Online Metric TSP

Christian Bertram
University of Copenhagen, Denmark

Abstract

In the *online metric traveling salesperson problem*, n points of a metric space arrive one by one and have to be placed (immediately and irrevocably) into empty cells of a size-n array. The goal is to minimize the sum of distances between consecutive points in the array. This problem was introduced by Abrahamsen, Bercea, Beretta, Klausen, and Kozma [ESA'24] as a generalization of the *online sorting problem*, which was introduced by Aamand, Abrahamsen, Beretta, and Kleist [SODA'23] as a tool in their study of online geometric packing problems.

Online metric TSP has been studied for a range of fixed metric spaces. For 1-dimensional Euclidean space, the problem is equivalent to online sorting, where an optimal competitive ratio of $\Theta(\sqrt{n})$ is known. For d-dimensional Euclidean space, the best-known upper bound is $O(2^d\sqrt{dn\log n})$, leaving a gap to the $\Omega(\sqrt{n})$ lower bound. Finally, for the uniform metric, where all distances are 0 or 1, the optimal competitive ratio is known to be $\Theta(\log n)$.

We study the problem for a general metric space, presenting an algorithm with competitive ratio $O(\sqrt{n})$. In particular, we close the gap for d-dimensional Euclidean space, completely removing the dependence on dimension. One might hope to simultaneously guarantee competitive ratio $O(\sqrt{n})$ in general and $O(\log n)$ for the uniform metric, but we show that this is impossible.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases online algorithm, metric space, TSP

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.80

Related Version *Full Version*: `https://arxiv.org/abs/2504.17716` [4]

Funding Christian Bertram is part of BARC, Basic Algorithms Research Copenhagen, supported by the VILLUM Foundation grant 54451.

Acknowledgements I thank Anders Aamand, Mikkel Abrahamsen, Théo Fabris, Jonas Klausen, and Mikkel Thorup for helpful and inspiring conversations, and the reviewers for valuable comments.

1 Introduction

1.1 Problem definition

The *online metric traveling salesperson problem* (online metric TSP), recently introduced by Abrahamsen, Bercea, Beretta, Klausen, and Kozma [2], is as follows. Given a sequence $x_1, \ldots, x_n$ of points arriving one by one (with repetitions allowed) from a metric space (M, d), assign them bijectively to array cells $A[1], \ldots, A[n]$. The goal is to minimize $\sum_{i=1}^{n-1} d(A[i], A[i+1])$, which represents the length of the walk $A[1], \ldots, A[n]$. The problem is *online* in the sense that after receiving x_i, we must immediately and irrevocably set $A[j] = x_i$ for some previously unused array index j, without knowledge of x_k for $k > i$.

We can think of this as a metric traveling salesperson problem, where n cities are sequentially revealed, one by one, and must be placed on a unique date in the salesperson's n-day calendar. The cost to be minimized is the length of the final n-day trip.

In [2], the metric space is fixed as 1-dimensional Euclidean space, d-dimensional Euclidean space, or a space with uniform/discrete metric (where all distances are 0 or 1). In this paper, we study the problem for a general metric space, allowing the algorithm to query the distance $d(x_i, x_j)$ between x_i and x_j, if it has received x_i and x_j. Alternatively, the ith input can be assumed to be the vector $(d(x_i, x_1), d(x_i, x_2), \ldots, d(x_i, x_{i-1}))$.

© Christian Bertram;
licensed under Creative Commons License CC-BY 4.0

The term "online TSP" has also been used for a different, older problem where a salesperson moves through the metric space at unit speed as cities are revealed [3]. In contrast, our problem consists of constructing a fixed travel plan through irrevocable online decisions.

1.2 Prior work

In [1], the *online sorting problem* is introduced as part of their study of various geometric translational packing problems. The online sorting problem is equivalent to online (metric) TSP in the Euclidean unit interval $[0, 1]$. They further essentially assume that the points 0 and 1 always show up in the input, which simplifies the analysis, as the optimal offline cost becomes 1. They present a deterministic algorithm with competitive ratio $O(\sqrt{n})$ for online sorting (where *competitive ratio* is the worst-case ratio between the algorithm's cost and the optimal offline cost), and show an $\Omega(\sqrt{n})$ lower bound for deterministic algorithms.

