A Universal Error Measure for Input Predictions Applied to Online Graph Problems

Giulia Bernardini [*]   Alexander Lindermayr [†]   Alberto Marchetti-Spaccamela [‡]

Nicole Megow [†]   Leen Stougie [§]   Michelle Sweering [¶]

Abstract

We introduce a novel measure for quantifying the error in input predictions. The error is based on a minimum-cost hyperedge cover in a suitably defined hypergraph and provides a general template which we apply to online graph problems. The measure captures errors due to absent predicted requests as well as unpredicted actual requests; hence, predicted and actual inputs can be of arbitrary size. We achieve refined performance guarantees for previously studied network design problems in the online-list model, such as Steiner tree and facility location. Further, we initiate the study of learning-augmented algorithms for online routing problems, such as the online traveling salesperson problem and the online dial-a-ride problem, where (transportation) requests arrive over time (online-time model). We provide a general algorithmic framework and we give error-dependent performance bounds that improve upon known worst-case barriers, when given accurate predictions, at the cost of slightly increased worst-case bounds when given predictions of arbitrary quality.

[*]University of Trieste, Italy, and CWI, Amsterdam, The Netherlands. *giulia.bernardini@units.it*. Partially supported by the Netherlands Organisation for Scientific Research (NWO) through project OCENW.GROOT.2019.015 "Optimization for and with Machine Learning (OPTIMAL)".

[†]Faculty of Mathematics and Computer Science, University of Bremen, Germany. *{linderal,nicole.megow}@uni-bremen.de*. Partially supported by the German Science Foundation (DFG) under contract 146371743 – TRR 89 Invasive Computing.

[‡]La Sapienza University of Rome, Italy, and INRIA-Erable, France. *alberto@diag.uniroma1.it*.

[§]CWI and Vrije Universiteit, Amsterdam, The Netherlands, and INRIA-Erable, France. *Leen.Stougie@cwi.nl*. Supported by the Netherlands Organisation for Scientific Research (NWO) through Gravitation-grant NETWORKS-024.002.003 and through project OCENW.GROOT.2019.015 "Optimization for and with Machine Learning (OPTIMAL)".

[¶]CWI, Amsterdam, The Netherlands. *Michelle.Sweering@cwi.nl*. Supported by the Netherlands Organisation for Scientific Research (NWO) through Gravitation-grant NETWORKS-024.002.003.

1 Introduction

We develop a novel measure for quantifying the error in input predictions and apply it to derive error-dependent performance guarantees of algorithms for online metric graph problems. Online graph problems are among the most fundamental online optimization problems, where an initially unknown input is revealed incrementally. The two main paradigms for the incremental information release are the *online-time* model and the *online-list* model. In the online-time model, initially unknown requests are revealed over time and can be served any time, whereas in the online-list model, requests are revealed one-by-one and must be served immediately before the next request appears. In this work, we address specifically the following online routing and network design problems.

Online-time routing problems. In the classical *Online Traveling Salesperson Problem* (OLTSP) and *Online Dial-a-Ride Problem* (OLDARP), a server can move at unit speed in a given metric space. Transportation requests appear online over time, each defining a start and end point in the metric space (in the TSP both points are equal). The task is to determine a tour to serve all requests (in any order) by moving to the corresponding start and end points. The objective is to minimize the makespan, i.e., the time point when all requests have been served and the server is back in the origin. These problems are well-studied [7, 8, 16, 17, 24], as well as other related variants [32, 33, 39].

Online-list network design problems. In the *Online Steiner Tree Problem*, requests are terminal nodes that are revealed one-by-one in a given metric space (typically represented as a complete edge-weighted graph) and must be connected to a fixed root by selecting edges via other (Steiner) nodes. In the closely related *Online Steiner Forest Problem*, a request is composed of two nodes which have to be connected by the selected set of edges. In both problems, the objective is to minimize the total cost of selected edges. In the more general *Online Facility Location Problem*, a facility can be opened at every vertex at a certain one-time cost at any time, and arriving client vertices are connected upon arrival to the closest open facility at the cost of the shortest path to it. The goal is to minimize the opening and connection cost. These problems are very well-studied [2–4, 9, 13, 15, 19, 21, 23, 26, 27, 31, 42, 49, 50].

The performance of online algorithms is typically assessed by worst-case analysis. An algorithm is called ρ-*competitive* if it computes, for any input instance, a solution with objective value within a multiplicative factor ρ of the optimal value that can be computed when knowing the full instance upfront. The *competitive ratio* of an algorithm is the smallest factor ρ for which it is ρ-competitive. For the above problems (nearly) tight bounds on the competitive ratio are known. For OLTSP and OLDARP, there have been shown best possible 2-competitive algorithms [7, 8]. For the online network design problems, the existence of $O(1)$-competitive algorithms has been ruled out [26, 31, 42] and algorithms with (tight) (poly-)logarithmic upper bounds have been shown [15, 26, 31, 42, 49].

The assumption in online optimization of not having any prior knowledge about future requests seems overly pessimistic. In particular, given the success of machine-learning methods and data-driven applications, one may expect to have access to predictions about future requests. However, simply trusting such predictions might lead to very poor solutions, as these predictions come with no quality guarantee. The recent vibrant line of research initiated in [40, 41] aims at incorporating such error-prone predictions into online algorithms, to go beyond worst-case barriers. The goal are *learning-augmented algorithms* with a performance that is close to that of an optimal offline algorithm when given accurate predictions (called *consistency*) and, at the same time, never being (much) worse than that of a best known algorithm without access to predictions (called *robustness*). Further, the performance of an algorithm shall degrade in a controlled way with increasing prediction error.

In this paper, we consider an *input predictions* model, i.e., there is given a set $\hat{R}$ of predictions for the actual online input R of a problem. We do not make any assumption on the quality of the prediction or on

2

its size. In particular, $\hat{R}$ might be substantially larger or smaller than R.

Defining an appropriate error measure is a crucial task in this line of research. There is no common agreement (yet) in the literature on what constitutes a good error measure. The philosophy behind our error measure is the following. Any learning-augmented algorithm needs to trust the predictions to some extent, as otherwise no improvement upon an online algorithm is possible. The error should then be able to sensitively bound how much any (reasonable) algorithm pays for erroneous predictions. Roughly, our error measure achieves this by approximating the extra cost that an algorithm trusting the predictions has when it serves the true instance; very informally, this is $\mathrm{OPT}_{\mathrm{trust}\hat{R}}(R) - \mathrm{OPT}(R)$. Several natural measures (for graph problems) have been proposed, such as the number of erroneous predictions [51], the ℓ_1-norm (e.g., distances between predicted and real points), or more involved perfect matching-based errors [12]. Although we cannot expect that a single error measure is appropriate for all problems, we propose a universal template based on the cost of a hyperedge cover in a bipartite hypergraph that is constructed in a problem-specific way.

1.1 Our contributions

Cover error for input predictions. Here we sketch the main idea of our error measure, which will be made precise in Section 2. We separately cover the errors incurred by *unexpected actual requests*, $R \setminus \hat{R}$, and *absent predicted requests*, $\hat{R} \setminus R$, as these pose a potential threat to an algorithm which trusts $\hat{R}$. For each of the two error types we consider a suitable weighted bipartite hypergraph with erroneous requests on the left side and define an error measure combining the costs of minimum hyperedge covers of the left side of each hypergraph. Let us concentrate on errors due to unexpected actual requests, being the nodes on the left side, with the predicted requests as the nodes on the right side. Each hyperedge links a single node on the right side with a subset of the nodes on the left side which it *covers*. Its contribution to the overall error, i.e., its cost in the hyperedge cover problem to cover all left side nodes, is related to the optimal cost for the subinstance induced by its nodes. E.g. in OLTSP this cost is the value of an optimal tour for some unexpected requests (left) when starting from some predicted request (right), which can be seen as a minimum detour that needs to be made from the predicted request to serve the unexpected requests. Bounding the number of left side requests in the hyperedges by k yields a hierarchical family $\{\Lambda_k\}_{k=1}^{\infty}$ of errors, with higher values of k giving errors that reflect more precisely the cost due to trusting wrong predictions.

The cover error fulfills several useful properties. First, it provides a framework that may apply to various problems by assigning appropriate costs to the hyperedges. E.g. for the online-time model it allows to integrate in a very precise way actual and predicted release dates, as we demonstrate in Section 3. This is a feature which previous metric graph errors seem to miss [12, 51], because they rely on counting incorrect predictions or disallow asymmetric cost functions. Our error also naturally supports different sizes of R and $\hat{R}$, reflecting the input sequence length being unknown in almost all online optimization problems in the literature. Although previously studied error measures [12, 51] do support this in theory, we will show in Section 2 that they fail to detect good predictions in certain scenarios, which results in imprecise weak performance bounds. In contrast, the cover error guarantees an almost optimal performance of the same algorithms in these cases. We therefore hope that the cover error will be useful for better analyzing existing learning-augmented algorithms and other problems in the future.

Algorithms with error-sensitive performance bounds. Our algorithmic results are twofold: we provide the first learning-augmented algorithms for online-time routing problems, and we give new error-dependencies for existing algorithms for online-list network design problems. The unifying element is

3

that we achieve these by problem-dependent implementations of our new cover error. We first introduce a general framework for OLTSP and OLDARP, in which we delay the moment in which we start following the optimal predicted tour by a multiplicative trust factor $\alpha \in (0, 1)$. For robustness, before starting to follow the predictions any ρ-competitive online algorithm is executed in a black-box fashion. We prove an error-dependency w.r.t. the first cover error in the hierarchy Λ_1 (as properly defined in Section 2). We denote by C^* the cost of an optimal tour on the actual requests R.

Theorem 1. OLTSP *and* OLDARP *admit learning-augmented algorithms that use a ρ-competitive algorithm as a subroutine and achieve a competitive ratio that is, for any $\alpha > 0$, bounded by*

$$\min\left\{(1 + \alpha)\left(1 + \frac{3 \cdot \Lambda_1}{C^*}\right), 1 + \rho + \frac{\rho}{\alpha}\right\}.$$

Hence, sufficiently good predictions help to beat the classic lower bound of 2 [8] for OLTSP.

When using the algorithm of [7] as a subroutine, we can further refine our algorithm by carefully aligning the used waiting strategies and prove an improved robustness guarantee.

Theorem 2. OLTSP *and* OLDARP *admit a learning-augmented algorithm that uses the 2-competitive algorithm of [7] as a subroutine and achieves a competitive ratio that is, for any $\alpha > 0$, bounded by*

$$\min\left\{(1 + \alpha)\left(1 + \frac{3 \cdot \Lambda_1}{C^*}\right), 2 + \frac{2}{\alpha}\right\}.$$

In general, online algorithms for OLTSP and OLDARP aim for tackling the uncertainty of the input rather than efficient running times, which is also the case for the above discussed results. Yet, we show in Section 3.2 that we can trade efficiency with slightly larger constant factors in the guarantees.

Further, we remark that simpler and tightened results are possible for restricted metric spaces: in Section 3.2 we provide improved bounds for OLTSP on the positive half of the real line. Here, a minimalistic prediction, a single value predicting the optimal makespan C^*, suffices to obtain an almost tight consistency-robustness tradeoff.

We complement these theoretical bounds with empirical results (see Section 5) on simulated real-world taxi instances in the city road network of Manhattan, which indicate the superior performance of our new algorithms compared to classic methods in both general and relevant restricted scenarios.

Further, we consider online-list graph problems and analyze the algorithmic framework provided by Azar, Panigrahi and Touitou [12] w.r.t. our new cover error. For each problem, we specify hyperedge cost functions which follow the same paradigm and prove new error-dependent bounds.

Theorem 3. *The algorithms in [12] for the online Steiner tree or online (capacitated) facility location problem incur, for any parameter $k \geq 1$, a cost of at most $O(1) \cdot \text{OPT} + O(\log k) \cdot \Lambda_k$.*

Theorem 4. *The algorithm in [12] for the online Steiner forest problem incurs, for any parameter $k \geq 1$, a cost of at most $O(1) \cdot \text{OPT} + O(k) \cdot \Lambda_k$.*

These bounds hold *simultaneously* for any k. The algorithm is still robust due to the robustness bound of $O(\log|R|)$ provided by Azar et al. in [12]. On the technical side, we can exploit a technical lemma by [12] that allows us to split the analysis in two parts. The actual proofs for bounding the algorithm's cost by the cost of an optimal solution and the cover error are completely different.

For certain input scenarios we substantially strengthen the bounds on the competitive ratio provided by Azar et al. [12]. Indeed, their bound never improves over $\Omega(\log(\max\{|R|, |\hat{R}|\} - \min\{|R|, |\hat{R}|\}))$, which

4

is not better than the best possible competitive ratio $O(\log|R|)$ for classic online algorithms, if $|\hat{R}|$ and $|R|$ differ significantly. We show that there are input scenarios for which their error measure overestimates the actual error substantially and, thus, gives poor performance bounds while the algorithm performs actually well; more precisely, we prove a constant competitive ratio for the algorithm in [12] w.r.t. our error measure whereas the bound w.r.t. the previous measure is $O(\log|R|)$.

1.2 Further related work

While untrusted predictions have been successfully integrated into online models for many different problems, none of the previous approaches and models seem to capture the complexity of combined routing and scheduling decisions. Related research includes work on scheduling [10, 11, 14, 30, 35, 37, 43, 45, 46], routing in metric spaces such as the k-server problem and more generally metrical task systems [5, 38], graph exploration [22] and online network design [1, 12, 51].

Further, there is hardly any work on integrating untrusted predictions into online problems in the online-time model. The only exception seems to be the work by Antoniadis et al. [6] on online speed scaling with a prediction model that includes release dates and deadlines. The nature of the speed-scaling problem, however, is very different from the routing problems we consider. Other works on non-clairvoyant scheduling with jobs arriving over time (minimizing flow time [10, 11], total completion time [37] and energy [14]) assume predictions on the job sizes or priorities; release dates are known in advance in [14], while [10, 11, 37] consider purely online problems w.r.t. job arrivals.

Very recently and independently of our work, two papers [28, 29] were announced that also study OLTSP in the learning-augmented setting. Hu et al. [29] consider OLTSP with different prediction models in general metric spaces. For arbitrary input predictions, their result has no error-dependency and a weaker consistency-robustness tradeoff compared to Theorem 2. Gouleakis et al. [28] exclusively study OLTSP on the real line. Assuming that the correct number of requests is known in advance, they study the power of predictions on the locations; their results are incomparable to ours.

1.3 Organization

We first introduce the cover error in Section 2, and then give our results for online-time routing problems and online-list network design problems in Section 3 resp. Section 4. Finally, we present empirical experiments in Section 5.

2 The cover error

Given an input prediction $\hat{R}$ and the actual input R, we design an error measure that *covers* every erroneously predicted item, i.e., all unexpected requests $R \setminus \hat{R}$ and all absent predicted requests $\hat{R} \setminus R$. As a concrete example think of OLTSP where a learning-augmented algorithm trusts (at least to a certain degree) the predicted requests in $\hat{R}$, and thus follows an optimal tour on $\hat{R}$. After serving a predicted request $(\hat{x}, \hat{r})$, it may serve some unexpected actual requests $R' \subseteq R \setminus \hat{R}$ that have already been released and are *relatively close* to $(\hat{x}, \hat{r})$ (both time- and location-wise). In our terminology, think of $(\hat{x}, \hat{r})$ covering R', and observe that the cost for this cover shall naturally be equal to the optimal cost for serving R' when starting in $\hat{x}$ at time $\hat{r}$. Conversely, the predicted requests that have not shown up and are nevertheless visited, $\hat{R} \setminus R$, should be covered by actual requests, to make up for the extra cost incurred by these superfluous visits.

