Learning-Augmented Online Sorting and TSP

Ioana O. Bercea
KTH Royal Institute of Technology, Stockholm, Sweden

Gerth Stølting Brodal
Aarhus University, Denmark

John Iacono
Universite libre de Bruxelles (ULB), Belgium

László Kozma
Dresden University of Technology, Germany

Debmalya Panigrahi
Duke University, Durham, NC, USA

Abstract

The *online sorting problem* is a natural online analog of classical sorting: n elements arrive one by one and must be placed irrevocably into an array of size n so as to minimize the sum of absolute differences between consecutive elements. Recent work by Aamand *et al.* [SODA 2023] and Abrahamsen *et al.* [ESA 2024] showed that the optimal competitive ratio for this problem is $\Theta(\sqrt{n})$, even when randomization is allowed. Bertram [ESA 2025] extended this bound to the online traveling salesman problem (TSP), of which online sorting is a special case on the line metric. These polynomial bounds raise the question of whether additional information can lead to improved performance. In this paper, we initiate the study of online sorting and TSP in the framework of *machine-learned predictions*. We characterize the exact tradeoff between *consistency* and *robustness* for online sorting with predictions, and prove a surprising lower bound showing that robustness is not lossless in this setting. This phenomenon sets online sorting apart from most previously studied online problems with predictions. We extend our results to online TSP with predictions on general metric spaces, where the same consistency–robustness tradeoff persists. Finally, we present a sharp contrast in the case of online TSP on the uniform metric. While Abrahamsen *et al.* gave an $O(\log n)$-competitive algorithm without predictions for the uniform metric, we show that predictions enable an algorithm that is simultaneously $O(1)$-consistent and $O(\log n)$-robust. We further extend this result to the setting of multiple predictions.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Online sorting, online traveling salesman problem, learned predictions

Digital Object Identifier 10.4230/LIPIcs.ESA.2026.78

Funding *Ioana O. Bercea*: Supported by Swedish Research Council grant 2024-05366.
John Iacono: Supported by the Fonds de la Recherche Scientifique – FNRS.
Debmalya Panigrahi: Supported in part by NSF grants CCF-2329230 and CCF-1955703.

Acknowledgements This work was initiated at the Dagstuhl Seminar 25181 on "Learned Predictions for Data Structures and Running Time" in April-May, 2025 that all the authors participated in.

1 Introduction

Sorting, where n elements have to be reordered to a monotonically non-decreasing (or non-increasing) order, is one of the fundamental pillars of combinatorial algorithms. In *online* sorting, we are given an online sequence of n elements $x_1, x_2, \ldots, x_n$ that have to be inserted into an n-element array $A[1 \ldots n]$. At time t, the t-th element x_t is provided to the algorithm and has to be placed in one of the empty cells of array A. Once placed, the location of x_t is fixed and x_t cannot be moved to other cells during the arrival of the subsequent

elements. The goal is to produce as close to a sorted array as possible. Formally, if $A[j]$ is the element occupying the j-th cell of array A, then the *cost* of the algorithm is defined as $c := \sum_{i=1}^{n-1} \big|A[i+1] - A[i]\big|$. Clearly, the optimal solution is a sorted array (increasing or decreasing), and its cost is $c_{\text{opt}} := \max_t x_t - \min_t x_t$. Following standard convention, an algorithm is said to be ρ-competitive if $c/c_{\text{opt}} \leq \rho$ for all instances of the problem.

Aamand *et al.* [1] introduced the online sorting problem and gave a deterministic algorithm with a competitive ratio of $O\big(\sqrt{n}\big)$ under the assumption that the range of the elements is known in advance. They also showed a matching lower bound for deterministic algorithms. Later, Abrahamsen *et al.* [2] showed that the same upper and lower bounds also apply in the general setting of an unknown range of elements, and extended the lower bound to randomized algorithms. They also introduced a natural generalization of the problem called the online traveling salesman problem (online TSP), where a sequence of points in a metric space have to be inserted in an array to minimize the sum of distances between consecutive points. The online sorting problem is the special case of online TSP on the line metric. Recently, Bertram [10] gave a deterministic algorithm for online TSP with a competitive ratio of $O\big(\sqrt{n}\big)$ for general metrics. This bound is tight by virtue of the existing lower bounds for the special case of online sorting. In contrast, for the uniform metric where all distances are 0 or 1, Abrahamsen *et al.* [2] gave matching upper and lower bounds of $O(\log n)$, establishing this as a variant of online TSP that admits better competitive bounds than online sorting.

While these recent works have provided a detailed picture of competitive bounds for online sorting/TSP, the large lower bounds and the fundamental nature of sorting/TSP motivate further exploration of these problems beyond worst-case settings. In this paper, we consider online sorting/TSP in the popular setting of (machine-learned) predictions (see references in related work). In this framework, at time t, along with the element x_t, the algorithm also gets a *predicted* index $\hat{i}_t \in [n]$ for x_t in the array A. We assume that the predicted indices over all the n elements form a permutation of the indices $[n]$ of the array, i.e., that the predictions correspond to a valid solution of the problem. The cost of the solution produced by the predictions is denoted $\hat{c}$. If the algorithm simply follows the predictions, then its cost would also be $\hat{c}$. Following standard terminology established by Purohit *et al.* [35], we call this a 1-*consistent* solution. More generally, an α-*consistent* solution is one whose cost is at most $\alpha \cdot \hat{c}$. The 1-consistent solution is ideal if the predicted solution is optimal, but provides no guardrails if the predicted solution has a large cost, possibly $\Omega(n)$ times c_{opt}. On the other hand, the online algorithm can ignore the predictions and run the algorithms from [1, 2, 10] to obtain a cost of $O\big(\sqrt{n}\big) \cdot c_{\text{opt}}$. In this case, we say that the solution is $O\big(\sqrt{n}\big)$-*robust*; in general, if the cost is no more than r times c_{opt}, then the algorithm is said to provide a robustness guarantee of r. (Thus, one can think of an r-robust algorithm as being r-competitive irrespective of any prediction.) Of course, the problem with the latter strategy is that it fails to take advantage of accurate predictions, e.g., even if they give an optimal solution. The main question that we study in this paper is the tradeoff between these two desiderata – consistency and robustness – for online sorting and TSP with predictions. Note that an r-robust algorithm is always r-consistent, since $\hat{c} \geq c_{\text{opt}}$, but not vice versa.

1.1 Our Results

For many online problems, we can simultaneously obtain optimal consistency and robustness (up to constants), usually by standard techniques (e.g., [5]) that combine two solutions for an online problem to obtain the better of the two costs up to constants (e.g., caching [30], ski rental [35, 21], set cover [9, 3], etc.). For online sorting/TSP, this would imply $O(1)$-consistent, $O\big(\sqrt{n}\big)$-robust algorithms. Our first contribution is a lower bound for online sorting (and therefore online TSP which is more general) that rules this out in a rather strong sense:

▶ **Theorem 1.** *For any n and r, with $1 \leq r \leq n$, for any algorithm for online sorting with predictions (randomization allowed) there is an input sequence (including predictions) of length n and optimal cost 1, where the algorithm, when run on that sequence, has expected robustness $\Omega(r)$ or expected consistency $\Omega(\frac{n}{r})$.*

By [1] and [2] we know that deterministic and randomized algorithms (with void predictions) must have robustness $\Omega(\sqrt{n})$, and there exist algorithms achieving robustness $O(\sqrt{n})$, and therefore also consistency $O(\sqrt{n})$ (when ignoring the predictions). Our lower bound raises the question of whether we can get better consistency bounds if we *relax* the robustness guarantee to be weaker than $O(\sqrt{n})$. We show that this is indeed the case. Given any $r = \Omega(\sqrt{n})$, we give a deterministic $O(n/r)$-consistent, $O(r)$-robust algorithm for online sorting. (Following [1], in Theorems 2 and 3, we assume that c_{opt} is known to the algorithm.)

▶ **Theorem 2.** *For any $r = \Omega(\sqrt{n})$, there is a deterministic $O(n/r)$-consistent and $O(r)$-robust algorithm for online sorting with predictions (when c_{opt} is known to the algorithm).*

Note that this result is tight by Theorem 1. We also extend this theorem to the more general problem of online TSP:

▶ **Theorem 3.** *For any $r = \Omega(\sqrt{n})$, there is a deterministic $O(n/r)$-consistent and $O(r)$-robust algorithm for online TSP with predictions (when c_{opt} is known to the algorithm).*

Finally, we consider online TSP with predictions on the uniform metric. Intuitively, in this case, the cost simply counts all pairs of *unequal* neighbors in the array. Recall that Abrahamsen *et al.* [2] had shown a qualitative separation between online TSP on the line metric (i.e., online sorting) and the uniform metric; for the latter, they gave an $O(\log n)$-competitive deterministic algorithm and a matching lower bound for randomized algorithms. We show that there is a qualitative separation between these problems with predictions as well. On the uniform metric, we give an algorithm for online TSP with predictions that simultaneously achieves the asymptotically best consistency and robustness bounds:

▶ **Theorem 4.** *On the uniform metric, there is an $O(1)$-consistent, $O(\log n)$-robust deterministic algorithm for online TSP with predictions.*

This theorem can be interpreted as a "best of both worlds" result – an algorithm can match the better cost up to constants between two solutions given online. What if the algorithm is given more than two solutions? This is called the multiple predictions setting (or prediction portfolios), studied previously in, e.g., [21, 8, 17, 3, 27, 16]. For online TSP on the uniform metric with h predictions, we generalize Theorem 4 to the following:

▶ **Theorem 5.** *There is a randomized algorithm for online TSP with h predictions on the uniform metric that is $O(\log h)$-consistent with the best of the h predictions, and a deterministic algorithm that is $O(h)$-consistent with the best of the h predictions.*

Intuitively, Theorem 5 captures the situation where we have access to multiple algorithms that produce valid online solutions, and we wish to do as well as the best among them (in hindsight). Observe that Theorem 5 implies Theorem 4, since we can set $h = 2$, and have the sequence $\hat{i}_t$ from Theorem 4 as one of the predictors, and the output of the $O(\log n)$-competitive algorithm from [2] as the other.

Our Techniques. The main idea behind the lower bound (Theorem 1) is simple. Suppose the adversary presents a set of elements in $[0, 1]$ that are spread out uniformly over the interval, and the prediction also spreads them uniformly in the array (similarly to previous

input sequences in [1, 2]). Now, the algorithm faces a continuum of choices between two extremes: it can follow the prediction and spread the elements uniformly, or it can allocate the elements to a contiguous block of array cells. The former is the correct choice if the adversary continues to present well spread-out numbers in the future, while the latter is the correct choice if the adversary reverts to only giving 0 or 1. Of course, the adversary will choose the future that does not match the algorithm's choice, thereby making the algorithm incur large cost. This tension is quantified by the number of *gaps* (maximal contiguous blocks of empty cells) in the array. If the algorithm creates a large number of gaps, then the adversary presents either all 0s or all 1s as the remaining elements, resulting in a large cost at the boundaries of these gaps. On the other hand, if the algorithm keeps the number of gaps small, then it necessarily places many different elements next to each other, also incurring large cost. In the first case, robustness suffers as the optimal solution places the 0s or 1s at one end incurring lower cost. In the second case, consistency suffers, as the predictions may turn out to be accurate, but we do not follow them.

At a high level, all our algorithms (Theorems 2–4) carefully balance these two approaches, adapting to the input and prediction sequences. The idea is to follow the predictions *approximately*, placing elements more compactly than the prediction to reduce the number of gaps. If at some stage the cost of the prediction becomes prohibitive and ensuring robustness becomes the goal, then the algorithm switches to a robust algorithm defined only on the empty spaces in the array. Deviating from the predictions on the fly introduces challenges both for the algorithm and its analysis. First, we need careful bookkeeping to ensure that future predictions are still adequately mapped to empty cells, and second, we must account for the effect of these changes on the cost and consistency.

