Learning-Augmented Online TSP on Rings, Trees, Flowers and (Almost) Everywhere Else

Evripidis Bampis
Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

Bruno Escoffier
Sorbonne Université, CNRS, LIP6, F-75005 Paris, France
Institut Universitaire de France, Paris, France

Themis Gouleakis
National University of Singapore, Singapore

Niklas Hahn
Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

Kostas Lakis
ETH Zürich, Switzerland

Golnoosh Shahkarami
Max-Planck-Institut für Informatik, Universität des Saarlandes, Saarbrücken, Germany

Michalis Xefteris
Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

Abstract

We study the Online Traveling Salesperson Problem (OLTSP) with predictions. In OLTSP, a sequence of initially unknown requests arrive over time at points (locations) of a metric space. The goal is, starting from a particular point of the metric space (the origin), to serve all these requests while minimizing the total time spent. The server moves with unit speed or is "waiting" (zero speed) at some location. We consider two variants: in the open variant, the goal is achieved when the last request is served. In the closed one, the server additionally has to return to the origin. We adopt a prediction model, introduced for OLTSP on the line [24], in which the predictions correspond to the locations of the requests and extend it to more general metric spaces.

We first propose an oracle-based algorithmic framework, inspired by previous work [14]. This framework allows us to design online algorithms for general metric spaces that provide competitive ratio guarantees which, given perfect predictions, beat the best possible classical guarantee (*consistency*). Moreover, they degrade gracefully along with the increase in error (*smoothness*), but always within a constant factor of the best known competitive ratio in the classical case (*robustness*).

Having reduced the problem to designing suitable efficient oracles, we describe how to achieve this for general metric spaces as well as specific metric spaces (rings, trees and flowers), the resulting algorithms being tractable in the latter case. The consistency guarantees of our algorithms are tight in almost all cases, and their smoothness guarantees only suffer a linear dependency on the error, which we show is necessary. Finally, we provide robustness guarantees improving previous results.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Online algorithms; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases TSP, Online algorithms, Learning-augmented algorithms, Algorithms with predictions, Competitive analysis

Digital Object Identifier 10.4230/LIPIcs.ESA.2023.12

Related Version *Full Version*: `https://arxiv.org/pdf/2305.02169.pdf` [13]

31st Annual European Symposium on Algorithms (ESA 2023).
Editors: Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman; Article No. 12; pp. 12:1–12:17

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Acknowledgements This work was partially funded by the grant ANR-19-CE48-0016 from the French National Research Agency (ANR). Kostas Lakis gratefully acknowledges financial support from the Foundation for Education and European Culture, Athens, Greece. Themis Gouleakis was supported by the National Research Foundation Fellowship for AI (Award NRF-NRFFAI-0002), an Amazon Research Award, and a Google South & Southeast Asia Research Award.

1 Introduction

In the classical Traveling Salesperson Problem (TSP), we are given a set of locations as well as the pairwise distances between them and the objective is to find a shortest tour visiting all the locations. TSP is one of the most fundamental and well studied problems in Computer Science [31]. We focus on the online version of the problem in a metric space, the Online Traveling Salesperson Problem (OLTSP), introduced in the seminal paper of Ausiello et al. [10]. In OLTSP, the input arrives over time, i.e., new requests (locations) that have to be visited by the traveler (or server) will appear during the travel. The time in which a request is communicated to the traveler is called its release time (or release date). The objective is the minimization of the total traveled time assuming that at any time the traveler either moves at unit speed or is "waiting" (zero speed) at some location[1]. We study both the *closed* variant, where the server is required to return to the origin after serving all requests, and the *open* variant, where the server does not have to return to the origin after serving all the requests. A series of papers considered many variations of OLTSP in different metric spaces (general metric space [10, 14], the line [10, 18, 24], the semi-line [9, 14, 19], the ring [14, 28] and the star [14]).

The motivation of studying OLTSP and its variations comes from applications in many different domains, such as e.g. logistics and robotics [8, 38]. In the framework of competitive analysis, the performance of an online algorithm is usually evaluated using the competitive ratio which is defined as the maximum ratio between the cost of the online algorithm and the cost of an optimal offline algorithm, which by definition has knowledge of the entire input in advance, over all input instances. However, it is admitted that the competitive analysis approach can be overly pessimistic as it is calculated considering worst-case instances, giving a lot of power to the adversary. Hence, many papers try to limit the power of the adversary [19], or give extra knowledge and hence more power to the online algorithm [1, 14, 27].

More recently, the framework of *Learning-Augmented (LA) algorithms* has emerged due to the vibrant successes of Machine Learning methods and Artificial Intelligence in predicting and learning the unknown (i.e., future inputs in the case of online algorithms) based on data [33]. In this line of research, the goal is to utilize predictions of the future input that have potentially been acquired using a learning algorithm, in order to have provably improved competitive ratio in the case that the predictions are accurate enough, while maintaining worst-case guarantees even if the prediction error is arbitrarily large. In particular, a (possibly erroneous) prediction of the input is given to the algorithm and the goal is to design algorithms with a good performance guarantee when the prediction is accurate (consistency), a not too bad (and bounded) performance when the prediction is wrong (robustness) and a gradual deterioration of the competitive ratio with respect to the prediction error (smoothness). We give more precise definitions in Section 2.

[1] Note that this choice of possible speeds is w.l.o.g., as any setting where the maximum speed is bounded can be reduced to this setting. Specifically, if the maximum speed is some $S > 0$, we can normalize the maximum speed to be 1 and multiply all distances by S. Also, our setting can simulate any other setting where the algorithm is free to chose either the unit speed of some other speed $0 < s < 1$. Moving at speed s is equivalent to dividing the time into intervals of length $dt \to 0$ and moving at unit speed only for time $s \cdot dt$ within each interval.

Recent works have proposed a variety of approaches to tackle OLTSP with predictions (LA-OLTSP). In [24], Gouleakis et al. studied a learning-augmented framework for OLTSP on the line. They introduced a prediction model in which the predictions correspond to the locations of the requests. They proposed LA algorithms for both the closed and the open variants that are consistent, smooth and robust. Bampis et al., in [14], considered the case of perfect predictions of the locations and studied different metric spaces (general metric space, semi-line, ring, and star) and proposed competitive online algorithms and lower bounds. In [26], Hu et al. proposed three different prediction models for OLTSP. In two of their models, each request is associated to a prediction for both its release time and its location while in the third one the prediction is just the release time of the last request. In [17], Bernardini et al. studied OLTSP with predictions of both the release time and the location of each request. They introduced a new error measure, the cover error, and they also considered other online graph problems. More recently, Chawla and Christou [20] studied the Online Time-Windows TSP with predictions of release time and location of the requests.