We can arguably expect that any well-performing algorithm should be as least as good as serving all unexpected requests $R \setminus \hat{R}$ and all absent predicted requests $\hat{R} \setminus R$ in such partitions which can be covered

5

(a) Instance in metric space **(b)** Min-cost k-hyperedge cover **(c)** $\Gamma_k(R, \hat{R}) = \Gamma_k(R \setminus \hat{R}, \hat{R})$

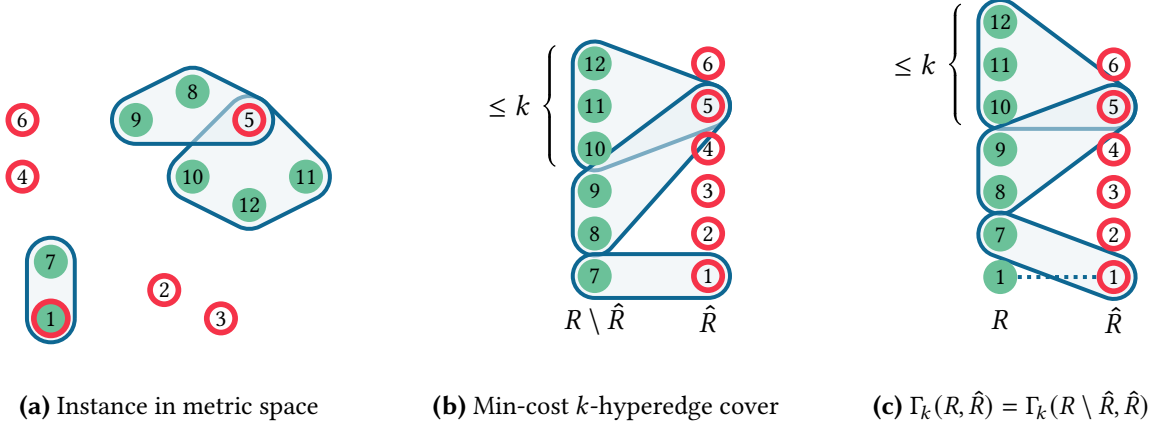



Figure 1: Example for a metric instance and input prediction with a min-cost k-hyperedge cover of the set of unexpected requests $R \setminus \hat{R}$. The actual requests are filled green and the predicted requests are encircled red. The labels show which points in the metric space correspond to which nodes in the bipartite graphs.

by, respectively, predicted and actual requests in the cheapest possible way.

We now embed this intuition into a precise definition. Let A and B be two sets of (possibly different) size and let $k \geq 1$. We define a bipartite hypergraph $G_k = (A \cup B, \mathcal{H})$ where $\mathcal{H}$ is the set of all hyperedges which have exactly one endpoint in B and at most k endpoints in A. A *k-hyperedge cover of A by B* is a set of hyperedges $\mathcal{H}' \subseteq \mathcal{H}$ in G_k such that every vertex in A is incident to at least one hyperedge in $\mathcal{H}'$. If every hyperedge $h \in \mathcal{H}$ of G_k has an associated cost $\gamma(h)$, a *minimum-cost k-hyperedge cover $\mathcal{H}'$* is a k-hyperedge cover which minimizes the total hyperedge cost $\sum_{h' \in \mathcal{H}'} \gamma(h')$. We denote the value of a min-cost k-hyperedge cover of A by B by $\Gamma_k(A, B)$. Finally, the cover error, denoted by $\Lambda_k(R, \hat{R})$, is given by

$$\Lambda_k(R, \hat{R}) = \Gamma_\infty(\hat{R}, R) + \Gamma_k(R, \hat{R}).$$

Notice that we allow arbitrary large hyperedges ($k = \infty$) to cover predicted requests $\hat{R}$. We emphasize that all results also hold for a symmetric error definition $\Gamma_k(\hat{R}, R) + \Gamma_k(R, \hat{R})$, because $\Gamma_i(A, B) \geq \Gamma_{i+1}(A, B)$, for any i. Nevertheless we use this asymmetric definition to obtain a stronger bound when covering $\hat{R}$. Intuitively, this is possible because all predicted requests are known in advance (as opposed to the actual requests, which arrive online).

We simply write Λ_k if $\hat{R}$ and R are clear from the context. Since our error measure shall give value zero if $\hat{R} = R$, we require that the cost of every hyperedge $\{a\} \cup \{b\}$ for some $a \in A$ and $b \in B$ is equal to zero if $a = b$. Then, all vertices in $A \cap B$ can be covered trivially by B, and we conclude that $\Gamma_k(A \setminus B, B) = \Gamma_k(A, B)$. Figure 1 depicts an example of a k-hyperedge cover.

It remains to specify the cost $\gamma(A', b)$ of a hyperedge $A' \cup \{b\}$. Although we will give precise definitions separately for every concrete problem, all definitions follow a certain paradigm. That is, the cost $\gamma(A', b)$ shall be equal to *the value of an optimal solution for the subinstance induced by A' with respect to b*. This anchoring requirement is the *single* detail which has to be specified for a concrete problem. Note that this matches our intuition discussed above for OLTSP.

Comparison to other error measures. We compare the cover error to previously proposed error measures for the (undirected) online Steiner tree problem. Xu and Moseley [51] define a prediction error $\eta = \max\{|\hat{R}|, |R|\} - |\hat{R} \cap R|$, the number of erroneous requests, and prove that their algorithm is $O(\log(\min\{|R|, \eta\}))$-competitive. Azar et al. [12] introduce the *metric error with outliers* $\lambda = (\Delta, D)$,

6

where D is the value of a min-cost perfect matching between two equally sized subsets of R and $\hat{R}$, and Δ is the total number of unmatched points in R and $\hat{R}$. They prove for their algorithms a multiplicative error dependency w.r.t. $\log(\min\{|R|, \Delta\})$ and an additive error dependency w.r.t. D.

We give a family of instances with n actual requests and an input prediction for which the algorithms of Azar et al. [12] and Xu and Moseley [51] perform arguably well, but their error measures and analyses yield a bound of $O(\log n) \cdot \text{OPT} + O(\epsilon)$, which could be achieved even without predictions. For some $\epsilon > 0$, the instance is composed of one terminal request at x_1 and $n - 1$ requests in an ϵ-ball around the Steiner point x_2, but no request is exactly on x_2. Both x_1 and x_2 are predicted. We can immediately observe that the number of erroneous requests [51] is $\eta = n - 1$. For the metric error with outliers [12], note that any perfect matching is composed of at most two matches, therefore $\Delta \geq n - 2$ and $D = O(\epsilon)$. On the other hand, our cover error is bounded by $\Lambda_k \leq \Lambda_1 = O(n \cdot \epsilon)$ for any k, because x_2 covers all requests in the ϵ-ball around it. Then, Theorem 3 concludes that the algorithm of Azar et al. [12] is indeed constant competitive for this instance when $\epsilon \to 0$.

3 Online metric TSP with predictions

Let $M = (X, d)$ be a metric space, consisting of a set of points X, with origin $o \in X$ and a metric d. In the *Online Metric Traveling Salesperson Problem* (OLTSP), a set of unknown requests R is released online over time. A request (x, r) is composed of a point $x \in X$ and a release date $r \in \mathbb{R}_{\geq 0}$, i.e., the time at which the request becomes known and can be served. The task of an algorithm is to route a server, which is initially in the origin and moves at unit speed, through all requests back to the origin. The objective is to minimize the makespan, i.e., the total time required for this task.

The OLTSP *with predictions*, is an OLTSP in which we are given additionally an a priori prediction $\hat{R}$ on the set of requests. We assume that the server receives a signal when it is back at the origin after serving all the actual requests. Unlike in the classic OLTSP, this is important, as otherwise an algorithm might continue considering predicted requests, thus, ruling out any robustness.

To specify our cover error, we define the cost of a hyperedge $R' \cup \{(x', r')\}$ as the extra cost of serving erroneous (unexpected or predicted absent) requests R' w.r.t. a request x' (actual or predicted):

$$\gamma^{\text{TSP}}(R', (x', r')) = \textit{optimal makespan for serving instance } R' \textit{ from origin } x' \textit{ and initial time } r'.$$

To get some intuition, consider $X = \mathbb{R}_{\geq 0}$ and an algorithm that does not move before time t if there are no requests. It will receive an adversarial request (t, t) and hence encounters a ratio of $\frac{3}{2}$ [17]. To overcome this when having (almost) accurate predictions, the server has to move towards (predicted) requests before they actually arrive. However, this pre-moving technique brings several challenges. If an algorithm moves the server without interruption to a predicted request at the beginning of the instance, an adversary would immediately spawn the single actual request at the origin, giving an unbounded robustness. If the server would directly move back, one can similarly argue that the consistency is at least $\frac{3}{2}$. Therefore, the key is to define a proper waiting strategy before moving towards predicted requests. We show that we can execute an arbitrary online algorithm while delaying pre-moving to gain information about the instance. This is very delicate, since too much delay clearly weakens the consistency, but too little delay gives weak robustness. In Appendix A we prove the following result.

Theorem 5. *Let $\alpha \in (0, 1/2)$ and let $\mathcal{A}$ be a $(1 + \alpha)$-consistent deterministic learning-augmented algorithm for OLTSP. Then, $\mathcal{A}$ can be β-robust only for $\beta \geq \frac{1}{\alpha} - 1$. This holds even on the half-line.*

7

Our final algorithm uses a hyperparameter $\alpha > 0$ to configure the waiting duration and thereby achieves a tight asymptotic consistency-robustness tradeoff. Intuitively, we can express our confidence in the prediction using α and get customized guarantees.

3.1 A general framework for OLTSP with predictions

Our strategy involves an initial delay phase in which we follow an arbitrary online algorithm, up to some predetermined time depending on the cost $\hat{C}$ of an optimal tour $\hat{T}$ of the predicted requests $\hat{R}$. After that, we start following $\hat{T}$, adjusting it whenever the actual requests deviate from the predictions. We call this greedy strategy PREDICTREPLAN (PREDREPLAN for short), due to the analogy with the classic REPLAN heuristic [7]. Let $p(t)$ be the server's location at time t.

Algorithm 1 PREDREPLAN

Follow $\hat{T}$. Whenever an unexpected request (x, r) is released, recompute and follow a fastest tour from $p(r)$ to the origin serving all unserved predicted requests as well as all the unserved unexpected requests. If the server receives an end signal in the origin, terminate.

While this algorithm might move towards predicted requests which are known to be absent to make the analysis clearer, a practical implementation ignores these and thereby only improve its performance.

We formally define the class of algorithms DELAYTRUST, that is parameterized by our trust parameter $\alpha > 0$, which scales the delay. Let $\mathcal{A}$ be any ρ-competitive online algorithm for OLTSP.

Algorithm 2 DELAYTRUST

  i: Follow $\mathcal{A}$ as long as for time t it holds $t \le \alpha\hat{C} - d(p(t), o)$
 ii: Move the server to the origin
iii: Follow the PREDREPLAN strategy until the end

We refer to the execution of each line as a *phase*. We now prove the main Theorem 1 for OLTSP, by showing for DELAYTRUST separately an error-dependent bound in Lemma 6 and a robustness bound in Lemma 7. Given an OLTSP instance, we denote by C^* the makespan of an optimal tour T^* serving all actual requests.

Lemma 6. *DELAYTRUST has a competitive ratio of at most* $(1 + \alpha)\left(1 + 3 \cdot \frac{\Lambda_1}{C^*}\right)$, *for any $\alpha \ge 0$.*

Proof. We first bound $\hat{C}$. Fix a min-cost ∞-hyperedge cover of $\hat{R}$ by R and an optimal tour T^* for R. For every hyperedge $\hat{R}' \cup \{(x, r)\}$ in the cover, we extend T^* by adding the optimal offline OLTSP tour for $\hat{R}'$ which starts at x at the time t at which T^* serves x. Note that, since $r \le t$, the makespan of this subtour is bounded by the cost of $\hat{R}' \cup \{(x, r)\}$. Since every predicted request is covered by at least one hyperedge, the constructed tour serves $\hat{R}$ and we conclude that $\hat{C} \le C^* + \Gamma_\infty(\hat{R}, R)$.

We now bound the makespan of the tour of the algorithm. If the algorithm terminates in Phases (i) or (ii), its makespan is at most $\alpha\hat{C} \le \alpha \cdot (C^* + \Gamma_\infty(\hat{R}, R)) \le (1 + \alpha) \cdot (C^* + \Lambda_1)$.

Otherwise, the algorithm reaches Phase (iii). There it first computes an optimal tour $\hat{T}$ of length at most $\hat{C}$ serving all unserved predicted requests. The makespan only increases when unexpected requests arrive. To this end, fix a min-cost 1-hyperedge cover of R by $\hat{R}$ and a hyperedge $\{(x', r')\} \cup \{(\hat{x}, \hat{r})\}$ of this cover. We upper bound the additional cost due to (x', r') by the cost of an excursion from the algorithm's current

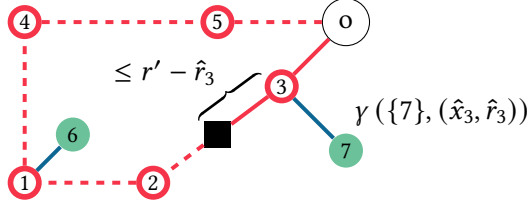
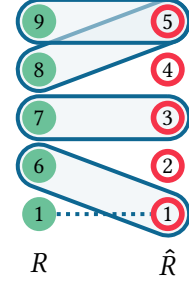
8

(a) The server (square) plans (dashed) and follows (solid) the predicted tour. Two hyperedges are completely released: $\{7\}$ is covered by 3, which is already served, and $\{6\}$ is covered by 1, which will be served in the future. Note that neither 1, 8 nor 9 have been released yet. Also notice that PREDREPLAN might find a faster tour.

(b) Min-cost 1-hyperedge cover of R

Figure 2

tour serving (x', r'). The algorithm might find a faster tour to serve all unserved requests and henceforth uses that. We distinguish two cases depending on the algorithm's remaining tour before request (x', r') arrived. If $\hat{x}$ is not part of this tour, we consider an excursion which immediately deviates from $p(r')$ to serve (x', r') and then returns to $p(r')$. By the triangle inequality, the length of this excursion is bounded by twice the distance between $p(r')$ and $\hat{x}$, plus the cost for optimally serving (x', r') from $\hat{x}$ when starting at time $\hat{r}$. Due to our assumption, $(\hat{x}, \hat{r})$ must have already been served at some time t with $r' \geq t \geq \hat{r}$. Thus, the algorithm's server is at most $r' - \hat{r}$ units away from $\hat{x}$ at time r', and the total time incurred for this excursion is bounded by

$$2 \cdot (r' - \hat{r}) + \gamma^{\text{TSP}}(\{(x', r')\}, (\hat{x}, \hat{r})) \leq 3 \cdot \gamma^{\text{TSP}}(\{(x', r')\}, (\hat{x}, \hat{r})).$$

Note that the inequality is due to the fact that (x', r') can only be served after its release date.

In the other case, the algorithm's server will visit $\hat{x}$ at some later point in time, especially at least once after time $\hat{r}$. We thus wait until the algorithm reaches $\hat{x}$ at some time $t \geq \hat{r}$, and then serve (x', r') using at most $\gamma^{\text{TSP}}(\{(x', r')\}, (\hat{x}, \hat{r}))$ additional time. See Figure 2 for an illustration of both cases.

Since every actual request is covered by one hyperedge, we conclude that Phase (iii) takes time at most $\hat{C} + 3 \cdot \Gamma_1(R, \hat{R})$. Adding the time for Phases (i) and (ii) gives a makespan of at most

$$(1 + \alpha)\hat{C} + 3 \cdot \Gamma_1(R, \hat{R}) \leq (1 + \alpha)\left(C^* + \Gamma_\infty(\hat{R}, R) + 3 \cdot \Gamma_1(R, \hat{R})\right) \leq (1 + \alpha)\left(C^* + 3 \cdot \Lambda_1\right). \qquad \square$$

Lemma 7. *DELAYTRUST has a competitive ratio of at most $1 + \rho + \frac{\rho}{\alpha}$, for any $\alpha > 0$ and any ρ-competitive algorithm used in Phase (i).*

Proof. If the algorithm terminates during Phase (i) or (ii), the competitive ratio is ρ. We are guaranteed to finish in one of these two phases if $\rho C^* \leq \alpha \hat{C}$.