In case of online sorting/TSP (Theorems 2 and 3), we can bound the additional cost due to not strictly following the predictions by using a consistency factor $O(n/r)$. The uniform metric case (Theorem 4) is more challenging, since we cannot afford to lose more than a constant compared to the prediction. In this case, we devise a compaction strategy that not only reduces the number of gaps in the array but does so at no penalty; in fact, we show that the compaction actually *lowers the cost* compared to the prediction. This strong property helps ensure a strict consistency bound for the uniform metric. Moreover, it also allows more flexibility in switching between different strategies, which we exploit further in our last result for the multiple predictions setting (Theorem 5). When switching from one strategy to another, we re-map the empty cells of the array to a virtual array, bounding the penalty due to this abstraction. Correspondingly, we also map the predictions to the empty cells of the array, which allows the algorithm to recurse on this virtual array. Crucially, the overall prediction cost can be additively decomposed into a non-recursive and a recursive part, which are then used to account for the non-recursive and recursive parts of the algorithm's cost.

1.2 Related Work

The online sorting problem was introduced by Aamand *et al.* [1] as a gadget for studying geometric packing problems such as packing polygons arriving online in a given geometric space. Such problems are widely studied in the literature, e.g., online strip packing [7, 15, 23, 40] and online rectangle bin packing [12, 19, 39, 20, 13, 14, 37, 22]. The online sorting problem itself has been studied in beyond worst-case settings earlier, such as for stochastic inputs [2, 25, 18], or with resource augmentation (meaning, in this context, an array with more than n cells) [1, 34, 6]. To the best of our knowledge, our work is the first to consider it in the predictions setting.

More broadly, augmenting online algorithms with machine learning predictions has emerged as an exciting new paradigm in the last few years that seeks to circumvent strong lower bounds in worst-case competitive analysis using side information. This framework was formalized in the work of Lykouris and Vassilvitskii [30], and has since developed into a robust literature in a variety of problem domains (see the survey [33]). In particular, the predictions setting has been explored in data structures both in theory (e.g., [32, 24, 38, 29, 16]) and practice (e.g., [28] and many follow up works). Our work can be seen as extending the predictions setting to the problem of constructing a sorted array (or more generally a TSP tour) online.

The term "online TSP" has been used to define a variety of problems in the past. In the 1970s, Rosenkrantz *et al.* [36] studied the problem of incrementally constructing a TSP tour over a sequence of points arriving online. This problem is strictly simpler than the online TSP problem studied in this paper since it only asks for commitment to the ordering of points and not their locations in the array. Indeed, this problem admits $O(\log n)$-competitive algorithms without predictions [36, 26]. Later, the term "online TSP" was used for other classes of problems: in the problem setting of [4, 11], requests appear over time and have to be served by a moving server, with the goal of minimizing the total time to serve all requests; in the setting of [31], requests appear online and a tour of small cost has to be maintained for all seen requests, with a limited budget for updating the set of edges in the tour. Although they share the same name, these problems are not related to ours.

Organization. We give the (randomized) lower bound for online sorting with predictions (Theorem 1) in Section 2, and the matching algorithm for the more general online TSP with predictions (Theorem 3) in Appendix A. For online TSP with predictions on the uniform metric, although Theorem 5 generalizes Theorem 4, we present them separately in Section 4 and Appendix B for improved readability. We close with some future directions in Section 5.

2 Lower Bound for Online Sorting with Predictions

In this section we show that it is impossible to have good consistency while ensuring robustness, even for a randomized algorithm. Our lower bound uses input sequences very similar to those used in the lower bounds by Aamand *et al.* [1] and Abrahamsen *et al.* [2]. But whereas [1] provide an adversary against deterministic algorithms, and [2] provide a single random input distribution hard for all deterministic and randomized algorithms, we provide for each deterministic or random algorithm a specific input sequence where the algorithms fail to achieve simultaneously good expected consistency and robustness (Theorem 1).

We show that it is possible to create input sequences and predictions so that part of the way through the sequence, the sequence could be completed so that all of the predictions are correct and constant consistency is possible, or that they are far from correct and robustness is the measure of interest. Thus an algorithm needs to follow the predictions, or at least come close to following them, if they end up being correct. Our sequences are, however, constructed in such a way that following the predictions can result in poor robustness, should the predictions not be correct. We now prove Theorem 1.

We assume a fixed randomized algorithm and for convenience assume $\frac{n}{r}$ and $\frac{r}{3}$ are integers. Consider a sequence $X = x_1, x_2, \ldots, x_n$ of elements to be inserted into the size-n array A:

$$X \coloneqq \overbrace{\frac{0}{r-1}, \frac{1}{r-1}, \ldots, \frac{r-1}{r-1}}^{\text{round } 1}, \ldots, \overbrace{\frac{0}{r-1}, \frac{1}{r-1}, \ldots, \frac{r-1}{r-1}}^{\text{round } \frac{n}{r}}$$

Also define $2\frac{n}{r}$ sequences X_j^0 and X_j^1, where $j \in [\frac{n}{r} - 1]$, which are the same as X but with all values after round j being replaced with 0 or 1:

$$X_j^0 \coloneqq \overbrace{\frac{0}{r-1}, \frac{1}{r-1}, \ldots, \frac{r-1}{r-1}}^{\text{round } 1}, \ldots, \overbrace{\frac{0}{r-1}, \frac{1}{r-1}, \ldots, \frac{r-1}{r-1}}^{\text{round } j}, \overbrace{0, 0, \ldots, 0}^{\text{round } j+1}, \ldots, \overbrace{0, 0, \ldots, 0}^{\text{round } \frac{n}{r}}$$

$$X_j^1 \coloneqq \overbrace{\frac{0}{r-1}, \frac{1}{r-1}, \ldots, \frac{r-1}{r-1}}^{\text{round } 1}, \ldots, \overbrace{\frac{0}{r-1}, \frac{1}{r-1}, \ldots, \frac{r-1}{r-1}}^{\text{round } j}, \overbrace{1, 1, \ldots, 1}^{\text{round } j+1}, \ldots, \overbrace{1, 1, \ldots, 1}^{\text{round } \frac{n}{r}}$$

The sequence of predictions $\hat{I} = \hat{i}_1, \hat{i}_2, \ldots, \hat{i}_n$ is an optimal sequence of predictions for the sequence X: It is the permutation such that if x_t is inserted into $A[\hat{i}_t]$ the elements will be sorted in increasing order. Specifically:

$$\hat{I} \coloneqq \begin{cases} \overbrace{0\frac{n}{r} + 1, 1\frac{n}{r} + 1, \ldots, (r-1)\frac{n}{r} + 1}^{\text{round } 1}, \\ \overbrace{0\frac{n}{r} + 2, 1\frac{n}{r} + 2, \ldots, (r-1)\frac{n}{r} + 2}^{\text{round } 2}, \\ \vdots \\ \overbrace{0\frac{n}{r} + \frac{n}{r}, 1\frac{n}{r} + \frac{n}{r}, \ldots, (r-1)\frac{n}{r} + \frac{n}{r}}^{\text{round } \frac{n}{r}} \end{cases}$$

Our basic approach is to show that inserting one of the sequences X_j^0 or X_j^1 gives an expected robustness of $\Omega(r)$ or inserting X gives an expected consistency of $\Omega(\frac{n}{r})$. We use $\hat{I}$ as the prediction sequence for all $2\frac{n}{r} + 1$ sequences X_j^0, X_j^1 or X.

After the algorithm has inserted a prefix of the n elements, some array cells will contain elements and some will be empty. Define a *transition* in the array to be an adjacency between an empty cell and a non-empty cell. Let T_j be the stochastic variable equal to the number of transitions that the algorithm on input X and predictions $\hat{I}$ has after round j, where the expectation is with respect to the possible randomized choices of the algorithm. There are two cases – either for some $j \in [\frac{n}{r}]$, $\Pr[T_j \geq \frac{r}{3}] \geq \frac{1}{4}$ or for all $j \in [\frac{n}{r}]$, $\Pr[T_j \geq \frac{r}{3}] < \frac{1}{4}$.

First Case. First consider the case where there exists a j where $\Pr[T_j \geq \frac{r}{3}] \geq \frac{1}{4}$, fix such a j for this paragraph. This j can not be $\frac{n}{r}$, as after the last round the array is full and there are no transitions. Observe the first j rounds of the sequences X, X_j^0 and X_j^1 are identical. Consider the non-empty array cells in each transition after round j, and compute the expected average of these elements. This expected average is either above or at most $\frac{1}{2}$, assume that it is above. In that case, when inserting X_j^0, each of the at least $\frac{r}{3}$ transitions after round j will result in 0's next to an element whose average is at least $\frac{1}{2}$. The contribution to the cost in these pairs alone is thus at least $\frac{r}{3} \cdot \frac{1}{2} = \frac{r}{6}$. The case where the average is at most $\frac{1}{2}$ is symmetric, where the insertion of X_j^1 will result in cost of at least $\frac{r}{6}$. Thus in this case insertion one of the sequences X_j^1 and X_j^0 will result in an expected cost of $\frac{r}{6} \cdot \frac{1}{4} = \frac{r}{24}$ and thus a robustness of $\Omega(r)$, since the optimal cost X_j^1 and X_j^0 is 1.

Second Case. Next we consider the case $\Pr[T_j \geq \frac{r}{3}] < \frac{1}{4}$ for all $j \in [\frac{n}{r}]$, i.e., $\Pr[T_j < \frac{r}{3}] \geq \frac{3}{4}$ for all $j \in [\frac{n}{r}]$. We now focus on the cost of inserting X (not any of the X_j^0 or X_j^1) and we know that $\hat{I}$ has perfect predictions that if followed on X will result in the array being sorted and with cost 1.

Call a round j *bad* if after rounds $j-1$ and j the number of transition T_{j-1} and T_j are both at most $\frac{r}{3}$. By the union bound, this happens with probability at least $\frac{1}{2}$. During round j, r elements are inserted. In a bad round at most $\frac{r}{3}$ of these can occupy the at most $\frac{r}{3}$ empty cells of a transition present after round $j-1$. Also at most $\frac{r}{3}$ of these r newly inserted elements in round j have an empty cell next to them after round j completes. This means that at least $\frac{r}{3}$ of the elements inserted in round j must have as neighbors on both sides in the array other elements inserted in round j, creating at least $\frac{r}{3}$ adjacencies in the array between elements inserted in round j. Any two elements inserted in round j have values that are at least $\frac{1}{r}$ apart, so the total cost incurred by adjacancies between elements inserted in a bad round j is at least $\frac{r}{3} \cdot \frac{1}{r} = \frac{1}{3}$.

The total cost of an algorithm is at least the cost incurred by elements which are placed in consecutive cells in the array in the same bad round. We now bound this quantity via linearity of expectation. Since in this case, bad rounds happen with probability at least $\frac{1}{2}$ the total expected cost of inserted-in-the-same-bad-round adjacency is at least the number of rounds times the probability of a bad round times a lower bound on the cost between the adjacant elements added in a bad round round: $\frac{n}{r} \cdot \frac{1}{2} \cdot \frac{1}{3} = \frac{n}{6r}$. Recall that in X the predictions are perfect with a predicted cost $\hat{c} = 1$, i.e., an expected cost of at least $\frac{n}{6r}$ gives an expected consistency of $\Omega(\frac{n}{r})$. This concludes the proof of Theorem 1.