In this work, we adopt the prediction model of [24]. We propose a general oracle-based framework that allows us to design consistent, smooth and robust LA algorithms for both the closed and open variants of the problem for different metric spaces. We also provide some interesting lower bounds.

1.1 Our contributions and techniques

In this paper, we propose a novel approach to improve the competitive ratio of OLTSP using predictions concerning the locations of requests in general metric spaces. Our algorithms provide tight competitive ratio guarantees in most cases. Moreover, we show how to get polynomial-time/FPT algorithms in specific metrics, namely rings, trees and flowers.

Bampis et al. [14] gave an algorithm for general metrics with a competitive ratio of 3/2 for the case of perfect predictions (known locations), which is tight. First, in Section 3, we modify this algorithm (still under the assumption of perfect predictions) and introduce our main oracle-based 3/2-competitive framework, which we call *Strategically Wait And Go* (`SWAG`, for pseudocode see Algorithm 1). The main idea is to consider a suitable subset of permutations of the requests, referred to as *Dominating permutations*, given by a so-called *Domination oracle* instead of all the permutations. This allows for a reduction of the running time, since the bottleneck is located in the cardinality of the set of considered permutations. This restriction of the permutation set preserves the consistency of 3/2.

Then, we introduce our main algorithm, *Learning-Augmented Strategically Wait And Go* (`LA-SWAG`, for pseudocode see Algorithm 2), which does not assume perfect predictions. `LA-SWAG` is consistent, smooth, and robust. More formally, in Section 4 we show the following.

▶ **Theorem 1** (Consistency and Smoothness). *`LA-SWAG` has a competitive ratio of at most* $3/2 + 5\eta$ *for both closed and open LA-OLTSP.*

Here, η is the error of the prediction (defined formally later) that captures the normalized sum of distances between predicted and actual locations of the requests. Note that for $\eta = 0$, we get a consistency of 3/2, which is tight for all cases except the open variant on trees.

Additionally, we show a smoothness lower bound of $3/2 + \eta/2$ for the open variant with $\eta \in [0, 1/3]$ (Proposition 14), implying that a linear dependency on η is required.

Regarding robustness, the algorithm in [24] achieves 3-robustness on the line. `LA-SWAG` improves this bound for general metrics and further so in specific metrics.

▶ **Theorem 2** (Robustness-Closed). *`LA-SWAG` is* 2.75*-robust for closed LA-OLTSP in general metric spaces, and* 2.5*-robust in Euclidean spaces and in trees.*

▶ **Theorem 3** (Robustness-Open). *LA-SWAG is* $(3 - 1/6)$*-robust for open LA-OLTSP in general metric spaces, and* $(3 - 1/3)$*-robust in trees.*

Our analysis for 2.5-robustness and $(3 - 1/3)$-robustness is tight even on the line. Moreover, we show a negative result concerning the consistency/robustness trade-off of any algorithm for the open variant (Lemma 15).

The main technical contribution of our work is the implementation of the domination oracles we have referred to. On a high level, we say that a permutation π_{dom} dominates another permutation π at time t, if the following conditions hold. Assuming q is the first unreleased request in π at time t, the distance traveled up to q is not longer in π_{dom}, and also a superset of the requests preceding q in π is visited before q. Moreover, π_{dom} induces a not longer path than π. These two key facts allow us to preserve 3/2-consistency.

For general metrics, we achieve this domination using a very similar idea as the one employed in the definition of the $O(n^2 2^n)$ dynamic programming solution of the classical TSP [16, 25]. That is, for any possible subset of released requests that might have been served by π before q, we simply build two optimal paths for the parts before and after q (without release times) and then we concatenate them to get a dominating permutation. We call the resulting sets *general dominating sets*. Any permutation is dominated by the one corresponding to the correct guess of requests served up to q. This yields a single-exponential time algorithm overall.

▶ **Theorem 4** (General Metrics). *LA-SWAG with an oracle* $\mathcal{D}$ *which uses the general dominating sets runs in single-exponential time and is* $\min\{3/2 + 5\eta, 2.75\}$*-competitive for the closed variant and* $\min\{3/2 + 5\eta, 3 - 1/6\}$*-competitive for the open variant of LA-OLTSP.*

The overarching insight behind how we reduce the runtime in specific metrics is the fact that we do not really need to try all possible subsets of requests served before q. For example, in trees, we first show a structural result about the optimal solutions (with release times). Specifically, we prove that the requests placed along a path from a leaf to the origin (considered the root) can be assumed to be served in a very specific order. This is the order in which the requests are encountered as one traverses the path from the leaf to the origin. This fact allows us to design a domination oracle which, roughly, provides a single permutation for any subset of *leaves* visited before q. Hence, we get an FPT algorithm parameterized by the number l of leaves of the tree.

▶ **Theorem 5** (Trees). *There exists a Domination oracle for LA-SWAG in trees which yields a time complexity of* $O(2^l \cdot n^3)$ *for the closed variant and* $O(2^l \cdot n^4)$ *for the open variant, where* l *is the number of leaves of the input tree.*

As a first step towards more general graphs, we deal with the concept of cycles by considering the ring. While we cannot retrieve the exact same structural result about online optimal solutions, we show something quite similar. Namely, we prove that the cyclic nature of the ring may be utilized only once by an optimal solution. After such a cyclic traversal, we can assume that the ring is split in half, yielding a tree which we know how to deal with.

▶ **Theorem 6** (Ring). *There exists a Domination oracle for LA-SWAG in the ring which yields a time complexity of* $O(n^3)$ *for the closed variant and* $O(n^5)$ *for the open variant.*

Finally, we combine the two previous sets of ideas to tackle flowers. Flowers are essentially comprised of a bunch of rings (petals) and a semi-line (stem), all of which are attached to the origin. It is still true that each single ring may be assumed to be traversed with a loop only once in this case. It turns out that we can consider at most 6 options for every petal.

▶ **Theorem 7** (Flowers). *There exists a Domination oracle for* LA-SWAG *in flowers which yields a time complexity of* $O(6^p \cdot n^3)$ *for the closed variant and* $O(6^p \cdot n^5)$ *for the open variant, where* p *is the number of petals of the input flower.*

We summarize our results in Tables 1 and 2. In the closed variant, there is a lower bound of 3/2 even in the case of a line [24]. In the open one, we show a lower bound of ≈ 1.468 even in the case of a line and there is a lower bound of 3/2 even in the case of a ring [14]. A full version of the paper including all proofs can be found in [13].

Table 1 Consistency, smoothness, robustness and runtime guarantees of LA-SWAG for different metrics and variants.