If we terminate within Phase (iii), then $\hat{C} < \frac{\rho}{\alpha} C^*$. Once the last request has arrived at some time $r_{last} \leq C^*$, our tour stays fixed. We distinguish two cases. If the last request arrives before the end of Phase (ii), then the cost of our tour comprises of the cost for finishing Phase (ii), which is at most $\alpha \hat{C}$, and the cost of PREDREPLAN for following the predicted tour, including all unexpected yet unserved requests, which is at most $\hat{C} + C^*$. The total cost is thus bounded from above by

$$\alpha \hat{C} + \hat{C} + C^* \leq \left(1 + (1 + \alpha) \cdot \frac{\rho}{\alpha}\right) \cdot C^* = \left(1 + \rho + \frac{\rho}{\alpha}\right) \cdot C^*.$$

9

In the second case, the last request arrives in Phase (iii). In this case the cost after r_{last} is the cost of following the predicted tour, adapted for incorporating the unexpected, yet unserved, requests. This is bounded above by the cost of returning to the origin, following the predicted tour $\hat{T}$, and finally following the optimal tour, T^*. Note that the cost of returning to the origin is at most $r_{last} - \alpha C^*$. Hence, we complete the proof by upper bounding the makespan, for any $\rho \geq 1$, by

$$r_{last} + r_{last} - \alpha\hat{C} + \hat{C} + C^* \leq \left(3 + (1-\alpha)\cdot\frac{\rho}{\alpha}\right)C^* = \left(3 - \rho + \frac{\rho}{\alpha}\right)C^* \leq \left(1 + \rho + \frac{\rho}{\alpha}\right)C^*. \qquad \square$$

3.2 Extensions and improvements

An improved algorithm for OLTSP with predictions. A best possible online algorithm for OLTSP is SMARTSTART, which is 2-competitive [7]. Using this in Phase (i) of DELAYTRUST, Theorem 1 yields a robustness factor of at most $3 + \frac{2}{\alpha}$. We exploit SMARTSTART's waiting strategy to serve yet unserved requests and expedite Phase (iii) avoiding unnecessary waiting time, obtaining an algorithm, SMARTTRUST, with improved robustness factor $2 + \frac{2}{\alpha}$. See Theorem 2 for OLTSP in Appendix B.1.

Algorithms with polynomial running time. Algorithms DELAYTRUST and SMARTTRUST require the computation of optimal TSP tours on subinstances. NP-hardness of TSP prohibits polynomial running time, unless P=NP. We provide performance guarantees for our learning-augmented algorithm framework when using polynomial-time ν-approximation algorithms for solving TSP, which guarantee to find a TSP tour within a factor ν of the optimum.

We use a modified efficient PREDREPLAN strategy which uses a ν-approximate solution instead of an optimal solution and further ensures that errors due to such approximations do not add up too much compared to our error budget Λ_1. Adjusting the proof of Theorem 1 yields the following result, whose proof is in Appendix B.2.

Theorem 8. *Given a ν-approximation algorithm for metric TSP, the competitive ratio of the polynomial time DELAYTRUST using a polynomial time ρ-competitive online algorithm in Phase (i) is, for any $\alpha > 0$, bounded by*

$$\min\left\{(1+\nu)(1+\alpha)\left(1 + \frac{3\cdot\Lambda_1}{2\cdot C^*}\right), \rho + (1+\nu)\left(1 + \frac{\rho}{\alpha}\right)\right\}.$$

Online metric Dial-a-Ride with predictions. The *Online Metric Dial-a-Ride Problem* (OLDARP) is a generalization of OLTSP where each request (x^s, x^d, r) has a starting location x^s and a destination x^d. To serve a request, the server must first visit x^s at some time not earlier than r, and then x^d. We assume that the server can carry at most one request at the time and cannot store it after pickup.

We show in Appendix B.3 that slight modifications of DELAYTRUST and SMARTTRUST yield Theorems 1 and 2 for this generalized setting. We define the cost function γ^{DaRP} for the cover error:

$$\gamma^{DaRP}(R', (x^s, x^d, r)) = \min\{\gamma^{TSP}(R', (x^s, r)), \gamma^{TSP}(R', (x^d, r + d(x^s, x^d)))\} + D,$$

where D is the maximum transportation distance in $R \cap \hat{R}$. Intuitively, an excursion can start from x^s after time r or from x^d after time $r + d(x^s, x^d)$ to serve R', whatever is the shorter of the two.

10

An improved algorithm for OLTSP on the half-line metric. When restricting the metric space to $X = \mathbb{R}_{\geq 0}$, the best possible online algorithm is $\frac{3}{2}$-competitive [17]. We design a learning-augmented algorithm tailored to this metric space and a minimalistic prediction, namely, a single value $\hat{C}$ predicting the optimal makespan C^*. We prove (Appendix B.4) the following error-dependent performance bound, which gives an almost tight consistency-robustness tradeoff w.r.t Theorem 5.

Theorem 9. *There is a learning-augmented algorithm for the half-line metric that has for every $\alpha \in (0, 1/2]$ a competitive ratio of at most*

$$\min \left\{ (1 + \alpha) \left(1 + \frac{\Lambda_1}{C^*} \right), \frac{3}{2\alpha} \right\}.$$

4 Online network design problems with predictions

This section sketches the applicability of our new error measure for the online-list problems Steiner Tree, Steiner Forest and (capacitated) facility location. To prove new error-dependent performance bounds as stated in Theorem 3 and Theorem 4, we revisit the algorithms proposed by Azar et al. [12] and analyze it w.r.t. our cover error measure. The key is an appropriate, problem-specific definition of the cost of a hyperedge and the corresponding analysis. Recall that the cost of a hyperedge $R' \cup \{x'\}$ should (intuitively) be equal to the value of an optimal solution for serving a set of unexpected (predicted absent) requests R' w.r.t. a predicted (actual) request x'. We define the cost functions:

Steiner tree: $\gamma^{\text{ST}}(R', x') = $ *cost of an optimal Steiner tree for terminals R' with root x'.*

Steiner forest: $\gamma^{\text{SF}}(R', x') = $ *cost of an optimal Steiner forest for terminal pairs R' when connecting via $x' = (s', t')$ is free.*

Facility location: $\gamma^{\text{FL}}(R', x') = $ *cost for opening facility x' and assigning clients R' to it.*

Our main technical contribution for online network design problems are the proofs of Theorem 3 and Theorem 4. We now give some intuition on how these proofs work by considering the online Steiner tree problem, and defer details and results for this and the other problems to Appendix C.

On a high level, the algorithm by Azar et al. [12] for the online Steiner tree problem does the following. Each new request (terminal) is connected to the current solution greedily by buying edges on a shortest path to a vertex of the current tree. When the Greedy cost increased sufficiently (with thresholds following a doubling-strategy), the algorithm spends a certain budget (depending on the spent Greedy cost) on connecting as many future predicted requests as possible to the current solution.

The proof of Theorem 3 splits the execution of this algorithm into two parts, where the first part considers the time until all predicted requests are satisfied, and the second part the remaining execution. We then use a sub-result provided by Azar et al. [12] which roughly bounds the total cost of the first part by the optimal solutions of R and $\hat{R}$, and the total cost of the second part by the cost of the algorithm for serving specific subsequences of the request sequence. To further bound the first part, we use the structure of a min-cost ∞-hyperedge cover of $\hat{R}$ to prove an upper bound of at most $O(1) \cdot \text{OPT} + O(1) \cdot \Gamma_\infty(\hat{R}, R)$. For the second part, we consider the total cost the Greedy algorithm incurs for a hyperedge of a min-cost k-hyperedge cover of R, and conclude by the bounded hyperedge size and Greedy properties that this is at most $O(\log k)$ times the hyperedge cost, yielding a total bound of $O(1) \cdot \text{OPT} + O(\log k) \cdot \Gamma_k(R, \hat{R})$. Here we especially use the fact that in our chosen partition of the algorithm's execution, any predicted terminal $\hat{x}$ which covers actual requests must have already been served in the first part.

11

(a) Noise only in request locations   **(b)** Partial instance predicted correctly

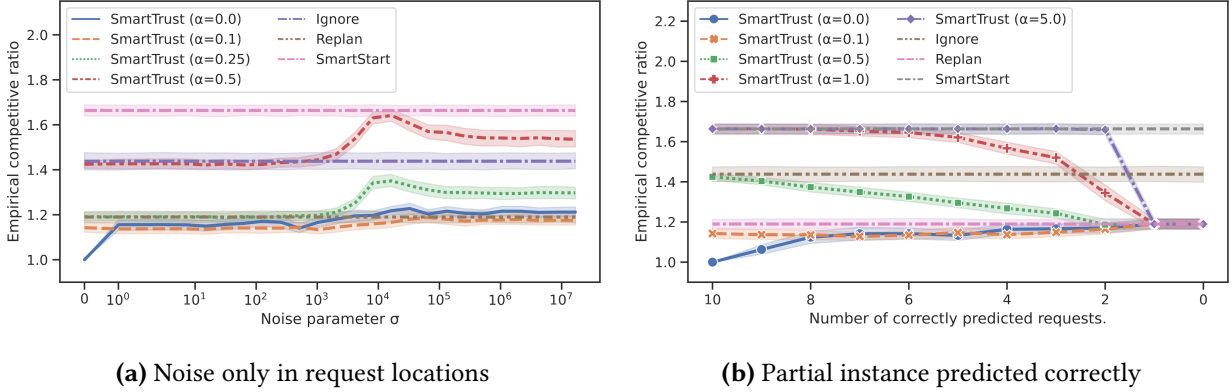



Figure 3: Experimental results for two different prediction settings (100 instances with 10 requests each)

5   Experiments

We performed various empirical experiments on real-world OLTSP instances that demonstrate the benefits of using our algorithms over classic online algorithms. We consider the road network of Manhattan [18, 44] and compose 100 instances of 10 requests each based on taxi pickup requests from a dataset offered by the NYC Taxi & Limousine Commission.[1] We compare SMARTTRUST with the classic online algorithms REPLAN [7], IGNORE [7, 24, 48] and SMARTSTART [7]; all algorithms use efficient TSP heuristics. We report for every experiment and instance the empirical competitive ratio, i.e. the average ratio between the algorithms performance and the approximated value of the optimal makespan, as well as error bars that denote the 95% confidence interval over all instances.

We sketch here two relevant experiments and defer further details to Appendix D. The first experiment considers synthetic predictions with Gaussian noise σ only in the request locations, i.e., the release dates are predicted correctly. The results (Figure 3a) show that SMARTTRUST with $\alpha = 0.1$ dominates classic algorithms even for arbitrarily bad predictions. In the second experiment only a part of the actual instance is predicted, which is an interesting and practice-relevant variant. Again, the results (Figure 3b) show that for small values of α, SMARTTRUST outperforms all classic algorithms.

Concluding remarks

The universal cover error can be applied to arbitrary problems with uncertain inputs. As it seems to be the first error measure that captures arrival times, it seems very natural to investigate, in particular, other online-time problems such as, e.g., scheduling problems. Further, it would be interesting to identify more compact or smaller predictions. While we predict a full input instance much less information might be sufficient to gain high-quality solutions. This can be only partial information about the input instance or predictions on algorithmic actions, such as an optimal tour instead of request sequences. In the latter case, we can directly apply our framework after approximating the predicted tour by some time-stamped discrete points and using those as input prediction.

---

[1]`https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page`, downloaded 02/05/22

12

References

[1] Matteo Almanza, Flavio Chierichetti, Silvio Lattanzi, Alessandro Panconesi, and Giuseppe Re. Online facility location with multiple advice. In *NeurIPS*, pages 4661–4673, 2021.

[2] Hyung-Chan An, Ashkan Norouzi-Fard, and Ola Svensson. Dynamic facility location via exponential clocks. *ACM Trans. Algorithms*, 13(2):21:1–21:20, 2017.

[3] Aris Anagnostopoulos, Russell Bent, Eli Upfal, and Pascal Van Hentenryck. A simple and deterministic competitive algorithm for online facility location. *Inf. Comput.*, 194(2):175–202, 2004.

[4] Spyros Angelopoulos. Improved bounds for the online steiner tree problem in graphs of bounded edge-asymmetry. In *SODA*, pages 248–257. SIAM, 2007.

[5] Antonios Antoniadis, Christian Coester, Marek Eliás, Adam Polak, and Bertrand Simon. Online metric algorithms with untrusted predictions. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pages 345–355. PMLR, 2020.

[6] Antonios Antoniadis, Peyman Jabbarzade Ganje, and Golnoosh Shahkarami. A novel prediction setup for online speed-scaling. In *SWAT*, volume 227 of *LIPIcs*, pages 9:1–9:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[7] Norbert Ascheuer, Sven Oliver Krumke, and Jörg Rambau. Online dial-a-ride problems: Minimizing the completion time. In *STACS*, volume 1770 of *Lecture Notes in Computer Science*, pages 639–650. Springer, 2000.

[8] Giorgio Ausiello, Esteban Feuerstein, Stefano Leonardi, Leen Stougie, and Maurizio Talamo. Algorithms for the on-line travelling salesman. *Algorithmica*, 29(4):560–581, 2001.

[9] Baruch Awerbuch, Yossi Azar, and Yair Bartal. On-line generalized steiner problem. In *SODA*, pages 68–74. ACM/SIAM, 1996.

[10] Yossi Azar, Stefano Leonardi, and Noam Touitou. Flow time scheduling with uncertain processing time. In *STOC*, pages 1070–1080. ACM, 2021.

[11] Yossi Azar, Stefano Leonardi, and Noam Touitou. Distortion-oblivious algorithms for minimizing flow time. In *SODA*, pages 252–274. SIAM, 2022.

[12] Yossi Azar, Debmalya Panigrahi, and Noam Touitou. Online graph algorithms with predictions. In *SODA*, pages 35–66. SIAM, 2022.

[13] Étienne Bamas, Marina Drygala, and Andreas Maggiori. An improved analysis of greedy for online steiner forest. In *SODA*, pages 3202–3229. SIAM, 2022.

[14] Étienne Bamas, Andreas Maggiori, Lars Rohwedder, and Ola Svensson. Learning augmented energy minimization via speed scaling. In *NeurIPS*, 2020.

[15] Piotr Berman and Chris Coulston. On-line algorithms for steiner tree problems (extended abstract). In *STOC*, pages 344–353. ACM, 1997.

13

[16] Antje Bjelde, Jan Hackfeld, Yann Disser, Christoph Hansknecht, Maarten Lipmann, Julie Meißner, Miriam Schlöter, Kevin Schewior, and Leen Stougie. Tight bounds for online TSP on the line. *ACM Trans. Algorithms*, 17(1):3:1–3:58, 2021.

[17] Michiel Blom, Sven Oliver Krumke, Willem de Paepe, and Leen Stougie. The online TSP against fair adversaries. *INFORMS J. Comput.*, 13(2):138–148, 2001.

[18] Geoff Boeing. Osmnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Comput. Environ. Urban Syst.*, 65:126–139, 2017.

[19] Niv Buchbinder, Shahar Chen, and Joseph Naor. Competitive analysis via regularization. In *SODA*, pages 436–444. SIAM, 2014.

[20] Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie Mellon University, 1976.

[21] Sina Dehghani, Soheil Ehsani, MohammadTaghi Hajiaghayi, Vahid Liaghat, and Saeed Seddighin. Greedy algorithms for online survivable network design. In *ICALP*, volume 107 of *LIPIcs*, pages 152:1–152:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[22] Franziska Eberle, Alexander Lindermayr, Nicole Megow, Lukas Nölke, and Jens Schlöter. Robustification of online graph exploration methods. In *AAAI*, pages 9732–9740. AAAI Press, 2022.

[23] David Eisenstat, Claire Mathieu, and Nicolas Schabanel. Facility location in evolving metrics. In *ICALP (2)*, volume 8573 of *Lecture Notes in Computer Science*, pages 459–470. Springer, 2014.

[24] Esteban Feuerstein and Leen Stougie. On-line single-server dial-a-ride problems. *Theor. Comput. Sci.*, 268(1):91–105, 2001.

[25] Dimitris Fotakis. A primal-dual algorithm for online non-uniform facility location. *J. Discrete Algorithms*, 5(1):141–148, 2007.

[26] Dimitris Fotakis. On the competitive ratio for online facility location. *Algorithmica*, 50(1):1–57, 2008.

[27] Dimitris Fotakis, Loukas Kavouras, and Lydia Zakynthinou. Online facility location in evolving metrics. *Algorithms*, 14(3):73, 2021.

[28] Themis Gouleakis, Konstantinos Lakis, and Golnoosh Shahkarami. Learning-augmented algorithms for online TSP on the line. *CoRR*, abs/2206.00655, 2022.

[29] 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.

[30] Sungjin Im, Ravi Kumar, Mahshid Montazer Qaem, and Manish Purohit. Non-clairvoyant scheduling with predictions. In *SPAA*, pages 285–294. ACM, 2021.

[31] Makoto Imase and Bernard M. Waxman. Dynamic steiner tree problem. *SIAM J. Discret. Math.*, 4(3):369–384, 1991.