3 Online Sorting with Predictions

In this section, we give an algorithm that for a given parameter $r = \Omega(\sqrt{n})$ establishes the upper bound for online sorting with predictions (Theorem 2). The algorithm consists of two phases, where the first phase continues as long as the predicted cost is sufficiently small. In the first phase, the algorithm partitions the array A into r contiguous blocks of n/r cells each. The j-th block goes from the $((j-1) \cdot n/r + 1)$-st cell to the $(j \cdot n/r)$-th cell of the array. We denote the indices in the j-th block $\mathcal{B}_j$. On the arrival of an element x_t with predicted index $\hat{i}_t$, the algorithm places x_t in the next available empty cell in the block j where $\hat{i}_t \in \mathcal{B}_j$. Since the predicted indices produce a valid permutation of $[n]$, there are at most n/r predicted indices in $\mathcal{B}_j$ for any j. Therefore, we never run out of space in any block.

Let $\hat{c}(t)$ be the cost of the predicted solution until time t, i.e., the cost of the array of size n where $x_1, \ldots, x_t$ would be placed at indices $\hat{i}_1, \ldots, \hat{i}_t$ and the remaining $n-t$ empty cells removed. Note that the cost $\hat{c}(t)$ is non-decreasing in t and $\hat{c} = \hat{c}(n)$. The first phase continues while $(n/r) \cdot \hat{c}(t) \leq r \cdot c_{\text{opt}}$, and the second phase starts if and when this condition is violated. We remark that knowledge of c_{opt} is used in this step by the algorithm to determine the transition from the first to the second phase.

At the beginning of the second phase (see Figure 1), suppose the number of empty cells is n'. The algorithm maps these n' cells to an empty virtual array A' of size n'. Then, it inserts the remaining n' elements in the virtual array A' using the online sorting algorithm from [1] that achieves a cost of $O(\sqrt{n'}) \cdot c'_{\text{opt}}$ on the virtual array, where $c'_{\text{opt}} \leq c_{\text{opt}}$ is the optimal cost for the elements of the second phase. Note that the predictions are ignored by the online algorithm in the second phase. Each element is inserted in the real array by mapping its cell in the virtual array back to the empty cell in the real array that it represents.

We now analyze the cost of the solution produced by the algorithm. First, note that the cost incurred by pairs of consecutive elements in the virtual array inserted in the second phase is at most $O(\sqrt{n'} \cdot c'_{\text{opt}})$ by [1], which is at most $O(\sqrt{n} \cdot c_{\text{opt}})$. Moreover, since the virtual array maps to at most r gaps in the real array, the cost incurred by pairs of consecutive indices in A filled in different phases is at most $2r \cdot c_{\text{opt}}$.

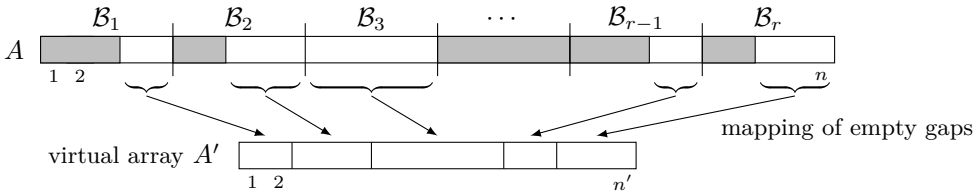

Figure 1 Algorithm for online sorting and online TSP with predictions, end of phase one. Grey area of each block are cells placed during phase one. The empty gaps form a new virtual array of size n' for phase two.

We are left to analyze the cost of the first phase. First, we show two simple lower bounds on the cost of the predicted solution. Let m denote the number of non-empty blocks in A after placing $x_1, \ldots, x_t$ and let $j_1 < \cdots < j_m$ denote the indices of the non-empty blocks.

▷ Claim 6. At time t, we have the following lower bounds on the cost of the predicted solution:

- $\hat{c}(t) \geq \sum_{k=1}^{m} \left(\max_{t' \leq t : \hat{i}_{t'} \in \mathcal{B}_{j_k}} x_{t'} - \min_{t' \leq t : \hat{i}_{t'} \in \mathcal{B}_{j_k}} x_{t'} \right)$, i.e., the total cost is at least the sum of ranges in the individual non-empty blocks.
- $\hat{c}(t) \geq \frac{1}{2} \cdot \sum_{k=1}^{m-1} \left(\max_{t' \leq t : \hat{i}_{t'} \in \mathcal{B}_{j_k} \cup \mathcal{B}_{j_{k+1}}} x_{t'} - \min_{t' \leq t : \hat{i}_{t'} \in \mathcal{B}_{j_k} \cup \mathcal{B}_{j_{k+1}}} x_{t'} \right)$, i.e., the total cost is at least half the sum of ranges across pairs of contiguous non-empty blocks.

Proof. Note that for any set of elements in arbitrary order, the total cost is at least the difference of the maximum and minimum elements. For the first claim, we use this observation separately for each block and observe that the total cost is at least the sum of costs in each block of the predicted solution. For the second claim, we use the same observation twice, once for elements in $\mathcal{B}_{j_1} \cup \mathcal{B}_{j_2}, \mathcal{B}_{j_3} \cup \mathcal{B}_{j_4}, \ldots$ and again for elements in $\mathcal{B}_{j_2} \cup \mathcal{B}_{j_3}, \mathcal{B}_{j_4} \cup \mathcal{B}_{j_5}, \ldots$. ◁

Now, observe that the first phase incurs two types of costs: those within a block and those across consecutive non-empty blocks. Let the first phase end at time t. The first type of cost is incurred for $\leq n/r - 1$ pairs of elements in any block j, and the cost for each pair is at most $\max_{t' \leq t : \hat{i}_{t'} \in \mathcal{B}_j} x_{t'} - \min_{t' \leq t : \hat{i}_{t'} \in \mathcal{B}_j} x_{t'}$. By the first property in Claim 6, this adds up to at most $\hat{c}(t) \cdot (n/r - 1)$ across all the non-empty blocks at time t. The second type of cost is incurred for at most $m - 1$ pairs and is at most $\max_{t' \leq t : \hat{i}_{t'} \in \mathcal{B}_{j_k} \cup \mathcal{B}_{j_{k+1}}} x_{t'} - \min_{t' \leq t : \hat{i}_{t'} \in \mathcal{B}_{j_k} \cup \mathcal{B}_{j_{k+1}}} x_{t'}$ for the k-th pair. By the second property in Claim 6, this adds to at most $2\hat{c}(t)$ across all the non-empty blocks at time t. It follows that the total cost for the first phase is at most $\hat{c}(t) \cdot (n/r + 1)$.

To complete the proof of Theorem 2, we consider two situations. If the second phase is not invoked, i.e., $(n/r) \cdot \hat{c} \leq r \cdot c_{\text{opt}}$, then the total cost is at most

$$\hat{c}(n) \cdot \left(\frac{n}{r} + 1 \right) = O\left(\hat{c} \cdot \frac{n}{r} \right) .$$

On the other hand, if the second phase starts at time $t+1$, then $\hat{c}(t) \cdot n/r \leq r \cdot c_{\text{opt}}$ and $\hat{c}(t+1) \cdot n/r > r \cdot c_{\text{opt}}$, i.e., $\hat{c} \cdot n/r > r \cdot c_{\text{opt}}$. Then, the total cost is

$$\begin{aligned} \hat{c}(t) \cdot \left(\frac{n}{r} + 1 \right) + 2r \cdot c_{\text{opt}} + O\left(\sqrt{n} \cdot c_{\text{opt}} \right) &= O\left(\left(r + \sqrt{n} \right) \cdot c_{\text{opt}} \right), \quad \text{since } \hat{c}(t) \cdot \frac{n}{r} \leq r \cdot c_{\text{opt}} \\ &= O(r \cdot c_{\text{opt}}), \qquad \text{since } r = \Omega\left(\sqrt{n} \right) . \end{aligned}$$

In both cases, we have established that the cost of the algorithm is $O\left(\min(\hat{c} \cdot n/r, r \cdot c_{\text{opt}}) \right)$.

4 Online TSP with Predictions on the Uniform Metric

In this section, we consider online TSP on the uniform metric on k points denoted $v_1, v_2, \ldots, v_k$. We assume w.l.o.g. (by scaling) that the distance between any pair of points, v_i, v_j for $i \neq j$, is 1. In an optimal solution, all elements corresponding to any point v_i are stored consecutively. For simplicity, we start with the case when the input comprises at least one copy of each v_i, resulting in an optimal cost of $k - 1$. Our goal in this section is to prove Theorem 4.

The algorithm that establishes Theorem 4 has two phases. Broadly speaking, these correspond to small and large cost in the predicted solution. The algorithm always starts in the first phase and either switches to the second phase during the instance when a specific condition (that we give below) is met, or never does so if the instance ends without meeting this condition. As before, let us denote by $\hat{c}$ the cost of the predicted solution.

First Phase. The algorithm would ideally like to follow the predictions in this phase. However, naïvely doing so is not robust. This can be illustrated even with $k = 2$, i.e., points 0 and 1 in the metric space. In the first half of the instance, the prediction places 0 in alternate cells of the array. If the algorithm also does so, then the instance now gives 1 for all the remaining elements resulting in an $\Omega(n)$ cost for the algorithm. (The cost of the prediction is also $\Omega(n)$, but this is immaterial for robustness.) This instance illustrates that the algorithm should place identical elements in contiguous cells as much as possible, while also ensuring that the cost of the solution does not increase with respect to that of the predicted solution. To do this systematically, we define a permutation σ_t that evolves with time t. Intuitively, this permutation maps the predicted sequence to a hybrid sequence that agrees with the algorithm's partially filled array at any time. Among other things, this allows the algorithm to interpret a new prediction that points to an empty index in the predicted sequence but might point to an already filled array cell in the algorithm's solution.

We describe the steps of the algorithm and the permutation σ_t below. We use i_t to denote the index of array A where element x_t is inserted, i.e., $A[i_t] = x_t$. First, consider $t = 1$. Let σ_1 be the identity permutation, i.e., $\sigma_1(i) = i$. On receiving x_1 and its predicted index $\hat{i}_1$, the algorithm sets $i_1 := \sigma_1(\hat{i}_1) = \hat{i}_1$. The algorithm then inserts x_1 at $A[i_1] = A[\hat{i}_1]$. In other words, the algorithm inserts the first element at the index given by the predicted solution. Now, we extend the above steps to arbitrary $t > 1$. On receiving element x_t and its predicted index $\hat{i}_t$, we set a temporary index $\tilde{i}_t := \sigma_{t-1}(\hat{i}_t)$. The following key property holds:

▶ **Lemma 7** (Backward Compatibility). *Fix any $t \geq 1$. For any $t' \leq t$, it holds that $\sigma_t(\hat{i}_{t'}) = i_{t'}$.*

Proof. The proof is by induction on t. For the base case, consider $t = 1$. We only need to establish the lemma for $t' = 1$. We have that σ_1 is the identity map. Therefore, $\sigma_1(\hat{i}_1) = \hat{i}_1$. To complete the base case, recall that the algorithm sets $i_1 = \hat{i}_1$.

We now prove the inductive case. Recall the definition of σ_t from σ_{t-1} described in Section 4 after the statement of Lemma 7. We have the following properties:

- If $\ell, r \neq \perp$, then $\sigma_t(i) = \sigma_{t-1}(i)$ for all i such that $\sigma_{t-1}(i) \leq \ell$ and $\sigma_{t-1}(i) \geq r$.
- If $\ell = \perp$, then $\sigma_t(i) = \sigma_{t-1}(i)$ for all i such that $\sigma_{t-1}(i) \geq r$.
- If $r = \perp$, then $\sigma_t(i) = \sigma_{t-1}(i)$ for all i such that $\sigma_{t-1}(i) \leq \ell$.