In [2], the online sorting algorithm from [1] is generalized to points from the real line $\mathbb{R}$, not necessarily including 0 and 1, by employing a careful *doubling technique*. They maintain the $O(\sqrt{n})$ competitive ratio, and generalize the $\Omega(\sqrt{n})$ lower bound to randomized algorithms.

In [2], they further generalize the problem to online metric TSP, and consider this problem for a few fixed metric spaces. For d-dimensional Euclidean space, they present an $O(2^d\sqrt{dn\log n})$ competitive ratio algorithm, leaving a gap to their $\Omega(\sqrt{n})$ lower bound. For use as a subroutine in this algorithm, they consider online TSP in the uniform metric, i.e. the metric where all distances are 0 or 1. They argue that this problem has interest in its own right, corresponding to minimizing the number of task switches in scheduling, and present a tight $\Theta(\log n)$ bound on the competitive ratio. Finally, they ask, as an open question, whether $O(\sqrt{n})$ is the optimal competitive ratio for arbitrary metrics.

1.3 Our results

Our main result is an optimal algorithm for online metric TSP in a general metric space, settling a question posed in [2].

▶ **Theorem 1.** *There exists a deterministic algorithm for online metric TSP with competitive ratio $O(\sqrt{n})$.*

This is optimal by the $\Omega(\sqrt{n})$ lower bound for the Euclidean unit interval [1], which holds even for randomized algorithms [2]. As a direct corollary, we get an optimal algorithm for online TSP in d-dimensional Euclidean space, improving upon the best-known $O(2^d\sqrt{dn\log n})$ competitive ratio algorithm [2], completely removing the dependence on dimension.

▶ **Corollary 2.** *There exists a deterministic algorithm for online TSP in d-dimensional Euclidean space with competitive ratio $O(\sqrt{n})$.*

Note, though, that the $O(\sqrt{n})$ bound of Theorem 1 does not match the known $\Theta(\log n)$ bound for online TSP with uniform metric [2]. One might hope for an even stronger algorithm, obtaining an asymptotically optimal competitive ratio for every fixed metric space. We show that no such algorithm exists, hinting that our algorithm is the best one can hope for.

▶ **Theorem 3.** *No randomized algorithm for online metric TSP obtains both $O(\log n)$ expected competitive ratio for the uniform metric and $O(\sqrt{n})$ expected competitive ratio in general.*

1.4 Structure of the paper

In Section 2, we recall relevant definitions and fix notation. In Section 3, we present our algorithm, proving Theorem 1 and Corollary 2. In Section 4, we confirm the worst-case optimality of our algorithm via known lower bounds, and then we show that no algorithm is simultaneously optimal for every fixed metric, proving Theorem 3.

2 Preliminaries

We will often let (M, d) be a metric space, where M is the set of points, and $d\colon M \times M \to \mathbb{R}_{\geq 0}$ is the metric. Our main examples are d-dimensional Euclidean space and the uniform metric.

The *uniform metric* is defined by $d(x, y) = 1$ for all $x \neq y$. This is also known as the *discrete metric*, but we adopt the former term for consistency with [2]. Any set of points can form a metric space under the uniform metric.

Though our algorithms run in polynomial time, we do not focus on their exact running times. Instead, we evaluate performance via *competitive analysis*, as is standard in the study of online algorithms. Let $\texttt{OPT}(X)$ denote the optimal offline cost for an input sequence $X \in M^n$. An online algorithm is said to have *competitive ratio* $C(n)$, for a function $C\colon \mathbb{N} \to \mathbb{R}$, if its cost is at most $C(n)\,\texttt{OPT}(X)$ for all $n \in \mathbb{N}$ and all $X \in M^n$.

In our setting, the cost is $\sum_{i=1}^{n-1} d(A[i], A[i+1])$, and $\texttt{OPT}(X)$ is the minimum value of this sum, minimized over all bijections from the input points in X to array cells $A[1], \ldots, A[n]$. By the triangle inequality, $\texttt{OPT}(X)$ is equal to the length of a shortest walk visiting all points in X. We will sometimes write $\texttt{OPT}(X)$ for a subset $X \subseteq M$, as the optimal offline cost is invariant under permutations and repetitions of the input points.

