SIAM J. COMPUT.
Vol. 45, No. 3, pp. 859–880





# THE POWER OF RECOURSE FOR ONLINE MST AND TSP*

NICOLE MEGOW†, MARTIN SKUTELLA‡, JOSÉ VERSCHAE§, AND ANDREAS WIESE¶

**Abstract.** We consider online versions of the minimum spanning tree (MST) problem and the traveling salesman problem (TSP) where recourse is allowed. The nodes of an unknown graph with metric edge cost appear one by one and must be connected in such a way that the resulting tree or tour has low cost. In the standard online setting, with irrevocable decisions, no algorithm can guarantee a constant-competitive ratio. In our model we allow recourse actions by giving a limited budget of edge rearrangements per iteration. It has been an open question for more than 20 years whether an online algorithm equipped with a constant (amortized) budget can guarantee constant-approximate solutions. As our main result, we answer this question affirmatively in an amortized setting. We introduce an algorithm that maintains a nearly optimal tree when given a constant amortized budget. Unlike in classical TSP variants, the standard double-tree and shortcutting approach does not give constant guarantees in the online setting. We propose a nontrivial robust shortcutting technique that allows translation of online MST results into TSP results at the loss of small factors.

**Key words.** online algorithms, minimum spanning tree, traveling salesman, recourse

**AMS subject classifications.** 68Q25, 68W25

**DOI.** 10.1137/130917703

## 1. Introduction.

In the *online minimum spanning tree* (MST) problem and *online traveling salesman problem* (TSP) we aim at constructing low-cost spanning trees (resp., tours) for an unknown graph that is revealed online. In each iteration a new node is revealed, together with all connections to previously known nodes, and an algorithm must make an irrevocable decision on how to connect it. Such problems appear naturally in applications related to multicast routing in multimedia distribution systems, video conferencing, software delivery, and groupware [16, 17]. These problems—in particular online MST and Steiner tree variants—have been studied extensively. Here, constant-competitive ratios are not achievable; the best possible performance ratio is $\Theta(\log t)$, where $t$ is the number of iterations [11].

However, in many of the above-mentioned applications it is possible to adapt solutions in some limited way when the node set changes [19, 20]. Such *recourse actions* may allow for better solutions. However, a large number of recourse actions—in particular, a complete reconstruction of solutions—might not be feasible or may cause unacceptable additional cost. Our goal is to understand the trade-off between

*Received by the editors April 19, 2013; accepted for publication (in revised form) April 12, 2016; published electronically June 29, 2016. A preliminary version of this paper appeared in the proceedings of ICALP 2012 [15]. This work was partially funded by the Berlin Mathematical School (BMS), the German Science Foundation (DFG) under grant ME 3825/2, the DFG Research Center MATHEON "Mathematics for key technologies" in Berlin, Nucleo Milenio Información y Coordinación en Redes ICR RC130003, the DFG Focus Program 1307 within the project "Algorithm Engineering for Real-time Scheduling and Routing," and a fellowship within the Postdoc-Programme of the German Academic Exchange Service (DAAD).

http://www.siam.org/journals/sicomp/45-3/91770.html

†Zentrum Mathematik, M9, Technische Universität München, Boltzmannstrasse 3, 85747 Garching bei München, Germany (nmegow@ma.tum.de).

‡Institut für Mathematik, Technische Universität Berlin, Straße des 17. Juni 136, 10623 Berlin, Germany (martin.skutella@tu-berlin.de).

§Facultad de Matemáticas and Escuela de Ingeniería, Pontificia Universidad Católica de Chile, Av. Vicuña Mackenna 4860, Santiago, Chile (jverschae@uc.cl).

¶Max-Planck-Institut für Informatik, Campus E1 4, 66123 Saarbrücken, Germany (wiese@mpi-inf.mpg.de).

the amount of adaptivity and the quality of solutions. As a main problem, we want to determine the amount of recourse that is necessary to allow for provably near-optimal solutions.

Looking from a different perspective, we construct solutions that satisfy some adequate concept of *robustness*, where we measure robustness by the (amortized) *recourse budget* that is necessary to guarantee solutions of a particular quality.

More precisely, we consider the online MST problem and online TSP with recourse: An undirected graph is revealed online. In each iteration a new node becomes known, together with the connection costs to all previously arrived nodes. We assume that the graph is complete and that the cost function is metric, i.e., it satisfies the triangle inequality. The objective is to construct in each iteration a low-cost spanning tree (resp., tour) of the revealed vertices, without any assumption on the vertices that might arrive in the future. We measure the quality of the solution sequence with the standard competitive analysis framework by comparing online solutions to the offline optimum on the currently known subgraph. Depending on the problem, the current offline optimum is the MST or the optimal TSP tour.