By the definitions of ℓ and r, the algorithm's array does not contain any of $x_1, x_2, \ldots, x_{t-1}$ between $A[\ell + 1]$ and $A[r - 1]$, both inclusive. Therefore, $\sigma_t(i) = \sigma_{t-1}(i)$ for all i such that $i_{t'} = \sigma_{t-1}(i)$ for some $t' < t$. Therefore, the lemma holds for all $t' < t$ by the inductive hypothesis.

We now consider the element x_t. We need to show that $\sigma_t(\hat{i}_t) = i_t$. Recall that $\sigma_{t-1}(\hat{i}_t) = \tilde{i}_t$. We consider the three cases in the algorithm separately:

- In the first case, $i_t = \ell + 1$ and the algorithm also sets $\sigma_t(\hat{i}_t) = \ell + 1$ since $\sigma_{t-1}(\hat{i}_t) = \tilde{i}_t$.
- In the second case, $i_t = r - 1$ and the algorithm also sets $\sigma_t(\hat{i}_t) = r - 1$ since $\sigma_{t-1}(\hat{i}_t) = \tilde{i}_t$.
- Finally, in the third case, $i_t = \tilde{i}_t$ and the algorithm also sets $\sigma_t(\hat{i}_t) = \sigma_{t-1}(\hat{i}_t) = \tilde{i}_t$.

We have now established the Backward Compatibility property for all $t' \leq t$, which completes the proof of the lemma. ◀

In other words, for all elements up to x_t, the algorithm's solution agrees with the prediction after the permutation σ_t is applied to the latter.

This property ensures that $A[\tilde{i}_t]$ is not occupied by $x_1, x_2, \ldots, x_{t-1}$, i.e., $A[\tilde{i}_t]$ is empty in the algorithm's solution when the algorithm is inserting element x_t. (We will prove this formally later.) Let ℓ and r represent the closest occupied array indices in the algorithm's solution to the left and right of $\tilde{i}_t$ respectively. Formally,

$$\ell = \max\{i : i < \tilde{i}_t \text{ and } \exists t' < t \text{ s.t. } A[i] = x_{t'}\}$$
$$r = \min\{i : i > \tilde{i}_t \text{ and } \exists t' < t \text{ s.t. } A[i] = x_{t'}\}.$$

If ℓ (resp., r) is undefined because the set we are maximizing (resp., minimizing) over is empty, then we say $\ell = \perp$ (resp., $r = \perp$). Let the element x_t be the point v_j in the metric space. There are three cases (see Figure 2 for illustration):

- If $A[\ell] = v_j$, then we set $i_t = \ell + 1$ and assign x_t to $A[\ell + 1]$. The intuition is that we would like to keep the two array cells containing v_j contiguous while deviating minimally from the predicted solution.

  To define σ_t from σ_{t-1}, we compose the latter with the following permutation:

  $$f_t(i) := \begin{cases} i + 1 & \text{if } \ell + 1 \leq i \leq \tilde{i}_t - 1 \\ \ell + 1 & \text{if } i = \tilde{i}_t \\ i & \text{otherwise.} \end{cases}$$

  Thus, $\sigma_t := f_t \circ \sigma_{t-1}$, where f_t maps the index $\tilde{i}_t$ to $\ell + 1$ and to make room for it, pushes each index in the range $[\ell + 1, \tilde{i}_t - 1]$ to the right by one spot. Note that remapping $\tilde{i}_t$ to $\ell + 1$ is necessary to ensure backward compatibility.

  If $A[\ell] \neq v_j$ (including $A[\ell] = \perp$), we go the second case.
- If $A[r] = v_j$, then we set $i_t = r - 1$ and assign x_t to $A[r - 1]$. The intuition is again the same as the previous case: we would like to keep the two array cells containing v_j contiguous while deviating minimally from the predicted solution.

  We again define σ_t from σ_{t-1} by composing the latter with a permutation:

  $$f_t(i) := \begin{cases} r - 1 & \text{if } i = \tilde{i}_t \\ i - 1 & \text{if } \tilde{i}_t + 1 \leq i \leq r - 1 \\ i & \text{otherwise.} \end{cases}$$

  Thus, $\sigma_t := f_t \circ \sigma_{t-1}$, where f_t maps the index $\tilde{i}_t$ to $r - 1$ and to make room for it, pushes each index in the range $[\tilde{i}_t + 1, r - 1]$ to the left by one spot. As in the previous case, note that remapping $\tilde{i}_t$ to $r - 1$ is necessary to ensure backward compatibility.

  If $A[r] \neq v_j$ (including $A[r] = \perp$), and we go the third case.
- We arrive at this case if neither ℓ nor r contains v_j. In this case, the algorithm has no incentive to try to keep x_t contiguous with either ℓ or r. So, in this case, we assign x_t to $A[\tilde{i}_t]$ and keep the permutation unchanged, $\sigma_t := \sigma_{t-1}$.

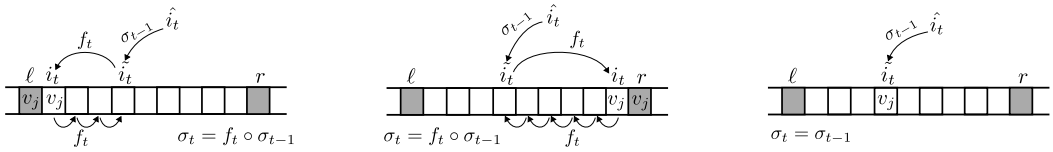

Figure 2 Inserting element $x_t = v_j$ into the array, occupied cells shown in gray. Predicted index $\hat{i}_t$ is first mapped to temporary index $\tilde{i}_t = \sigma_{t-1}(\hat{i}_t)$. Left: if $A[\ell] = v_j$, then element is inserted at $i_t = \sigma_t(\hat{i}_t) = \ell + 1$. Middle: otherwise, if $A[r] = v_j$, then element is inserted at $i_t = \sigma_t(\hat{i}_t) = r - 1$. Right: if neither holds, then element is inserted at $i_t = \sigma_t(\hat{i}_t) = \sigma_{t-1}(\hat{i}_t) = \tilde{i}_t$. Permutation σ_t is obtained by augmenting σ_{t-1} with cycle f_t, shown with horizontal arrows.

Transition from First to Second Phase. We now define the condition that causes the algorithm to switch from the first to the second phase. To define this condition, we introduce a new parameter of the predicted sequence that lower bounds the cost of the prediction $\hat{c}$. We call this parameter *switch count* and denote it $\tilde{c}_t$. To define switch count, we introduce the operation of *condensing* an array, similarly to Section 3.

▶ **Definition 8** (Condensing an array)**.** In this operation, we are given an array A of size n and a selection of $n' \leq n$ indices in $[n]$, denoted $S = s_1 < s_2 < \cdots < s_{n'}$. We say that an array B of size n' is formed by condensing A with respect to S if $B[k] = A[s_k]$ for all $k \in [n']$. In other words, the k-th cell in the array B is a copy of the contents in the k-th cell in A when only considering the indices in S.

The switch count $\tilde{c}_t$ at time t is defined as the cost of the predicted sequence with respect to the predicted indices for $x_1, x_2, \ldots, x_t$, i.e., $S = \{\hat{i}_{t'} : t' \leq t\}$. It is easy to see that condensing an array with respect to any index set cannot increase its cost (we show this formally later). This means that at any time t, the switch count $\tilde{c}_t$ is a lower bound on the cost of the predicted solution $\hat{c}$. The algorithm terminates the first phase and moves to the second phase when $\tilde{c}_t \geq k \lg n$. This tallies with the intuition that the first phase (that uses predictions) should give way to the second phase (that ignores predictions) when the cost of the predicted sequence becomes large.

Second Phase. The algorithm in the second phase ignores predictions and runs the $O(\log n)$-competitive algorithm from [2] on the empty cells to insert $x_{t+1}, x_{t+2}, \ldots, x_n$. To make this formal, we first condense the array A with respect to the empty cells, i.e., $S = [n] \setminus \{i_{t'} : t' \leq t\}$, to obtain a virtual array B. We then run the algorithm from [2] on this virtual array, and invert the mapping produced when condensing A to B to actually insert the element in A.

We establish Theorem 4 by analysing the algorithm given above.

Analysis of the First Phase. As a simple corollary of the Backward Compatibility property (Lemma 7), we get that $A[\tilde{i}_t]$ is not occupied by any of $x_1, x_2, \ldots, x_{t-1}$, which validates the definitions of ℓ and r. Moreover, it also implies that the algorithm produces a valid solution, i.e., there are no collisions.

▶ **Corollary 9.** *Fix any $t > 1$. It holds that $\tilde{i}_t \neq i_{t'}$ and also $i_t \neq i_{t'}$ for any $t' < t$.*

Proof. Fix any $t' < t$. Note that since the predictions form a permutation, we have $\hat{i}_{t'} \neq \hat{i}_t$. Therefore,

$$\begin{aligned} \tilde{i}_t = \sigma_{t-1}(\hat{i}_t) &\neq \sigma_{t-1}(\hat{i}_{t'}) && \text{(since } \sigma_{t-1} \text{ is a permutation and } \hat{i}_{t'} \neq \hat{i}_t) \\ &= i_{t'} && \text{(by Backward Compatibility).} \end{aligned}$$

This proves the first claim in the lemma. We now show the second claim. Note that i_t is one of the three indices $\ell+1$, $r-1$, or $\tilde{i}_t$. By the definitions of ℓ and r, the array A is empty between indices $\ell+1$ and $r-1$, both inclusive, after the algorithm has inserted the first $t-1$ elements. (If $\ell = \perp$, then A is empty between indices 1 and $r-1$. Similarly, if $r = \perp$, then A is empty between $\ell+1$ and n.) It follows that $i_t \neq i_{t'}$. ◀

So far, we have established that the algorithm avoids collisions. We now bound the cost of the solution produced in the first phase. Recall that the algorithm maintains a parameter called the switch cost $\tilde{c}_t$ of the predicted solution. First, we show that the switch cost is a lower bound on the cost of the predicted solution. We prove a more general lemma:

▶ **Lemma 10.** *Suppose array B of size n' is formed by condensing a larger array A of size $n \geq n'$ with respect to any set of indices $S = s_1 < s_2 < \cdots < s_{n'}$. Then, the cost of B is a lower bound of the cost of A.*

Proof. We show that whenever B incurs a cost, so too does A. Suppose B incurs a cost of 1 between its k-th and $k+1$-st cell, i.e., $B[k] \neq B[k+1]$. Then, $A[s_k] = B_k \neq A[s_{k+1}] = B_{k+1}$. No matter what the contents of the cells between $A[s_k]$ and $A[s_{k+1}]$ are, A incurs at least a cost of 1 summing over all consecutive pairs of cells in the range s_k to s_{k+1}. ◀

Since $\tilde{c}_t$ is the cost of an array obtained by condensing the predicted solution with respect to its indices for $x_1, x_2, \ldots, x_t$, it follows that:

▶ **Corollary 11.** *At any time t, the parameter $\tilde{c}_t$ is a lower bound on the cost $\hat{c}$ of the predicted solution.*

Next, we upper bound the cost of the algorithm's solution in the first phase against the parameter $\tilde{c}_t$. We define the cost of the first phase as the number of pairs of consecutive non-identical elements in array A at the end of the first phase. Note that we are not considering the cost of adjacent elements inserted by the two different phases yet – this will be considered later in the analysis.