	Smoothness (Consistency for $\eta = 0$)	Robustness		Runtime	
	Closed/Open	Closed	Open	Closed	Open
Tree	$3/2 + 5\eta$	2.5	$3 - 1/3$	$O(2^l \cdot n^3)$	$O(2^l \cdot n^4)$
Ring		2.75	$3 - 1/6$	$O(n^3)$	$O(n^5)$
Flower		2.75	$3 - 1/6$	$O(6^p \cdot n^3)$	$O(6^p \cdot n^5)$
Euclidean		2.5	$3 - 1/6$	$O(n^2 \cdot 2^n)$	$O(n^2 \cdot 2^n)$
General		2.75	$3 - 1/6$	$O(n^2 \cdot 2^n)$	$O(n^2 \cdot 2^n)$

Table 2 Consistency of LA-SWAG and other tractable algorithms. The upper bounds with * are given by an FPT algorithm and a polytime algorithm otherwise. Tight bounds are denoted in bold.

	Consistency			
	Closed		Open	
	Previous work	**This paper**	Previous work	**This paper**
Line	**3/2** [24]	**3/2**	5/3 [24]	3/2
Star	7/4+ϵ [14]	**3/2***	2 [14]	3/2*
Tree	2 [10]	**3/2***	2 [14]	3/2*
Ring	5/3 [14]	**3/2**	2 [14]	**3/2**
Flower	2 [10]	**3/2***	2 [14]	**3/2***

1.2 Further related works

The offline version of the problem, in which the locations and release times are known in advance, has been studied in [18, 38] for both closed and open variants. For OLTSP, a 2-competitive algorithm for the closed variant and a 2.5-competitive algorithm for the open variant have been proposed in general metric spaces by [10]. Specifically on the line, there exist lower bounds of 1.64 [10] and 2.04 [18] for the closed and open variants, respectively.

In addition to online TSP, there are several works that have explored learning augmented settings. The online caching problem with predictions was investigated by [33], and the initial results were improved by [3, 40, 42]. Adopting the LA approach, algorithms were developed for the ski-rental problem [2, 23, 39, 41] as well as for scheduling problems [4, 11, 35, 37]. There is also literature on learning augmented algorithms for classical data structures [30], bloom filters [34], routing problems [15, 22, 29], online selection and matching problems [5, 21] and a more general framework of online primal-dual algorithms [12]. There is a survey [36] and an updated list of papers [32] in this area.

2 Preliminaries

Online TSP (OLTSP)

The input of OLTSP consists of a metric space M with a distinguished point $\mathcal{O}$ (the origin), and a set $Q = \{q_1, ..., q_n\}$ of n requests. Every request q_i is a pair (x_i, t_i), where x_i is a point of M and $t_i \geq 0$ is a real number. We use t to quantify time. The number t_i represents the moment after which the request q_i can be served (release time). A server located at the origin at time $t = 0$, which can move with unit speed, must serve all the requests after their release times with the goal of minimizing the total completion time (makespan).

We consider a wide class of continuous metric spaces M whose corresponding distance metric $d(x, y)$ is defined as the shortest path from $x \in M$ to $y \in M$ and is continuous in M, as in [10]. We call this class general metric spaces (or general metrics). The release times for continuous metric spaces can be any non-negative real number.

For the rest of the paper, we denote the total completion time of an online algorithm ALG by $|\,\mathrm{ALG}\,|$ and that of an optimal (offline) solution OPT by $|\,\mathrm{OPT}\,|$. We recall that an algorithm ALG is ρ-competitive if on all instances we have $|\,\mathrm{ALG}\,| \leq \rho \cdot |\,\mathrm{OPT}\,|$.

Learning-augmented algorithms

In order to measure the quality of the predictions, we will define a prediction error η. LA algorithms have three main properties. We use the formal definitions in [24] here. We say that an algorithm is

- α-*consistent*, if it is α-*competitive* when $\eta = 0$,
- β-*robust*, if it is always β-*competitive* regardless of η, and
- γ-*smooth* for a continuous function $\gamma(\eta)$, if it is $\gamma(\eta)$-*competitive*.

In general, if c is the best competitive ratio achievable without predictions, it is desirable to have $\alpha < c$, $\beta \leq k \cdot c$ for some constant k and also the function γ should increase from α to β along with the error η.

Our prediction setting

Let $Q = \{q_1, \ldots, q_n\}$ be the set of requests. As we mentioned, each request q_i has a corresponding release time t_i and a location x_i. We have a set of predictions $P = \{p_1, \ldots, p_n\}$ in which p_i predicts x_i, the location of request q_i. The algorithm gets these predictions as well as the number of requests n as an offline input. The actual values of x_i and t_i only become known at time-point t_i.

The predictions' quality can vary and is unknown to the algorithm. We can evaluate the quality by defining a measure η. Essentially, η measures the sum of all the distance between the predicted location and the actual ones normalized by the length of a shortest path serving all the requests.

▶ **Definition 8** (Prediction Error). *The prediction error of an instance is defined by* $\eta = \frac{\sum_{i=1}^{n} d(x_i, p_i)}{F}$*, where* F *is the length of a shortest path serving all the requests (and returning to the origin in the closed case).*

Note that the prediction error is scale invariant (i.e., it will not change if all the distances are multiplied by a constant factor), and F acts as the normalization factor.

3 Oracle-based framework: the SWAG algorithm

In this section, we define an oracle-based algorithm, SWAG, designed for the case of perfect predictions. The oracle provides the algorithm with a set of permutations of the requests. We show that if the oracle satisfies some conditions, then SWAG has a competitive ratio of 3/2.

SWAG is actually a (slightly) modified version of the general algorithm in [14]. The principle of this latter algorithm is the following:

- First, to wait at $\mathcal{O}$ until a chosen time T. This time T depends both on the requests' locations and on their release times.
- Then, to choose a route of serving requests that minimizes some criterion involving the length of the corresponding route and the fraction of it which is released at time T, and to follow this route, waiting at unreleased requests.

The calculation of the chosen time T on the first step is done by computing some values on *all* the $n!$ permutations of the requests. The improvement we show here is that we can get the same competitive ratio (i.e., 3/2) while considering not the whole set of permutations, but some well chosen (and ideally small) subset that satisfies a certain property. This subset of permutations is given to the algorithm by the oracle.

Then, based on this framework, to derive an efficient 3/2-competitive algorithm, one only has to build an efficient oracle. We will show for example in Section 5.2 that for lines or rings, we can devise a polytime oracle building a polysize subset of permutations, leading to a polytime 3/2-competitive algorithm for these metrics.

