Robustification of Online Graph Exploration Methods

Franziska Eberle,[1] Alexander Lindermayr,[2] Nicole Megow,[2] Lukas Nölke,[2] Jens Schlöter[2]

[1] Department of Mathematics, London School of Economics
[2] Faculty of Mathematics and Computer Science, University of Bremen
f.eberle@lse.ac.uk, {linderal,nmegow,noelke,jschloet}@uni-bremen.de

Abstract

Exploring unknown environments is a fundamental task in many domains, e.g., robot navigation, network security, and internet search. We initiate the study of a learning-augmented variant of the classical, notoriously hard online graph exploration problem by adding access to machine-learned predictions. We propose an algorithm that naturally integrates predictions into the well-known Nearest Neighbor (NN) algorithm and significantly outperforms any known online algorithm if the prediction is of high accuracy while maintaining good guarantees when the prediction is of poor quality. We provide theoretical worst-case bounds that gracefully degrade with the prediction error, and we complement them by computational experiments that confirm our results. Further, we extend our concept to a general framework to *robustify* algorithms. By interpolating carefully between a given algorithm and NN, we prove new performance bounds that leverage the individual good performance on particular inputs while establishing robustness to arbitrary inputs.

Introduction

In online mapping problems, a searcher is tasked to explore an unknown environment and create a complete map of its topology. However, the searcher has only access to local information, e.g., via optical sensors, and must move through the environment to obtain new data. Such problems emerge in countless real-life scenarios with a prominent example being the navigation of mobile robots, be it a search-and-rescue robot, an autonomous vacuum cleaner, or a scientific exploration robot in the deep sea or on Mars. Less obvious but equally important applications include crawling the Internet or social networks for information and maintaining security of large networks (Berman 1996; Rao et al. 1986; Gasieniec and Radzik 2008).

We investigate the online graph exploration problem on an undirected connected graph $G = (V, E)$ with n vertices. Every edge $e \in E$ has non-negative cost $c(e)$, and every vertex $v \in V$ has a unique label. Starting in a designated vertex $s \in V$, the task of the searcher is to find a tour that visits all vertices of G and returns to s. A priori, the searcher does not know the graph. Instead, she gains local information when traversing it: When for the first time visiting (*exploring*) a vertex, all incident edges, as well as their costs, and the labels of their end points are revealed. This exploration model is also known as *fixed graph scenario* (Kalyanasundaram and Pruhs 1994). In order to explore a vertex v, the searcher may traverse a path consisting of known edges that starts in the searcher's current location and ends in v. When the searcher traverses an edge e, she pays its cost $c(e)$. The goal is to minimize the total cost for exploring all vertices.

Due to the lack of information, the searcher cannot expect to find an optimal tour. We resort to standard *competitive analysis* to measure the quality of our search algorithms. That is, we compare the length of the tour found by the searcher with the optimal tour that can be found if the graph is known in advance. If the ratio between the costs of these two tours is bounded by $\rho \geq 1$ for every instance, then we say that the online algorithm is *ρ-competitive*. The *competitive ratio* of an algorithm is the minimum ρ for which it is ρ-competitive. The *offline* problem of finding an optimal tour on a known graph is the well-known Traveling Salesperson Problem (TSP), which is NP-hard (Lawler et al. 1985).

Indeed, it appears extremely difficult to obtain solutions of cost within a constant factor of an optimal tour. The best known competitive ratio for arbitrary graphs is $\mathcal{O}(\log n)$, attained by the following two algorithms. The Nearest Neighbor algorithm (NN) (Rosenkrantz, Stearns, and Lewis 1977) greedily explores the unknown vertex closest to the current position. While its performance is usually good in practice (Applegate et al. 2006), a matching lower bound of $\Omega(\log n)$ holds even for very simple graphs, e.g., unweighted graphs (Hurkens and Woeginger 2004) or trees (Fritsch 2021). The second algorithm is the hierarchical Depth First Search algorithm (HDFS) (Megow, Mehlhorn, and Schweitzer 2012) that, roughly speaking, executes depth-first searches (DFS) on subgraphs with low edge costs, thereby limiting its traversal to a minimum spanning tree (MST). Here, a matching lower bound is attained on a weighted path.

Only for rather special graph classes it is known how to obtain constant-competitive tours. Notable examples are planar graphs, with a competitive ratio of 16 (Kalyanasundaram and Pruhs 1994; Megow, Mehlhorn, and Schweitzer 2012), graphs of bounded genus g with a ratio of $16(1 + 2g)$ and graphs with k distinct weights with a ratio of $2k$ (Megow, Mehlhorn, and Schweitzer 2012). The latter emerges as somewhat of an exception since the HDFS algorithm achieves a

Copyright © 2022, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

performance that is both, good on a specific yet interesting graph class and still acceptable on arbitrary instances. Beyond the above, results are limited to the most basic kind of graphs, such as unweighted graphs (Miyazaki, Morimoto, and Okabe 2009), cycles and tadpole graphs (Miyazaki, Morimoto, and Okabe 2009; Brandt et al. 2020), and cactus and unicyclic graphs (Fritsch 2021). Conversely, the best known lower bound on the competitive ratio of an online algorithm is $10/3$ (Birx et al. 2021). Despite ongoing efforts, it remains a major open question whether there exists an $O(1)$-competitive exploration algorithm for general graphs.

The assumption of having no prior knowledge about the graph may be overly pessimistic. Given the tremendous success of artificial intelligence, we might have access to predictions about good exploration decisions. Such predictions, e.g., machine-learned ones, are typically imperfect; they usually have a good quality but may be arbitrarily bad.

A new line of research is concerned with the design of online algorithms that have access to predictions of unknown quality (Lykouris and Vassilvitskii 2018; Purohit, Svitkina, and Kumar 2018; Medina and Vassilvitskii 2017). Ideally, algorithms have the following properties: (i) good predictions lead to a better performance than the best worst-case bound achievable when not having access to predictions; (ii) the algorithm never performs (asymptotically) worse than the best worst-case algorithm even if the prediction is of poor quality; and (iii) the performance gracefully degrades with decreasing prediction quality. More formally, we define a parameter $\eta \geq 0$, called *prediction error*, that measures the quality of a given prediction, where $\eta = 0$ refers to the case that the prediction is correct, we also say *perfect*. We assess an algorithm's performance by the competitive ratio as a function of the prediction error. If an algorithm is $\rho(\eta)$-competitive for some function ρ, we call it α-*consistent* for $\alpha = \rho(0)$ and β-*robust* if $\rho(\eta) \leq \beta$ for any prediction error $\eta \geq 0$ (Purohit, Svitkina, and Kumar 2018).

For the online graph exploration problem, we consider predictions that suggest a known, but unexplored vertex as next target to a learning-augmented algorithm. In other words, a *prediction* is a function that, given the current state of the exploration, outputs an explorable vertex. Predictions may be computed dynamically and use all data collected so far, which is what one would expect in practice. This rather abstract requirement allows the implementation of various prediction models. In this paper, we consider two kinds of predictions, namely *tour predictions* and *tree predictions*, where the suggested vertex is the next unexplored vertex of a TSP tour or of a Depth First Search (DFS) tour corresponding to some predicted spanning tree, respectively. The prediction error η is the difference between the total exploration cost of following these per-step suggestions blindly and that of following a perfect prediction w.r.t. the given prediction model (tour respectively tree predictions).