▶ **Lemma 12.** *Fix any $t \geq 1$. The cost of the algorithm's solution at time t is at most $\tilde{c}_t$.*

Before proving this lemma, we introduce some new notation. We denote the predicted sequence by the function $\hat{\pi}$ that maps t to $\hat{i}_t$. The sequence obtained by applying the permutation σ_t to $\hat{\pi}$ is then denoted $\hat{\pi}_t$, i.e., $\hat{\pi}_t(t') = \sigma_t(i_{t'})$. We also denote by $\tilde{c}_{t'}(\hat{\pi}_t)$ the cost of the sequence $\hat{\pi}_t$ on the elements $x_1, x_2, \ldots, x_{t'}$. In contrast, recall that $\tilde{c}_{t'} = \tilde{c}_{t'}(\hat{\pi})$ denotes the cost of the predicted sequence on the same elements. We have the following claim, which essentially says that applying the permutation σ_t to $\hat{\pi}$ does not increase the cost $\tilde{c}_{t'}$ for any prefix of elements $x_1, x_2, \ldots, x_{t'}$ such that $t' \geq t$.

▷ Claim 13. Fix any $t \geq 1$ and any prefix of elements $x_1, x_2, \ldots, x_{t'}$ such that $t' \geq t$. Then, $\tilde{c}_{t'}(\hat{\pi}_t) \leq \tilde{c}_{t'}$.

Proof. We proceed by induction on t. For the base case $t = 1$, the permutation σ_1 is the identity map; hence, $\hat{\pi}_1 = \hat{\pi}$ and the lemma holds trivially.

We now consider the inductive case. Fix any t. By the inductive hypothesis, $\tilde{c}_{t'}(\hat{\pi}_{t-1}) \leq \tilde{c}_{t'}$ for all $t' \geq t-1$ (and therefore, for all $t' \geq t$). In contrast, we want to show $\tilde{c}_{t'}(\hat{\pi}_t) \leq \tilde{c}_{t'}$ for all $t' \geq t$. Therefore, it suffices to show that $\tilde{c}_{t'}(\hat{\pi}_t) \leq \tilde{c}_{t'}(\hat{\pi}_{t-1})$ for any $t' \geq t$.

Fix any $t' \geq t$. If $\sigma_{t-1} = \sigma_t$, then we have $\hat{\pi}_t = \hat{\pi}_{t-1}$, and therefore, $\tilde{c}_{t'}(\hat{\pi}_t) = \tilde{c}_{t'}(\hat{\pi}_{t-1})$. So, we only consider the two cases in the algorithm where $\sigma_{t-1} \neq \sigma_t$:

- In the first case, we have that $A[\ell] = x_t$ and the algorithm sets $i_t = \ell + 1$. In this case, the sequences defined by $\{\hat{\pi}_t(t'') : t'' \in [n]\}$ and $\{\hat{\pi}_{t-1}(t'') : t'' \in [n]\}$ only differ on the indices $\ell+1, \ell+2, \ldots, \tilde{i}_t$. Excluding $\tilde{i}_t$ which is occupied in $\hat{\pi}_{t-1}$ by element x_t, we denote the (possibly empty) set of other occupied indices at time t' in this range by

  $$S_{\text{prev}} := \{i \in [\ell + 1, \tilde{i}_t - 1] : \exists t'' \leq t' \text{ s. t. } \hat{\pi}_{t-1}(t'') = i\}.$$

  There are two subcases. The first (simpler) subcase is when $S_{\text{prev}} = \varnothing$. I.e., for every t'' such that $\hat{\pi}_{t-1}(t'') \in [\ell + 1, \tilde{i}_t - 1]$, we have $t'' > t'$. In this case, after condensing the cells occupied by $x_1, x_2, \ldots, x_{t'}$ in $\hat{\pi}_{t-1}$ and $\hat{\pi}_t$, we get the same array. Therefore, $\tilde{c}_{t'}(\hat{\pi}_t) = \tilde{c}_{t'}(\hat{\pi}_{t-1})$.
  Now, consider the case when $S_{\text{prev}} \neq \varnothing$. We denote the smallest and largest index in S_{prev} by $s_{\min}$ and $s_{\max}$ respectively. Note that $s_{\min} = s_{\max}$ if $|S_{\text{prev}}| = 1$. We also denote by s_{next} the smallest index $i > \tilde{i}_t$ such that $\hat{\pi}_{t-1}(t'') = i$ for any $t'' \leq t'$. Note that if $r \neq \perp$, then $s_{\text{next}} \in [\tilde{i}_t + 1, r]$. If $r = \perp$, either $s_{\text{next}} \in [\tilde{i}_t + 1, n]$ or $s_{\text{next}} = \perp$ if there for every i such that $\hat{\pi}_{t-1}(t'') = i$, we have that $i \leq \tilde{i}_t$.
  On condensing with respect to the indices of the elements $x_1, x_2, \ldots, x_{t'}$, there are at most three pairs of consecutive indices that have different pairs of elements between $\hat{\pi}_{t-1}$ and $\hat{\pi}_t$. These are:
  - in $\hat{\pi}_{t-1}$, the pairs are $(\ell, s_{\min})$, $(s_{\max}, \tilde{i}_t)$, and $(\tilde{i}_t, s_{\text{next}})$, while
  - in $\hat{\pi}_t$, the pairs are $(\ell, \ell + 1)$, $(\ell + 1, s_{\min})$, and $(s_{\max}, s_{\text{next}})$.

  If $s_{\text{next}} = \perp$, then the last pairs $(\tilde{i}_t, s_{\text{next}})$ and $(s_{\max}, s_{\text{next}})$ are not defined, and the first two pairs are the only differing ones.
  Now, suppose $x_t = v_j$. By the definition of this case in the algorithm, we have $A[\ell] = v_j$. This implies by the Backward Compatibility property (Lemma 7) that $\tilde{c}_{t'}(\hat{\pi}_t)$ does not incur any cost for the pair $(\ell, \ell + 1)$. Next, consider the pair $(\ell + 1, s_{\min})$. If this pair incurs unit cost in $\tilde{c}_{t'}(\hat{\pi}_t)$, then so does the pair $(\ell, s_{\min})$ in $\tilde{c}_{t'}(\hat{\pi}_{t-1})$. Finally, consider the pair $(s_{\max}, s_{\text{next}})$. If this pair incurs unit cost in $\tilde{c}_{t'}(\hat{\pi}_t)$, then so does at least one of the pairs $(s_{\max}, \tilde{i}_t)$ and $(\tilde{i}_t, s_{\text{next}})$ in $\tilde{c}_{t'}(\hat{\pi}_{t-1})$. Overall, this means that $\tilde{c}_{t'}(\hat{\pi}_t) \leq \tilde{c}_{t'}(\hat{\pi}_{t-1})$.
- The second case is symmetric to the first case. In this case, we have that $A[r] = x_t$ and the algorithm sets $i_t = r - 1$. In this case, the sequences defined by $\{\hat{\pi}_t(t'') : t'' \in [n]\}$ and $\{\hat{\pi}_{t-1}(t'') : t'' \in [n]\}$ only differ on the indices $\tilde{i}_t, \tilde{i}_t + 2, \ldots, r - 1$. Excluding $\tilde{i}_t$ which is occupied in $\hat{\pi}_{t-1}$ by element x_t, we denote the (possibly empty) set of other occupied indices at time t' in this range by

  $$S_{\text{next}} := \{i \in [\tilde{i}_t + 1, r - 1] : \exists t'' \leq t' \text{ s. t. } \hat{\pi}_{t-1}(t'') = i\}.$$

  There are two subcases. The first (simpler) subcase is when $S_{\text{next}} = \varnothing$. I.e., for every t'' such that $\hat{\pi}_{t-1}(t'') \in [\tilde{i}_t + 1, r - 1]$, we have $t'' > t'$. In this case, after condensing the cells occupied by $x_1, x_2, \ldots, x_{t'}$ in $\hat{\pi}_{t-1}$ and $\hat{\pi}_t$, we get the same array. Therefore, $\tilde{c}_{t'}(\hat{\pi}_t) = \tilde{c}_{t'}(\hat{\pi}_{t-1})$.
  Now, consider the case when $S_{\text{next}} \neq \varnothing$. We denote the smallest and largest index in S_{next} by $s_{\min}$ and $s_{\max}$ respectively. Note that $s_{\min} = s_{\max}$ if $|S_{\text{next}}| = 1$. We also denote by s_{prev} the largest index $i < \tilde{i}_t$ such that $\hat{\pi}_{t-1}(t'') = i$ for any $t'' \leq t'$. Note that if $\ell \neq \perp$, then $s_{\text{prev}} \in [\ell, \tilde{i}_t - 1]$. If $\ell = \perp$, either $s_{\text{prev}} \in [1, \tilde{i}_t - 1]$ or $s_{\text{prev}} = \perp$ if there for every i such that $\hat{\pi}_{t-1}(t'') = i$, we have that $i \geq \tilde{i}_t$.
  On condensing with respect to the indices of the elements $x_1, x_2, \ldots, x_{t'}$, there are at most three pairs of consecutive indices that have different pairs of elements between $\hat{\pi}_{t-1}$ and $\hat{\pi}_t$. These are:
  - in $\hat{\pi}_{t-1}$, the pairs are $(s_{\text{prev}}, \tilde{i}_t)$, $(\tilde{i}_t, s_{\min})$, and $(s_{\min}, s_{\max})$, while
  - in $\hat{\pi}_t$, the pairs are $(s_{\text{prev}}, s_{\min})$, $(s_{\max}, r - 1)$, and $(r - 1, r)$.

If $s_{\text{prev}} = \perp$, then the first pairs $(s_{\text{prev}}, \tilde{i}_t)$ and $(s_{\text{prev}}, s_{\min})$ are not defined, and the last two pairs are the only differing ones.

Now, suppose $x_t = v_j$. By the definition of this case in the algorithm, we have $A[r] = v_j$. This implies by the Backward Compatibility property (Lemma 7) that $\tilde{c}_{t'}(\hat{\pi}_t)$ does not incur any cost for the pair $(r-1, r)$. Next, consider the pair $(s_{\max}, r-1)$. If this pair incurs unit cost in $\tilde{c}_{t'}(\hat{\pi}_t)$, then so does the pair $(s_{\max}, r)$ in $\tilde{c}_{t'}(\hat{\pi}_{t-1})$. Finally, consider the pair $(s_{\text{prev}}, s_{\min})$. If this pair incurs unit cost in $\tilde{c}_{t'}(\hat{\pi}_t)$, then so does at least one of the pairs $(s_{\text{prev}}, \tilde{i}_t)$ and $(\tilde{i}_t, s_{\min})$ in $\tilde{c}_{t'}(\hat{\pi}_{t-1})$. Overall, this means that $\tilde{c}_{t'}(\hat{\pi}_t) \leq \tilde{c}_{t'}(\hat{\pi}_{t-1})$.

This completes the proof of the claim. ◁

We use this claim to prove Lemma 12.

Proof of Lemma 12. Suppose the first phase of the algorithm ends at time t. Then, using $t' = t$ in Claim 13, we can conclude that $\tilde{c}_t(\hat{\pi}_t) \leq \tilde{c}_t$.

But, note that by Backward Compatibility (Lemma 7), the algorithm's array A agrees with $\hat{\pi}_t$ on elements $x_1, x_2, \ldots, x_t$. Now, if we condense the algorithm's array A with respect to the indices occupied by $x_1, x_2, \ldots, x_t$, then the cost of the resulting virtual array is $\tilde{c}_t(\hat{\pi}_t) \leq \tilde{c}_t$. Therefore, the algorithm's cost in the first phase is at most $\tilde{c}_t$. ◀

By Corollary 11, the above lemma shows that if the algorithm never enters the second phase, then its cost is no more than that of the predicted solution. But, if the algorithm enters the second phase, then we need to also bound two other costs: the cost due to consecutive non-identical elements inserted in A by the two different phases, and the cost due to consecutive non-identical elements inserted in A by the second phase alone. We call the former *interphase cost* and bound it first.