More formally, we consider SWAG, which uses the following notation. For a given order σ on the requests (where $\sigma[i]$ denotes the i-th request in the order), we denote:

- by ℓ_σ the length of the route associated to σ (starting at $\mathcal{O}$), i.e., $\ell_\sigma = d(\mathcal{O}, \sigma[1]) + \sum_{j=1}^{n-1} d(\sigma[j], \sigma[j+1])$ in the open case, $\ell_\sigma = d(O, \sigma[1]) + \sum_{j=1}^{n-1} d(\sigma[j], \sigma[j+1]) + d(\sigma[n], \mathcal{O})$ in the closed case;
- by $\alpha_\sigma(t)$ the fraction of the length of the largest fully released prefix of the route associated to σ at time t over ℓ_σ. More formally, if all n requests are released, then $\alpha_\sigma(t) = 1$ for all σ, otherwise, if requests $\sigma[1], \ldots, \sigma[k-1]$ are released at t but $\sigma[k]$ is not, then the route is fully released up to $\sigma[k]$, and

$$\alpha_\sigma(t) = \big(d(\mathcal{O}, \sigma[1]) + \sum_{j=1}^{k-1} d(\sigma[j], \sigma[j+1])\big)/\ell_\sigma \ .$$

  Note that this requires $\ell_\sigma > 0$. If $\ell_\sigma = 0$, i.e., all requests are at $\mathcal{O}$, we set $\alpha_\sigma = 1$.

The key idea behind these subsets is the following. To achieve the same competitive ratio, it is sufficient that for any possible permutation σ, the set S contains a permutation σ' that induces a tour/path that is not longer than that of σ, and its unreleased portion of the tour at time t is not larger than that of σ. We will say that σ' dominates σ and define this notion formally below. Furthermore, since the released parts only change when a new request is released, it is sufficient to only update the subset then.

▶ **Definition 9** (Dominating Permutation). *Let σ be a permutation of the n requests and t a given time. We define $Dom(\sigma, t)$ to be the set of permutations that dominate permutation σ at time t. A permutation $\sigma' \in Dom(\sigma, t)$ if and only if:*

$$\ell_{\sigma'} \leq \ell_\sigma \quad \text{and} \quad (1 - \alpha_{\sigma'}(t))\ell_{\sigma'} \leq (1 - \alpha_\sigma(t))\ell_\sigma \ .$$

We also say that σ' is a corresponding dominating permutation of σ (at time t).

Algorithm 1 Strategically Wait And Go (SWAG).

Input: Offline: request locations $x_1, \ldots, x_n$
Online: release times $t_1, \ldots, t_n$
Parameter: an oracle $\mathcal{D}$ which outputs a set of permutations on requests

Call the oracle $\mathcal{D}$ to get an initial set $S(0)$ of permutations at $t = 0$. Set $S = S(0)$.
while *true* **do**
  At each release time t_i, request a new set $S(t_i)$ of permutations and update $S = S(t_i)$. For every $\sigma \in S$, compute ℓ_σ and $\alpha_\sigma(t_i)$.
  If $\exists\, \sigma_0 \in S$ s.t. (1) $t \geq \ell_{\sigma_0}/2$ and (2) $\alpha_{\sigma_0}(t) \geq 1/2$, set $T = t$ and **break**.
end
At time T:
- Compute an order σ_1 which minimizes, over all orders $\sigma' \in S$, $(1 - \beta_{\sigma'})\ell_{\sigma'}$, where $\beta_{\sigma'} = \min\{\alpha_{\sigma'}(T), 1/2\}$.
- Follow the tour/path associated to σ_1. Serve the requests in this order, waiting at a request location if this request is not released.

We show that SWAG with an oracle $\mathcal{D}$ is 3/2-consistent if $\mathcal{D}$ is a *domination oracle*.

▶ **Definition 10** (Domination Oracle). *An oracle $\mathcal{D}$ which outputs at time t a set $S(t)$ of permutations is a domination oracle if*

1. *$S(t) \subseteq S(t')$ for every $t \leq t'$, and*
2. *for all t there exists a permutation $\sigma' \in S(t)$ such that $\sigma' \in Dom(\sigma_{\mathrm{OPT}}, t)$. Here, σ_{OPT} is the permutation corresponding to the serving order of requests in an optimal solution.*

▶ **Lemma 11.** *SWAG is 3/2-consistent for both closed and open variants of OLTSP with perfect predictions if it uses a domination oracle.*

Note that this matches the lower bounds in [24, 14], making the result tight in almost all cases. Regarding the computational complexity of SWAG, we have the following lemma.

▶ **Lemma 12.** *If N is the maximum number of permutations which the oracle $\mathcal{D}$ outputs at each time t and $T_{\mathcal{D}}$ is the total time required (by the oracle) to compute all permutations, then the running time of SWAG is $O(\max\{n^2 \cdot N, T_{\mathcal{D}}\})$.*

4 Performance guarantees for LA-OLTSP: the LA-SWAG algorithm

In this section, we deal with imperfect predictions; more specifically, we show how to adapt SWAG to also get smoothness and robustness upper bounds.

We note that if the predictions are perfect, then LA-SWAG is at least as good as SWAG (it works the same, the only difference being that it optimally serves the remaining unserved requests when everything is released). So in particular it is 3/2-consistent as well, provided that the oracle satisfies the conditions of Lemma 11, which is tight in almost all cases.

We also note that the algorithm could serve the requests in a more clever way: instead of going first to the predicted location of a request and then to its true location, the algorithm could go to the true location directly if it is released (or as soon as it is). However, no gain can be obtained; the tightness of analysis given in the full version of the paper would still hold.

Regarding the running time of LA-SWAG, it can be shown that if T_{TSP} is the time required for the computation of an optimal path serving a subset of the requests, then from Lemma 12 we get the following corollary. Note that, for the cases we consider, we achieve suitable upper bounds on T_{TSP}. We describe how this is done in the full version of the paper.

Algorithm 2 Learning-Augmented Strategically Wait And Go (LA-SWAG).

Input: Offline: predicted request locations $p_1, \ldots, p_n$
Online: release times $t_1, \ldots, t_n$, true request locations $x_1, \ldots, x_n$
Parameter: an oracle $\mathcal{D}$ which outputs a set of permutations on requests

(Breaking rule) At any time t: if all requests are released, follow an optimal path serving all unserved requests (returning to $\mathcal{O}$ if in the closed variant) and **break**.

Run SWAG until the starting time T of the server, and the computation of σ_1.

At time T, follow the tour/path σ_1, serving the requests in the following order:
for $i = 1, \ldots, n$ **do**
- first go to $p_{\sigma_1[i]}$; if $q_{\sigma_1[i]}$ is not released, wait there until it is released;
- then, go to $x_{\sigma_1[i]}$ and serve the request.

end
In the closed version, go back to $\mathcal{O}$.