Our results Our contribution is twofold. Firstly, we present a learning-augmented online algorithm for the graph exploration problem that has a constant competitive ratio when the prediction error is small, while being robust to poor-quality predictions. Our algorithm interpolates carefully between the algorithms NN and Follow the Prediction (FP), where the latter blindly follows a given prediction.

Theorem 1. *For any $\lambda > 0$, there is an algorithm for the online graph exploration problem that uses a predicted spanning tree or tour such that the algorithm is $\kappa(3 + 4\lambda)$-consistent and $\left(1 + \frac{1}{2\lambda}\right)(\lceil \log(n) \rceil + 1)$-robust, where $\kappa = 1$, for tour predictions, and $\kappa = 2$, for tree predictions. With growing prediction error, the competitive ratio degrades gracefully with linear dependence on η.*

The parameter λ can steer the algorithm towards one of the underlying algorithms, e.g., towards NN when $\lambda \to \infty$. It reflects our trust in the quality of the provided predictions.

Further, we show that our predictions (tour and tree) are learnable in the sense of PAC learnability (Valiant 1984; Vapnik and Chervonenkis 2013) under the assumptions that the given graph is complete and its size known. We show a bound on the sample complexity that is polynomial in the number of nodes and give learning algorithms with a polynomial running time in the case of tree predictions and an exponential running time for tour predictions. The learnability results also approximately bound the expected prediction error η, which potentially can be taken into account when setting λ.

Our second main result is a general framework to *robustify* algorithms. Given an online algorithm $\mathcal{A}$ with a certain worst-case performance for particular classes of instances but unknown, possibly unbounded, performance in general, the robustification framework produces an algorithm with the same good performance on special instances while guaranteeing the best-known worst-case performance $\mathcal{O}(\log n)$ on general instances. As it turns out, the idea of interpolating between two algorithms that we used to design a learning-augmented algorithm can be generalized to interpolating between the actions of an arbitrary algorithm $\mathcal{A}$ and NN.

Theorem 2. *For any $\lambda > 0$, there is a robustification framework $\mathcal{R}$ for the online graph exploration problem that, given an online algorithm $\mathcal{A}$ and an instance $\mathcal{I} = (G, s)$, produces a solution of cost at most $R_{\mathcal{I}} = \min\{(3 + 4\lambda) \cdot \mathcal{A}_{\mathcal{I}},\ \left(1 + \frac{1}{2\lambda}\right)(\lceil \log(n) \rceil + 1) \cdot \mathrm{OPT}_{\mathcal{I}}\}$, where $\mathrm{OPT}_{\mathcal{I}}$ and $\mathcal{A}_{\mathcal{I}}$ denote the cost of an optimal solution and of the one obtained by $\mathcal{A}$ on instance $\mathcal{I}$, respectively.*

This seems useful in situations where one may suspect that an instance is of a certain type for which there exist good algorithms. One would like to use a tailored algorithm without sacrificing the general upper bound and good average-case performance of NN in case the suspicion is wrong. Two illustrative examples are as follows. *(i) Planar graphs:* Many spatial networks, e.g., urban street networks, can often be assumed to be (almost) planar (Barthelemy 2018; Boeing 2020). Here, the graph exploration algorithm BLOCKING (Kalyanasundaram and Pruhs 1994; Megow, Mehlhorn, and Schweitzer 2012) seems the best choice, given its competitive ratio of 16. Yet, on general instances, the competitive ratio may be unbounded and is known to be worse than $\omega(\log n)$ (Megow, Mehlhorn, and Schweitzer 2012), underlining the need for robustification. *(ii) Bounded number of weights:* Here, HDFS (Megow, Mehlhorn, and Schweitzer 2012) is the logical choice with a competitive ratio propor-

tional to the number of weights and an asymptotically best-known competitive ratio on general instances. Even here, robustification is useful as it provides the good average-case performance of NN and the slightly better competitive ratio for general instances.

Interestingly, when considering the surprisingly good average-case performance of NN in practice, our robustification framework may also be interpreted to be robustifying NN and not the specifically tailored algorithm. Either algorithm can possibly make up for the other's shortcomings.

Our robustification scheme is conceptually in line with other works combining algorithms with different performance characteristics (Mahdian, Nazerzadeh, and Saberi 2012; Fiat et al. 1991; Blum and Burch 2000; Azar, Broder, and Manasse 1993). However, it is nontrivial to implement such concept for online graph exploration with the particular way in which information is revealed. Since the graph is revealed depending on an algorithm's decisions, the key difficulty lies in handling the cost of different algorithms in different metrics. This fact also prohibits the application of previous learning-augmented algorithms, e.g., for metrical task systems, in our setting.

We complement our theoretical results by empirically evaluating the performance of our algorithms on several real-world instances as well as artificially generated instances. The results confirm the power of using predictions and the effectivity of the robustification framework.

Further related work The recent introduction of learning-augmented online algorithms (Lykouris and Vassilvitskii 2018; Medina and Vassilvitskii 2017; Purohit, Svitkina, and Kumar 2018) spawned a multitude of exciting works. These provide methods and concepts for a flurry of problems including, e.g., rent-or-buy problems (Purohit, Svitkina, and Kumar 2018; Gollapudi and Panigrahi 2019), scheduling/queuing and bin packing (Purohit, Svitkina, and Kumar 2018; Lattanzi et al. 2020; Mitzenmacher 2020; Angelopoulos et al. 2020; Bamas et al. 2020; Azar, Leonardi, and Touitou 2021; Im et al. 2021), caching (Rohatgi 2020; Lykouris and Vassilvitskii 2018; Antoniadis et al. 2020), the secretary problem (Dütting et al. 2021), revenue optimization (Medina and Vassilvitskii 2017), and matching (Kumar et al. 2019; Lavastida et al. 2021). It is a very active research area. We are not aware of learning-augmented algorithms for online graph exploration.

Several works empirically study the use of machine learning to solve TSP (Khalil et al. 2017; Vinyals, Fortunato, and Jaitly 2015; Bello et al. 2017; Kool, van Hoof, and Welling 2019) without showing theoretical guarantees. For example, Khalil et al. (Khalil et al. 2017) use a combination of reinforcement learning, deep learning, and graph embedding to learn a greedy policy for TSP. As the policy might depend on information that is not accessible online, e.g., the degree of an unexplored vertex, and constructs the tour in an offline manner, the results do not directly transfer to our online setting. However, similar approaches are conceivable and might be an application for the robustification framework, especially since there already exist empirical results in related settings for the exploration of unknown environments. For example, one approach (Luperto and Amigoni 2019) uses constructive machine learning tools to predict unknown indoor environments, another approach (Chiotellis and Cremers 2020) considers online graph exploration as a reinforcement learning problem and solves it using graph neural networks (cf. (Zhou et al. 2020)), and a third approach (Dai et al. 2019) uses reinforcement learning to explore a maximum number of states in an unknown environment using a limited budget. As those approaches do not give theoretical guarantees, they are potential applications for the robustification framework. One work (Elmiger et al. 2020) uses reinforcement learning to find instances with a high competitive ratio for NN.