Interphase Cost. We start with the following claim, which uses the terminology of gaps and guards from the previous section:

▷ Claim 14. Fix any time t and consider the array A produced by the algorithm after insertion of $x_1, x_2, \ldots, x_t$. If a gap has a left and a right guard, then the array cells corresponding to the left and right guards are occupied by non-identical elements, i.e., are different points in the metric space.

Proof. The proof is by induction over time. Clearly, the claim holds after $t = 1$ since there is no gap with a left and a right guard. Now, suppose the claim holds after time $t-1$. The only way the claim can be violated at time t is if the insertion of x_t violates it. We consider each of the three cells that x_t can be inserted at:

- Suppose x_t is inserted at $A[\ell+1]$. If $A[r] = \perp$, then x_t is the left guard of a gap that does not have a right guard, and hence the claim cannot be violated. If $A[r] \neq \perp$, then the claim can only be violated if $A[\ell+1] = A[r]$. But, this would mean $A[\ell] = A[r]$, which violates the inductive hypothesis.
- Suppose x_t is inserted at $A[r-1]$. If $A[\ell] = \perp$, then x_t is the right guard of a gap that does not have a left guard, and hence the claim cannot be violated. If $A[\ell] \neq \perp$, then the claim can only be violated if $A[\ell] = A[r-1]$. But, this would mean $A[\ell] = A[r]$, which violates the inductive hypothesis.
- Finally, if x_t is inserted at $A[\tilde{i}_t]$, then the claim can only be violated if $A[\ell] = A[\tilde{i}_t]$ or $A[\tilde{i}_t] = A[r]$. But, for either of these, we would have been in one of the first two cases.

This completes the proof of the claim. ◁

We can use this claim to bound the interphase cost. In fact, we can bound the sum of the interphase cost and the cost incurred by the first phase. Suppose that x_t is the last element processed in phase one of the algorithm.

▶ **Lemma 15.** *The sum of the interphase cost and the cost incurred by the first phase of the algorithm is at most* $2\tilde{c}_t + 2$.

Proof. Suppose the total number of right and left guards at the end of the first phase at time t is g. (If an element is both the right guard of a gap and the left guard of the next gap, we count it twice in g.) Except possibly the two gaps at the two ends, every other gap has a left and a right guard, which are non-identical elements by Claim 14. After condensing on the occupied cells, each such pair of guards incurs unit cost in $\tilde{c}_t(\hat{\pi}_t)$. This total cost is $\geq \frac{g-2}{2}$. Additionally, every pair of consecutive non-identical elements inserted in the first phase also contributes unit cost to $\tilde{c}_t(\hat{\pi}_t)$, and this cost is distinct from that contributed by the guards. Let us denote this cost c_1. Then, $(g-2)/2 + c_1 \leq \tilde{c}_t(\hat{\pi}_t)$, i.e.,

$$g + c_1 \leq 2\tilde{c}_t(\hat{\pi}_t) + 2 \leq 2\tilde{c}_t + 2, \text{ by Claim 13.}$$

The interphase cost is at most g and the first phase cost is c_1. The lemma follows. ◀

Analysis of the Second Phase. We are only left to bound the cost of consecutive non-identical elements, both of which are inserted in the second phase. This is at most the cost incurred by the algorithm of [2] on the virtual array obtained by condensing A at the beginning of the first phase. The following lemma follows immediately from the $O(\log n)$-competitiveness of the algorithm of [2]:

▶ **Lemma 16.** *The cost of the second phase of the algorithm is at most* $O(k \log n)$.

Theorem 4 now follows by adding the costs of the first and second phases along with the interphase cost, using Lemmas 12, 15, and 16.

A minor shortcoming of the above algorithm is that it assumes knowledge of k (recall that the condition for switching to the second phase depends on k). We remove this assumption in a more general setting via Theorem 5 (Appendix B). Recall that Theorem 5 yields an algorithm that is consistent with the best of h prediction sequences revealed simultaneously, and the resulting algorithm is oblivious to k. In particular, for $h = 2$ and using a prediction sequence, resp., the output of the algorithm of [2], we obtain Theorem 4 as a corollary.

5 Closing Remarks

Like many online problems, online sorting/TSP suffers from strong lower bounds in the worst-case model that act as a deterrent to the design of useful and interesting algorithms. To obviate this shortcoming, we study the popular predictions setting in this paper where the online algorithm is additionally provided a predicted solution in the form of a permutation of the elements. We derive tight upper and lower bounds on the tradeoff between consistency and robustness for online sorting/TSP with predictions. Interestingly, we show that unlike the vast majority of online problems, robustness is not "lossless" for online sorting/TSP, i.e., the consistency bounds degrade by super-constant terms when one also demands robustness of the solution. We then consider online TSP on the uniform metric and show a qualitative distinction: for this problem, one can indeed obtain optimal robustness while being constant-consistent. We further generalize this observation to the case of multiple predictions.

Our work raises several interesting questions. Are our multiple predictions results (deterministic and randomized) for the uniform metric tight? Is there a natural parametrization of metric spaces, e.g., the aspect ratio, that explains the contrasting behavior of online TSP with predictions on the uniform and line metrics? Can the assumption of known c_{opt} in online TSP be removed without degrading the results? The natural idea for the latter question would be to enclose the algorithm in a guess-and-double loop that guesses the value of c_{opt} and doubles the guess every time the current guess is breached. Implementing this idea, however, does not seem straightforward since switching to a new iteration implies a penalty for re-mapping the empty cells to a virtual array. Indeed, we can show that the natural implementation of our online sorting/TSP algorithms in a guess-and-double framework does not work, and we leave this as an interesting open question.

A broader agenda might include other forms of predictions for online sorting/TSP (for instance, removing the assumption that the predictions form a valid permutation) or a comprehensive understanding of online sorting/TSP in other beyond worst-case settings, not necessarily using predictions.

References

1 Anders Aamand, Mikkel Abrahamsen, Lorenzo Beretta, and Linda Kleist. Online sorting and translational packing of convex polygons. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 1806–1833. SIAM, 2023. doi:10.1137/1.9781611977554.CH69.

2 Mikkel Abrahamsen, Ioana O. Bercea, Lorenzo Beretta, Jonas Klausen, and László Kozma. Online sorting and online TSP: randomized, stochastic, and high-dimensional. In Timothy M. Chan, Johannes Fischer, John Iacono, and Grzegorz Herman, editors, *32nd Annual European Symposium on Algorithms, ESA 2024, September 2-4, 2024, Royal Holloway, London, United Kingdom*, volume 308 of *LIPIcs*, pages 5:1–5:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.ESA.2024.5.

3 Keerti Anand, Rong Ge, Amit Kumar, and Debmalya Panigrahi. Online algorithms with multiple predictions. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 582–598. PMLR, 2022. URL: https://proceedings.mlr.press/v162/anand22a.html.

4 Giorgio Ausiello, Esteban Feuerstein, Stefano Leonardi, Leen Stougie, and Maurizio Talamo. Algorithms for the on-line travelling salesman 1. *Algorithmica*, 29(4):560–581, 2001. doi:10.1007/S004530010071.

5 Yossi Azar, Andrei Z. Broder, and Mark S. Manasse. On-line choice of on-line algorithms. In Vijaya Ramachandran, editor, *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas, USA*, pages 432–440. ACM/SIAM, 1993. URL: http://dl.acm.org/citation.cfm?id=313559.313847.

6 Yossi Azar, Debmalya Panigrahi, and Or Vardi. Nearly tight bounds for the online sorting problem. *CoRR*, abs/2508.14287, 2025. doi:10.48550/arXiv.2508.14287.

7 Brenda S. Baker and Jerald S. Schwarz. Shelf algorithms for two-dimensional packing problems. *SIAM J. Comput.*, 12(3):508–525, 1983. doi:10.1137/0212033.

8 Maria-Florina Balcan, Tuomas Sandholm, and Ellen Vitercik. Generalization in portfolio-based algorithm selection. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 12225–12232. AAAI Press, 2021. doi:10.1609/AAAI.V35I14.17451.

9 Étienne Bamas, Andreas Maggiori, and Ola Svensson. The primal-dual method for learning augmented algorithms. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/e834cb114d33f729dbc9c7fb0c6bb607-Abstract.html.

10 Christian Bertram. Online metric TSP. In Anne Benoit, Haim Kaplan, Sebastian Wild, and Grzegorz Herman, editors, *33rd Annual European Symposium on Algorithms, ESA 2025, Warsaw, Poland, September 15-17, 2025*, volume 351 of *LIPIcs*, pages 80:1–80:9. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.ESA.2025.80.

11 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 Transactions on Algorithms (TALG)*, 17(1):1–58, 2020. doi:10.1145/3422362.

12 Don Coppersmith and Prabhakar Raghavan. Multidimensional on-line bin packing: Algorithms and worst-case analysis. *Operations Research Letters*, 8(1):17–20, 1989. doi:10.1016/0167-6377(89)90027-8.

13 János Csirik, J. B. G. Frenk, and Martine Labbé. Two-dimensional rectangle packing: On-line methods and results. *Discret. Appl. Math.*, 45(3):197–204, 1993. doi:10.1016/0166-218X(93)90009-D.

14 János Csirik and André van Vliet. An on-line algorithm for multidimensional bin packing. *Oper. Res. Lett.*, 13(3):149–158, 1993. doi:10.1016/0167-6377(93)90004-Z.

15 János Csirik and Gerhard J. Woeginger. Shelf algorithms for on-line strip packing. *Inf. Process. Lett.*, 63(4):171–175, 1997. doi:10.1016/S0020-0190(97)00120-8.

16 Michael Dinitz, Sungjin Im, Thomas Lavastida, Benjamin Moseley, Aidin Niaparast, and Sergei Vassilvitskii. Binary search with distributional predictions. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang, editors, *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL: http://papers.nips.cc/paper_files/paper/2024/hash/a4b293979b8b521e9222d30c40246911-Abstract-Conference.html.

17 Michael Dinitz, Sungjin Im, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Algorithms with prediction portfolios. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL: http://papers.nips.cc/paper_files/paper/2022/hash/7f9220f90cc85b0da693643add6618e6-Abstract-Conference.html.

18 Dimitris A. Fotakis, Andreas Kalavas, Charalampos Platanos, and Thanos Tolias. A polylogarithmic algorithm for stochastic online sorting. *CoRR*, abs/2508.12527, 2025. doi:10.48550/arXiv.2508.12527.

19 Gábor Galambos. A 1.6 lower-bound for the two-dimensional on-line rectangle bin-packing. *Acta Cybern.*, 10(1-2):21–24, 1991. URL: https://cyber.bibl.u-szeged.hu/index.php/actcybern/article/view/3389.

20 Gábor Galambos and André van Vliet. Lower bounds for 1-, 2- and 3-dimensional on-line bin packing algorithms. *Computing*, 52(3):281–297, 1994. doi:10.1007/BF02246509.

21 Sreenivas Gollapudi and Debmalya Panigrahi. Online algorithms for rent-or-buy with expert advice. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 2319–2327. PMLR, 2019. URL: http://proceedings.mlr.press/v97/gollapudi19a.html.

22 Xin Han, Francis Y. L. Chin, Hing-Fung Ting, Guochuan Zhang, and Yong Zhang. A new upper bound 2.5545 on 2D online bin packing. *ACM Trans. Algorithms*, 7(4):50:1–50:18, 2011. doi:10.1145/2000807.2000818.