We control the amount of recourse, i.e., how much the solution changes along iterations, by means of a budget that limits the number of edges that can be inserted in each iteration. We say that an algorithm needs budget $k$ if the number of inserted edges in each iteration is bounded by $k$. Similarly, the algorithm uses an *amortized budget* $k$ if up to iteration $t$ the total number of inserted edges is at most $t \cdot k$. Notice that by this definition, algorithms for the standard online MST problem (without recourse) have a budget of 1, whereas the online TSP without recourse is given a budget of 2.

It has been a longstanding open question whether a constant budget suffices to maintain constant-approximate solutions [4, 11]. As our main result we answer this question affirmatively.

**Related work.** The online MST and Steiner tree problems have been studied intensively. The best possible competitive ratio for online algorithms is known to be $\Theta(\log t)$, where $t$ is the number of iterations [11]. A simple greedy algorithm that connects a new node to the current tree through a shortest edge achieves this bound. Even in the special case of Euclidean distances, there is a lower bound of $\Omega(\frac{\log t}{\log \log t})$ on the competitive guarantee of any online algorithm [1].

Unlike in stochastic programming [6], where recourse actions are an important concept when optimizing under limited information, the literature on recourse models for online optimization seems rather sparse. Regarding our model, we are aware only of the work by Imase and Waxman [11] that deals with the online minimum Steiner tree problem with recourse (or *dynamic Steiner tree*). The model they introduce is slightly more general, as nodes not only arrive at but may also *depart* from the terminal set. For this setting, they give an algorithm that is 8-competitive and performs in $t$ iterations at most $O(t^{3/2})$ rearrangements. This translates by our definitions into an algorithm that requires an amortized recourse budget of $O(t^{1/2})$. In the more restricted setting considered in this paper, with no node leaving the terminal set, their algorithm achieves a competitive guarantee of 4. Furthermore, for the online MST problem, their algorithm is even 2-competitive when given the mentioned nonconstant amortized budget. The question of whether there is a constant-competitive algorithm that uses only a constant (amortized) budget has been left open, but the answer was conjectured to be affirmative.

A related online MST variant has been studied by Dynia, Korzeniowski, and

Kutylowski [8]. Here, in each iteration the cost of some edge increases or decreases by one unit. The task is to maintain a sequence of *optimal* MSTs with the goal of minimizing the number of rearrangements. They give a best possible deterministic algorithm that is $O(t^2)$-competitive and a randomized algorithm with expected competitive ratio $O(t \log t)$.

The TSP is one of the most prominent problems in combinatorial optimization [2, 13]. Despite remarkable recent progress, the best known approximation algorithm for the offline metric TSP is still the classic 3/2-approximation by Christofides [7]. Various online variants for TSP have been studied. In what is probably the most popular model, introduced by Ausiello et al. [3], the nodes that must be visited appear online over time as the salesman traverses its tour. The part of the tour that has not yet been visited can be adjusted arbitrarily when new nodes arrive, which is in strong contrast to our model. Small constant-competitive factors were shown when minimizing the total travel time in a metric space. In another online TSP variant, the nodes appear when the salesman moves, depending on its position. Whenever a new node is visited, all its neighbors are revealed. This, again, is in strong contrast to our problem, where the nodes appear independently of the current tour. The goal is to find a tour for exploring the entire unknown graph. Constant-competitive algorithms were shown for planar graphs [12] and, generally, graphs of bounded genus [14], even without any assumptions on the cost function. Despite the obvious interest in online TSP, we are not aware of any results for the recourse version considered in this paper.

**Our contribution.** Our main contribution, presented in section 3, is an online algorithm for the online MST problem with recourse that is $(1 + \varepsilon)$-competitive, for any $\varepsilon > 0$, when given an amortized budget of $O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$. This is the first significant improvement on the 2-competitive algorithm with nonconstant budget given by Imase and Waxman [11]. We complement our result by showing that any $(1+\varepsilon)$-competitive algorithm for the online MST problem needs an amortized budget of $\Omega(\frac{1}{\varepsilon})$. Thus, our algorithm is best possible up to logarithmic factors. Using a standard argument, we immediately obtain a $(2+\varepsilon)$-competitive algorithm for the online Steiner tree problem with the same amortized budget.