A recent line of research considers data-driven algorithm design with theoretical performance guarantees, where the task is to select the algorithm with the best expected performance for an unknown distribution over instances from a fixed set of algorithms; see (Balcan 2020) for a survey of recent results. We are not aware of such results for online graph exploration. While our results on PAC learnability have a similar flavor, there are major differences. In contrast to data-driven algorithms, we learn predictions to minimize the error η, which is related to the worst-case guarantees of our algorithms but does not directly bound their expected objective values. Instead, a function depending on the error (cf. Theorem 1) upper bounds the expected objective values. This may be seen as a disadvantage, but also means that our learned predictions are independent of the used algorithm.

Another line of research studies graph exploration *with advice* (Böckenhauer, Fuchs, and Unger 2018; Dobrev, Královic, and Markou 2012; Komm et al. 2015). Here, an algorithm is also equipped with advice that can convey arbitrary information and the goal is to find a competitive solution while using advice of small encoding size. The advice is assumed to be correct which is crucially different from our model.

A General Robustification Scheme

In this section, we introduce the robustification scheme $\mathcal{R}$ from Theorem 2 that, given an algorithm $\mathcal{A}$ for the online graph exploration problem, robustifies its worst-case performance guarantee. In the course of the exploration, the set of vertices known to the searcher can be partitioned into *explored* and *unexplored* vertices, i.e., vertices that have already been visited by the searcher, or not, respectively. The robustification scheme uses the algorithm $\mathcal{A}$ as a blackbox. That is, we treat $\mathcal{A}$ as a function that, given the current position, currently known subgraph, and set of already explored vertices, returns the next vertex to explore. The learning-augmented algorithm from Theorem 1 emerges as an application of the robustification scheme and is discussed in the next section.

The Robustification Scheme

Intuitively, the robustification scheme $\mathcal{R}$, summarized in Algorithm 1, balances the execution of algorithm $\mathcal{A}$ with that of NN by executing the algorithms in alternating phases. These phases are budgeted so that their costs are roughly proportional to each other, with a parameter $\lambda > 0$ dictating the proportionality. Specifically, whenever $\mathcal{A}$ is at position v and about to explore a vertex u via some path $P_u^{\mathcal{A}}$, we interrupt $\mathcal{A}$ and, instead, start from v a phase of exploration via NN. This

Algorithm 1: Robustification scheme $\mathcal{R}$.

Input: Partially explored graph G, start vertex s, algorithm $\mathcal{A}$, and parameter $\lambda > 0$

1 $G_{\mathcal{A}} \leftarrow G$ `// subgraph revealed to` $\mathcal{A}$
2 **while** *G has an unexplored vertex* **do**
3 $\quad u \leftarrow$ next unexplored node to be visited by $\mathcal{A}$, computed via Algorithm 2
4 $\quad P_u^{\mathcal{A}} \leftarrow$ shortest s-u-path in G
5 $\quad u' \leftarrow$ nearest unexplored neighbor of s, $b \leftarrow 0$
6 $\quad$ **while** $b < \lambda \cdot c(P_u^{\mathcal{A}})$ *and* $s \neq u$ **do**
7 $\quad\quad$ traverse a shortest s-u'-path $P_{u'}$ and update G
8 $\quad\quad s \leftarrow u'$, $b \leftarrow b + c(P_{u'})$
9 $\quad\quad u' \leftarrow$ nearest unexplored neighbor of s
10 $\quad$ traverse a shortest known path to u, set $s \leftarrow u$ and update G
11 $\quad$ update $G_{\mathcal{A}}$ to reflect exploration of u
12 traverse a shortest path in G to the start vertex

Algorithm 2: Computes next unexplored node visited by $\mathcal{A}$ and updates $G_{\mathcal{A}}$.

Input: Partially explored graph G, blackbox graph $G_{\mathcal{A}}$, start vertex s and algorithm $\mathcal{A}$

1 **while** *next vertex u explored by $\mathcal{A}$, given $G_{\mathcal{A}}$ and s, is explored in G* **do**
2 $\quad$ update $G_{\mathcal{A}}$ by adding previously unknown edges incident to u, and mark u as explored
3 $\quad s \leftarrow u$
4 **return** u, $G_{\mathcal{A}}$

phase ends when the cost incurred by NN reaches $\lambda c(P_u^{\mathcal{A}})$ or when NN is about to explore u (Lines 6 to 9). Only afterwards does the scheme explore the vertex u and resumes exploration via $\mathcal{A}$ (Line 10).

Note that we do not reveal to $\mathcal{A}$ information gained by exploring vertices during the nearest-neighbor phase (Lines 6 to 9). If $\mathcal{A}$ decides to explore a vertex u next that is already known to $\mathcal{R}$, we only simulate $\mathcal{A}$ without actually executing any traversals (Line 3 resp. Algorithm 2). This is possible since the new information that $\mathcal{A}$ would obtain by exploring u is already known to $\mathcal{R}$.

Recall that, given an online algorithm $\mathcal{A}$ and a graph exploration instance $\mathcal{I}$, the terms $\mathcal{A}_{\mathcal{I}}$ and $\mathrm{OPT}_{\mathcal{I}}$ refer to the costs incurred by $\mathcal{A}$ and an optimal solution on instance $\mathcal{I}$, respectively. To prove Theorem 2, we bound the cost incurred by $\mathcal{R}$ during the NN phases in terms of $\mathrm{OPT}_{\mathcal{I}}$.

Lemma 3. *The cost κ^N of all traversals in Line 7 is at most* $\frac{1}{2}(\lceil \log n \rceil + 1)\mathrm{OPT}_{\mathcal{I}}$.

The lemma can be shown by following the approach of (Rosenkrantz, Stearns, and Lewis 1977). While one consecutive nearest-neighbor search is considered there, our algorithm starts and executes multiple (incomplete) nearest-neighbor searches with different starting points. The adapted proof can be found in the full version (Eberle et al. 2021).

Proof of Theorem 2. Fix $\lambda > 0$, an algorithm $\mathcal{A}$ for the graph exploration problem, and an instance $\mathcal{I}$. Denote by $\mathcal{R}_{\mathcal{I}}$ the cost incurred on instance $\mathcal{I}$ by the robustification scheme $\mathcal{R}$ applied to $\mathcal{A}$ with parameter λ. We show $\mathcal{R}_{\mathcal{I}} \leq (3 + 4\lambda)\mathcal{A}_{\mathcal{I}}$ and $\mathcal{R}_{\mathcal{I}} \leq \left(1+\frac{1}{2\lambda}\right)(\lceil \log(n) \rceil + 1)\mathrm{OPT}_{\mathcal{I}}$ separately. For each iteration i of the outer while loop, denote by κ_i^N the traversal cost incurred by the inner while loop (Line 7), and by $\kappa_i^{\mathcal{A}}$ the cost of the traversal in Line 10. Then, $\mathcal{R}_{\mathcal{I}} = \sum_i (\kappa_i^{\mathcal{A}} + \kappa_i^N)$.