▶ **Corollary 13.** *If N is the maximum number of permutations which the oracle $\mathcal{D}$ outputs at each time t, $T_{\mathcal{D}}$ is the total time required (by the oracle) to compute all permutations and T_{TSP} is the time for the computation of an optimal path that serves a subset of the requests, then the running time of LA-SWAG is $O(\max\{n^2 \cdot N, T_{\mathcal{D}}, T_{TSP}\})$.*

4.1 Smoothness

In this subsection, we show that LA-SWAG is smooth with respect to the measure of error η. To show the main result, we relate the performance of LA-SWAG on a hypothetical instance using the predictions as requests. As noted above, LA-SWAG inherits its 3/2-consistency from SWAG. Leveraging triangle inequality, we further get upper and lower bounds on the performance of LA-SWAG in terms of the performance of LA-SWAG and OPT on the actual instance, respectively. This allows us to determine their maximum ratio.

▶ **Theorem 1** (Consistency and Smoothness). *LA-SWAG has a competitive ratio of at most $3/2 + 5\eta$ for both closed and open LA-OLTSP.*

Giving a simple example with 2 requests, we further show that a linear dependency on η is necessary for the open variant for any algorithm.

▶ **Proposition 14** (Smoothness Lower Bound). *No algorithm can have a better competitive ratio than $\left(\frac{3}{2} + \frac{\eta}{2}\right)$ for the open variant on an instance with prediction error $\eta \in [0, 1/3]$.*

The example is built symmetrically, such that the adversary will be able to react to any choice made by an algorithm, thereby forcing it to take a detour. At the same time, OPT is able to quickly finish its route without having to wait or turn back.

4.2 Robustness

LA-SWAG can easily be shown to be 3-robust, as it can always go back to the origin when all requests are released and then follow an optimal tour. Since the distance to the origin is at most $|\,\mathrm{OPT}\,|$ when all requests are released, $3|\,\mathrm{OPT}\,|$ is an immediate upper bound. We first show improved upper bounds on the robustness of LA-SWAG, and then complement this analysis with some lower bounds.

The first step in the analysis is to assume LA-SWAG waits for some portion of $|\,\mathrm{OPT}\,|$ in the origin, say c. Then, a $(3-c)$ robustness follows. Otherwise, by construction of LA-SWAG, we get a suitable bound on the distances of predictions from the origin, depending on c.

For the following, we focus on the case where LA-SWAG leaves early. In the closed variant, we observe that the furthest away from the origin LA-SWAG can be when the last request is released is $3/4|\,\mathrm{OPT}\,|$, thereby improving the robustness bound to 2.75. This stems from the fact that the furthest prediction and the furthest request each cannot have a distance more than $1/2|\,\mathrm{OPT}\,|$ (corresponding to $c = 1/4$) from the origin as we consider the closed version, but LA-SWAG could be in between the two locations.

For trees or in a Euclidean space, the same argument can be used, but here it holds that any point in between the two locations cannot be further away from the origin than either of the two locations, which gives us overall a 2.5-robustness in these cases.

▶ **Theorem 2** (Robustness-Closed). *LA-SWAG is 2.75-robust for closed LA-OLTSP in general metric spaces, and 2.5-robust in Euclidean spaces and in trees.*

In the open variant, we can use a similar approach. While going back to the origin and following OPT is always a viable option when everything is released, we can also go the the final request OPT visited and then take OPT backwards to serve all requests. It turns out that these two choices are sufficient to improve the robustness below 3. We again consider general metrics first and show an improvement to $3 - 1/6$, before we again study the case of trees to show that a bigger improvement to $3 - 1/3$ is possible in this scenario.

▶ **Theorem 3** (Robustness-Open). *LA-SWAG is $(3 - 1/6)$-robust for open LA-OLTSP in general metric spaces, and $(3 - 1/3)$-robust in trees.*

We show in the full version that the analyses for 2.5-robustness in the closed variant and $(3 - 1/3)$-robustness in the open variant are tight. Moreover, we show the following consistency-robustness tradeoff for any algorithm in the open variant. For a good consistency result, the algorithm has to trust the predictions more, which will lead to a bad robustness.

▶ **Proposition 15.** *Let* ALG *be an algorithm for the open variant with consistency guarantee $2 - \lambda$, where $\lambda \in [0, 1]$. Then,* ALG *cannot be $(2 + \lambda - \epsilon)$-robust, for any $\epsilon > 0$.*

5 Domination oracles

In this section, we implement a domination oracle for general metrics using a Dynamic Programming inspired technique and then we exploit structural insights about specific metrics to obtain more efficient oracles. We focus on the closed variant; we explain in the full version how to deal with the open variant. In the following, $R(t)$ and $U(t)$ will refer to the released and unreleased requests at time t. We may drop the dependency on t.

5.1 General metrics

In the case of perfect predictions, using SWAG and Lemma 11 we can reduce the factorial running time of the algorithm in [14] for general metrics. We will show that an exponential number of permutations suffices to have a domination oracle. After that, using the scheme of LA-SWAG and the theorems proved in the previous section we will get an exponential-time algorithm that is 3/2-consistent, smooth and robust.

Consider an arbitrary permutation σ of the n requests at time t. A corresponding dominating tour for this permutation can be constructed in the following way; find the first unreleased request $u \in U(t)$ in the permutation σ. The released requests before u in

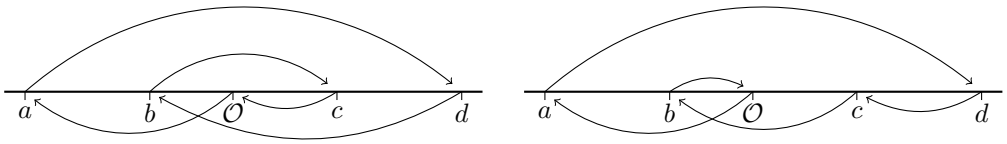

Figure 1 Two permutations illustrating the notion of domination as in Example 16. Left side: OPT, right side: σ'.

the permutation form a subset $R'(t) \subseteq R(t)$ of released requests. Then, a corresponding general dominating tour would be the optimal TSP path that starts from the origin, serves all requests in $R'(t)$ and ends at request u plus the optimal TSP path that starts from u, serves all requests in $\big(R(t) \setminus R'(t)\big) \cup \big(U(t) \setminus \{u\}\big)$ and returns to the origin. We will call these tours (corresponding) *general dominating tours*. It can be shown that at time t a corresponding general dominating tour σ' of a permutation σ dominates σ. In Example 16, we illustrate the notion of a dominating tour.

▶ **Example 16.** Consider the following example on the line in the closed variant. There are 4 requests a, b, c, d, located at $-10, -4, 5, 11$, respectively. Assume the release times to be $t_a = 10, t_b = 46, t_c = 55, t_d = 31$. The optimal tour OPT is $\mathcal{O} \to a \to d \to b \to c \to \mathcal{O}$ with $|\operatorname{OPT}| = 60$, never having to wait and reaching requests just as they are released. Thus, we also have $\ell_{\operatorname{OPT}} = 60$.