23 Xin Han, Kazuo Iwama, Deshi Ye, and Guochuan Zhang. Strip packing vs. bin packing. In Ming-Yang Kao and Xiang-Yang Li, editors, *Algorithmic Aspects in Information and Management, Third International Conference, AAIM 2007, Portland, OR, USA, June 6-8, 2007, Proceedings*, volume 4508 of *Lecture Notes in Computer Science*, pages 358–367. Springer, 2007. doi:10.1007/978-3-540-72870-2_34.

24 Chen-Yu Hsu, Piotr Indyk, Dina Katabi, and Ali Vakilian. Learning-based frequency estimation algorithms. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=r1lohoCqY7.

25 Yang Hu. Nearly optimal bounds for stochastic online sorting. *CoRR*, abs/2508.07823, 2025. doi:10.48550/arXiv.2508.07823.

26 Makoto Imase and Bernard M. Waxman. Dynamic steiner tree problem. *SIAM J. Discret. Math.*, 4(3):369–384, 1991. doi:10.1137/0404033.

27 Eniko Kevi and Kim Thang Nguyen. Online covering with multiple experts. *CoRR*, abs/2312.14564, 2023. doi:10.48550/arXiv.2312.14564.

28 Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein, editors, *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 489–504. ACM, 2018. doi:10.1145/3183713.3196909.

29 Honghao Lin, Tian Luo, and David P. Woodruff. Learning augmented binary search trees. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 13431–13440. PMLR, 2022. URL: https://proceedings.mlr.press/v162/lin22f.html.

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

31 Nicole Megow, Martin Skutella, José Verschae, and Andreas Wiese. The power of recourse for online mst and tsp. *SIAM Journal on Computing*, 45(3):859–880, 2016. doi:10.1137/130917703.

32 Michael Mitzenmacher. A model for learned bloom filters and optimizing by sandwiching. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 462–471, 2018. URL: https://proceedings.neurips.cc/paper/2018/hash/0f49c89d1e7298bb9930789c8ed59d48-Abstract.html.

33 Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. In Tim Roughgarden, editor, *Beyond the Worst-Case Analysis of Algorithms*, pages 646–662. Cambridge University Press, 2020. doi:10.1017/9781108637435.037.

34 Jubayer Nirjhor and Nicole Wein. Improved online sorting. In Jannik Matuschke and José Verschae, editors, *Approximation and Online Algorithms - 23rd International Workshop, WAOA 2025, Warsaw, Poland, September 18-19, 2025, Proceedings*, volume 16077 of *Lecture Notes in Computer Science*, pages 187–197. Springer, 2025. doi:10.1007/978-3-032-06706-7_13.

35 Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ML predictions. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 9684–9693, 2018. URL: https://proceedings.neurips.cc/paper/2018/hash/73a427badebe0e32caa2e1fc7530b7f3-Abstract.html.

36 Daniel J. Rosenkrantz, Richard Edwin Stearns, and Philip M. Lewis II. An analysis of several heuristics for the traveling salesman problem. *SIAM J. Comput.*, 6(3):563–581, 1977. doi:10.1137/0206041.

37 Steven S. Seiden and Rob van Stee. New bounds for multidimensional packing. *Algorithmica*, 36(3):261–293, 2003. doi:10.1007/S00453-003-1016-7.
38 Kapil Vaidya, Eric Knorr, Michael Mitzenmacher, and Tim Kraska. Partitioned learned bloom filters. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL: https://openreview.net/forum?id=6BRLOfrMhW.
39 André van Vliet. An improved lower bound for on-line bin packing algorithms. *Inf. Process. Lett.*, 43(5):277–284, 1992. doi:10.1016/0020-0190(92)90223-I.
40 Deshi Ye, Xin Han, and Guochuan Zhang. A note on online strip packing. *J. Comb. Optim.*, 17(4):417–423, 2009. doi:10.1007/S10878-007-9125-X.

A Online TSP with Predictions on General Metric Spaces

In this section, we extend the upper bound of Section 3 to online TSP. We assume that the optimal value c_{opt} is known.

▶ **Theorem 3.** *For any $r = \Omega(\sqrt{n})$, there is a deterministic $O(n/r)$-consistent and $O(r)$-robust algorithm for online TSP with predictions (when c_{opt} is known to the algorithm).*

The technical ideas in this section mirror those in Section 3, although the introduction of the metric leads to some technical complications.

We first introduce some notation and preliminary claims. We consider a general metric space $\mathcal{U}$ with a metric $d : \mathcal{U} \times \mathcal{U} \to \mathbb{R}$. Note that for the special case of online sorting we have $\mathcal{U} = [0, 1]$ and $d(x, y) = |x - y|$.

Condensed Cost of a Prediction. Consider a subset of array indices $\mathcal{I} \subseteq [n]$ consisting of s indices $i_1 < i_2 < \cdots < i_s$. Given an array A with elements from $\mathcal{U}$, we let $A[\mathcal{I}]$ denote the array obtained by removing from A all cells not in $\mathcal{I}$, and we define the *cost of A with respect to $\mathcal{I}$* as:

$$c(A[\mathcal{I}]) = \sum_{k=1}^{s-1} d(A[i_k], A[i_{k+1}]) \ .$$

Consider an input sequence $X = x_1, \ldots, x_n$ with predictions $\hat{I} = \hat{i}_1, \ldots, \hat{i}_n$. Let $\hat{A}$ be the resulting array if we follow the predictions, i.e., $\hat{A}[\hat{i}_t] = x_t$ for $1 \leq t \leq n$. Let $\hat{I}_t = \hat{i}_1, \ldots, \hat{i}_t$, for $1 \leq t \leq n$. We define the condensed cost of the predictions at time t as

$$\hat{c}(t) = c(\hat{A}[\hat{I}_t]) \ .$$

We note that $\hat{c}(t)$ is non-decreasing, since $\hat{I}_{t+1} \supseteq \hat{I}_t$ and d is a metric. Finally, we let $\hat{c} = \hat{c}(n)$.

Lower Bounds on the Condensed Cost. For a subset of indices $\mathcal{I} \subseteq [n]$ and a metric d, we define the *diameter* of A with respect to $\mathcal{I}$ as

$$\mathsf{diam}(A[\mathcal{I}]) = \max_{i,j \in \mathcal{I}} d(A[i], A[j]) \ .$$

Similar to Claim 6, we have the following lower bounds:

▷ Claim 17. Let $\mathcal{I}_1, \ldots, \mathcal{I}_m$ be disjoint subsets of $\mathcal{I} \subseteq [n]$ such that all the indices in $\mathcal{I}_k$ are smaller than all the indices in $\mathcal{I}_{k+1}$ for all $1 \leq k < m$. Then, for any array A of size n, we have that:

- $c(A[\mathcal{I}]) \geq \sum_{k=1}^{m} c(A[\mathcal{I}_k]) \geq \sum_{k=1}^{m} \mathsf{diam}(A[\mathcal{I}_k])$,
- $c(A[\mathcal{I}]) \geq \frac{1}{2} \cdot \sum_{k=1}^{m-1} c(A[\mathcal{I}_k \cup \mathcal{I}_{k+1}]) \geq \frac{1}{2} \cdot \sum_{k=1}^{m-1} \mathsf{diam}(A[\mathcal{I}_k \cup \mathcal{I}_{k+1}])$.

Proof. Consider any subset $\mathcal{T} \subseteq [n]$ consisting of indices $i_1 < i_2 < \cdots < i_s$. We claim that $c(A[\mathcal{T}]) \geq \mathsf{diam}(A[\mathcal{T}])$. Namely, let $i_{j_1} \leq i_{j_2}$ with $i_{j_1}, i_{j_2} \in \mathcal{T}$ denote two indices where the diameter is realized, i.e., $\mathsf{diam}(A[\mathcal{T}]) = d(A[i_{j_1}], A[i_{j_2}])$. We then have that:

$$c(A[\mathcal{T}]) = \sum_{k=1}^{s-1} d(A[i_k], A[i_{k+1}]) \geq \sum_{k=j_1}^{j_2-1} d(A[i_k], A[i_{k+1}]) \geq d(A[i_{j_1}], A[i_{j_2}]) \ ,$$

where the last inequality is true because d satisfies the triangle inequality.

To prove the first claim, we invoke the above inequality for $\mathcal{T} = \mathcal{I}_k$ for every $k \in 1, \ldots, m$ and notice that, by definition, $c(A[\mathcal{I}]) \geq \sum_{k=1}^{m} c(A[\mathcal{I}_k])$. To prove the second claim, we invoke the first claim twice: once for $\mathcal{I}_1 \cup \mathcal{I}_2, \mathcal{I}_3 \cup \mathcal{I}_4, \ldots$ and a second time for $\mathcal{I}_2 \cup \mathcal{I}_3, \mathcal{I}_4 \cup \mathcal{I}_5, \ldots$. ◁

Algorithm. The algorithm consists of two phases. In the first phase, the algorithm partitions the array A into r contiguous blocks of n/r cells each. The j-th block goes from the $((j-1) \cdot n/r + 1)$-st cell to the $(j \cdot n/r)$-th cell of the array. The algorithm fills the cells in each block in a left to right manner: On the arrival of an element x_t with predicted index $\hat{i}_t$, the algorithm places x_t in the leftmost available empty cell in the block containing $\hat{i}_t$. Since the prediction produce a valid permutation of $[n]$ for each input sequence, there are at most n/r predicted indices in any block. Therefore, the algorithm never runs out of space in any block. For each $t \in [n]$, let $\hat{c}(t)$ be the cost of the predicted solution until time t. The first phase continues for all time steps as long as $(n/r) \cdot \hat{c}(t) \leq r \cdot c_{\text{opt}}$, and concludes if and when this condition is violated. Then, the algorithm switches to the second phase.

At the beginning of the second phase (see Figure 1), suppose the number of empty array cells is n'. The algorithm maps these n' cells to an empty virtual array A' of size n' and uses the $O(\sqrt{n'})$-competitive algorithm of Bertram [10] to insert the remaining elements into A'. Note that the predictions are ignored by the online algorithm in the second phase. Each element is inserted into A by mapping its cell in A' back to the empty cell in A that it represents.

Analysis. We now analyze the cost of the solution produced by the algorithm. Let c'_{opt} denote the cost of the optimal solution from the second phase (restricted to the array A'). First, note that the cost incurred by pairs of adjacent elements inserted in the second phase (i.e., in A') is at most $O(\sqrt{n'}) \cdot c'_{\text{opt}}$. Since $c'_{\text{opt}} \leq c_{\text{opt}}$ and $n' \leq n$, we have that the cost of the solution within the array A' is at most $O(\sqrt{n} \cdot c_{\text{opt}}) = O(r \cdot c_{\text{opt}})$, since by assumption $r = \Omega(\sqrt{n})$. Moreover, since A' maps to at most r gaps in A, the cost incurred by adjacent elements filled in different phases is at most $2r \cdot c_{\text{opt}}$. In total, this second phases incurs a cost of $O(r \cdot c_{\text{opt}})$.

We are left to analyze the cost of the first phase. We will bound this cost in terms of $\hat{c}(t)$ and in particular, bound the cost we incur by not placing elements exactly where the prediction tells us but rather only within the same block. Specifically, let $\hat{I}_t = \hat{i}_1, \ldots, \hat{i}_t$ denote the indices that would be occupied if we were to follow the prediction. Similarly, let I_t represent the indices that are occupied when our algorithm processes X_t. Note that $c(A[I_t])$ is an upper bound on the cost of the first phase. We show the following claim:

▷ Claim 18. At time t, we have that $c(A[I_t]) \leq (n/r+1) \cdot \hat{c}(t)$.