Proof of $\mathcal{R}_I \leq (3 + 4\lambda)\mathcal{A}_I$: For iteration i of the outer while loop, in which $\mathcal{A}$ wants to explore u, let $P_i^{\mathcal{A}}$ be the shortest s-u-path in Line 4. Since Line 3 resp. Algorithm 2 only simulate traversals, the $P_i^{\mathcal{A}}$ may not match the actual traversals that are due to algorithm $\mathcal{A}$. Specifically, it might be the case that $\sum_i c(P_i^{\mathcal{A}}) \neq \mathcal{A}_{\mathcal{I}}$. However, since $P_i^{\mathcal{A}}$ is a shortest path in the currently known graph G which contains $G_{\mathcal{A}}$ after executing the simulated traversals, it cannot exceed the sum of the corresponding simulated traversals. Thus, $\sum_i c(P_i^{\mathcal{A}}) \leq \mathcal{A}_{\mathcal{I}}$.

Consider an iteration i and the traversal cost $\kappa_i^N + \kappa_i^{\mathcal{A}}$ incurred during this iteration. We start by upper bounding κ_i^N. Let κ_i' be the cost of the inner while loop (Lines 6 to 9) *excluding* the last iteration. By definition, $\kappa_i' < \lambda \cdot c(P_i^{\mathcal{A}})$.

Let P_i be the path traversed in the last iteration of the inner while loop, s' its start vertex, and u' its end vertex. Recall that u is the endpoint of $P_i^{\mathcal{A}}$. Before executing the inner while loop, the cost of the shortest path from the current vertex to u was $c(P_i^{\mathcal{A}})$. By executing the inner while loop, excluding the last iteration, the cost of the shortest path from the new current vertex to u can only increase by at most $\kappa_i' < \lambda \cdot c(P_i^{\mathcal{A}})$ compared to the cost of $P_i^{\mathcal{A}}$. Since P_i is the path to the nearest neighbor of the current vertex, $c(P_i)$ cannot be larger than the cost of the shortest path to vertex u. Thus, $c(P_i) \leq c(P_i^{\mathcal{A}}) + \kappa_i' < (1 + \lambda) \cdot c(P_i^{\mathcal{A}})$, and $\kappa_i^N = \kappa_i' + c(P_i) \leq (1 + 2\lambda) \cdot c(P_i^{\mathcal{A}})$.

To bound $\kappa_i^{\mathcal{A}}$, consider the traversal of the shortest s-u-path in Line 10. Before executing the inner while loop, the cost of the shortest path from the current vertex to u was $c(P_i^{\mathcal{A}})$. By executing the while loop, the cost of the shortest path from the new current vertex to u can increase by at most $\kappa_i^N \leq (1 + 2\lambda) \cdot c(P_i^{\mathcal{A}})$ compared to $c(P_i^{\mathcal{A}})$. This implies $\kappa_i^{\mathcal{A}} \leq (2 + 2\lambda) \cdot c(P_i^{\mathcal{A}})$. Using $\kappa_i^{\mathcal{A}} + \kappa_i^N \leq (3 + 4\lambda) \cdot c(P_i^{\mathcal{A}})$, we conclude

$$\mathcal{R}_{\mathcal{I}} = \sum_i \left(\kappa_i^{\mathcal{A}} + \kappa_i^N\right) \leq (3+4\lambda) \sum_i c(P_i^{\mathcal{A}}) \leq (3+4\lambda)\mathcal{A}_{\mathcal{I}}.$$

Proof of $\mathcal{R}_{\mathcal{I}} \leq \left(1 + \frac{1}{2\lambda}\right)(\lceil \log(n) \rceil + 1)\mathrm{OPT}_{\mathcal{I}}$: We have $\kappa_i^{\mathcal{A}} \leq c(P_i^{\mathcal{A}}) + \kappa_i^N$. If the inner while loop was aborted due to $s = u$, then $\kappa_i^{\mathcal{A}} = 0$. Otherwise, $\kappa_i^N \geq \lambda \cdot c(P_i^{\mathcal{A}})$, and thus, $\kappa_i^{\mathcal{A}} \leq (1 + \frac{1}{\lambda})\kappa_i^N$. We conclude $\mathcal{R}_{\mathcal{I}} = \kappa^N + \sum_i \kappa_i^{\mathcal{A}} \leq \left(2 + \frac{1}{\lambda}\right)\kappa^N$. Lemma 3 directly implies the result. □

Reducing the Overhead for Switching Algorithms

The robustification scheme $\mathcal{R}$ balances the execution of a blackbox algorithm $\mathcal{A}$ and a nearest-neighbor search via the parameter λ, that allows us to configure the proportion at which the algorithms are executed. Even for arbitrarily small $\lambda > 0$, the worst-case cost of $\mathcal{R}$ on instance $\mathcal{I}$ is

still $(3+4\lambda)\mathcal{A}_\mathcal{I} \approx 3\mathcal{A}_\mathcal{I}$. The loss of the factor 3 is due to the overhead created by switching between the execution of algorithm $\mathcal{A}$ and NN. To reduce this overhead, we propose a modification of $\mathcal{R}$. At the cost of a slightly worse worst-case guarantee, we significantly improve the average-case performance as our experimental results show. The main reason is that worst-case instances with cost roughly $3\mathcal{A}_\mathcal{I}$ are very particular. We give the full algorithm and its analysis in the full version of our paper (Eberle et al. 2021).

Here, we briefly highlight the differences to Algorithm 1. Intuitively, the modified robustification scheme $\overline{\mathcal{R}}$ reduces the overhead for switching between algorithms by extending the individual phases. To prevent from frequently interrupting $\mathcal{A}$, we introduce a budget for its execution that is now proportional to the cost of *all* past nearest neighbor phases. In turn, we also increase the budget for NN to match these larger phases of executing $\mathcal{A}$: We introduce budgets $b_\mathcal{A}$ and b_N, initialized to 0, that represent the budget generated by $\mathcal{A}$ for executing NN and the budget generated by NN for executing $\mathcal{A}$, respectively. Essentially, we then modify $\mathcal{R}$ by repeating Lines 3 to 5, i.e., exploration according to $\mathcal{A}$, until the budget $\frac{1}{\lambda}b_N$ would be exceeded, and by replacing $c(P^\mathcal{A})$ with $(b_\mathcal{A} + c(P^\mathcal{A}))$ in Line 6. Further, we update $b_\mathcal{A}$ and b_N such that they reflect the cost incurred during the last phase of $\mathcal{A}$ and during all NN-phases, respectively.