3 Our algorithm

Our algorithm is based on the algorithms for online sorting presented in [1] and [2]. Let us first give a high-level explanation of their approach. Denote by X' the set of currently received points. They consider an interval J containing X', and partition J into $\sqrt{n}$ subintervals. Specifically, they assume without loss of generality that $x_1 = 0$, and let J be the smallest interval of the form $[-2^k, 2^k]$ containing X', increasing k as necessary. Note that $\texttt{OPT}(X') \geq 2^{k-1}$, so the subintervals have size $O(\texttt{OPT}(X')/\sqrt{n})$. Similarly, they partition the array A into $2\sqrt{n}$ subarrays, called *blocks*. The algorithm generally places the input points into the array, such that no block contains points from different subintervals. This ensures, that they only pay a small $O(\texttt{OPT}(X')/\sqrt{n})$ cost between points inside each block, and pay a large $O(\texttt{OPT}(X'))$ cost only between blocks. The former cost is paid $O(n)$ times, and the latter $O(\sqrt{n})$ times, totaling a cost of $O(\sqrt{n}\,\texttt{OPT}(X'))$. It is always possible to place the first half of the points in this manner, after which they consider the remaining empty array cells as one contiguous array, and recursively fill this simulated array with the remaining input points. This leads to a total cost of $O(\sqrt{n}\,\texttt{OPT}(X))$ as the cost per recursion falls geometrically. Each time k is increased, we incur some additional cost of changing the intervals, but this only happens when the bound on $\texttt{OPT}(X')$ doubles, so again it all sums to $O(\sqrt{n}\,\texttt{OPT}(X))$.

Let us now give a high-level explanation of how we will extend this algorithm from the Euclidean line to a general metric space. There are two main problems to solve: the setting of $J = [-2^k, 2^k]$ and bound of $\texttt{OPT}(X') \geq 2^{k-1}$ no longer make sense. One might naively try letting J be a ball of radius 2^k, but this cannot generally be covered by $\sqrt{n}$ balls of radius $O(2^k/\sqrt{n})$. To see this, think of d-dimensional Euclidean space, where 2^d unit cubes are required to cover a cube of sidelength 2. This is related to why there is a term of size 2^d in

the best-known bound for d-dimensional Euclidean space [2]. We instead present an online algorithm maintaining a set of at most $\sqrt{n}$ balls of radius $r = \Theta(\texttt{OPT}(X')/\sqrt{n})$ covering all received input points. Only when $\texttt{OPT}(X')$ doubles, we allow changing the radius r and resetting the set. Of course, we don't know the value of $\texttt{OPT}(X')$, but it turns out that we can successfully use the minimum spanning tree weight as a (polynomial-time computable) proxy. This proxy will be essential when showing that our covering only needs $\sqrt{n}$ balls.

We begin the formal presentation of our algorithm with the above mentioned proxy.

▶ **Definition 4.** Let (M, d) be a metric space and $X \subseteq M$ a finite subset of points. When the metric d is clear from context, we denote by $\texttt{MST}(X)$ the total weight of a minimum spanning tree of the complete graph on X, where edge weights are given by d.

Computing exact metric TSP length is NP-hard, as follows by a straightforward reduction from the undirected Hamiltonian cycle problem, one of Karp's 21 NP-complete problems [5]. Luckily, there is a simple 2-approximation based on minimum spanning trees [6]. This well-known argument yields the following lemma, which tells us that $\texttt{MST}(X)$ is a good proxy of $\texttt{OPT}(X)$. We provide a proof in the full version of the paper [4].

▶ **Lemma 5.** *Let (M, d) be a metric space and $X \subseteq M$ a finite subset of points. Then $\texttt{MST}(X) \leq \texttt{OPT}(X) \leq 2\,\texttt{MST}(X)$.*

As sketched, our algorithm will cover the input points by a dynamic set of balls. Formally, we will work with a set of *centers* forming a *net* in the following standard manner.