[32] Patrick Jaillet and Michael R. Wagner. Online routing problems: Value of advanced information as improved competitive ratios. *Transp. Sci.*, 40(2):200–210, 2006.

14

[33] Patrick Jaillet and Michael R. Wagner. Generalized online routing: New competitive ratios, resource augmentation, and asymptotic analyses. *Oper. Res.*, 56(3):745–757, 2008.

[34] Vladimir Kolmogorov. Blossom V: a new implementation of a minimum cost perfect matching algorithm. *Math. Program. Comput.*, 1(1):43–67, 2009.

[35] Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In *SODA*, pages 1859–1877. SIAM, 2020.

[36] E. L. Lawler, Jan Karel Lenstra, Alexander H. G. Rinnooy Kan, and David B. Shmoys, editors. *The Traveling Salesman Problem – A Guided Tour of Combinatorial Optimization*. John Wiley & Sons Ltd., Chichester, 1985.

[37] Alexander Lindermayr and Nicole Megow. Permutation predictions for non-clairvoyant scheduling. In *SPAA*, pages 357–368. ACM, 2022.

[38] Alexander Lindermayr, Nicole Megow, and Bertrand Simon. Double coverage with machine-learned advice. In *ITCS*, volume 215 of *LIPIcs*, pages 99:1–99:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[39] Maarten Lipmann, Xiwen Lu, Willem de Paepe, René Sitters, and Leen Stougie. On-line dial-a-ride problems under a restricted information model. *Algorithmica*, 40(4):319–329, 2004.

[40] Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 3302–3311. PMLR, 2018.

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

[42] Adam Meyerson. Online facility location. In *FOCS*, pages 426–431. IEEE Computer Society, 2001.

[43] Michael Mitzenmacher. Scheduling with predictions and the price of misprediction. In *11th Innovations in Theoretical Computer Science Conference (ITCS)*, volume 151 of *LIPIcs*, pages 14:1–14:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[44] OpenStreetMap contributors. `https://www.openstreetmap.org`, 2017.

[45] Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ML predictions. In *NeurIPS*, pages 9684–9693, 2018.

[46] Ziv Scully, Isaac Grosof, and Michael Mitzenmacher. Uniform bounds for scheduling with job size estimates. In *ITCS*, volume 215 of *LIPIcs*, pages 114:1–114:30. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[47] Anatoliy I. Serdyukov. O nekotorykh ekstremalnykh obkhodakh v grafakh (on some extremal walks in graphs). *Upravlyaemye Sistemy (in Russian)*, 17:76–69, 1978.

[48] David B. Shmoys, Joel Wein, and David P. Williamson. Scheduling parallel machines on-line. *SIAM J. Comput.*, 24(6):1313–1331, 1995.

[49] Seeun Umboh. Online network design algorithms via hierarchical decompositions. In *SODA*, pages 1373–1387. SIAM, 2015.

15

[50] Jeffery R. Westbrook and Dicky C. K. Yan. Greedy algorithms for the on-line steiner tree and generalized steiner problems. In *WADS*, volume 709 of *Lecture Notes in Computer Science*, pages 622–633. Springer, 1993.

[51] Chenyang Xu and Benjamin Moseley. Learning-augmented algorithms for online steiner tree. In *AAAI*, pages 8744–8752. AAAI Press, 2022.

16

A A lower bound on the consistency-robustness tradeoff

Theorem 5. *Let $\alpha \in (0, 1/2)$ and let $\mathcal{A}$ be a $(1 + \alpha)$-consistent deterministic learning-augmented algorithm for OLTSP. Then, $\mathcal{A}$ can be β-robust only for $\beta \geq \frac{1}{\alpha} - 1$. This holds even on the half-line.*

Proof. Let $\epsilon > 0$ be a small constant such that $\epsilon \leq 1 - 2\alpha$. In the following we consider two instances. The first instance consists of the two requests $\sigma_1 = (0, 2\alpha + \epsilon)$ and $\sigma_2 = (1, 1)$. Since $\alpha \leq (1 - \epsilon)/2$, in the optimal solution the server immediately moves to 1, serving σ_2 at time 1, and is back at the origin at time 2, serving σ_1. Suppose that algorithm $\mathcal{A}$ has access to a perfect prediction. Thus, it has to finish the instance within time $2(1 + \alpha)$ due to its consistency. We can make two observations on the behavior of algorithm $\mathcal{A}$. Firstly, $\mathcal{A}$ must serve σ_2 before σ_1, as otherwise, its server must be at the origin at time $2\alpha + \epsilon$ and can finish the instance at the earliest at time $2(1 + \alpha) + \epsilon$, a contradiction. Secondly, at time 1, $\mathcal{A}$'s server cannot be strictly to the left of the point $1 - 2\alpha$, otherwise again its consistency would be contradicted.

Now consider a second instance, consisting of the single request $\sigma = (0, 2\alpha + \epsilon)$. Clearly, an optimal solution finishes at time $2\alpha + \epsilon$. Suppose that algorithm $\mathcal{A}$ gets the same prediction as in the first instance. Since $\mathcal{A}$ is deterministic and the two instances are the same until time 1, it will behave the same as in the first instance until time 1. By our observations from the first instance we conclude that at time 1, $\mathcal{A}$'s server is at least at distance $1 - 2\alpha$ from the origin, and it has not yet served σ_1. Thus, $\mathcal{A}$ can finish the second instance at the earliest at time $1 + 1 - 2\alpha$, yielding a robustness factor of at least $1/\alpha - 1$ for an arbitrarily small ϵ. $\square$

B Online routing problems with predictions

B.1 An improved algorithm for OLTSP with predictions

We investigate the particular algorithm SMARTSTART by Ascheuer et al. [7] to be applied in Phase (i) of DELAYTRUST. We show that the general framework DELAYTRUST can carefully be adjusted to better exploit the properties of SMARTSTART and obtain an improved robustness guarantee.

Algorithm 3 SMARTSTART [7]

Whenever the server is at the origin at some time t, compute an optimal tour S of length $\ell(S)$ serving all the released requests currently unserved. If $\ell(S) \leq t$, follow S while ignoring all the requests that are released in the meanwhile. Otherwise, restart the algorithm at time $\ell(S)$.

Ascheuer et al. [7] showed that this algorithm has a competitive ratio of 2 for OLDARP and, thus, OLTSP. Theorem 1 directly implies the following result.

Corollary 10. *DELAYTRUST using SMARTSTART in Phase (i) has a competitive ratio bounded from above by*

$$\min \left\{ (1 + \alpha) \left(1 + \frac{3 \cdot \Lambda_1}{C^*} \right), 3 + \frac{2}{\alpha} \right\},$$

for any $\alpha > 0$.

When SMARTSTART is used in Phase (i), we can carefully adjust our DELAYTRUST strategy. We exploit the criteria by which the server waits at the origin and expedite, in this case, immediately to Phase (ii), to avoid unnecessary waiting time. We define the following class of algorithms, parameterized by $\alpha > 0$, that we name SMARTTRUST.

17

Algorithm 4 SMARTTRUST
---
  i: Execute SMARTSTART with the following stopping criteria. If SMARTSTART decides to follow a tour S of length $\ell(S)$ at time t such that $t + \ell(S) > \alpha\hat{C}$, go to Phase (ii). If SMARTSTART sleeps or idles at time $\alpha\hat{C}$, go to Phase (iii).

  ii: Wait until time at least $\alpha \cdot \hat{C}/2$, then go to Phase (iii).

  iii: Follow the PREDREPLAN strategy until the end.

It is easy to verify that Lemma 6 holds for SMARTTRUST. Further, in Lemma 11 we show that we can improve upon the robustness factor given in Lemma 7, which then implies Theorem 2 for OLTSP.

Lemma 11. *SMARTTRUST has a competitive ratio of at most $2 + 2/\alpha$, for any $\alpha > 0$.*

Proof. Let C denote the makespan of SMARTTRUST's tour. If the algorithm terminates during Phase (i), then the competitive ratio is 2 by Ascheuer et al. [7]. Suppose the algorithm enters Phases (ii) and (iii). Let C_S be the cost of SMARTSTART when serving the whole actual online instance: because SMARTSTART is 2-competitive, it holds $C_S \leq 2C^*$. Since SMARTTRUST reaches Phase (iii), is must be $\alpha\hat{C} < C_S$, hence it holds $\alpha\hat{C} < 2C^*$.

Let $t_{(iii)}$ be the time at which Phase (iii) starts, t_{abort} be the time at which Phase (ii) begins (i.e., SMARTSTART is aborted at time t_{abort}), and r_{last} be the last actual release date. We distinguish two cases.

Case $r_{last} < t_{(iii)}$. Since SMARTTRUST reaches Phase (iii), it also entered Phase (ii) at time t_{abort}. Let C_{abort} be the length of an optimal tour serving all actual requests that are unserved at time t_{abort}. Note that $t_{abort} + C_{abort} > \alpha\hat{C}$, as otherwise this tour would have been started in Phase (i), because of the workings of SMARTSTART. We further distinguish two subcases:

$t_{abort} \geq \alpha \cdot \hat{C}/2$. In this case, SMARTTRUST does not have to wait in Phase (ii), and thus we have $t_{abort} = t_{(iii)}$. Observe that $t_{abort} + C_{abort} \leq C_S \leq 2 \cdot C^*$. Since $r_{last} < t_{(iii)} = t_{abort}$, the length of Phase (iii) is at most $\hat{C} + C_{abort}$, and we conclude

$$C \leq t_{abort} + C_{abort} + \hat{C} \leq 2 \cdot C^* + \hat{C} \leq \left(2 + \frac{2}{\alpha}\right) \cdot C^*.$$

$t_{abort} < \alpha \cdot \hat{C}/2$. Let $C_{(iii)}$ be the length of an optimal tour of all unserved actual requests at time $t_{(iii)}$. Since we assume $r_{last} < t_{(iii)}$, Phase (iii) takes at most $C_{(iii)} + \hat{C}$ time. Using the fact that Phase (iii) starts at time $\alpha\hat{C}/2$ we obtain

$$C \leq \frac{\alpha}{2}\hat{C} + C_{(iii)} + \hat{C} = C_{(iii)} + \left(1 + \frac{\alpha}{2}\right)\hat{C}$$
$$\leq C^* + \left(1 + \frac{\alpha}{2}\right)\frac{2}{\alpha}C^* = \left(2 + \frac{2}{\alpha}\right)C^*.$$

Case $r_{last} \geq t_{(iii)}$. Once the last request has arrived in Phase (iii) at time r_{last}, our tour stays fixed. The cost of the algorithm after r_{last} is the cost of following the predicted tour, adapted for incorporating the unexpected, yet unserved, requests. This is bounded above by the cost of returning to the origin, following the predicted tour and finally following the optimal tour. Note, that the cost for returning to the origin is at most $r_{last} - t_{(iii)}$. Further, $r_{last} \leq C^*$ and Phase (ii) ensures that $t_{(iii)} \geq \alpha\hat{C}/2$. Hence,

18

the algorithm's makespan satisfies

$$C \le r_{last} + (r_{last} - t_{(iii)}) + \hat{C} + C^* \le r_{last} + \left(r_{last} - \frac{\alpha}{2}\hat{C}\right) + \hat{C} + C^*$$

$$\le 2r_{last} + \left(1 - \frac{\alpha}{2}\right)\frac{2}{\alpha}C^* + C^* \le \left(3 + \left(1 - \frac{\alpha}{2}\right)\frac{2}{\alpha}\right)C^* = \left(2 + \frac{2}{\alpha}\right)C^*. \quad \square$$

We next show that the bound on the robustness of SMARTTRUST given in Lemma 11 is tight.

Lemma 12. *SMARTTRUST has a robustness factor of at least $2 + 2/\alpha$, for any $\alpha > 0$.*

Proof. Consider $\mathbb{R}$ as metric space. Let $\hat{\sigma} = (-1/2, 1/2)$ be the only predicted request, while the only actual request is $\sigma = (\alpha/4 + \epsilon, \alpha/4)$. Clearly, $C^* = \alpha/2 + 2\epsilon$ and $\hat{C} = 1$. In Phase (i), SMARTTRUST executes SMARTSTART until time $\alpha\hat{C} = \alpha$. Since the length of the tour that serves σ is equal to $\alpha/2 + 2\epsilon$, when σ is released SMARTSTART decides to sleep until time $\alpha/2 + 2\epsilon$. When SMARTSTART wakes up, the condition of Phase (i) forbids to execute the tour, and SMARTTRUST immediately goes to Phase (iii). We conclude that at time $\alpha/2 + 2\epsilon$ SMARTTRUST is at the origin and σ is still unserved. Then, the algorithm needs at least $1 + \alpha/2 + 2\epsilon$ time to serve σ and follow the predicted tour. This gives a robustness factor of at least

$$\frac{\alpha/2 + 2\epsilon + 1 + \alpha/2 + 2\epsilon}{\alpha/2 + 2\epsilon} = \frac{2\alpha + 2 + 8\epsilon}{\alpha + 4\epsilon} \xrightarrow{\epsilon \to 0} 2 + \frac{2}{\alpha}. \qquad \square$$

B.2 Algorithms with polynomial running time

In this section, we adapt DELAYTRUST to run in polynomial time and prove new performance bounds.

To this end, we first need to choose a polynomial time algorithm $\mathcal{A}$ for Phase (i) of DELAYTRUST. There are classic online algorithms, such as REPLAN [8] and SMARTSTART [7], which can be implemented to run in polynomial time by using an approximation algorithm to compute solutions for offline metric TSP instances. Using a ν-approximation algorithm for this task, the mentioned online algorithms achieve a competitive ratio of at most $\frac{3}{2} + \nu$ [8] resp. $\frac{1}{4}(4\nu + 1 + \sqrt{1 + 8\nu})$ [7].

Further, we have to adapt PREDREPLAN to run in polynomial time. Since PREDREPLAN relies on computing tours through requests with release dates, we also have to approximate such tours in polynomial time, unless $P = NP$. To do so one can use an approximation algorithm for offline metric TSP on the set of remaining requests (both predicted and unexpected) disregarding the release times. We then visit these requests and wait, in case of predicted requests, for release times, if needed.

Lemma 13. *Given a polynomial time ν-approximate algorithm for offline metric TSP without release times, we obtain a polynomial time $(1 + \nu)$-approximate algorithm for offline metric TSP with release times by following the returned tour and waiting if we arrive at a request early.*

Proof. The given algorithm returns a tour of length at most $\nu \cdot C^*$ in polynomial time. The total time spent waiting for the visited request to arrive is at most C^*, because no new requests arrive after time C^*. Therefore the algorithm takes at most $\nu \cdot C^* + C^* = (1 + \nu) \cdot C^*$ in polynomial time. $\qquad \square$

The polynomial-time DELAYTRUST uses a slight modification of PREDREPLAN in Phase (iii). This is necessary, since only replacing optimal solutions by $(1 + \nu)$-approximate solutions using Lemma 13 when recomputing tours in PREDREPLAN can invalidate bounds which we require for proving the error-dependency via previously considered excursions. We therefore only recompute a tour through unserved requests using a $(1 + \nu)$-approximation algorithm if this does not worsen our targeted error-dependent bound. On the

19

Algorithm 5 Polynomial-Time PREDREPLAN

1: Initially, use Lemma 13 to compute and follow a tour on the predicted requests $\hat{R}$.
2: Whenever an unexpected request $(x, r) \in R \setminus \hat{R}$ appears, find the predicted request $(\hat{x}, \hat{r}) \in \hat{R}$ which minimizes $\gamma^{\mathrm{TSP}}(\{(\hat{x}, \hat{r})\}, (x, r))$. Modify the current remaining tour to the origin as follows. If $\hat{x}$ is part of the current remaining tour, add an excursion that starts at $\hat{x}$ at some time $t \geq \hat{r}$, serves (x, r) and finally returns to $\hat{x}$. Otherwise, add an immediate deviation from $p(r)$ to (x, r) and back to $p(r)$ on the current tour. Let T_1 be the computed tour. Additionally, compute a new $(1 + \nu)$-approximate tour T_2 from $p(r)$ to the origin through all unserved requests (including (x, r)) using Lemma 13. Follow the shorter tour in $\{T_1, T_2\}$.
3: If the server receives a signal in the origin, terminate.

---

other hand, this still ensures that we always follow a $(1 + \nu)$-approximate tour through the remaining requests, which is important for being robust.