Theorem 4. *Given an algorithm $\mathcal{A}$ for the online graph exploration problem and $\lambda > 0$, the modified robustification scheme $\overline{\mathcal{R}}$ solves a graph exploration instance $\mathcal{I}$ with cost*

$$\overline{\mathcal{R}}_\mathcal{I} \leq \min\left\{(3+4\lambda)\mathcal{A}_\mathcal{I}, \left(1+\frac{1}{\lambda}\right)(\lceil\log(n)\rceil+1)\mathrm{OPT}_\mathcal{I}\right\}.$$

The analysis of the modified robustification scheme $\overline{\mathcal{R}}$ remains essentially the same. The second term in the minimum of the worst-case guarantee is slightly higher since the algorithm might terminate directly after executing an extended phase of $\mathcal{A}$ without giving NN a chance to compensate for possible errors. The first term, $(3+4\lambda)\mathcal{A}_\mathcal{I}$, remains the same since the cost of an iteration of the outer while-loop remains bounded by $(3+4\lambda)$ times the cost of an extended phase of $\mathcal{A}$ plus the cost of reaching the next target of $\mathcal{A}$ after that phase.

Online Graph Exploration with Untrusted Predictions

In this section, we apply the previously introduced robustification scheme in the context of learning-augmented algorithms and present an algorithm that satisfies the criteria in Theorem 1. This algorithm is provided with untrusted predictions that come in the form of a fixed exploration order of the vertices, given by a spanning tree or a tour. The learnability of such predictions is discussed at the end of this section.

A Learning-Augmented Online Algorithm

We consider prediction models which upfront fix an exploration order τ_p of the vertices. Such an order can be predicted directly (*tour predictions*) or is given by the traversal order of a Depth First Search (DFS) on a predicted spanning tree T_p (*tree predictions*). Recall that a prediction is a function that outputs for a given exploration state an explorable vertex, and, given an order τ_p, this vertex is the first unexplored vertex in τ_p w.r.t. the current state. Due to this mapping, we also call τ_p a prediction. Denote by $c(\tau_p)$ the cost of executing FP with the prediction τ_p. A *perfect prediction* τ^* is, in the case of tour predictions, an optimal tour and, in the case of tree predictions, the DFS traversal order of a Minimum Spanning Tree (MST) T^*. The prediction error is defined as $\eta = c(\tau_p) - c(\tau^*)$. Regarding tree predictions, the definition of DFS ensures that each edge in T^* is traversed at most twice, thus $c(\tau^*) \leq 2c(T^*)$. Using $c(T^*) \leq \mathrm{OPT}_\mathcal{I}$, this implies the following lemma.

Lemma 5. *For an instance $\mathcal{I}$, following tree predictions has cost* $\mathrm{FP}_\mathcal{I} \leq 2\mathrm{OPT}_\mathcal{I} + \eta$.

Naively calling FP might lead to an arbitrarily bad competitive ratio of FP. Theorem 2 provides us with a tool to mitigate this possibility. Using FP within the robustification scheme of Algorithm 1 allows us to bound the worst-case performance. Denote by $\mathcal{R}(\mathrm{FP}, G)$ the performance of this strategy. With Lemma 5, and given an instance $\mathcal{I}$ with tree predictions, we can upper bound $\mathcal{R}(\mathrm{FP}, G)$ by

$$\min\{(3+4\lambda)(\kappa\mathrm{OPT}_\mathcal{I}+\eta), (1+\frac{1}{2\lambda})(\lceil\log(n)\rceil+1)\mathrm{OPT}_\mathcal{I}\},$$

with $\kappa = 2$. For tour predictions, observe that $\mathrm{FP}_\mathcal{I} = \mathrm{OPT}_\mathcal{I} + \eta$. Thus, we obtain the same bound on $\mathcal{R}(\mathrm{FP}, G)$ but with $\kappa = 1$. This concludes the proof of Theorem 1.

PAC Learnability of the Predictions

We briefly discuss the learnability of tree and tour predictions. To allow for predicted tours or trees that are consistent with the graph to be explored, we assume that the set of n (labeled) vertices is fixed and the graph G is complete. This may seem like a strong restriction of general inputs to the graph exploration problem. However, notice that the cost $c(e)$ of an edge e is still only revealed when the first endpoint of e is explored. There is no improved online algorithm known for this special case.

Firstly, we show PAC learnability of tree predictions. Our goal is to predict a spanning tree in G of low expected cost when edge costs are drawn randomly from an unknown distribution D. We assume that we can sample cost vectors c efficiently and i.i.d. from D to obtain a training set. Denote by $\mathcal{H}$ the set of all labeled spanning trees in G, and, for each $T \in \mathcal{H}$, let $\eta(T, c) = c(T) - c(T^*)$ denote the error of T with respect to the edge costs c, where T^* is an MST of G with respect to the edge costs c. As c is drawn randomly from D, the value $\eta(T, c)$ is a random variable. Our goal is to learn a prediction $T_p \in \mathcal{H}$ that (approximately) minimizes the expected error $\mathbb{E}_{c\sim D}[\eta(T, c)]$ over all $T \in \mathcal{H}$.

We show that there is an efficient learning algorithm that determines a tree prediction that has nearly optimal expected cost with high probability and has a sample size polynomial in n and $\eta_{\max}$, an upper bound on $\eta(T, c)$. The existence and value of such an upper bound depends on the unknown distribution D. Thus, to select the correct value of $\eta_{\max}$ when determining the training set size, we require such minimal prior knowledge of D, which does not seem unreasonable in applications.

Theorem 6. *Let $\eta_{\max}$ be an upper bound on $\eta(T, c)$ and $\bar{T} = \arg\min_{T \in \mathcal{H}} \mathbb{E}_{c \sim D}[\eta(T, c)]$. Under the assumptions above (in particular, that the graph is complete and has a fixed number n of vertices), and for any $\varepsilon, \delta \in (0, 1)$, there exists a learning algorithm that returns a $T_p \in \mathcal{H}$ such that $\mathbb{E}_{c \sim D}[\eta(T_p, c)] \leq \mathbb{E}_{c \sim D}[\eta(\bar{T}, c)] + \varepsilon$ with probability at least $(1 - \delta)$. It does so using a training set of size $m \in \mathcal{O}\left(\frac{(n \cdot \log n - \log \delta) \cdot \eta_{\max}^2}{\varepsilon^2}\right)$ and in time polynomial in n and m.*

We show a similar result for learning a predicted tour. Again, assume that $G = (V, E)$ is complete and let $\mathcal{T}$ be the set of all tours, i.e., the set of all permutations of V. Let $\eta(\tau, c) = c(\tau) - c(\tau^*)$, where $\tau \in \mathcal{T}$, τ^* is an optimal tour w.r.t. cost vector c, and $c(\tau)$ is the cost of tour τ assuming that the next vertex v in τ is always visited via a shortest path in the graph induced by the previous vertices in τ and v. Our goal is to learn a predicted tour $\tau_p \in \mathcal{T}$ that (approximately) minimizes the expected error $\mathbb{E}_{c \sim D}[\eta(\tau, c)]$ over all $\tau \in \mathcal{T}$. As $|\mathcal{T}| = n! \in \mathcal{O}(n^n)$, and assuming an upper bound $\eta_{\max}$ on $\eta(\tau, c)$, we apply ERM with the same sample complexity as in Theorem 6. However, the problem of computing a $\tau \in \mathcal{T}$ that minimizes the empirical error in training set S contains TSP. Thus, unless $P \neq NP$, we settle for an exponential running time; cf. our full version (Eberle et al. 2021).