Let us focus on the setting in which only the first request has been released, i.e., for $t \in [10, 31)$. At this time, we have $R(t) = \{a\}$ and $U(t) = \{b, c, d\}$. We can see that d is the first unreleased request in permutation OPT at this time, so $u = d$, and $R' = R$. The first part of the general dominating permutation σ' of OPT is thus $\mathcal{O} \to a \to d$, and the second part $d \to c \to b \to \mathcal{O}$. This means that $\ell_{\sigma'} = 50 < 60 = \ell_\sigma$. An illustration of OPT and σ' is given in Figure 1.

When we run `SWAG`, we can see why this notion of domination makes sense: Clearly, the released lengths of OPT and σ' are 31 for $t \in [10, 31)$ (for both permutations, the server can already serve a and continue to d without having to wait anywhere – but d itself cannot yet be served). This also means that $\alpha_{\operatorname{OPT}} \geq 1/2$ and $\alpha_{\sigma'} \geq 1/2$. Thus, both of the permutations would start the second part of the algorithm when t would be large enough. Since $\ell_{\sigma'} \leq \ell_\sigma$, it suffices to consider the permutation σ' instead of σ, as regardless of the release time of a, no earlier starting time could be computed when using OPT than when using σ'. Additionally, $\beta_{\sigma'}(t) = \beta_{\operatorname{OPT}}(t) = 1/2$ and thus OPT would not be chosen as permutation to follow in the second part of the algorithm.

Note that, if a, b, and d are released but c is not at some time t', then OPT would not be dominated at t' by σ'. This is because the unreleased part of σ' would be greater than that of OPT. Clearly, OPT would be dominated at time t' by another permutation, e.g., $\mathcal{O} \to a \to b \to d \to c \to \mathcal{O}$. ◀

If we assume that $t_1 \leq t_2 \leq \cdots \leq t_n$ are the release times of the requests, $t_0 = 0$ and $t_{n+1} = \infty$, then at time t_i, with $i \leq n$, we compute the set of general dominating permutations D_i given the sets $R(t_i)$ and $U(t_i)$. Together with the previously computed sets, it is then provided to `SWAG` as set $S(t_i) = \bigcup_{j=0}^{i} D_j$. It is easy to show that an oracle $\mathcal{D}$ which outputs at any time $t \in [t_i, t_{i+1})$ the general dominating sets $S(t) = \bigcup_{j=0}^{i} D_j$ for each $i \in \{0, 1, 2, \ldots, n\}$, is a domination oracle and `SWAG` is 3/2-consistent (Lemma 11). Condition 1 of the domination oracle is satisfied as $S(t) \subseteq S(t')$ for every $t \leq t'$. Condition 2 is also satisfied since for each time t we have a corresponding dominating permutation for every possible subset of $R(t)$ and choice of u, and thus for every permutation σ.

Concerning the running time of the algorithm, we begin by bounding the number of general dominating permutations which are contained in the set $\bigcup_{j=0}^{i} D_j$ for all $i \leq n$. Formally, we prove the following lemma.

▶ **Lemma 17.** *There always exists a set $\bigcup_{j=0}^{i} D_j$ which consists of at most 2^n distinct general dominating permutations, for all $i \in \{0, 1, \ldots, n\}$.*

In order to get an LA algorithm, we run `LA-SWAG` with the oracle of general dominating permutations. Since all possible general dominating permutations can be precomputed at $t = 0$ using the dynamic programming algorithm for classical TSP in time $O(n^2 \cdot 2^n)$, we use Corollary 13 and Lemma 17 to show the following result for the running time of `LA-SWAG`.

▶ **Lemma 18.** *`LA-SWAG` that uses the general dominating sets runs in $O(n^2 \cdot 2^n)$ time complexity for both closed and open variants of OLTSP.*

Using theorems 1, 2, and 3 we show that `LA-SWAG` with the general dominating permutations is a single-exponential time algorithm with the following performance guarantees.

▶ **Theorem 4** (General Metrics). *`LA-SWAG` with an oracle $\mathcal{D}$ which uses the general dominating sets runs in single-exponential time and is $\min\{3/2 + 5\eta, 2.75\}$-competitive for the closed variant and $\min\{3/2 + 5\eta, 3 - 1/6\}$-competitive for the open variant of LA-OLTSP.*

5.2 Specific metrics

In the following we explain how to implement domination oracles for trees, rings, and flowers. These will dominate a so-called *sensible* set of permutations, which we define later.

We also need to implement the "cleanup" step of `LA-SWAG`, where a computation of an optimal classical (offline, without release times) TSP path/tour is required. In fact, these subroutines are also employed within the domination oracles themselves. We describe how this is done efficiently (in $O(n)$ time) in the full version of the paper.

▶ **Definition 19** (Safe Set). *Let X be a set of request locations. A set of permutations Π is safe for X if for any assignment f of release times to each location in X, there exists a permutation $\pi \in \Pi$ which is optimal for the resulting input $Q = \{(x, f(x)) \mid x \in X\}$.*

Note that the set of all permutations is safe. However, we can define *sensible* sets of permutations that follow some desirable structure but are still safe. The guarantees of `LA-SWAG` follow even when dominating a sensible set.

In general, the idea is to ensure domination as follows. Let Q_1 be the requests served by a permutation π before its current request q. We construct a permutation π_{dom}, which serves a superset of Q_1 before q and is not longer than π on both parts, before and after q.

Trees

We consider metric spaces that are shaped like a tree. By that we mean continuous spaces that can be embedded into trees with edges weighted by their lengths. The requests may appear on nodes of the tree or at some point along the edges.

Suppose that an algorithm serves a request q at time t. There is only one path $\mathcal{P}_q$ from q to $\mathcal{O}$, and it must be traversed after t. Hence, any request q' on $\mathcal{P}_q$ will be visited at some point after t anyway, rendering the earlier serving of such requests useless. From this, it follows that any sequence of requests S_q encountered along a path $\mathcal{P}_q$ towards the origin may be safely assumed to be a subsequence of the optimal permutation.

▶ **Definition 20** (Sensible Set for Trees). *Let $\mathcal{T}$ be a tree rooted at the origin $\mathcal{O}$ and X a set of request locations on $\mathcal{T}$. The set $\Pi_s(\mathcal{T}, X)$ of sensible permutations consists of all permutations π where the following holds. Let $\pi = x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)}$. For any $x_{\pi(i)}$ and any $x_{\pi(j)}$ on the path $\mathcal{P}_{x_{\pi(i)}}$ from $x_{\pi(i)}$ to the origin, we have $j > i$.*

▷ Claim 21. For any tree $\mathcal{T}$ and any set of request locations X, the set $\Pi_s(\mathcal{T}, X)$ is safe.