Finding the predicted request $(\hat{x}, \hat{r})$ for every unexpected request (x, r) in Step 2 can be done efficiently in time $O(|R \setminus \hat{R}| \cdot |\hat{R}|)$.

The remaining part of this section is dedicated to the proof of Theorem 8.

Theorem 8. *Given a ν-approximation algorithm for metric TSP, the competitive ratio of the polynomial time DELAYTRUST using a polynomial time ρ-competitive online algorithm in Phase (i) is, for any $\alpha > 0$, bounded by*

$$\min \left\{ (1 + \nu)(1 + \alpha) \left(1 + \frac{3 \cdot \Lambda_1}{2 \cdot C^*} \right), \rho + (1 + \nu) \left(1 + \frac{\rho}{\alpha} \right) \right\}.$$

In the following we prove Theorem 8 by separately proving the error-dependent bound in Lemma 14 and the robustness bound in Lemma 15. We use $\hat{C}$ to denote the length of an optimal tour on the predicted requests and $\tilde{C}$ for the length of a tour which is computed by a (fixed) $(1+\nu)$-approximation on the predicted requests using Lemma 13. Hence $\tilde{C} \leq (1 + \nu)\hat{C}$.

Lemma 14. *Given a ν-approximation algorithm for metric TSP, the competitive ratio of polynomial-time DELAYTRUST using a polynomial time ρ-competitive online algorithm in Phase (i) is, for any $\alpha > 0$, bounded by $(1 + \nu)(1 + \alpha) \left(1 + \frac{3}{2} \cdot \frac{\Lambda_1}{C^*} \right).$*

Proof. We first bound $\hat{C}$. Fix a min-cost ∞-hyperedge cover of $\hat{R}$ by R and an optimal tour T^* for R. For every hyperedge $\hat{R}' \cup \{(x, r)\}$ in the cover, we extend T^* by adding the optimal offline OLTSP tour for $\hat{R}'$ which starts at x at the time t at which T^* serves x. Note that, since $r \leq t$, the makespan of this subtour is bounded by the cost of $\hat{R}' \cup \{(x, r)\}$. Since every predicted request is covered by at least one hyperedge, the constructed tour serves $\hat{R}$ and we conclude that $\hat{C} \leq C^* + \Gamma_\infty(\hat{R}, R)$, whence the length of the approximate tour on all predicted requests is bounded by $(1 + \nu)\hat{C} \leq (1 + \nu)(C^* + \Gamma_\infty(\hat{R}, R))$.

We now bound the makespan of the tour of the algorithm. If the algorithm terminates in Phases (i) or (ii), its makespan is at most $(1 + \nu) \cdot \alpha\hat{C} \leq (1 + \nu) \cdot \alpha \cdot (C^* + \Gamma_\infty(\hat{R}, R)) \leq (1 + \nu)(1 + \alpha)(C^* + \Lambda_\infty)$.

Otherwise, the algorithm reaches Phase (iii). There it first computes a tour that serves all predicted requests of length $\tilde{C}$. Everytime an unexpected request (x', r') appears, the tour selection policy of the polynomial-time PREDREPLAN algorithm ensures that the makespan of the remaining tour increases at most by the length of the computed excursion to serve (x', r'). Let $(\hat{x}, \hat{r})$ be the predicted request that the algorithm computes and uses for the excursion. We distinguish two cases.

20

If $\hat{x}$ is not part of this tour, the algorithm immediately deviates from $p(r')$ to serve (x', r') and then returns to $p(r')$. By the triangle inequality, the length of this excursion is bounded by twice the distance between $p(r')$ and $\hat{x}$, plus the cost for optimally serving (x', r') from $\hat{x}$ when starting at time $\hat{r}$. Since $\hat{x}$ is not part of the remaining tour, $(\hat{x}, \hat{r})$ must have already been served at some time t with $r' \geq t \geq \hat{r}$. Thus, the algorithm's server is at most $r' - \hat{r}$ units away from $\hat{x}$ at time r'. Therefore, the total time incurred for this excursion is bounded by

$$2 \cdot (r' - \hat{r}) + \gamma^{\text{TSP}}(\{(x', r')\}, (\hat{x}, \hat{r})) \leq 3 \cdot \gamma^{\text{TSP}}(\{(x', r')\}, (\hat{x}, \hat{r})).$$

Note that the inequality is due to the fact that (x', r') can only be served after its release date. In the other case, the algorithm serves (x', r') via an excursion from $\hat{x}$ at some time $t \geq \hat{r}$. This takes at most $\gamma^{\text{TSP}}(\{(x', r')\}, (\hat{x}, \hat{r}))$ additional time.

We conclude that Phase (iii) takes time at most $\tilde{C} + 3 \cdot \Gamma_1(R, \hat{R})$. Adding the time for Phases (i) and (ii) and using that $v \geq 1$ gives a makespan of at most

$$
\begin{aligned}
(1 + \alpha)\tilde{C} + 3 \cdot \Gamma_1(R, \hat{R}) &\leq (1 + v)(1 + \alpha)\hat{C} + 3 \cdot \Gamma_1(R, \hat{R}) \\
&\leq (1 + v)(1 + \alpha)\left(C^* + \Gamma_\infty(\hat{R}, R)\right) + 3 \cdot \Gamma_1(R, \hat{R}) \\
&\leq (1 + v)(1 + \alpha)\left(C^* + \frac{3}{2} \cdot \Lambda_1\right).
\end{aligned}
$$

$\square$

Lemma 15. *Given a polynomial time v-approximate algorithm to solve metric TSP, polynomial-time DE-LAYTRUST has a competitive ratio of at most $\rho + (1 + v)(1 + \frac{\rho}{\alpha})$, for any polynomial time ρ-competitive algorithm used in Phase (i).*

Proof. If the algorithm terminates during Phase (i) or (ii), the competitive ratio is ρ. We are guaranteed to finish in one of these two phases if $\rho C^* \leq \alpha \tilde{C}$.

Now suppose we do not terminate within Phase (i) or (ii). Then, $\tilde{C} < \frac{\rho}{\alpha} C^*$. Once the last request has arrived at some time $r_{last} \leq C^*$, our tour stays fixed. We distinguish two cases. In the first case, the last request arrives before the end of Phase (ii). In this case, the cost of our tour comprises of the cost for finishing Phase (ii), which is at most $\alpha \tilde{C}$, and the cost of PredReplan. The latter is bounded by following the approximate tour, incorporating all unserved requests. This is shorter than following the approximate tour on the unexpected yet unserved requests and all predicted requests (served or not). The latter tour has optimal length not larger than $\hat{C} + C^*$. Thus, the length of the approximate tour is upper-bounded by $(1 + v)(\hat{C} + C^*)$. Adding the cost of finishing Phase (ii) yields a bound on the total cost of

$$\alpha \tilde{C} + (1 + v)(\hat{C} + C^*) \leq \alpha \tilde{C} + (1 + v)(\tilde{C} + C^*) \leq \left(\rho + (1 + v)\left(1 + \frac{\rho}{\alpha}\right)\right) C^*.$$

In the second case, the last request arrives in Phase (iii). In this case the cost after r_{last} is at most the cost of starting in the position at time r_{last} and serving all yet unserved requests using an approximate tour. The optimal cost is bounded by the cost of going back to the origin, which is at most $r_{last} - \alpha \tilde{C}$, and following an optimal tour, which takes time at most $\hat{C} + C^*$. Hence the total cost of the approximate tour is

21

bounded by

$$r_{last} + (1 + v)\left(r_{last} - \alpha\tilde{C} + \hat{C} + C^*\right) \le r_{last} + (1 + v)\left(r_{last} - \alpha\tilde{C} + \tilde{C} + C^*\right)$$

$$\le (3 + 2v)C^* + (1 + v)(1 - \alpha)\tilde{C}$$

$$\le (3 + 2v)C^* + (1 + v)\left(\frac{\rho}{\alpha} - \rho\right)C^*$$

$$= (3 + 2v)C^* - (1 + v)\rho C^* + (1 + v)\frac{\rho}{\alpha}C^*.$$

Since $\rho \ge 1$, we conclude that this is at most

$$(2 + v)C^* + (1 + v)\frac{\rho}{\alpha}C^* \le C^* + (1 + v)\left(1 + \frac{\rho}{\alpha}\right)C^* \le \left(\rho + (1 + v)\left(1 + \frac{\rho}{\alpha}\right)\right)C^*. \quad \square$$

We finally observe that Theorem 8 gives, using the polynomial-time version of the SMARTSTART algorithm in Phase (i) and Christofides' $\frac{3}{2}$-approximation algorithm [20, 47] to solve metric TSP instances, for any $\alpha > 0$, a competitive ratio of at most

$$\min\left\{(1 + \alpha)\left(\frac{5}{2} + \frac{15 \cdot \Lambda_1}{4 \cdot C^*}\right), 5.152 + \frac{6.629}{\alpha}\right\}.$$

B.3 Online metric Dial-a-Ride with predictions

In this section we show how the techniques we developed in Section 3 can be adapted to solve the more general OLDARP.

The *Online Metric Dial-a-Ride Problem* (OLDARP) is a generalization of OLTSP where each request (x^s, x^d, r) consists of a starting location x^s, a destination x^d and a release time r. To serve a request (x^s, x^d, r), the server must first visit x^s, at some time not earlier than r, and then x^d. We assume that the server can carry at most one request at the time and a move from x^s to x^d cannot be interrupted, i.e., there is no storage possible. In OLDARP *with predictions* we are given additionally predicted requests $\hat{R}$ specified by $(\hat{x}^s, \hat{x}^d, \hat{r})$.

We first introduce slight modifications of DELAYTRUST which enable it for OLDARP.

(a) In Phase (i), DELAYTRUST executes an algorithm that is ρ-competitive for OLDARP.

(b) While in Phase (i), DELAYTRUST only picks up a request if it can serve it and return to the origin until time $\alpha\hat{C}$.

(c) PREDREPLAN only recomputes tours if the server capacity is currently empty.

In the remaining part of this section we highlight how one can adapt the proofs for Theorem 1 and Theorem 2 from OLTSP to OLDARP.

But first we define the cost of a hyperedge for the cover error for OLDARP. By lifting the cost computed by γ^{TSP} to subinstances of OLDARP (i.e. for transportation requests), we define the cost of a hyperedge $R' \cup \{(x^s, x^d, r)\}$ as

$$\gamma^{\text{DaRP}}(R', (x^s, x^d, r)) = \min\left\{\gamma^{\text{TSP}}(R', (x^s, r)), \gamma^{\text{TSP}}(R', (x^d, r + d(x^s, x^d)))\right\} + D,$$

where $D = \max_{(x^s, x^d, r) \in R \cap \hat{R}} d(x^s, x^d)$. Further, for a request $(x^s, x^d, r) \in R \cap \hat{R}$ we set $\gamma^{\text{DaRP}}(\{(x^s, x^d, r)\}, (x^s, x^d, r)) = 0$. We emphasize that we require the additional quantity D in the hyperedge cost only for covering the unexpected actual requests, $R \setminus \hat{R}$, but not for covering absent predicted request, $\hat{R} \setminus R$.

We first proof the error-dependent bound in Theorem 1 for OLDARP.

22

Lemma 16. *The competitive ratio of DELAYTRUST for OLDARP is at most $(1 + \alpha)\left(1 + 3 \cdot \frac{\Lambda_1}{C^*}\right)$, for any $\alpha \geq 0$.*

Proof. We first bound $\hat{C}$. Fix a min-cost ∞-hyperedge cover of $\hat{R}$ by R and an optimal tour T^* for R. For every hyperedge $\hat{R}' \cup \{(x^s, x^d, r)\}$ in the cover, we extend T^* by adding the optimal offline OLDARP tour for $\hat{R}'$ which is considered in $\gamma^{\mathrm{DaRP}}(\hat{R}', (x^s, x^d, r))$. Since every predicted request is covered by at least one hyperedge, the constructed tour serves $\hat{R}$ and we conclude that $\hat{C} \leq C^* + \Gamma_\infty(\hat{R}, R)$.

We now bound the makespan of the tour of the algorithm. If the algorithm terminates in Phases (i) or (ii), its makespan is at most $\alpha\hat{C} \leq \alpha \cdot (C^* + \Gamma_\infty(\hat{R}, R)) \leq (1 + \alpha) \cdot (C^* + \Lambda_1)$.

Otherwise, the algorithm reaches Phase (iii). There it first computes an optimal tour $\hat{T}$ serving all predicted requests of length $\hat{C}$. The makespan only increases when unexpected requests arrive. To this end, fix a min-cost 1-hyperedge cover of R by $\hat{R}$ and a hyperedge $\{(x^s, x^d, r)\} \cup \{(\hat{x}^s, \hat{x}^d, \hat{r})\}$ of this cover. We upper bound the additional cost due to (x^s, x^d, r) by the cost of an excursion which serves (x^s, x^d, r) from the algorithm's current tour. The algorithm might find a faster tour to serve all remaining requests and henceforth uses that. In the following, we assume that the minimum in $\gamma^{\mathrm{DaRP}}(\{(x^s, x^d, r)\}, (\hat{x}^s, \hat{x}^d, \hat{r}))$ is attained by $\gamma^{\mathrm{TSP}}(\{(x^s, x^d, r)\}, (\hat{x}^s, \hat{r}))$, and only note differences for the other case.

We distinguish two cases depending on the algorithm's remaining tour before request (x^s, x^d, r) arrived. The first case assumes that $\hat{x}^s$ (resp. $\hat{x}^d$) is not part of this tour. This implies that $\hat{x}^s$ (resp. $\hat{x}^d$) must have already been visited at some time t with $r \geq t \geq \hat{r}$ (resp. $r \geq t \geq \hat{r} + d(\hat{x}^s, \hat{x}^d)$). We consider an excursion which starts at the next point in time when the server reaches a point $\hat{x}$ of a predicted request, serves (x^s, x^d, r) and returns to $\hat{x}$. We now bound the additional cost for this excursion. If the server follows another excursion at time r, it must have visited point $\hat{x}$ before this excursion, so especially before time r, but after time $\hat{r}$ (resp. $\hat{r} + d(\hat{x}^s, \hat{x}^d)$). If the server serves a correctly predicted request at time r, it will reach $\hat{x}$ (which is in this case the destination point of the currently served request) latest at time $r + D$. In both cases, the distance between $\hat{x}^s$ (resp. $\hat{x}^d$) and $\hat{x}$ is at most $D + r - \hat{r}$ (resp. $D + r - (\hat{r} + d(\hat{x}^s, \hat{x}^d))$). By the triangle inequality we observe that the length of the excursion for (x^s, x^d, r) is bounded by twice the distance from $\hat{x}$ to $\hat{x}^s$ (resp. $\hat{x}^d$) and the cost for serving (x^s, x^d, r) from $\hat{x}^s$ at time $\hat{r}$ (resp. $\hat{r} + d(\hat{x}^s, \hat{x}^d)$), that is

$$2 \cdot (D + r - \hat{r}) + \gamma^{\mathrm{TSP}}(\{(x^s, x^d, r)\}, (\hat{x}^s, \hat{r})) \leq 3 \cdot \gamma^{\mathrm{DaRP}}(\{(x^s, x^d, r)\}, (\hat{x}^s, \hat{x}^d, \hat{r})).$$

Note that the inequality is due to the fact that (x^s, x^d, r) can only be served after its release date, i.e., $r - \hat{r} \leq \gamma^{\mathrm{TSP}}(\{(x^s, x^d, r)\}, (\hat{x}^s, \hat{r})) - D$ (resp. $r - (\hat{r} + d(\hat{x}^s, \hat{x}^d)) \leq \gamma^{\mathrm{TSP}}(\{(x^s, x^d, r)\}, (\hat{x}^d, \hat{r} + d(\hat{x}^s, \hat{x}^d))) - D$).

In the second case, the algorithm's server will visit $\hat{x}^s$ (resp. $\hat{x}^d$) at some later point in time, especially at least once after time $\hat{r}$ (resp. $\hat{r} + d(\hat{x}^s, \hat{x}^d)$). We thus wait until the algorithm reaches $\hat{x}^s$ (resp. $\hat{x}^d$) at some time $t \geq \hat{r}$ (resp. $t \geq \hat{r} + d(\hat{x}^s, \hat{x}^d)$), and then serve (x^s, x^d, r) using at most $\gamma^{\mathrm{DaRP}}(\{(x^s, x^d, r)\}, (\hat{x}^s, \hat{x}^d, \hat{r}))$ additional time.