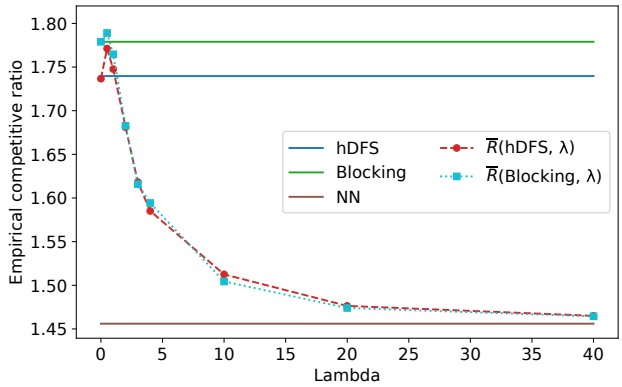
Experimental Analysis

We present the main results of our empirical experiments and discuss their significance with respect to our algorithms' performance. More details can be found in the full version of the paper (Eberle et al. 2021).

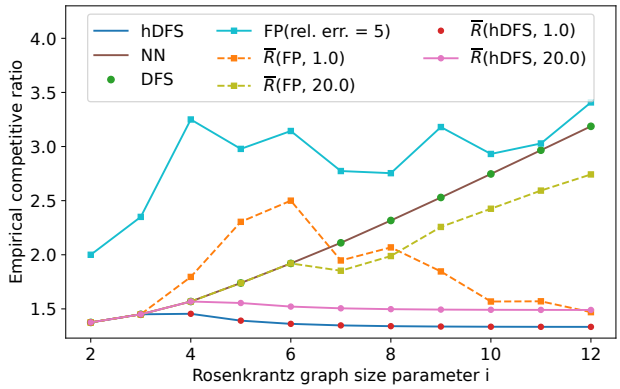
We analyze the performance of the robustification scheme for various instances, namely, real world city road networks, symmetric graphs of the TSPlib library (Reinelt 1991, 1995), and special artificial graphs. We use the *empirical competitive ratio* as performance measure; for an algorithm $\mathcal{A}$, it is defined as the average of the ratio $\mathcal{A}_{\mathcal{I}}/\mathrm{OPT}_{\mathcal{I}}$ over all input instances I in our experiments. Since the offline optimum $\mathrm{OPT}_{\mathcal{I}}$ is the optimal TSP tour which is NP-hard to compute, we lower bound this value by the cost of an MST for instance $\mathcal{I}$, which we can compute efficiently. This leads to larger empirical competitive ratios, but the relative differences between any two algorithms remains the same.

To evaluate learning-augmented algorithms, we compute a (near) perfect prediction and iteratively worsen it to get further predictions. Again, due to the intractability of the underlying TSP problem, we use heuristics to determine a "perfect" prediction, namely Christofides' algorithm (Christofides 1976) and 2-opt (Croes 1958). Such weaker input disfavors our algorithms as having better predictions can only improve the performance of our learning-augmented algorithms. The *relative prediction error* is defined as the ratio between the prediction error and the cost of an MST for the instance.

For the experiments, we consider the classical exploration algorithms depth first search (DFS), nearest neighbor (NN), and hierarchical depth first search (HDFS), as well as the constant-competitive algorithm for graphs of bounded genus, including planar graphs, called BLOCKING (Kalyanasundaram and Pruhs 1994; Megow, Mehlhorn, and Schweitzer 2012). Regarding learning-augmented algorithms we look at



(a) Results for TSPLib instances.



(b) Results for Rosenkrantz graphs.

Figure 1: Evaluation of the robustification scheme on TSPlib and Rosenkrantz graphs.

the algorithm that follows a prediction (FP). We denote by $\overline{\mathcal{R}}(\mathcal{A}, \lambda)$ the modified robustification scheme with parameter $\lambda > 0$ applied to an algorithm $\mathcal{A}$.

TSPLib instances We consider the 72 graphs of the TSPlib library with at most 600 nodes (for performance reasons) and evaluate the classical exploration algorithms as well as their robustified variants. The results are displayed in Figure 1a. Observe that NN outperforms HDFS and BLOCKING. While for small values of λ the performance of the robustified algorithms stays close to that of their base variants, it improves quickly with an increasing λ and eventually converges to that of NN. This illustrates that if NN performs well, our robustification scheme exploits this and improves algorithms performing worse. Note that TSPlib provides complete graphs and our implementation of DFS explores a closest unexplored child first. Thus, DFS and NN act identically.

Rosenkrantz graphs This experiment looks at graphs on which NN is known to perform badly. Specifically, we consider a family of graphs that are artificially constructed in (Rosenkrantz, Stearns, and Lewis 1977) in order to show a lower bound of $\Omega(\log n)$ on the competitive ratio. Each member of the family corresponds to a size parameter i and consists of $n = \Theta(2^i)$ nodes. The cost of a NN tour in-

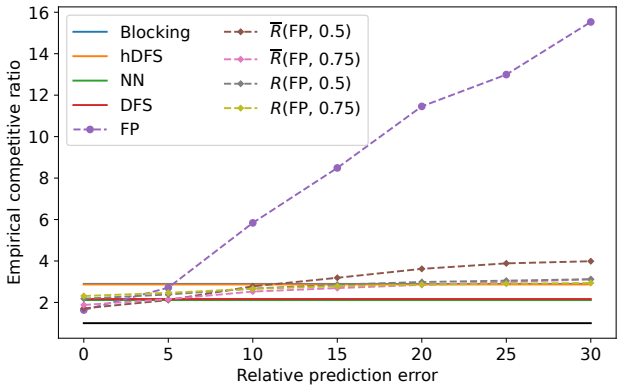
creases linearly with i. We refer to this family as Rosenkrantz graphs. There exist variations of the Rosenkrantz construction, that suggest that we can expect similar experimental results, even for Euclidean graphs and unit weights (Hurkens and Woeginger 2004). Besides NN, we consider the algorithms BLOCKING, HDFS and FP (relative error of 5) with robustification parameters 0, 1, and 20, respectively. Again, DFS acts like NN on these graphs. For the sake of clarity, we deferred results for BLOCKING to the full version (Eberle et al. 2021).

The results (Figure 1b) show that the slight robustification $\overline{\mathcal{R}}(\text{FP},\ 1)$ improves FP's performance significantly. This remains true for large i, even though NN is performing increasingly bad here. If we increase the amount of robustification, i.e., $\overline{\mathcal{R}}(\text{FP},\ 20)$, the slope is equal to NN's slope, but it still outperforms NN and FP. Surprisingly, for HDFS, this drawback does not appear: $\overline{\mathcal{R}}(\text{HDFS},\ 20)$ does indeed perform worse than for smaller λ's, but its competitive ratio does not grow as i increases. For $\overline{\mathcal{R}}(\text{HDFS},\ 1)$ there is almost no drop in performance when compared to HDFS.