Proof. We consider a partitioning on $\hat{I}_t$ and I_t that is induced by the blocks. Namely, let m denote the number of non-empty blocks in A after placing $x_1, \ldots, x_t$ and let $j_1 < \cdots < j_m$ denote the indices of the non-empty blocks. Note that these m blocks are also going to be the only non-empty ones if we were to follow the prediction: this is because we never place an element outside its predicted block. These non-empty blocks therefore induce a natural partition of I_t and $\hat{I}_t$ into $\mathcal{I}_{j_1}, \ldots, \mathcal{I}_{j_\ell}$ and $\hat{\mathcal{I}}_{j_1}, \ldots, \hat{\mathcal{I}}_{j_m}$, respectively.

We decompose $c(A[I_t])$ into two components. The first component is the cost incurred within each block, i.e., $\sum_{k=1}^{m} c(A[\mathcal{I}_{j_k}])$. For this, we have by definition that, for each $\mathcal{I}_{j_k}$, it holds that $c(A[\mathcal{I}_{j_k}]) \leq (|\mathcal{I}_{j_k}|-1) \cdot \mathsf{diam}(A[\mathcal{I}_{j_k}])$ and, moreover, since $|\mathcal{I}_{j_k}| \leq n/r$, we have that $c(A[\mathcal{I}_{j_k}]) \leq (n/r-1) \cdot \mathsf{diam}(A[\mathcal{I}_{j_k}])$. Therefore:

$$\sum_{k=1}^{m} c(A[\mathcal{I}_{j_k}]) \leq (n/r-1) \cdot \sum_{k=1}^{m} \mathsf{diam}(A[\mathcal{I}_{j_k}]) \ .$$

Since each block stores the same elements of the input sequence, we have that $\mathsf{diam}(A[\mathcal{I}_{j_k}]) = \mathsf{diam}(\hat{A}[\hat{\mathcal{I}}_{j_k}])$ (recall that $\hat{A}$ is the array with $\hat{A}[\hat{i}_t] = x_t$ for all $t = 1, \ldots, n$). By the first property of Claim 17, we get that:

$$\sum_{k=1}^{m} c(A[\mathcal{I}_{j_k}]) \leq (n/r-1) \cdot \sum_{k=1}^{m} \mathsf{diam}(\hat{A}[\hat{\mathcal{I}}_{j_k}]) \leq (n/r-1) \cdot \hat{c}(t) \ .$$

Now we are left to bound the distance between elements in successive non-empty blocks, i.e., the distances between $A[r_{j_k}]$ and $A[\ell_{j_{k+1}}]$ for each $1 \leq k \leq m-1$, where r_{j_k} and ℓ_{j_k} are the maximum and minimum indices in $\mathcal{I}_{j_k}$. Fix such a $k \in [m-1]$. Then the distance $d(A[r_{j_k}], A[\ell_{j_{k+1}}]) \leq \mathsf{diam}(A[\mathcal{I}_{j_k} \cup \mathcal{I}_{j_{k+1}}]) = \mathsf{diam}(\hat{A}[\hat{\mathcal{I}}_{j_k} \cup \hat{\mathcal{I}}_{j_{k+1}}])$. By the second property in Claim 17, we then have that:

$$\sum_{k=1}^{m-1} d(A[r_{j_k}], A[\ell_{j_{k+1}}]) \leq \sum_{k=1}^{m-1} \mathsf{diam}(\hat{A}[\hat{\mathcal{I}}_{j_k} \cup \hat{\mathcal{I}}_{j_{k+1}}]) \leq 2 \cdot \hat{c}(t) \ .$$

We note that by definition $c(A[I_t]) = \sum_{k=1}^{m} c(A[\mathcal{I}_{j_k}]) + \sum_{k=1}^{m-1} d(A[r_{j_k}], A[\ell_{j_{k+1}}])$ and add the bounds we obtained on each term separately. This completes the proof. ◁

To complete the proof of Theorem 3, we consider two situations. If the second phase is not invoked, i.e., $(n/r) \cdot \hat{c}(n) \leq r \cdot c_{\mathrm{opt}}$, then the total cost is $(n/r+1) \cdot \hat{c}(n)$ by Claim 18. If the second phase starts at some time $t+1$, i.e., $(n/r) \cdot \hat{c}(n) > r \cdot c_{\mathrm{opt}}$, then the total cost is at most

$$\hat{c}(t) \cdot (n/r+1) + O(r \cdot c_{\mathrm{opt}}) = O(r \cdot c_{\mathrm{opt}}), \text{ since } \hat{c}(t) \cdot n/r \leq r \cdot c_{\mathrm{opt}} \ .$$

In both cases, we have established that the cost of the algorithm is $O\left(\min(n/r \cdot \hat{c}(n), r \cdot c_{\mathrm{opt}})\right)$. This completes the proof of Theorem 3.

B Online TSP with Multiple Predictions on the Uniform Metric

In this section we consider online TSP on the uniform metric with multiple predictors and prove the following theorem:

▶ **Theorem 5.** *There is a randomized algorithm for online TSP with h predictions on the uniform metric that is $O(\log h)$-consistent with the best of the h predictions, and a deterministic algorithm that is $O(h)$-consistent with the best of the h predictions.*

Recall that we have access to h different predictors. At time t, when element x_t is revealed, the algorithm also receives predicted indices $i_t^{(j)} \in [n]$, for $j = 1, \ldots, h$. We assume that for every j, the sequence $i_t^{(j)}$ for $t \in [n]$ forms a permutation of the indices $[n]$ of the array.

We now describe the algorithm. The algorithm computes at every timestep t the partial costs $c_t^{(j)}$ corresponding to the first t predictions of the j-th predictor, for $j \in [h]$. Similarly to our earlier quantity $\tilde{c}_t$, the quantities $c_t^{(j)}$ denote the cost of an array obtained by condensing the solution for the elements $x_1, x_2, \ldots, x_t$, obtained by following predictor j for the entire sequence.

Note that $c_n^{(j)}$ is the total cost of the j-th predictor; we denote this quantity by $c^{(j)}$. We observe that, by Lemma 10, the quantity $c_t^{(j)}$ is non-decreasing in t, for every j.

The algorithm runs in *epochs*. Epoch i, for $i = 0, 1, \ldots$ runs while the cost of the best predictor (so far), $\min_{j \in [h]}\{c_t^{(j)}\}$ is in $[2^i, 2^{i+1})$.

When we are in epoch i, we call the predictors with index $j \in [h]$ whose cost $c_t^{(j)}$ is currently in $[2^i, 2^{i+1})$ *admissible*. Observe that with the assumption of sentinel elements, the initial cost is 1, and therefore, at the beginning of epoch 0, all h predictors are admissible.

Further observe that for all t, we have $|c_{t+1}^{(j)} - c_t^{(j)}| \leq 2$, i.e., the cost of a predictor can increase in one step by at most 2. Thus, when we finish epoch i, i.e., none of the predictors are admissible for i, then we necessarily move into epoch $i + 1$ with at least one admissible predictor.

Within each epoch, the algorithm runs in phases. A phase in our current algorithm is similar to the first phase in the algorithm of Section 4. During a phase, the algorithm follows just one admissible predictor, ignoring all other predictors.

We apply the same modification to the predictions within one phase as in Section 4, applying a dynamic permutation σ_t that remaps the predictions (placing an element at the boundary of the gap it falls into, when an equal element guards the gap). As before, the cost of following the predictions within a phase can only improve by this transformation, while keeping the interphase cost manageable.

Precisely, at the beginning of each phase, the algorithm chooses one of the (currently) admissible predictors uniformly at random, and follows the predictions of that predictor, until it becomes inadmissible. When that happens, the phase ends and a new phase starts. This continues until the set of admissible predictors becomes empty, which marks the end of the epoch, and the start of the next epoch, as well as of a new phase.

At the end of a phase (or of an entire epoch), before switching to a different predictor in the next phase, we remap the empty cells of the array into a smaller but contiguous virtual array via a mapping τ. In the next phase, starting with time t' the remaining predictions for $x_{t'}, \ldots, x_n$ will first be condensed into a virtual array, and then mapped (by inverting τ) into the actual empty cells. This is, again, identical to the step of switching from the first to the second phase in the algorithm of Section 4. As we note next, condensing the remaining predictions into a (contiguous) virtual array cannot increase their total cost.

The above description is for a randomized algorithm. If randomization is not allowed, then we replace the uniform random choice of an admissible predictor above, by an arbitrary (deterministic) choice.

Analysis. We rely on a simple but crucial property of predictors. For $t' \leq t$ let $c_{t':t}^{(j)}$ be the condensed cost of the j-th predictor, for the predictions from timestep t' to timestep t, i.e., applied for the input elements $x_{t'}, \ldots, x_t$, in an initially empty array. By Lemma 10, we have $c_{t':t}^{(j)} \leq c_{1:t}^{(j)} = c_t^{(j)}$.

Now we bound the cost of the algorithm.

As before, we separately bound the cost within each phase, and the interphase cost (due to switching to a different predictor and remapping the remaining empty cells to a virtual array).

Suppose that the current phase lasts from timestep t' to timestep t and in this phase we follow predictor j. Since at the beginning of the phase we remap empty cells into a virtual array, the cost within the phase can be bounded identically to the first phase of the algorithm in Section 4, by Lemma 12, as $c^{(j)}_{t':t}$.

The cost due to switching and remapping the empty cells into a virtual array is $c^{(j)}_{t':t} + 2$, by an argument identical to that of Lemma 15.

Thus, the total cost of the phase and the interphase cost due to switching at the end of the phase are $2c^{(j)}_{t':t} + 2 \leq 2c^{(j)}_t + 2$. If this phase happens within epoch i, then this cost is at most $2 \cdot 2^{i+1}$.

Suppose the algorithm finishes in epoch N. Then, none of the predictors were admissible for epoch $N-1$, in particular the total cost of the best predictor is $C = \min_{j\in[h]}\{c^{(j)}\} \geq 2^N$.

Let n_i denote the number of phases of the algorithm in epoch i. Then, adding up the costs within each phase and the interphase costs, and using the above upper bound, we obtain that the total cost of the algorithm is at most $\sum_{i=0}^{N} 2^{i+2} \cdot n_i$.

In case of the deterministic algorithm, clearly $n_i \leq h' \leq h$, and thus the total cost of the algorithm is at most

$$h \cdot 2^{N+3} \leq h \cdot 8C \leq O(C \cdot h).$$

For the randomized algorithm we will show that $E[n_i] \leq \ln h + 1$. Then, the expected cost of the algorithm is at most

$$(\ln h + 1) \cdot 2^{N+3} \leq (\ln h + 1) \cdot 8C \leq O(C \cdot \log h).$$

This proves Theorem 5.

To upper bound $E[n_i]$, suppose that $h' \leq h$ predictors are admissible at the beginning of epoch i, and label them from 1 to h' according to the order in which they will become inadmissible (breaking ties arbitrarily).

Our algorithm chooses the prediction sequences to follow in epoch i according to a uniform random permutation $\pi_1, \ldots, \pi_{h'}$ of $[h']$. This means that after a phase in which we follow prediction sequence π_j, we will follow the first among $\pi_{j+1}, \pi_{j+2}, \ldots$ that is still admissible after prediction sequence π_j became inadmissible. It is easy to see that we only follow a given prediction sequence in $\pi_1, \ldots, \pi_{h'}$ if its label is a *left-to-right maximum* in the order π, i.e., if it becomes inadmissible later than all those before it. As the expected number of left-to-right maxima of a random permutation of length n is $H_n \leq \ln n + 1$, the claim follows.