Since every actual request is covered by one hyperedge, we conclude that Phase (iii) takes time at most $\hat{C} + 3 \cdot \Gamma_1(R, \hat{R})$. Adding the time for Phases (i) and (ii) gives a makespan of at most

$$(1 + \alpha)\hat{C} + 3 \cdot \Gamma_1(R, \hat{R}) \leq (1 + \alpha)\left(C^* + \Gamma_\infty(\hat{R}, R) + 3 \cdot \Gamma_1(R, \hat{R})\right) \leq (1 + \alpha)(C^* + 3 \cdot \Lambda_1).$$

□

For the robustness bound of Theorem 1, note that at time r_{last} when the last request arrives the server might be in the processo of serving a ride, which has remaining cost C_{ride} at time r_{last}. While we have to add this value to the time until the algorithm computes its final tour T_{final} due to Modification (c), we can also subtract it from the length of T_{final}, because this particular ride is already served when computing

23

T_{final}. Using the other arguments of the proof of Lemma 7 for OLTSP, we conclude the stated robustness bound in Theorem 1 for OLDARP.

Similarly, one can lift SMARTTRUST to OLDARP and prove Theorem 2 for OLDARP.

B.4 Online TSP on the half-line with predictions

In this section we consider the special case of OLTSP in the very restricted metric space $X = \mathbb{R}_{\geq 0}$, the half-line, and show strengthened results.

Note that a request (x, r) cannot be served before time $\max\{x, r\}$; and once it has been served, at least x more time units are needed for returning to the origin. Hence, for a given set of requests, R, any algorithm has a makespan of at least $\max_{(x,r) \in R}\{r + x, 2x\}$. Further, an optimal offline algorithm can find a tour of this value by immediately going to the point $\max_{(x,r) \in R}\{r + x, 2x\}/2$ and then back to the origin, while serving all other requests on the way. We denote this minimal travel time by $C^* = \max_{(x,r) \in R}\{r + x, 2x\}$.

We propose a much more compact prediction model in which we predict a single value $\hat{C}$ instead of individual requests. This value is a prediction on the makespan of an optimal tour C^*.

By the argumentation above, we can compute for a given value $\hat{C}$ an optimal tour (assuming that the prediction is correct) by moving at time 0 to $\hat{C}/2$ and then returning to the origin. However, fully trusting the prediction may lead to solutions of unbounded robustness. Moreover, the lower bound on the tradeoff between consistency and robustness in Theorem 5 holds for OLTSP with predictions, even on the half-line.

This insight also shows us how to interpret the cover error in this prediction setting. From the point of view of an optimal solution, we can always reduce R to the single request $(C^*/2, 0)$. Symmetrically, one can interpret the prediction $\hat{C}$ as input prediction $\hat{R} = \{(\hat{C}/2, 0)\}$. The bipartite graph which we then consider for hyperedge costs has therefore only a single vertex on each side. Recall that the cost of a hyperedge $R' \cup \{x'\}$ should capture the optimal solution for instance R' with respect to x'. Here, the cost of a hyperedge $\{(x_1, r_1)\} \cup \{(x_2, r_2)\}$ is equal to the additional cost of an optimal solution to serve (x_1, r_1) if it can optimally serve (x_2, r_2) for free. All these observations give the following useful property.

Proposition 17. $\Lambda_1 = |\hat{C} - C^*|$.

Proof.
$$\Lambda_1 = \Gamma_1(R, \hat{R}) + \Gamma_1(\hat{R}, R) = 2 \cdot \max\left\{0, \frac{C^*}{2} - \frac{\hat{C}}{2}\right\} + 2 \cdot \max\left\{0, \frac{\hat{C}}{2} - \frac{C^*}{2}\right\} = |\hat{C} - C^*|.$$

$\square$

We propose a learning-augmented algorithm that combines the best possible online algorithm Move-Right-If-Necessary (MRIN) [17] with predictions in a three-stage framework similar to Section 3.1. but carefully exploits the particular functioning of MRIN. The $\frac{3}{2}$-competitive algorithm MRIN by Blom et al. [17] works as follows.

Algorithm 6 MRIN [17]

At any time t, if there is an unserved request to the right of the current position of the server, i.e., (x, r) with $r \leq t$ and $x > p(t)$, the server moves to the right with unit speed and, otherwise, it moves towards the origin with unit speed.

Given an instance of OLTSP with prediction $\hat{C}$, we define MRINTRUST, a class of algorithms parameterized by $\alpha \in (0, 0.5]$. The strategy consists again of three phases.

As the main result of this section, we show the following bound on the competitive ratio of MRINTRUST.

24

Algorithm 7 MRINTRUST

   i: Execute MRIN until time $\alpha\hat{C}$. Let $p_{(ii)} = p(\alpha\hat{C})$ be the position of the server at the end of this phase.
  ii: Move the server to the point $p_{(iii)} = \frac{1}{2}((1-\alpha)\hat{C} + p_{(ii)})$.
 iii: Execute MRIN again (starting from $p_{(iii)}$).

Theorem 9. *There is a learning-augmented algorithm for the half-line metric that has for every $\alpha \in (0, 1/2]$ a competitive ratio of at most*

$$\min\left\{(1+\alpha)\left(1 + \frac{\Lambda_1}{C^*}\right), \frac{3}{2\alpha}\right\}.$$

Before proving the two bounds separately in Lemmas 18 and 19, we state the following useful observation. Later we show that the bound in Lemma 18 is tight.

Observation 1. *The point $p_{(iii)}$ (start of Phase (iii)) is not to the left of the server's position $p_{(ii)}$ at the start of Phase (ii), that is, $p_{(iii)} \geq p_{(ii)}$.*

This is because $p_{(ii)} \leq \alpha\hat{C}$ and $\alpha \in (0, 1/2]$ imply

$$p_{(iii)} = \frac{1}{2}\left((1-\alpha)\hat{C} + p_{(ii)}\right) \geq \frac{1}{2}\alpha\hat{C} + \frac{1}{2}p_{(ii)} \geq p_{(ii)}.$$

Lemma 18. *MRINTRUST has a competitive ratio of at most $\frac{3}{2\alpha}$, for any $\alpha \in (0, 1/2]$.*

Proof. Let C be the makespan of a tour determined by MRINTRUST. If the algorithm terminates in Phase (i), then $C \leq \frac{3}{2} \cdot C^*$, because MRIN is 3/2-competitive [17]. Otherwise, Phase (i) requires $\alpha\hat{C}$ time and Phase (ii) requires $p_{(iii)} - p_{(ii)}$ time. By denoting the time used in Phase (iii) by $C_{(iii)}$, it follows

$$C = \alpha\hat{C} + (p_{(iii)} - p_{(ii)}) + C_{(iii)}. \tag{1}$$

Recall that in Phases (i) and (iii) the algorithm follows the MRIN. Consider the execution of MRIN on the same instance. Due to Phase (ii), in Phase (iii) MRINTRUST does not have to serve more requests than the ones served by MRIN after time $\alpha\hat{C}$. But, since MRINTRUST starts Phase (iii) at point $p_{(iii)}$ and MRIN is at point $p_{(ii)} \leq p_{(iii)}$ at time $\alpha\hat{C}$, Phase (iii) may take additional time equal to $p_{(iii)} - p_{(ii)}$ compared to MRIN to move back if there are no requests to the right of $p_{(ii)}$. As MRIN takes at most $\frac{3}{2}C^*$ time for the instance, we conclude that

$$\alpha\hat{C} + C_{(iii)} \leq \frac{3}{2}C^* + (p_{(iii)} - p_{(ii)}).$$

By Equation (1) and by the definition of $p_{(iii)}$ we obtain

$$C \leq 2(p_{(iii)} - p_{(ii)}) + \frac{3}{2}C^* \leq (1-\alpha)\hat{C} + \frac{3}{2}C^*. \tag{2}$$

Since we assumed that MRINTRUST does not terminate in Phase (i), the time required by MRIN for the same instance is at least $\alpha\hat{C}$. Thus, $\alpha\hat{C} \leq \frac{3}{2}C^*$, and together with (2) we obtain the claimed bound:

$$C \leq \frac{3(1-\alpha)}{2\alpha}C^* + \frac{3}{2}C^* = \frac{3}{2\alpha}C^*. \qquad \square$$

Lemma 19. *MRINTRUST has a competitive ratio of at most $(1+\alpha) \cdot \left(1 + \frac{\Lambda_1}{C^*}\right)$, for any $\alpha \in (0, 1/2]$.*

25

Proof. We can assume that the instance consists of at least one request, as otherwise, we would immediately receive an end-signal in the origin and clearly achieve the stated competitive ratio. For the case where MRINTRUST reaches Phase (iii), let $t_{(iii)}$ denote the time when Phase (iii) begins, that is, by the definition of $p_{(ii)}$ and $p_{(iii)}$, $t_{(iii)} = \alpha\hat{C} + (p_{(iii)} - p_{(ii)}) = \frac{1}{2}((1+\alpha)\hat{C} - p_{(ii)})$. We denote again by C the cost given by MRINTRUST and we distinguish two cases. In each case we prove that $C \leq (1+\alpha)(C^* + |\hat{C} - C^*|)$, and then assert the stated competitive ratio using Proposition 17.

Case $C^* \leq \hat{C}$. We start by showing that $C \leq (1+\alpha)\hat{C}$. If the algorithm does not reach Phase (ii), clearly $C \leq \alpha\hat{C}$. Now suppose that the algorithm reaches Phase (ii) (and therefore Phase (iii)) and the server is at position $p_{(iii)}$ at time $t_{(iii)}$. Observe that, in this case, every request released at time $t_{(iii)} + \delta$ cannot be to the right of $p_{(iii)} - \delta$ for any $0 \leq \delta \leq p_{(iii)}$. Indeed, the existence of such a request $(x, t_{(iii)} + \delta)$ with $x > p_{(iii)} - \delta$ would imply

$$
\begin{aligned}
C^* &\geq x + t_{(iii)} + \delta \\
&> p_{(iii)} - \delta + t_{(iii)} + \delta \\
&= p_{(iii)} + t_{(iii)} = \frac{1}{2}((1-\alpha)\hat{C} + p_{(ii)}) + \frac{1}{2}((1+\alpha)\hat{C} - p_{(ii)}) = \hat{C},
\end{aligned}
$$

a contradiction. Therefore, in Phase (iii) the algorithm first serves all the requests to the right of $p_{(iii)}$ released before time $t_{(iii)}$ (if there is any) and then the server goes straight back to the origin while serving all remaining requests. Since all requests are in the interval $[0, C^*/2] \subseteq [0, \hat{C}/2]$, the algorithm needs at most $\hat{C}$ time for phases (ii) and (iii), giving a total time of $(1+\alpha)\hat{C}$.

We thus obtain the desired bound on the algorithm's cost:

$$
C \leq (1+\alpha)\hat{C} = (1+\alpha)C^* + (1+\alpha)(\hat{C} - C^*).
$$

Case $C^* > \hat{C}$. In this case the algorithm must enter Phase (ii), as otherwise $C \leq \alpha\hat{C} < \alpha C^* \leq C^*$ contradicts that C^* is the optimal makespan. Thus, the server reaches position $p_{(iii)}$ at time $t_{(iii)}$, at the start of Phase (iii). We claim that MRINTRUST spends at most $B := \max\{C^* - \hat{C}, C^*/2 - p_{(iii)}\}$ time for moving *rightwards or for waiting at the origin after time $t_{(iii)}$*. This allows us to bound the algorithm's makespan, given by the time $t_{(iii)}$ spent on phases (i) and (ii), plus the time $p_{(iii)}$ needed to go back to the origin from the point reached at the start of Phase (iii), plus twice the time spent going to the right (or sitting at the origin waiting) in Phase (iii), as follows:

$$
\begin{aligned}
C &\leq t_{(iii)} + p_{(iii)} + 2B = \frac{1}{2}((1+\alpha)\hat{C} - p_{(ii)}) + \frac{1}{2}((1-\alpha)\hat{C} + p_{(ii)}) + 2B \\
&= \hat{C} + \max\left\{2C^* - 2\hat{C}, C^* - (1-\alpha)\hat{C} - p_{(ii)}\right\} \\
&\leq \max\left\{2C^* - \hat{C}, C^* + \alpha\hat{C}\right\} \\
&\leq (1+\alpha)C^* + (C^* - \hat{C}),
\end{aligned}
$$

which proves the desired bound.

We prove the claim by contradiction, for which we assume that the algorithm exceeds the bound of B when serving a request (x, r). This request must exist, as otherwise the server would not move rightwards or wait at the origin. If $r \leq t_{(iii)}$, the server directly moves from $p_{(iii)}$ to x at the beginning of Phase (iii). Our assumption implies $x - p_{(iii)} > B$, and thus

$$
C^* \geq 2x > 2B + 2p_{(iii)} \geq 2\left(\frac{C^*}{2} - p_{(iii)}\right) + 2p_{(iii)} = C^*,
$$

26

a contradiction. If $r > t_{(iii)}$, denote by L the total time spent by the server moving leftwards between time $t_{(iii)}$ and r. Thus, $r - t_{(iii)} - L$ is equal to the time the server spent moving rightwards or waiting in the origin between time $t_{(iii)}$ and r. Let $p(r)$ be the position of the server at time r. Note that $p(r) \leq x$. Due to our assumption, $x - p(r) + (r - t_{(iii)} - L) > B$. Since the server moved L units to the left after time $t_{(iii)}$, it cannot be to the left of point $p_{(iii)} - L$ at time r, that is, $p(r) \geq p_{(iii)} - L$. We conclude

$$
\begin{aligned}
C^* &\geq x + r \\
&> B - (r - t_{(iii)} - L) + p(r) + r \\
&\geq (C^* - \hat{C}) - (r - t_{(iii)} - L) + (p_{(iii)} - L) + r \\
&= C^* - \hat{C} + t_{(iii)} + p_{(iii)} = C^*,
\end{aligned}
$$

again a contradiction. □

We next show that the robustness factor $\frac{3}{2\alpha}$ is tight for our algorithm.

Lemma 20. *MRINTRUST has a robustness factor at least $\frac{3}{2\alpha}$, for any $\alpha \in (0, 1/2]$.*

Proof. Consider an instance consisting of a single request $\sigma = (\alpha/3, \alpha/3 + \epsilon)$ for a small $\epsilon > 0$, and suppose that we give MRINTRUST the prediction $\hat{C} = 1$. The algorithm serves σ at time $2/3 \cdot \alpha + \epsilon$ and the server is at position ϵ at the start of Phase (ii), i.e., at time α. Thus, it does not receive an end-signal but the server reaches point $p_{(iii)} = \frac{1}{2}(1 - \alpha + \epsilon)$ at time $t_{(ii)} = \frac{1}{2}(1 + \alpha - \epsilon)$. As there are no further requests, it moves back to the origin. Since an optimal solution serves σ at time $\alpha/3 + \epsilon$ and is back at the origin at time $2/3 \cdot \alpha + \epsilon$, we conclude that the prediction is not perfect and that the robustness ratio of the algorithm is at least

$$
\frac{\frac{1}{2}(1 + \alpha - \epsilon) + \frac{1}{2}(1 - \alpha + \epsilon)}{\frac{2}{3}\alpha + \epsilon} = \frac{1}{\frac{2}{3}\alpha + \epsilon} \xrightarrow{\epsilon \to 0} \frac{3}{2\alpha}.
$$

□

C   Online network design problems with predictions

This section covers our results for the following online-list graph problems: online Steiner tree, online Steiner Forest and (capacitated) facility location. We prove new and improved error-dependent performance bounds with respect to the cover error for algorithms based on the general framework introduced by Azar et al. [12].

In the following subsections we precisely define the problems mentioned above and specify problem-related details for the definition of the cover error. Before that, we give a short overview over the algorithmic framework of Azar et al. [12], introduce required notation and terminology which strictly follows [12], and prove an auxiliary lemma which helps to abstract parts of the connection between their algorithm and the cover error.