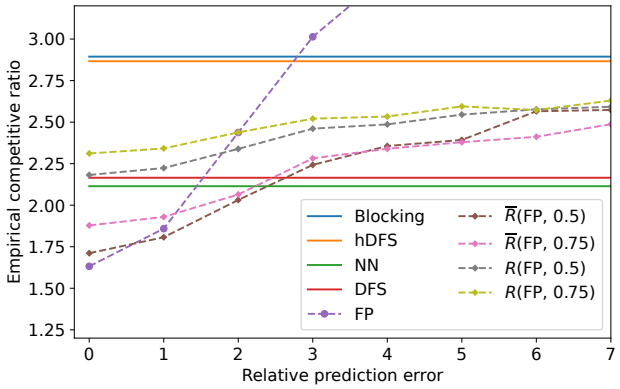
In summary, we have indications that even a slight robustification clearly improves algorithms that otherwise perform badly without the performance degrading too much when NN performs poorly. Even more interestingly, these experiments actually show that the robustification scheme applied to an online algorithm robustifies NN as well. Since NN generally performs notably well, c.f. Figure 1a, this may be useful in practice as protection against unlikely but possible bad scenarios for NN; in particular, in safety relevant applications where solutions have to satisfy strict performance bounds.

City road networks Finally, we provide experiments to evaluate our learning-augmented algorithm in the context of the real-world task of exploring a city road network. To this end, we consider the ten largest (by population) capitals in Europe with a population less than one million. Our instances represent the road networks of these cities, built with OSMnx (Boeing 2017) from OpenStreetMap data (OpenStreetMap contributors 2017). For each instance, we generate 150 predictions with relative errors ranging from 0 up to 30. The average results (Figures 2a and 2b) indicate that, for relative errors less than 2.5, we improve upon the best performing classical algorithms NN and DFS by using $\overline{\mathcal{R}}(\text{FP},\ \lambda)$ with $\lambda < 1$, while the increase of FP for large errors is significantly smaller after robustification. Moreover, $\overline{\mathcal{R}}(\text{FP},\ 0.5)$ and $\overline{\mathcal{R}}(\text{FP},\ 0.75)$ perform only slightly worse than HDFS and BLOCKING for large relative errors.

We conclude that, given predictions of somewhat reasonable quality, it is possible to beat the best known online algorithms in terms of solution quality, while still providing the security that, even if some predictions turn out to be bad, the consequences are not too harsh. While "somewhat reasonable" appears to be a relative error of roughly 2.5, recall that our perfect predictions are only approximate tours, which may be a constant factor away from the optimal tour. With logistic companies in mind, where margins are tight and every potential for optimization needs to be taken advantage of (while still making sure that trucks do arrive eventually), this seems to be a potentially useful outcome.



(a) Whole picture: results for large relative errors.



(b) Zoomed-in: results for small relative errors.

Figure 2: Average performance of classical and learning-augmented algorithms on city networks.

Conclusion

We initiate the study of learning-augmented algorithms for the classical online graph exploration problem. By carefully interpolating between the algorithm that blindly follows any given prediction and Nearest Neighbor, we are able to give a learning-augmented online algorithm whose theoretical worst-case bound linearly depends on the prediction error. In particular, if the prediction is close to perfect, this substantially improves upon any known online algorithm without sacrificing the worst-case bound. We complement these theoretical results by computational experiments on various instances, ranging from symmetric graphs of the TSPLib library and Rosenkrantz graphs to city road networks. Moreover, we design a framework to robustify any given online algorithm by carefully interpolating between this algorithm and Nearest Neighbor. This is potentially very interesting also in the area of stochastic optimization or when designing data-driven algorithms, that typically provide probabilistic guarantees but may perform very poorly in the worst case. It remains open whether online graph exploration (without additional information) allows for any constant-competitive algorithm.

Acknowledgments

Map data copyrighted OpenStreetMap contributors and available from https://www.openstreetmap.org. This research was supported by the German Science Foundation (DFG) under contracts ME 3825/1 and 146371743 — TRR 89 Invasive Computing.

References

Angelopoulos, S.; Dürr, C.; Jin, S.; Kamali, S.; and Renault, M. P. 2020. Online Computation with Untrusted Advice. In *ITCS*, volume 151 of *LIPIcs*, 52:1–52:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Antoniadis, A.; Coester, C.; Eliás, M.; Polak, A.; and Simon, B. 2020. Online metric algorithms with untrusted predictions. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, 345–355. PMLR.

Applegate, D. L.; Bixby, R. E.; Chvatál, V.; and Cook, W. J. 2006. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press.

Azar, Y.; Broder, A. Z.; and Manasse, M. S. 1993. On-line Choice of On-line Algorithms. In *SODA*, 432–440. ACM/SIAM.

Azar, Y.; Leonardi, S.; and Touitou, N. 2021. Flow time scheduling with uncertain processing time. In *STOC*, 1070–1080. ACM.

Balcan, M. 2020. Data-Driven Algorithm Design. In *Beyond the Worst-Case Analysis of Algorithms*, 626–645. Cambridge University Press.

Bamas, É.; Maggiori, A.; Rohwedder, L.; and Svensson, O. 2020. Learning Augmented Energy Minimization via Speed Scaling. In *NeurIPS*, 15350–15359.

Barthelemy, M. 2018. Spatial Networks. In *Encyclopedia of Social Network Analysis and Mining. 2nd Ed*. Springer.

Bello, I.; Pham, H.; Le, Q. V.; Norouzi, M.; and Bengio, S. 2017. Neural Combinatorial Optimization with Reinforcement Learning. In *ICLR (Workshop)*. OpenReview.net.

Berman, P. 1996. On-line Searching and Navigation. In *On-line Algorithms*, volume 1442 of *Lecture Notes in Computer Science*, 232–241. Springer.

Birx, A.; Disser, Y.; Hopp, A. V.; and Karousatou, C. 2021. An improved lower bound for competitive graph exploration. *Theor. Comput. Sci.*, 868: 65–86.

Blum, A.; and Burch, C. 2000. On-line Learning and the Metrical Task System Problem. *Mach. Learn.*, 39(1): 35–58.

Böckenhauer, H.; Fuchs, J.; and Unger, W. 2018. Exploring Sparse Graphs with Advice. In *WAOA*, volume 11312 of *Lecture Notes in Computer Science*, 102–117. Springer.

Boeing, G. 2017. OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Comput. Environ. Urban Syst.*, 65: 126–139.

Boeing, G. 2020. Planarity and street network representation in urban form analysis. *Environment and Planning B: Urban Analytics and City Science*, 47(5): 855–869.

Brandt, S.; Foerster, K.; Maurer, J.; and Wattenhofer, R. 2020. Online graph exploration on a restricted graph class: Optimal solutions for tadpole graphs. *Theor. Comput. Sci.*, 839: 176–185.

Chiotellis, I.; and Cremers, D. 2020. Neural Online Graph Exploration. *CoRR*, abs/2012.03345.

