amit Kumar. Maximum coverage problem with group budget constraints and applications. In *International Workshop on Approximation, Randomization, and Combinatorial Optimization (APPROX)*, 2004.

16 Ke Chen and Sariel Har-Peled. The orienteering problem in the plane revisited. In *Symposium on Computational Geometry (SoCG)*, 2006.

17 Michael Dinitz, Sungjin Im, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Faster matchings via learned duals. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

18 Paul Dütting, Silvio Lattanzi, Renato Paes Leme, and Sergei Vassilvitskii. Secretaries with advice. In *ACM Conference on Economics and Computation (EC)*, 2021. `doi:10.1145/3465456.3467623`.

19 Jon C. Ergun, Zhili Feng, Sandeep Silwal, David Woodruff, and Samson Zhou. Learning-augmented k-means clustering. In *International Conference on Learning Representations (ICLR)*, 2022.

20 Thomas Wilhelm Erlebach, Murilo Santos de Lima, Nicole Megow, and Jens Schloter. Learning-augmented query policies for minimum spanning tree with uncertainty. In *Embedded Systems and Applications (ESA)*, 2022.

21 Jie Gao, Su Jia, Joseph S. B. Mitchell, and Lu Zhao. Approximation algorithms for time-window TSP and prize collecting TSP problems. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2016.

22 Themis Gouleakis, Konstantinos Lakis, and Golnoosh Shahkarami. Learning-augmented algorithms for online TSP on the line. *CoRR*, abs/2206.00655, 2022. `arXiv:2206.00655`.

23 Hsiao-Yu Hu, Hao-Ting Wei, Meng-Hsi Li, Kai-Min Chung, and Chung-Shou Liao. Online TSP with predictions. *CoRR*, abs/2206.15364, 2022. `arXiv:2206.15364`.

24 Sungjin Im, Ravi Kumar, Mahshid Montazer Qaem, and Manish Purohit. Non-clairvoyant scheduling with predictions. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2021.

25 Zhihao Jiang, Debmalya Panigrahi, and Kevin Sun. Online algorithms for weighted paging with predictions. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, 2020.

26 Yoshiyuki Karuno and Hiroshi Nagamochi. 2-approximation algorithms for the multi-vehicle scheduling problem on a path with release and handling times. *Discret. Appl. Math.*, 129:433–447, 2003.

27 Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2020.

28 Thomas Lavastida, Benjamin Moseley, R. Ravi, and Chenyang Xu. Learnable and instance-robust predictions for online matching, flows and load balancing. In *European Symposium on Algorithms (ESA)*, 2021.

29 Alexander Lindermayr and Nicole Megow. Algorithms with predictions. URL: `https://algorithms-with-predictions.github.io/`.

30 Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. *J. ACM*, 68(4):24:1–24:25, 2021.

31 Dhruv Rohatgi. Near-optimal bounds for online caching with machine learned advice. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2020.

32 John N. Tsitsiklis. Special cases of traveling salesman and repairman problems with time windows. *Networks*, 22(3):263–282, 1992.

33 Chenyang Xu and Benjamin Moseley. Learning-augmented algorithms for online steiner tree. *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 36:8744–8752, 2022.