The framework of Azar et al. [12] combines in an iterative manner the execution of a plain online algorithm ON on the actual request set with partial solutions for the predicted input computed using a v-approximation algorithm for the price-collection variant of the corresponding offline problem. The execution of the framework for the ith request is called the ith iteration. Whenever in an iteration the total cost paid by the online algorithm so far doubles w.r.t. to the last iteration it doubled, an offline solution

27

for some part of $\hat{R}$ is added using the subroutine Partial. Such an iteration is called *major* iteration, and we call the sequence of following iterations until the next major iteration (including this) a *phase*. Let m be the total number of phases for a fixed input. We denote by Q_j the set of requests served by the online algorithm in phase j, and for any $Q'_j \subseteq Q_j$, $\mathrm{ON}_j(Q'_j)$ is the total cost of the online algorithm incurred to serve the requests in Q'_j in phase j. We write ON_j for $\mathrm{ON}_j(Q_j)$. Note that the online algorithm is always allowed to use the augmented solutions by the prediction with zero cost. Further, $\hat{B}_i$ denotes the total cost of the online algorithm until the latest previous major iteration before iteration i, and $\mathrm{OPT}(R')$ denotes the value of an optimal solution for input R'.

Lemma 21. *Suppose that the following two conditions hold for some function $\mu : \mathbb{N} \rightarrow \mathbb{R}^+$ and some $k \geq 1$:*

(a) $\mathrm{OPT}(\hat{R}) \leq \mathrm{OPT} + \Gamma_\infty(\hat{R}, R)$

(b) If there is a major iteration i in which all predicted requests become satisfied, we require $\mathrm{ON}_j(H) \leq O(\mu(k)) \cdot \gamma(H, \hat{x})$ for any $H \subseteq Q_j$ such that $|H| \leq k$, $j \in \{m-1, m\}$ and $\hat{x} \in \hat{R}$.

Then, the total cost of the framework in [12] is at most $O(1) \cdot \mathrm{OPT} + O(\mu(k)) \cdot \Lambda_k$.

Proof. If there exists a major iteration in which all predicted requests become satisfied, let this be the ith iteration. Otherwise, let i be the last iteration. Iteration i satisfies the following bounds [12, Lemma 2.2]:

1. The cost of the first i iterations is at most $O(1) \cdot \mathrm{OPT} + O(\hat{B}_{i-1})$.

2. If iteration i is not the last iteration, the total cost of the iterations after iteration i is at most $O(1) \cdot \max\{\mathrm{ON}_{m-1}, \mathrm{ON}_m\}$.

We first bound $\hat{B}_{i-1}$. Let $i' < i$ be the iteration in which $\hat{B}_{i-1}$ was set. Thus, some predicted requests were not satisfied by Partial in iteration i'. By the definition of the algorithm, we conclude that the total cost Partial must pay for satisfying all predicted requests in iteration i', that is $c(\text{Partial}(\hat{R}, 0))$, is strictly larger than $3\nu\hat{B}_{i'} = 3\nu\hat{B}_{i-1}$. By [12, Lemma 2.1], $\text{Partial}(\hat{R}, 0)$ approximates the least expensive solution that satisfies all predicted requests within a factor of 3ν. This implies that the optimal solution for $\hat{R}$, which is such a solution, has cost of at least $\hat{B}_{i-1}$, i.e. $\mathrm{OPT}(\hat{R}) \geq \hat{B}_{i-1}$. Condition (a) therefore implies that the cost of the first i iterations is at most

$$O(1) \cdot \mathrm{OPT} + O(\hat{B}_{i-1}) \leq O(1) \cdot \mathrm{OPT} + O(1) \cdot \mathrm{OPT}(\hat{R}) \leq O(1) \cdot \mathrm{OPT} + O(1) \cdot \Gamma_\infty(\hat{R}, R).$$

Second, we bound $\max\{\mathrm{ON}_{m-1}, \mathrm{ON}_m\}$ and assume that all predicted requests $\hat{R}$ are satisfied in iteration i. Let $j \in \{m-1, m\}$ and let $Q_j \subseteq R$ be the subset of requests considered by the online algorithm in phase j. Fix a min-cost k-hyperedge cover of R by $\hat{R}$, a hyperedge $R' \cup \{\hat{x}\}$ of the cover and let $H_j = Q_j \cap R'$ be the subset of requests of the hyperedge which ON_j serves. Recall that $|R'| \leq k$. Condition (b) implies

$$\mathrm{ON}_j(H_j) \leq O(\mu(k)) \cdot \gamma(H_j, \hat{x}),$$

By the choice of the hyperedge cover, every request in Q_j is in at least one hyperedge. Summing over all hyperedges of the cover gives

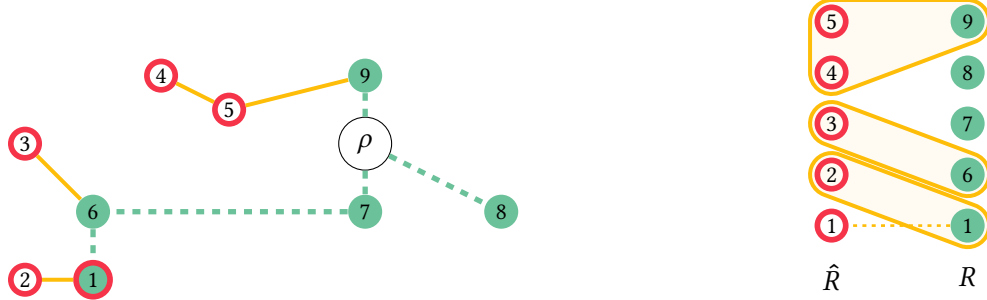
$$\mathrm{ON}_j(Q_j) \leq O(\mu(k)) \cdot \Gamma_k(R, \hat{R}),$$

and

$$\max\{\mathrm{ON}_{m-1}, \mathrm{ON}_m\} \leq O(\mu(k)) \cdot \Gamma_k(R, \hat{R}).$$

We conclude the statement by adding up both bounds, that is

$$O(1) \cdot \mathrm{OPT} + O(\Gamma_\infty(\hat{R}, R)) + O(\mu(k)) \cdot \Gamma_k(R, \hat{R}) \leq O(1) \cdot \mathrm{OPT} + O(\mu(k)) \cdot \Lambda_k. \qquad \square$$

28

(a) Augmentation of the optimal Steiner tree for R (dashed) to a Steiner tree which serves $\hat{R}$ using solutions of subinstances (solid) induced by hyperedges of the cover.

(b) Min-cost k-hyperedge cover of $\hat{R}$

Figure 4

C.1 Online Steiner tree

In the online (rooted) Steiner tree problem, a sequence of vertices $R \subseteq V$ (called terminals) of a graph $G = (V, E)$ with a distinct root vertex ρ is revealed one-by-one. An online algorithm maintains a solution S of selected edges and adds edges to S such that every arriving terminal is connected to ρ via edges in S. Every edge $e \in E$ has an associated cost $c_e \in \mathbb{R}^+$, and the objective of the algorithm is to minimize the total cost of selected edges $\sum_{e \in S} c_e$.

We define the cost $\gamma^{\mathrm{ST}}(R', x')$ of a hyperedge $R' \cup \{x'\}$ as the value of the optimal offline Steiner Tree for terminals R' with root x'.

Azar et al. [12] use the Greedy algorithm [31] as online algorithm in their framework, which connects an arriving terminal to the root using the shortest path to the current solution. This algorithm is $O(\log|R|)$-competitive in the online setting [31]. We denote it by $\mathrm{ON}^{\mathrm{ST}}$.

Theorem 22. *The algorithm in [12] for the (undirected) online Steiner tree problem incurs, for every $k \geq 1$, cost of at most $O(1) \cdot \mathrm{OPT} + O(\log k) \cdot \Lambda_k$.*

Proof. We show the bound on the competitive ratio using Lemma 21. It remains to prove the two required properties of Lemma 21.

First, consider a min-cost ∞-hyperedge cover of $\hat{R}$ by R. For every hyperedge $\hat{R}' \cup \{x\}$ in the cover, we connect $x \in R$ with all predicted requests in $\hat{R}'$ using the Steiner tree considered in $\gamma^{\mathrm{ST}}(\hat{R}', x)$. Adding an optimal Steiner Tree for the terminal set R then also satisfies all requests in $\hat{R}$ (see Figure 4), and since the augmentation cost are at most the hyperedge costs, we conclude

$$\mathrm{OPT}(\hat{R}) \leq \mathrm{OPT} + \Gamma_\infty(\hat{R}, R).$$

Second, assume that there is a major iteration i in which all predicted requests become satisfied. Let $j \in \{m - 1, m\}$, $H_j \subseteq Q_j$ such that $|H_j| \leq k$ and $\hat{x} \in \hat{R}$. In the proof of [12, Lemma 3.2] it is shown that

$$\mathrm{ON}^{\mathrm{ST}}_j(H_j) \leq O(\log k) \cdot \mathrm{OPT}_j(H_j),$$

where $\mathrm{OPT}_j(H_j)$ is the value of the optimal Steiner Tree for H_j in which edges that were bought before phase j have zero cost. Since we assume that $\hat{x}$ has already been served in or before iteration i, a feasible solution for H_j in phase j is given by connecting H_j optimally to $\hat{x}$. Therefore, $\mathrm{OPT}_j(H_j) \leq \gamma^{\mathrm{ST}}(H_j, \hat{x})$, and thus

$$\mathrm{ON}^{\mathrm{ST}}_j(H_j) \leq O(\log k) \cdot \mathrm{OPT}_j(H_j) \leq O(\log k) \cdot \gamma^{\mathrm{ST}}(H_j, \hat{x}). \qquad \square$$

29

C.2 Online Steiner forest

In the online Steiner forest problem, a sequence of vertex pairs $R \subseteq V \times V$ (called terminal pairs) of a graph $G = (V, E)$ is revealed one-by-one. An online algorithm maintains a solution S of selected edges and adds edges to S such that, of every arriving terminal pair (s, t), the vertices s and t are connected only via edges in S. Every edge $e \in E$ has an associated cost $c_e \in \mathbb{R}^+$, and the objective of the algorithm is to minimize the total cost of selected edges $\sum_{e \in S} c_e$.

We define the cost $\gamma^{\mathrm{SF}}(R', (s', t'))$ of a hyperedge $R' \cup \{(s', t')\}$ as the value of the optimal offline Steiner forest for terminal pairs R' where the distance between s' and t' has zero cost.

Azar et al. use a variant of an $O(\log|R|)$-competitive online algorithm of Berman and Coulston [15] in their framework, which we denote by $\mathrm{ON}^{\mathrm{SF}}$.

Theorem 23. *The algorithm in [12] for the online Steiner forest problem incurs, for every $k \geq 1$, cost of at most $O(1) \cdot \mathrm{OPT} + O(k) \cdot \Lambda_k$.*

Proof. We show the bound on the competitive ratio using Lemma 21. It remains to prove the two required properties of Lemma 21.

First, consider an optimal solution for R and fix a min-cost ∞-hyperedge cover of $\hat{R}$ by R. For every hyperedge $\hat{R}' \cup \{(s, t)\}$ in the cover, we augment the solution for R with the solution of the subinstance which is considered in $\gamma^{\mathrm{SF}}(\hat{R}', (s, t))$. Note that any terminal pair $(\hat{s}, \hat{t}) \in \hat{R}$ which is connected via connecting to s and t in this solution, remains connected, because s and t are connected in the solution of R. Since every client in $\hat{R}$ is contained in at least one hyperedge, the final solution satisfies $\hat{R}$ and we conclude

$$\mathrm{OPT}(\hat{R}) \leq \mathrm{OPT} + \Gamma_\infty(\hat{R}, R).$$

Second, assume that there is a major iteration i in which all predicted requests become satisfied. Let $j \in \{m - 1, m\}$, $H_j \subseteq Q_j$ such that $|H_j| \leq k$ and $(\hat{s}, \hat{t}) \in \hat{R}$. Azar et al. [12] note that for every $(s, t) \in H_j$, $\mathrm{ON}^{\mathrm{SF}}$ pays at most twice the distance to any previous request. Since we are assuming that $(\hat{s}, \hat{t})$ has already been connected before phase j, we conclude that $\mathrm{ON}_j^{\mathrm{SF}}(\{(s, t)\}) \leq 2 \cdot \gamma^{\mathrm{SF}}(\{(s, t)\}, (\hat{s}, \hat{t}))$. Summing over all requests in H_j gives

$$\mathrm{ON}_j^{\mathrm{SF}}(H_j) \leq 2k \cdot \max_{(s,t) \in H_j} \gamma^{\mathrm{SF}}(\{(s, t)\}, (\hat{s}, \hat{t})) \leq 2k \cdot \gamma^{\mathrm{SF}}(H_j, (\hat{s}, \hat{t})) = O(k) \cdot \gamma^{\mathrm{SF}}(H_j, (\hat{s}, \hat{t})).$$

□

C.3 Online facility location

In the online facility location problem, a sequence of vertices $R \subseteq V$ (called clients) of a graph $G = (V, E)$ with edge costs c_e arrive one-by-one. An online algorithm has to connect an arriving client x immediately to the closest open facility $v \in V$ with cost equal to the shortest path between v and x. The algorithm is also always allowed to open a (closed) facility v by paying opening cost f_v. The objective is to minimize the total connection and opening costs.

For a fixed solution of an instance, we denote for a client x the connected facility by v_x.

We define the cost $\gamma^{\mathrm{FL}}(R', x')$ of a hyperedge $R' \cup \{x'\}$ as the cost for opening facility x' and assigning clients R' to it, that is

$$\gamma^{\mathrm{FL}}(R', x') = f_{x'} + \sum_{r' \in R'} d(x', r').$$

30

We further slightly refine the cost of a k-hyperedge cover. For a k-hyperedge cover $\mathcal{H}'$ of R_1 by R_2 and for $x_2 \in R_2$, let $\mathcal{H}'(x_2)$ denote the subset of R_1 covered by x_2 in $\mathcal{H}'$. We define the cost of $\mathcal{H}'$ as follows:

$$\sum_{h \in \mathcal{H}'} \gamma^{\mathrm{FL}}(h) - \sum_{x_2 \in R_2 : |\mathcal{H}'(x_2)|=1} f_{x_2}.$$

Intuitively, this modification ensures that the error does not pay opening costs for facilities which cover only a single client in R_1, and therefore makes it more sensitive. We still write $\Gamma_k(R_1, R_2)$ to denote the minimal cost of any such cover. Azar et al. [12] use in their framework a $O(\log|R|)$-competitive online algorithm due to Fotakis [25], which we denote by $\mathrm{ON}^{\mathrm{FL}}$.

Theorem 24. *The algorithm in [12] for the online facility location problem incurs, for every $k \geq 1$, cost of at most $O(1) \cdot \mathrm{OPT} + O(\log k) \cdot \Lambda_k$.*

Proof. If there exists a major iteration in which all predicted requests become satisfied, let this be the ith iteration. Otherwise, let i be the last iteration. Iteration i satisfies the following bounds [12, Lemma 2.2]:

1. The cost of the first i iterations is at most $O(1) \cdot \mathrm{OPT} + O(\hat{B}_{i-1})$.

2. If iteration i is not the last iteration, the total cost of the iterations after iteration i is at most $O(1) \cdot \max\{\mathrm{ON}_{m-1}, \mathrm{ON}_m\}$.

First, we bound $\hat{B}_{i-1}$. Let $i' < i$ be the iteration in which $\hat{B}_{i-1}$ was set. Thus, some predicted requests were not satisfied by $\textsc{Partial}$ in iteration i'. By the definition of the algorithm, we conclude that the total cost $\textsc{Partial}$ must pay for satisfying all predicted requests in iteration i', that is $c(\textsc{Partial}(\hat{R}, 0))$, is strictly larger than $3\nu\hat{B}_{i'} = 3\nu\hat{B}_{i-1}$. By [12, Lemma 2.1], $\textsc{Partial}(\hat{R}, 0)$ approximates the least expensive solution that satisfies all predicted requests within a factor of 3ν. This implies that the optimal solution for $\hat{R}$, denoted by $\mathrm{OPT}(\hat{R})$, which is such a solution, satisfies

$$\hat{B}_{i-1} \leq \mathrm{OPT}(\hat{R}). \tag{3}$$

We now bound the additional cost in $\mathrm{OPT}(\hat{R})$ compared to $\mathrm{OPT}(R)$ using the cover error. To this end, consider an optimal solution for instance R and fix a min-cost ∞-hyperedge cover $\mathcal{H}$ of $\hat{R}$ by R. In the following, we modify the optimal solution for R by both disconnecting some actual requests from their facilities and assigning all requests $\hat{R}$ to existing facilities or new facilities. Note that the latter assignment might not be feasible because clients are not connected to their clostest open facility, but in this case we can clearly compute a feasible assignment without increasing the assignment costs. For every hyperedge $\hat{R}' \cup \{x\}$ in $\mathcal{H}$ we distinguish two cases depending on $\mathcal{H}(x)$. If $|\mathcal{H}(x)| = 1$, let $\hat{x}$ be the single request covered by x, i.e. $\hat{R}' = \{\hat{x}\}$. In the optimal solution for R, we disconnect x from its closest facility v_x and connect $\hat{x}$ to v_x. Compared to $\mathrm{OPT}(R)$, this increases the cost by at most

$$d(\hat{x}, v_x) - d(x, v_x) \leq d(x, \hat{x}) = \gamma^{\mathrm{FL}}(\{\hat{x}\}, x) - f_x.$$

If $|\mathcal{H}(x)| > 1$, we augment the optimal solution for R by the optimal solution considered in the cost function of the hyperedge, i.e. (possibly) open a facility at x and connect all clients in $\hat{R}'$ to x. This increases $\mathrm{OPT}(R)$ by at most $\gamma^{\mathrm{FL}}(\hat{R}', x)$.