▶ **Definition 6.** Let (M, d) be a metric space and $X \subseteq M$ a set of points. Let $r \geq 0$ be a real number. A subset $C \subseteq X$ is an *r-net* of X if

- for every $x \in X$, there exists a $c \in C$ with $d(c, x) \leq r$, and
- for every $c, c' \in C$ with $c \neq c'$, we have $d(c, c') > r$.

If so, we sometimes refer to each point in C as a *center*, and to r as the *radius* of the net.

Nets will be useful to us, because they are small in terms of our proxy, in the sense of the following lemma.

▶ **Lemma 7.** *Let (M, d) be a metric space, in which C is an r-net of a set $X \subseteq M$. Then $(|C| - 1)r \leq 2\,\texttt{MST}(X)$.*

Proof. By the triangle inequality, we have $\texttt{OPT}(X) \geq \texttt{OPT}(C)$. Combining this with Lemma 5, we get $2\,\texttt{MST}(X) \geq \texttt{OPT}(X) \geq \texttt{OPT}(C) \geq \texttt{MST}(C)$. Every edge in the complete graph on C has length at least r, and a tree on $|C|$ points has $|C| - 1$ edges, so $\texttt{MST}(C) \geq (|C| - 1)r$. ◀

When receiving a new point x, we will generally update our r-net by the following simple subroutine.

Increase-Net(C, r, x)
1. If no $c \in C$ has $d(x, c) \leq r$ then add x to C.

This subroutine allows us to maintain an r-net, for a fixed radius r. We state this as the following lemma, which follows directly from the definition.

▶ **Lemma 8.** *Let (M, d) be a metric space, in which C is an r-net of a set $X \subseteq M$, and let $x \in X$. Then running* Increase-Net(C, r, x) *modifies C only by insertion, such that C becomes an r-net of $X \cup \{x\}$.*

To facilitate analysis, let us introduce two pieces of notation. For a partially filled array A, we write $c(A) = \sum_{i \in I} d(A[i], A[i+1])$ for the *cost* of A, where I is the set of indices i for which both $A[i]$ and $A[i+1]$ are non-empty. Note that this matches the usual cost, when A is full. Secondly, we write $G(A)$ for the number of *gaps* in A, i.e. the number of non-trivial maximal contiguous empty subarrays of A.

We now present "half" of our algorithm, namely an algorithm handling the first $\lceil n/2 \rceil$ input points. First, we give a quick overview. The algorithm will partition the array A into $2\lfloor \sqrt{n} \rfloor$ blocks, and maintain an r-net C with at most $\lfloor \sqrt{n} \rfloor$ centers. If we get too many centers, we will reset the net with a larger radius. Each block may be *assigned* to a center, such that every center is assigned at most one block. Initially, every block is unassigned. Generally, if a newly received point lies within the ball of a center, it will be placed inside the block assigned to that center. The exact algorithm is as follows.

Fill-Most-Blocks(n, A, X)

1. Let $N_1 = \lfloor \sqrt{n} \rfloor$ and $N_2 = 2N_1$.
2. Partition A into N_2 subarrays, called *blocks*, of length at least $\lfloor n/N_2 \rfloor$.
3. Initialize $r = 0$ and $C = \emptyset$.
4. For each point x in the stream X:
   a. Increase-Net(C, r, x).
   b. If $|C| > N_1$:
      I. Unassign all blocks.
      II. Set $r = 4\,\mathtt{MST}(X')/N_1$, where X' is the set of known input points.
      III. Set $C = \{x\}$.
   c. Let $c \in C$ be a center with $d(x, c) \leq r$.
   d. If a full block B is assigned to c:
      I. Unassign B from c.
   e. If no block is assigned to c:
      I. Assign an unassigned non-full block to c.
   f. Let B be the block assigned to c.
   g. Place x in the left-most empty cell of B.