Dominating set

For the above sensible set, we will define a dominating set whose cardinality is of order $O(2^l \cdot n)$, where l is the number of leaves of the underlying tree. The idea is to look at subsets of paths of leaves to the origin. That is, for each such subset and each selection of intermediate request q, we consider the permutation which first optimally visits the selected leaves finishing at q and then "cleans up" the rest of the requests starting from q and ending at the origin. The set of these permutations for all different choices of leaf subsets and current request is called $Dom^{\curlyvee}(R, U, \mathcal{T}, L)$, where $\mathcal{T}$ is the input tree and L is its set of leaves.

▶ **Lemma 22** (Tree Domination). *Let π be a sensible permutation for a tree $\mathcal{T}$ and L be its set of leaves. Then, $\exists\, \pi_{dom} \in Dom^{\curlyvee}(R, U, \mathcal{T}, L)$ dominating π.*

Using the above lemma, we obtain our main theorem for trees.

▶ **Theorem 5** (Trees). *There exists a Domination oracle for* `LA-SWAG` *in trees which yields a time complexity of $O(2^l \cdot n^3)$ for the closed variant and $O(2^l \cdot n^4)$ for the open variant, where l is the number of leaves of the input tree.*

Ring

If a permutation makes more than one "full" loops around the ring, it may as well keep only the last. Moreover, no serving needs to be done before the loop. After such a loop, we can assume that the permutation moves along the ring as if it were a line, i.e., it never crosses the antipodal point of the origin. Hence, we can define the sensible permutations by branching on whether a loop is performed and then reduce to the tree case.

▶ **Definition 23** (Sensible Set for the Ring). *Let X be a set of request locations on the ring. The sensible set of permutations $\Pi_s^{\circ}(X)$ consists of all permutations π resulting from the concatenation of π_{loop} and π_{line} where π_{loop} is a subpermutation covering $X' \subseteq X$ in a cyclic fashion and $\pi_{line} \in \Pi_s(T, X \setminus X')$, where T is the tree resulting from splitting the ring at the midpoint across from the origin.*

▷ Claim 24. For any set of request locations X, the set $\Pi_s^{\circ}(X)$ is safe.

Dominating set

It suffices to define the dominators under the assumption that π does indeed have a loop in the beginning, since we can take the union with the tree dominators.

A crucial distinction is whether the current request q of π is part of the loop or not. In the first case, π has only traveled a portion of the loop so far and in the second case it has traveled a full loop and serves as if on a tree afterwards. For every choice of q, we define two permutations, one pertaining to the first case and another to the second one.

The first permutation serves all released requests within π's traveled portion of the loop and then moves to q. The second permutation serves all released requests with a full loop and then moves to q. Both finish by cleaning up the rest of the requests. We call the resulting set of permutations $Dom^{\circ}(R, U)$.

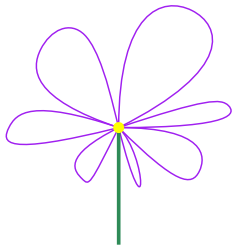

Figure 2 A flower with 7 petals (purple), connected with the stem (green) at the origin (yellow).

▶ **Lemma 25** (Ring Domination). *Let π be a sensible permutation whose tour contains a full loop in the beginning. $Dom^{\circ}(R, U)$ contains a permutation π_{dom} which dominates π.*

Thus, our main theorem for the ring follows.

▶ **Theorem 6** (Ring). *There exists a Domination oracle for* `LA-SWAG` *in the ring which yields a time complexity of $O(n^3)$ for the closed variant and $O(n^5)$ for the open variant.*

Flowers

We extend the reasoning used for the ring and trees to the so-called flowers. A flower consists of a number of rings (petals), all of which are attached to the origin point (receptacle). For reasons of artistic completion, a semi-line (stem) disjoint from the petals is also attached to the origin point. An illustration is given in Figure 2.

Sensible set

We can assume that any sensible permutation traverses a petal of the flower in a cyclic fashion at most once, for the same reason that this is true for the ring. Thus, the sensible set here is defined to include the permutations which have at most one loop for each petal, such loops are the first visit on the corresponding petal and the restrictions of the sensible set on trees also apply to every petal after such a loop is carried out.

Dominating set

First, choose the current request q of the permutation π to be dominated. Then, we guess the subset $\mathcal{P}$ of petals "looped" by π. We also guess the petals $\mathcal{P}_{done} \subseteq \mathcal{P}$ already looped before q. For a set of such choices, we traverse all petals in $\mathcal{P}_{done}$ in an arbitrary order. Next, we "snip" every petal except the ones in $\mathcal{P}$, yielding a tree $\mathcal{T}$ (actually, a star) and the petals in $\mathcal{P}$.

Now, we also guess the subset L' of the leaves L of $\mathcal{T}$ to be visited before q. Having fixed that, we append to our permutation the optimal path from $\mathcal{O}$ to q that visits every leaf in L' and the possible arc of π to q. Finally, we append the optimal path from q that "cleans up" the remaining unserved requests. The resulting set is called $Dom^{✿}(R, U)$.

▶ **Lemma 26** (Flower Domination). *Let π be a sensible permutation for the flower. The set $Dom^{✿}(R, U)$ contains a permutation π_{dom} which dominates π.*

Note that to make all these different guesses, one has at most 6 choices for every petal. So, if the graph has p petals, the cardinality of the dominators is $O(6^p \cdot n) = O(2^{2.59p} \cdot n)$. The below complexity comes from the term related to the cardinality of the dominating set.

▶ **Theorem 7** (Flowers). *There exists a Domination oracle for* `LA-SWAG` *in flowers which yields a time complexity of* $O(6^p \cdot n^3)$ *for the closed variant and* $O(6^p \cdot n^5)$ *for the open variant, where* p *is the number of petals of the input flower.*

6 Conclusion

We studied Online TSP augmented with predictions regarding the locations of the requests. Our algorithm, `LA-SWAG`, achieves a competitive ratio of 3/2 under the assumption of perfect predictions, which is tight in most cases we considered. Additionally, it is smooth and provides robustness guarantees below 3, improving over previous work. The runtime of `LA-SWAG` is single-exponential; however, we show how to remove the exponential dependency on the number of requests for specific metric spaces.

We believe that our techniques can be generalized to obtain FPT algorithms for other classes of graphs also; cactus graphs, graphs of bounded treewidth in general, as well as planar graphs are interesting options. Additionally, extending the algorithm to the Dial-A-Ride Problem seems like a reasonable direction to follow.

Another interesting direction is to leverage the ideas behind the PTAS for classical TSP on the Euclidean plane [6, 7] (or any other space which admits an approximation scheme for that matter) to obtain consistency guarantees which approach 3/2 arbitrarily close as the computational time is allowed to increase. Note that, since our smoothness proof follows from consistency itself and robustness from solving a classical TSP instance, these results would automatically extend, modulo some worsening of the bounds as a function of the approximation quality. For this, one possible approach is to extend the notion of a sensible set to that of ϵ-sensible, meaning that a $(1+\epsilon)$-approximation of the optimal Online TSP solution is guaranteed to exist within the set. A similar relaxation would also make sense for the idea of dominating sets.