Since every request in $\hat{R}$ is covered by at least one hyperedge, every client in $\hat{R}$ is assigned to a facility in the final solution and therefore

$$\mathrm{OPT}(\hat{R}) \leq \mathrm{OPT} + \Gamma_\infty(\hat{R}, R). \tag{4}$$

31

Second, we assume that all predicted requests $\hat{R}$ become satisfied in iteration i. We will now bound $\max\{\text{ON}^{\text{FL}}_{m-1}, \text{ON}^{\text{FL}}_{m}\}$. Let $j \in \{m-1, m\}$ and let $Q_j \subseteq R$ be the subset of requests considered by the online algorithm in phase j. Fix a min-cost k-hyperedge cover $\mathcal{H}$ of R by $\hat{R}$. Let F_0 be the set of open facilities in the beginning of phase j. Fix a hyperedge $R' \cup \{\hat{x}\} \in \mathcal{H}$ such that $Q_j \cap R' \neq \emptyset$ and let $H_j = Q_j \cap R'$ be the subset of requests of the hyperedge which ON_j serves in phase j. Recall that $|R'| \leq k$. We distinguish two cases. If $|\mathcal{H}(\hat{x})| = 1$, let $\{x\} = H_j = R'$ be the request covered by $\hat{x} \in \hat{R}$ and observe that the definition of the amortized costs [12, Definition 5.2] gives

$$\text{ON}^{\text{FL}}(x) \leq 2 \cdot d(x, F_0) \leq 2d(x, \hat{x}) + 2d(\hat{x}, F_0) = 2(\gamma^{\text{FL}}(\{x\}, \hat{x}) - f_{\hat{x}}) + 2(\hat{x}, F_0).$$

If $|\mathcal{H}(\hat{x})| > 1$, let $H_j = \{x_{i_1}, \ldots, x_{i_{k'}}\}$ be the covered requests indexed by their arrival in R. Note that $k' \leq k$. For the earliest request in H_j holds $\text{ON}^{\text{FL}}_j(x_{i_1}) \leq 2 \cdot f_{\hat{x}} + 2 \cdot d(\hat{x}, x_{i_1})$ [12, Definition 5.2]. For every $2 \leq \ell \leq k'$, let L_{i_ℓ} be the set of served requests by ON^{FL}_j until (including) iteration i_ℓ. Using again [12, Definition 5.2] gives

$$\begin{aligned}
\text{ON}^{\text{FL}}_j(x_{i_\ell}) &\leq 2 \cdot d(x_{i_\ell}, F_{i_\ell - 1}) \\
&\leq 2 \cdot d(\hat{x}, F_{i_\ell - 1}) + 2 \cdot d(\hat{x}, x_{i_\ell}) \\
&\leq \frac{2}{|L_{i_\ell - 1} \cap H_j|} \left(f_{\hat{x}} + 2 \cdot \sum_{v \in H_j} d(v, \hat{x}) \right) + 2 \cdot d(\hat{x}, x_{i_\ell}) \\
&\leq \frac{4}{\ell - 1} \cdot \gamma^{\text{FL}}(H_j, \hat{x}) + 2 \cdot d(\hat{x}, x_{i_\ell}),
\end{aligned}$$

where the first inequality is due to the definition of the amortized costs [12, Definition 5.2], the second due to the triangle inequality, and the third due to an argument by Fotakis [25, Corollary 1] applied to $\hat{x}$ and H_j. Using $k' \leq k$ and summing over all requests in H_j gives

$$\begin{aligned}
\text{ON}^{\text{FL}}_j(H_j) = \sum_{\ell=1}^{k'} \text{ON}^{\text{FL}}_j(x_{i_\ell}) \\
&\leq 2 \cdot f_{\hat{x}} + \sum_{\ell=2}^{k'} \left(\frac{4}{\ell - 1} \cdot \gamma^{\text{FL}}(H_j, \hat{x}) \right) + \sum_{\ell=1}^{k'} 2 \cdot d(\hat{x}, x_{i_\ell}) \\
&\leq \sum_{\ell=2}^{k'} \left(\frac{4}{\ell - 1} \cdot \gamma^{\text{FL}}(H_j, \hat{x}) \right) + 2 \cdot \gamma^{\text{FL}}(H_j, \hat{x}) \\
&\leq O(\log k) \cdot \gamma^{\text{FL}}(H_j, \hat{x}).
\end{aligned}$$

Since every request in Q_j is covered by some hyperedge, summing over all hyperedges in $\mathcal{H}$ gives

$$\text{ON}^{\text{FL}}_j(Q_j) \leq O(\log k) \cdot \Gamma_k(R, \hat{R}) + \sum_{\hat{x} \in \hat{R}} d(\hat{x}, F_0). \tag{5}$$

Since F_0 contains all facilities opened by a solution of PARTIAL in iteration i, $\sum_{\hat{x} \in \hat{R}} d(\hat{x}, F_0)$ is at most the connection cost of this solution. Using [12, Lemma 2.2] concludes that the cost of this solution is bounded by

$$O(1) \cdot \text{OPT} + O(\hat{B}_{i-1}).$$

32

Using (3) and (4) therefore implies

$$\sum_{\hat{x} \in \hat{R}} d(\hat{x}, F_0) \leq O(1) \cdot \text{OPT} + O(1) \cdot \Gamma_\infty(\hat{R}, R).$$

Thus, the bound in (5) is at most

$$\text{ON}_j^{\text{FL}}(Q_j) \leq O(\log k) \cdot \Gamma_k(R, \hat{R}) + O(\text{OPT}) + O(1) \cdot \Gamma_\infty(\hat{R}, R) \leq O(\text{OPT}) + O(\log k) \cdot \Lambda_k.$$

We finally conclude the statement by adding up both bounds, that is

$$O(1) \cdot \text{OPT} + O(1) \cdot \Gamma_\infty(\hat{R}, R) + O(1) \cdot \text{OPT} + O(\log k) \cdot \Lambda_k = O(1) \cdot \text{OPT} + O(\log k) \cdot \Lambda_k. \qquad \square$$

C.4 Soft-capacitated facility location

In the soft-capacitated facility location problems, every facility on a vertex v has a given capacity β_v on the number of clients that can be connected to it. Additionally, a solution is allowed to open multiple facilities, each with cost f_v, at a vertex v.

There exists a folklore reduction from the soft-capacitated variant to the standard facility location problem. For details we refer to [12].

This reduction also implies a reduction between costs of a hyperedge in the capacitated and non-capacitated case, as they are defined by values of solutions of subinstances. We therefore conclude that our result for the non-capacitated facility location problem naturally extends to the soft-capacitated problem.

Theorem 25. *The algorithm in [12] for the online soft-capacitated facility location problem incurs, for every $k \geq 1$, cost of at most $O(1) \cdot \text{OPT} + O(\log k) \cdot \Lambda_k$.*

D Experiments

We performed various empirical experiments on real-world instances that demonstrate the benefits of using our algorithm over classic online algorithms in relevant real-world scenarios. The source code includes instructions to reproduce them. We ran the experiments on an AMD Ryzen 9 3900X processor under Ubuntu 20.04.

Setup As metric space we use the shortest path completion of the road network of Manhattan. This network is composed of 4583 nodes, representing intersections and impasses, and 8130 edges, representing street segments. We construct this network with OSMnx [18] from OpenStreetMap data [44]. We generate real-world OLTSP instances by reconstructing taxi requests in Manhattan. The NYC Taxi & Limousine Commission offers public datasets of taxi trips online [2]. Specifically, we use the Yellow Taxi Trip Records from January 2021. We extract instances of a fixed length and use the pick-up location as request position. Since the dataset only determines a certain taxi zone as location, we randomly select a node of the city road network that is contained in this area. For the release date of a request, use the relative pick-up time-stamp and scale it to simulate the unit-speed behavior. We assume an average speed of 100 meters per minute. To generate synthetic predictions, we use independently sampled Gaussian noise of a given standard deviation σ. To compute predicted release dates, we add such noise to the actual release dates, while for predicted locations we randomly search a node that has a distance to the actual requested node of

---

[2]`https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page`

33

(a) Full picture   **(b)** Zoomed-in

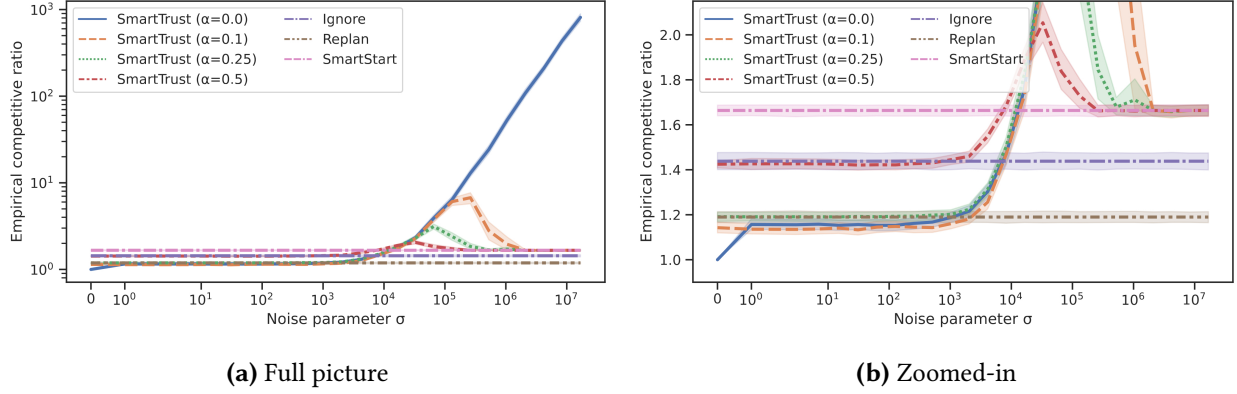



Figure 5: Results when adding noise to the request locations and release dates. (100 instances with 10 requests each)

approximately the given noise. We consider three different prediction settings: (i) noise in release dates and locations, (ii) noise only in locations, and (iii) having a randomly selected fraction of a certain size of the actual instance as prediction.

Algorithms We compare SMARTTRUST with the classic online algorithms REPLAN, IGNORE and SMART-START. We use the polynomial implementation of SMARTSTART as described in [7] (this variant uses a different waiting parameter $\theta = \frac{1+\sqrt{13}}{2} \approx 2.303$ internally). REPLAN greedily recomputes and follows a shortest tour from the current server position to the origin through all unserved requests whenever a new request arrives. IGNORE computes and then follows an optimal tour through all unserved requests whenever the server is in the origin, but ignores arriving requests while it is following a tour. The implementation of PREDREPLAN in SMARTTRUST ignores predicted requests which are already known to be absent. All algorithms compute repeatedly optimal or approximate TSP tours, which is known to be NP-hard [36]. In our implementation, Christofides' algorithm [20, 47] computes a heuristic for TSP tours in all algorithms, which itself uses Blossom V [34] to compute a min-cost perfect matching. Christofides' algorithm does not respect release dates. To approximate the optimal solution of a (predicted) OLTSP instance, we compute an approximate tour through all requests of the instance using Christofides' algorithm and then follow this tour greedily, i.e. wait at every point until the latest request at this point appeared.

Results For every prediction setting, we simulate all algorithms on every instance for every prediction parameter. We compute in every run its empirical competitive ratio, i.e. the average ratio between the algorithm's makespan and the approximated value of the optimal makespan. Further, we report error bars that denote the 95% confidence interval over all instances.

The results for the prediction setting (i) with noise in the release dates and in the locations are visualized in Figure 5. Although REPLAN performs really well on these real-world instances, SMARTTRUST with $\alpha = 0.0$ and $\alpha = 0.1$ still improves upon REPLAN while noise is small. For large noise, SMARTTRUST with $\alpha = 0.0$ has an unbounded competitive ratio as expected, as it does not ensure robustness by executing a competitive online algorithm in Phase (i). For a noise parameter $\sigma \approx 10^5$, SMARTTRUST achieves its worst performance. Larger noise ($\sigma \geq 10^6$) mainly results in late predicted release dates, since the largest path between two nodes in Manhattan is only $\approx 25 \cdot 10^5$ meters long. Therefore, $\hat{C}$ still grows, and at some point SMARTTRUST simply serves the whole instance in the first phase, which lets its performance coincide with the performance of SMARTSTART.

34

(a) Noise only in request locations **(b)** Partial instance predicted correctly

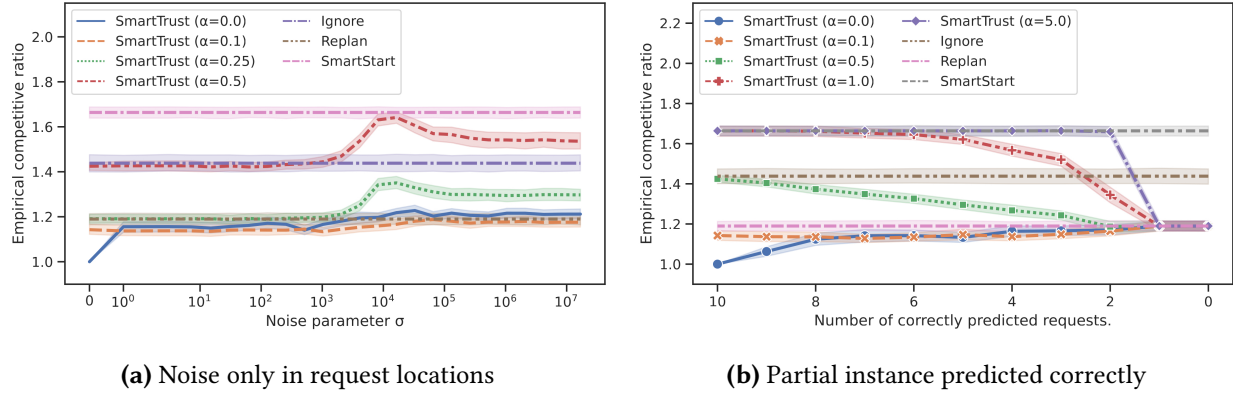



Figure 6: Results for prediction settings where parts of the actual instance are predicted correctly. (100 instances with 10 requests each)

It turns out that mainly erroneous release dates cause this performance spike. Indeed, in the prediction setting (ii) with correct release dates, the results (Figure 6a) indicate a better performance of SMARTTRUST. Specifically, for $\alpha = 0.1$ it completely dominates REPLAN over the whole range of reasonable expected prediction quality (by observing that due to the diameter of our network, worse noise parameters would have no significant impact). Intuitively, in this setting SMARTTRUST with $\alpha > 0$ only slightly pre-moves to predicted locations, but notices absent requests early enough and does not wait until they are predicted to arrive.

The third prediction setting (iii) is a scenario where a part of the actual instance is predicted correctly, and no further erroneous requests are given. One can think of this setting as a taxi company that knows some fixed pick-up requests of the coming day upfront, but still has uncertainty about short-term requests. On the one side, we can observe in Figure 6b that for having no predicted requests, the performance of SMARTTRUST equals the performance of REPLAN, as expected. On the other side, if α is large, we serve a large part of the actual requests in Phase (i), and achieve a performance close to SMARTSTART. For small values of α and some known requests, SMARTTRUST clearly outperforms REPLAN.

35