▶ **Lemma 9.** *Let (M, d) be a metric space and X be an online stream of $\lceil n/2 \rceil$ points in M. Let A be an empty array of length n. The deterministic algorithm* Fill-Most-Blocks(n, A, X) *irrevocably places each point from X in an empty cell of A, such that when all points have been placed, we have $G(A) \leq 2\sqrt{n}$ and $c(A) \leq 11\sqrt{n}\,\mathtt{OPT}(X)$.*

Proof. The following two claims show that the algorithm is well-defined.

▷ Claim. After step 4b, C is an r-net of a set containing x.

Proof. Clearly, the claim holds if step 4(b)III just ran, so let us show, that it also holds before and between these resets of the net. This follows by induction, where the base case is step 3 producing an r-net, and the induction step is step 4a with Lemma 8. ◁

▷ Claim. In step 4(e)I, there is always an unassigned non-full block to assign.

Proof. Assume for the sake of contradiction, that no unassigned non-full block exists. Then every block is either assigned or full. There are at most $|C| \leq N_1$ assigned blocks, so at least $N_2 - |C| \geq N_2 - N_1 = N_1$ blocks are full. Note that every assigned block contains at least one point, as right after assigning a block in step 4(e)I, it is given a point in step 4g. So at least

N_1 blocks contain at least $\lfloor n/N_2 \rfloor$ points, and the remaining blocks contain at least one point. This means that the total number of points in A is at least $N_1\lfloor n/N_2 \rfloor + N_1 \geq N_1 n/N_2 = n/2$. This contradicts the stream containing only $\lceil n/2 \rceil$ points, as in step 4(e)I the point x has yet to be placed. ◁

The following two claims show that the algorithm is correct.

▷ Claim. At termination, $G(A) \leq 2\sqrt{n}$.

Proof. Since every point is placed in the left-most empty cell of a block, there is at most one gap per block, so $G(A) \leq N_2 \leq 2\sqrt{n}$. ◁

▷ Claim. At termination, $c(A) \leq 11\sqrt{n}\,\texttt{OPT}(X)$.

Proof. The cost between two neighboring blocks is at most $\texttt{OPT}(X)$, so the total cost between neighboring blocks is at most $N_2\,\texttt{OPT}(X)$. It remains to bound the cost between neighboring cells inside a block. So let x and x' be points placed in neighboring cells inside a block B.

Consider first the case where the net was not reset (steps 4(b)I to 4(b)III) between x and x' being placed. Then B was assigned to the same center c with the same radius r both when x and x' were placed. Let X' be the set of points read right after both x and x' were placed. By the triangle inequality, $\texttt{OPT}(X') \leq \texttt{OPT}(X)$, so by Lemma 5, $d(x,x') \leq d(x,c) + d(c,x') \leq 2r = 8\,\texttt{MST}(X')/N_1 \leq 8\,\texttt{OPT}(X')/N_1 \leq 8\,\texttt{OPT}(X)/N_1$. Since at most $\lceil n/2 \rceil$ points are placed, the total cost between such pairs of points is at most $(\lceil n/2 \rceil - 1)8\,\texttt{OPT}(X)/N_1 \leq 4n\,\texttt{OPT}(X)/N_1$.

Consider now the case where the net was reset between x and x' being placed. Then $d(x,x') \leq \texttt{MST}(X')$ where X' is the set of points read at any point after placing both x and x'. This is a worse bound, but this case can only appear once per block per reset of the net. So the total cost for such pairs of points is at most $N_2\,\texttt{MST}(X')$ for each reset, where X' is the set of points read when resetting. Let X_1 be the set of points read at the point of a reset, or at the beginning of the algorithm, and X_2 be the set of points read at the point of the following reset or when the algorithm terminates. Right before the latter reset, $|C| > N_1$ and $r = 4\,\texttt{MST}(X_1)/N_1$. This is true even if X_1 is the set at the beginning of the algorithm, as then $X_1 = \emptyset$ and $r = 0$. So Lemma 7 gives us that $2\,\texttt{MST}(X_2) \geq (|C| - 1)r \geq N_1 4\,\texttt{MST}(X_1)/N_1 = 4\,\texttt{MST}(X_1)$, which simplifies to $\texttt{MST}(X_2) \geq 2\,\texttt{MST}(X_1)$. The total cost of this type across all rebuilds is thus no more than the geometric series $N_2 \sum_{i=0} 2^{-i}\,\texttt{MST}(X) = 2N_2\,\texttt{MST}(X) \leq 2N_2\,\texttt{OPT}(X)$, where the last inequality uses Lemma 5.