References

1. Luca Allulli, Giorgio Ausiello, and Luigi Laura. On the power of lookahead in on-line vehicle routing problems. In *COCOON*, 2005.
2. Spyros Angelopoulos, Christoph Dürr, Shendan Jin, Shahin Kamali, and Marc P. Renault. Online computation with untrusted advice. In *ITCS*, 2020.
3. Antonios Antoniadis, Christian Coester, Marek Eliás, Adam Polak, and Bertrand Simon. Online metric algorithms with untrusted predictions. In *ICML*, 2020.
4. Antonios Antoniadis, Peyman Jabbarzade Ganje, and Golnoosh Shahkarami. A novel prediction setup for online speed-scaling. *CoRR abs/2112.03082*, 2021. `arXiv:2112.03082`.
5. Antonios Antoniadis, Themis Gouleakis, Pieter Kleer, and Pavel Kolev. Secretary and online matching problems with machine learned advice. In *NeurIPS*, 2020.
6. Sanjeev Arora. Polynomial time approximation schemes for euclidean TSP and other geometric problems. In *FOCS*, 1996.
7. Sanjeev Arora. Nearly linear time approximation schemes for euclidean TSP and other geometric problems. In *FOCS*, 1997.
8. Norbert Ascheuer, Martin Grötschel, Sven O. Krumke, and Jörg Rambau. Combinatorial online optimization. In Peter Kall and Hans-Jakob Lüthi, editors, *Operations Research Proceedings 1998*, pages 21–37, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.
9. Giorgio Ausiello, Marc Demange, Luigi Laura, and Vangelis Paschos. Algorithms for the on-line quota traveling salesman problem. *Inf. Process. Lett.*, 92(2):89–94, 2004.
10. Giorgio Ausiello, Esteban Feuerstein, Stefano Leonardi, Leen Stougie, and Maurizio Talamo. Algorithms for the on-line travelling salesman. *Algorithmica*, 29:560–581, 2001.

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

12 Étienne Bamas, Andreas Maggiori, and Ola Svensson. The primal-dual method for learning augmented algorithms. In *NeurIPS*, 2020.

13 Evripidis Bampis, Bruno Escoffier, Themis Gouleakis, Niklas Hahn, Kostas Lakis, Golnoosh Shahkarami, and Michalis Xefteris. Learning-augmented online TSP on rings, trees, flowers and (almost) everywhere else. *CoRR*, abs/2305.02169, 2023. `arXiv:2305.02169`.

14 Evripidis Bampis, Bruno Escoffier, Niklas Hahn, and Michalis Xefteris. Online TSP with Known Locations. *To appear in WADS*, 2023. CoRR abs/2210.14722.

15 Evripidis Bampis, Bruno Escoffier, and Michalis Xefteris. Canadian traveller problem with predictions. In Parinya Chalermsook and Bundit Laekhanukit, editors, *Approximation and Online Algorithms*, pages 116–133, Cham, 2022. Springer International Publishing.

16 Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *J. ACM*, 9(1):61–63, January 1962.

17 Giulia Bernardini, Alexander Lindermayr, Alberto Marchetti-Spaccamela, Nicole Megow, Leen Stougie, and Michelle Sweering. A universal error measure for input predictions applied to online graph problems. In *NeurIPS*, 2022.

18 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.

19 Michiel Blom, Sven O. Krumke, Willem E. De Paepe, and Leen Stougie. The online TSP against fair adversaries. *INFORMS J. Comput.*, 13(2):138–148, 2001.

20 Shuchi Chawla and Dimitris Christou. Online time-windows TSP with predictions. *CoRR*, abs/2304.01958, 2023. `arXiv:2304.01958`.

21 Paul Dütting, Silvio Lattanzi, Renato Paes Leme, and Sergei Vassilvitskii. Secretaries with advice. In *EC*, 2021.

22 Franziska Eberle, Alexander Lindermayr, Nicole Megow, Lukas Nölke, and Jens Schlöter. Robustification of online graph exploration methods. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(9):9732–9740, June 2022.

23 Sreenivas Gollapudi and Debmalya Panigrahi. Online algorithms for rent-or-buy with expert advice. In *ICML*, 2019.

24 Themistoklis Gouleakis, Konstantinos Lakis, and Golnoosh Shahkarami. Learning-Augmented Algorithms for Online TSP on the Line. In *AAAI*, 2023.

25 Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. *J. Soc. Indust. Appl. Math.*, 10(1):196–210, 1962.

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

27 Patrick Jaillet and Michael R. Wagner. Online routing problems: Value of advanced information as improved competitive ratios. *Transportation Science*, 40:200–210, May 2006.

28 Vinay A. Jawgal, V. N. Muralidhara, and P. S. Srinivasan. Online travelling salesman problem on a circle. In T.V. Gopal and Junzo Watada, editors, *Theory and Applications of Models of Computation*, pages 325–336, Cham, 2019. Springer International Publishing.

29 Murali Kodialam and T. V. Lakshman. Prediction augmented segment routing. In *2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR)*, pages 1–6, 2021.

30 Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *SIGMOD*, 2018.

31 Eugene L. Lawler, Jan K. Lenstra, Alexander H. G. Rinnooy Kan, and David B. Shmoys. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Wiley Series in Discrete Mathematics & Optimization, 1991.

32 Alexander Lindermayr and Nicole Megow. Alps. `https://algorithms-with-predictions.github.io/`.

33 Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. *J. ACM*, 68(4):24:1–24:25, 2021. doi:10.1145/3447579.

34 Michael Mitzenmacher. A model for learned bloom filters and optimizing by sandwiching. In *NeurIPS*, 2018.

35 Michael Mitzenmacher. Scheduling with predictions and the price of misprediction. In *ITCS*, 2020.

36 Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. In *Beyond the Worst-Case Analysis of Algorithms*. Cambridge University Press, 2020.

37 Benjamin Moseley, Sergei Vassilvitskii, Silvio Lattanzi, and Thomas Lavastida. Online scheduling via learned weights. In *SODA*, 2020.

38 Harilaos N. Psaraftis, Marius M. Solomon, Thomas L. Magnanti, and Tai-Up Kim. Routing and scheduling on a shoreline with release times. *Manag. Sci.*, 36(2):212–223, 1990.

39 Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. In *NeurIPS*, 2018.

40 Dhruv Rohatgi. Near-optimal bounds for online caching with machine learned advice. In *SODA*, 2020.

41 Shufan Wang and Jian Li. Online algorithms for multi-shop ski rental with machine learned predictions. In *AAMAS*, 2020.

42 Alexander Wei. Better and simpler learning-augmented online caching. In *APPROX/RANDOM*, 2020.