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

Croes, G. A. 1958. A Method for Solving Traveling-Salesman Problems. *Operations Research*, 6(6): 791–812.

Dai, H.; Li, Y.; Wang, C.; Singh, R.; Huang, P.; and Kohli, P. 2019. Learning Transferable Graph Exploration. In *NeurIPS*, 2514–2525.

Dobrev, S.; Královic, R.; and Markou, E. 2012. Online Graph Exploration with Advice. In *SIROCCO*, volume 7355 of *Lecture Notes in Computer Science*, 267–278. Springer.

Dütting, P.; Lattanzi, S.; Leme, R. P.; and Vassilvitskii, S. 2021. Secretaries with Advice. In *EC*, 409–429. ACM.

Eberle, F.; Lindermayr, A.; Nölke, L.; Megow, N.; and Schlöter, J. 2021. Robustification of Online Graph Exploration Methods. *CoRR*, abs/2112.05422.

Elmiger, J.; Faber, L.; Khanchandani, P.; Richter, O. P.; and Wattenhofer, R. 2020. Learning Lower Bounds for Graph Exploration With Reinforcement Learning. In *Learning Meets Combinatorial Algorithms at NeurIPS2020*.

Fiat, A.; Karp, R. M.; Luby, M.; McGeoch, L. A.; Sleator, D. D.; and Young, N. E. 1991. Competitive Paging Algorithms. *J. Algorithms*, 12(4): 685–699.

Fritsch, R. 2021. Online graph exploration on trees, unicyclic graphs and cactus graphs. *Inf. Process. Lett.*, 168: 106096.

Gasieniec, L.; and Radzik, T. 2008. Memory Efficient Anonymous Graph Exploration. In *WG*, volume 5344 of *Lecture Notes in Computer Science*, 14–29.

Gollapudi, S.; and Panigrahi, D. 2019. Online Algorithms for Rent-Or-Buy with Expert Advice. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, 2319–2327. PMLR.

Hurkens, C. A. J.; and Woeginger, G. J. 2004. On the nearest neighbor rule for the traveling salesman problem. *Oper. Res. Lett.*, 32(1): 1–4.

Im, S.; Kumar, R.; Qaem, M. M.; and Purohit, M. 2021. Non-Clairvoyant Scheduling with Predictions. In *SPAA*, 285–294. ACM.

Kalyanasundaram, B.; and Pruhs, K. 1994. Constructing Competitive Tours from Local Information. *Theor. Comput. Sci.*, 130(1): 125–138.

Khalil, E. B.; Dai, H.; Zhang, Y.; Dilkina, B.; and Song, L. 2017. Learning Combinatorial Optimization Algorithms over Graphs. In *NIPS*, 6348–6358.

Komm, D.; Královic, R.; Královic, R.; and Smula, J. 2015. Treasure Hunt with Advice. In *SIROCCO*, volume 9439 of *Lecture Notes in Computer Science*, 328–341. Springer.

Kool, W.; van Hoof, H.; and Welling, M. 2019. Attention, Learn to Solve Routing Problems! In *ICLR (Poster)*.

Kumar, R.; Purohit, M.; Schild, A.; Svitkina, Z.; and Vee, E. 2019. Semi-Online Bipartite Matching. In *ITCS*, volume 124 of *LIPIcs*, 50:1–50:20. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

Lattanzi, S.; Lavastida, T.; Moseley, B.; and Vassilvitskii, S. 2020. Online Scheduling via Learned Weights. In *SODA*, 1859–1877. SIAM.

Lavastida, T.; Moseley, B.; Ravi, R.; and Xu, C. 2021. Learnable and Instance-Robust Predictions for Online Matching, Flows and Load Balancing. In *ESA*, volume 204 of *LIPIcs*, 59:1–59:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Lawler, E.; Lenstra, J.; Rinnoy Kan, A.; and Shmoys, D. 1985. *The Traveling Salesman Problem – A Guided Tour of Combinatorial Optimization*. Wiley.

Luperto, M.; and Amigoni, F. 2019. Predicting the global structure of indoor environments: A constructive machine learning approach. *Auton. Robots*, 43(4): 813–835.

Lykouris, T.; and Vassilvitskii, S. 2018. Competitive Caching with Machine Learned Advice. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, 3302–3311. PMLR.

Mahdian, M.; Nazerzadeh, H.; and Saberi, A. 2012. Online Optimization with Uncertain Information. *ACM Trans. Algorithms*, 8(1): 2:1–2:29.

Medina, A. M.; and Vassilvitskii, S. 2017. Revenue Optimization with Approximate Bid Predictions. In *NIPS*, 1858–1866.

Megow, N.; Mehlhorn, K.; and Schweitzer, P. 2012. Online graph exploration: New results on old and new algorithms. *Theor. Comput. Sci.*, 463: 62–72.

Mitzenmacher, M. 2020. Scheduling with Predictions and the Price of Misprediction. In *ITCS*, volume 151 of *LIPIcs*, 14:1–14:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Miyazaki, S.; Morimoto, N.; and Okabe, Y. 2009. The Online Graph Exploration Problem on Restricted Graphs. *IEICE Trans. Inf. Syst.*, 92-D(9): 1620–1627.

OpenStreetMap contributors. 2017. Planet dump retrieved from https://planet.osm.org. https://www.openstreetmap.org. Accessed: 2021-05-11.

Purohit, M.; Svitkina, Z.; and Kumar, R. 2018. Improving Online Algorithms via ML Predictions. In *NeurIPS*, 9684–9693.

Rao, N. S. V.; Iyengar, S. S.; Jorgensen, C. C.; and Weisbin, C. R. 1986. Robot navigation in an unexplored terrain. *J. Field Robotics*, 3(4): 389–407.

Reinelt, G. 1991. TSPLIB - A Traveling Salesman Problem Library. *INFORMS J. Comput.*, 3(4): 376–384.

Reinelt, G. 1995. TSPLIB. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/. Accessed: 2021-05-11.

Rohatgi, D. 2020. Near-Optimal Bounds for Online Caching with Machine Learned Advice. In *SODA*, 1834–1845. SIAM.

Rosenkrantz, D. J.; Stearns, R. E.; and Lewis, P. M. 1977. An Analysis of Several Heuristics for the Traveling Salesman Problem. *SIAM J. Comput.*, 6(3): 563–581.

Valiant, L. G. 1984. A Theory of the Learnable. *Commun. ACM*, 27(11): 1134–1142.

Vapnik, V. N.; and Chervonenkis, A. Y. 2013. On the Uniform Convergence of the Frequencies of Occurrence of Events to Their Probabilities. In *Empirical Inference*, 7–12. Springer.

Vinyals, O.; Fortunato, M.; and Jaitly, N. 2015. Pointer Networks. In *NIPS*, 2692–2700.

Zhou, J.; Cui, G.; Hu, S.; Zhang, Z.; Yang, C.; Liu, Z.; Wang, L.; Li, C.; and Sun, M. 2020. Graph neural networks: A review of methods and applications. *AI Open*, 1: 57–81.