The total cost $c(A)$ at termination is thus at most $(N_2 + 4n/N_1 + 2N_2)\,\texttt{OPT}(X)$. It can be checked that $N_2 + 4n/N_1 + 2N_2 \leq 11\sqrt{n}$, finishing the proof. ◁

The combination of the above claims finishes the proof. ◀

We are now ready to present our full algorithm, which recursively applies Fill-Most-Blocks, analogously to [1, 2]. The algorithm begins by placing the first half of the input points into the array A using Fill-Most-Blocks. It then treats the remaining empty cells of A as a contiguous array A_{empty}. The algorithm proceeds recursively on A_{empty}, treating it as a standard array; however, when a point is placed in A_{empty}, it is actually placed into the corresponding cell of A. The exact algorithm is given below as Recursively-Fill-Most-Blocks.

▶ **Theorem 10.** *Let (M,d) be a metric space and X be an online stream of n points in M. Let A be an empty array of length n. The deterministic algorithm* Recursively-Fill-Most-Blocks(n,A,X) *irrevocably places each point from X in an empty cell of A, such that when all points have been placed, we have $c(A) \leq 52\sqrt{n}\,\texttt{OPT}(X)$.*

Recursively-Fill-Most-Blocks(n, A, X)
1. If $n = 0$ then return.
2. Let X_{prefix} be the stream consisting the first $\lceil n/2 \rceil$ points of X.
3. Fill-Most-Blocks$(n, A, X_{\text{prefix}})$.
4. Consider the empty cells of A as one contiguous array A_{empty}.
5. Let X_{suffix} be the stream consisting the remaining $\lfloor n/2 \rfloor$ points of X.
6. Recursively-Fill-Most-Blocks$(\lfloor n/2 \rfloor, A_{\text{empty}}, X_{\text{suffix}})$.

Proof. We will show a cost of at most $15(2+\sqrt{2})\sqrt{n}\,\texttt{OPT}(X)$ using strong induction on n. The base case $n = 0$ is trivial, so let us handle the inductive step. Let A' denote the array A after step 3. Then $G(A') \leq 2\sqrt{n}$ and $c(A') \leq 11\sqrt{n}\,\texttt{OPT}(X_{\text{prefix}}) \leq 11\sqrt{n}\,\texttt{OPT}(X)$ by Lemma 9 and the triangle inequality. By induction, we have $c(A_{\text{empty}}) \leq 15(2+\sqrt{2})\sqrt{n/2}\,\texttt{OPT}(X)$. The final cost $c(A)$ is the sum of $c(A')$, $c(A_{\text{empty}})$, and the costs between neighboring cells in A where exactly one of the cells was empty in A'. The latter is at most $2G(A')\,\texttt{OPT}(X) \leq 4\sqrt{n}\,\texttt{OPT}(X)$, as there are at most two such pairs of neighboring cells per gap in A'. The total cost becomes $c(A) \leq (11 + 15(2+\sqrt{2})/\sqrt{2} + 4)\sqrt{n}\,\texttt{OPT}(X) = 15(2+\sqrt{2})\,\texttt{OPT}(X)$. ◀

From Theorem 10, we immediately get Theorem 1 and Corollary 2. We have thus generalized the optimal upper bound for online sorting [2] to online metric TSP, in particular improving the best-known upper bound for online TSP in d-dimensional Euclidean space. In Section 4 we show, that our algorithm is optimal for both general online metric TSP, online TSP in d-dimensional Euclidean space, and more.

4 Optimality

It follows directly from a known lower bound for online sorting [1, 2] that our algorithm is optimal. Recall that online sorting is equivalent to online TSP in 1-dimensional Euclidean space.

▶ **Theorem 11** (Theorem 1 in [2]). *The (deterministic and randomized) competitive ratio of online TSP in the Euclidean unit interval* $[0, 1]$ *is* $\Omega(\sqrt{n})$.

In particular, this gives a lower bound of $\Omega(\sqrt{n})$ for general online metric TSP, showing that our $O(\sqrt{n})$ algorithm Recursively-Fill-Most-Blocks from Section 3 is optimal. Perhaps surprisingly, online metric TSP is no harder than online sorting.

Since the Euclidean unit interval lies inside d-dimensional Euclidean space, Theorem 11 also shows that our $O(\sqrt{n})$ algorithm is optimal for online TSP in d-dimensional Euclidean space. We have thus closed the gap for this problem, improving upon the best known upper bound of $O(2^d\sqrt{dn\log n})$ [2], notably removing the dependence on dimension.

Studying the proof of the above lower bound, we find the following generalization. Intuitively, the generalization gives us a lower bound of $\Omega(\sqrt{n})$ for online TSP in any metric space where we can draw a straight line segment (of length ℓ with endpoints a_0 and a_1) and pick m evenly spaced points along it.

▶ **Corollary 12.** *Let* (M, d) *be a metric space, such that for every* $m \in \mathbb{N}$*, there exist two points* $a_0, a_1 \in M$*, and a set* $X \subseteq M$ *of* m *points, such that, for* $\ell = \texttt{OPT}(X \cup \{a_0, a_1\})$*,*
- $d(x, y) \geq \ell/m$ *for all distinct* $x, y \in X$*, and*
- $d(a_0, x) + d(x, a_1) \geq \ell$ *for all* $x \in X$*.*

Then the competitive ratio of online TSP in (M, d) *is* $\Omega(\sqrt{n})$.

Proof. This follows from the proof of Theorem 11 presented in Section 2 of [2]. They work in the Euclidean unit interval, but only use the points $a_0 = 0$, $a_1 = 1$, and the set of points $X = \{0, 1/\sqrt{n}, \ldots, (\sqrt{n}-1)/\sqrt{n}\}$. Ignoring the concrete values of the points a_0, a_1, and those in X, it can easily be checked, that they only use the properties stated above, where $m = \sqrt{n}$ and $\ell = 1$. The proof also follows through for any other $\ell > 0$, where all distances in their proof simply scale by ℓ. ◀

This tells us that our algorithm is also optimal for online TSP in e.g. (subgroups of) normed vector spaces and Riemannian manifolds.

4.1 Impossibility of optimality for every fixed metric space

We have seen, that our $O(\sqrt{n})$ algorithm is optimal for general online metric TSP, as well as for online TSP in many fixed metric spaces. It is not optimal for every fixed metric space, though, as an algorithm with competitive ratio $O(\log n)$ is known for the uniform metric [2]. Motivated by this gap, one might ask whether there exists a stronger algorithm, which optimally solves online TSP in (M, d), for every fixed metric space (M, d). In this subsection, we show that no such algorithm exists. This hints, that the weaker optimality of our algorithm is the best one can hope for.

Specifically, we show that no algorithm is optimal both for the uniform metric and for a general metric space. This is the content of Theorem 3, which we restate and prove below.

▶ **Theorem 3.** *No randomized algorithm for online metric TSP obtains both $O(\log n)$ expected competitive ratio for the uniform metric and $O(\sqrt{n})$ expected competitive ratio in general.*

Proof. Assume for the sake of contradiction that such an algorithm $\mathcal{A}$ exists. Let us first informally explain our basic idea. Let U be a set of $n^{4/5}$ points with pairwise distance 1, and let x be a point with distance $n^{4/5}$ to every point in U. Consider an input consisting $n^{1/5}$ consecutive copies of U. Then the optimal offline cost is $\texttt{OPT}(U) = |U| - 1 = n^{4/5} - 1$. The only way for $\mathcal{A}$ to match this optimal cost, would be to maintain $\Omega(|U|) = \Omega(n^{4/5})$ gaps until completion. Similarly, since $\mathcal{A}$ actually must obtain nearly-optimal cost $O(n^{4/5} \log n)$, it must have at least $n^{3/5}$ gaps at some point. But when this happens, we can trick $\mathcal{A}$ by changing the remaining input to be copies of x, filling up all these gaps. Then the cost becomes at least $n^{3/5}n^{4/5}$ which breaks the promise of competitive ratio $O(\sqrt{n})$, since $\texttt{OPT}(U \cup \{x\}) = 2n^{4/5} - 1$.

We now formally prove the theorem. Let $\mathcal{X}$ be the random input served by Oblivious-Random-Adversary(n), which we define below.

Oblivious-Random-Adversary(n)
1. Let U be a set of $n^{4/5}$ points with pairwise distance 1.
2. Let x be a point with distance $n^{4/5}$ to every point in U.
3. While less than n points have been served:
   a. With probability $n^{-3/5}$:
      I. Let m be the number of points served.
      II. Serve $n - m$ copies of x.
   b. Otherwise:
      I. Serve a copy of the points in U.

By closely following the proof of Theorem 11, replacing their random input by $\mathcal{X}$, we get that the expected cost of any deterministic algorithm on $\mathcal{X}$ is $\Omega(n)$. A full proof can be found in the full version of the paper [4].

Let $\mathcal{A}(\mathcal{X})$ denote the cost of $\mathcal{A}$ on $\mathcal{X}$. Considering $\mathcal{A}$ as a random variable over deterministic algorithms, we get $\mathbb{E}[\mathcal{A}(\mathcal{X})] \in \Omega(n)$. We will now derive an upper bound contradicting this lower bound. To utilize our assumed competitive ratio for the uniform metric, note that $\mathcal{X}$ follows the uniform metric when $x \notin \mathcal{X}$. From the definition of Oblivious-Random-Adversary, we have $\Pr[x \in \mathcal{X}] = 1 - \Pr[x \notin \mathcal{X}] = 1 - (1 - n^{-3/5})^{n^{1/5}} \leq n^{-2/5}$, using Bernoulli's inequality. Using this bound, we get

$$\begin{aligned}\mathbb{E}[\mathcal{A}(\mathcal{X})] &= \Pr[x \in \mathcal{X}]\,\mathbb{E}[\mathcal{A}(\mathcal{X}) \mid x \in \mathcal{X}] + \Pr[x \notin \mathcal{X}]\,\mathbb{E}[\mathcal{A}(\mathcal{X}) \mid x \notin \mathcal{X}] \\ &\leq n^{-2/5}\,\mathbb{E}[\mathcal{A}(\mathcal{X}) \mid x \in \mathcal{X}] + \mathbb{E}[\mathcal{A}(\mathcal{X}) \mid x \notin \mathcal{X}].\end{aligned}$$

By assumption, $\mathcal{A}$ has competitive ratio $O(\sqrt{n})$ in general, and competitive ratio $O(\log n)$ when x is not in the input. It's easy to see that $\texttt{OPT}(\mathcal{X}) \in O(n^{4/5})$, so we have $\mathbb{E}[\mathcal{A}(\mathcal{X}) \mid x \in \mathcal{X}] \in O(n^{4/5}\sqrt{n})$ and $\mathbb{E}[\mathcal{A}(\mathcal{X}) \mid x \notin \mathcal{X}] \in O(n^{4/5} \log n)$. In total, $\mathbb{E}[\mathcal{A}(\mathcal{X})] \in O(n^{-2/5} n^{4/5} \sqrt{n} + n^{4/5} \log n) \subseteq o(n)$, contradicting $\mathbb{E}[\mathcal{A}(\mathcal{X})] \in \Omega(n)$. ◀

5 Open questions

Both [1] and [2] additionally study a variant of the problem with extra space. In this variant, the array is of length γn for some $\gamma > 1$, and empty cells are ignored in the cost function. A significant gap between the best-known upper and lower bound remains, even for online sorting of reals with extra space [1]. For the uniform metric, a tight $\Theta(1 + \log(\gamma/(\gamma - 1)))$ competitive ratio is known [2]. We repeat it as an open question to tighten the gap for online sorting of reals with extra space, and suggest the introduction of an algorithm for online metric TSP with extra space.

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

4 Christian Bertram. Online metric TSP. *CoRR*, abs/2504.17716, 2025. doi:10.48550/arXiv.2504.17716.

5 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.

6 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.