Our algorithm is simple and easy to implement, but it captures subtleties in the structure of the problem that allow for an improved analysis. Similarly to the algorithm proposed in [11], we implement the following natural idea: When a new node appears, we (i) connect it to its closest neighbor and (ii) iteratively perform edge swaps if they yield a sufficient improvement of the solution. The key difficulty when implementing this idea is to balance the number of swaps and the cost of the solution. As our crucial refinement of this approach, we introduce two *freezing rules* that effectively avoid performing unnecessary swaps. The first rule prevents removing edges whose cost is very small. The second rule is more subtle and prohibits an edge swap if the removed edge can be traced back to a subgraph whose MST has negligible cost compared to the current MST.

In section 4 we study the nonamortized version of the online MST problem. First, we notice that our previous results imply that the amortized budget framework is significantly more powerful than the nonamortized counterpart. Indeed, we contrast our findings for a constant amortized budget with a simple example showing that no online algorithm can be $(2 - \varepsilon)$-competitive, for any $\varepsilon > 0$, if it uses a nonamortized constant budget. In a preliminary version of this paper [15] we studied a natural greedy online algorithm with budget 2. We proved that it has a constant-competitive ratio if a certain property is satisfied. We also conjectured that this property always

holds. We show a family of graphs that does not satisfy this property, falsifying our conjecture. However, this does not rule out that the algorithm is constant competitive independently of this property.

A very natural approach to solving the online TSP with recourse is to combine the algorithms proposed for the online MST problem with the classical double-tree and shortcutting technique [13]. Indeed, with this technique most offline variants of TSP are equivalent to MST from an approximation point of view, and performance guarantees differ only by a factor of 2. Hence, one might be tempted to assume that the same conversion technique applies directly to the online model with recourse. However, we observe that this is not true. In section 5 we give examples in which two trees differ in just a single edge, but the standard shortcutting technique leads to completely different tours, no matter which Eulerian walk on the doubled tree edges is chosen. We overcome this difficulty by introducing a *robust* variant of the shortcutting technique: We choose the Eulerian tour in a specific way and keep track of which copy of a node in the Eulerian tour is visited by the TSP tour. With this robust shortcutting technique we show that any algorithm for the online MST problem with recourse can be converted to an algorithm for the online TSP by increasing the competitive ratio by a factor 2 and the budget by a factor 4.

**Subsequent work.** After the appearance of a preliminary version of this work [15], Gu, Gupta, and Kumar [9] answered the main open question posed in [11]. Namely, they give a constant-competitive algorithm with budget 2 for the nonamortized case. In contrast to our analysis, their algorithm relies on a clever use of the primal-dual method. Moreover, they show that increasing the budget from 1 to 2 only every $1/\delta$ iterations suffices to obtain a $2^{O(1/\delta)}$-competitive algorithm, effectively showing that it is possible to trade solution quality for robustness. Finally, they are able to improve the analysis for the amortized setting. They show that the freezing rules of the greedy algorithm are not necessary, and that even without them the required amortized budget is $2/\varepsilon$. This result is asymptotically tight by the $\Omega(1/\varepsilon)$ bound provided in this work. Subsequently, Gupta and Kumar [10] studied the dynamic Steiner tree problem, in which terminal nodes might also depart from the terminal set. For this problem they are able to obtain a constant-competitive algorithm with constant amortized budget. If there are only node departures, that is, if no new terminal appears, they can obtain the same result even in the nonamortized scenario.

**2. Problem definitions.** An instance of the online MST problem with recourse is defined as follows. A sequence of nodes $v_0, v_1, \dots$ arrives online one by one. In iteration $t \geq 0$, node $v_t$ appears, together with all edges $v_t v_s$ for $s \in \{0, \dots, t-1\}$. The cost $c(e) \geq 0$ of an edge $e$ is revealed when the edge appears. We assume that the edges are undirected and that the costs satisfy the triangle inequality, that is, $c(vw) \leq c(vz) + c(zw)$ for all nodes $v, w, z$. For each iteration $t$, the current graph is denoted by $G_t = (V_t, E_t)$, where $V_t = \{v_0, \dots, v_t\}$ and $E_t = V_t \times V_t$, that is, $G_t$ is a complete graph. We are interested in constructing an online sequence of edge subsets $T_0, T_1, T_2, \dots$, where $T_0 = \emptyset$ and for each $t \geq 1$ the graph $(V_t, T_t)$ is a spanning tree of $G_t$. To simplify notation we will refer to the tree $(V_t, T_t)$ by its set of edges $T_t$. We say that the sequence needs budget $k$ if $|T_t \setminus T_{t-1}| \leq k$ for all $t \geq 1$. A relaxed version of this concept is obtained by considering the average or *amortized* budget $k$, $\sum_{s=1}^{t} |T_s \setminus T_{s-1}| \leq k \cdot t$.

In online TSP with recourse, the nodes of a complete metric graph arrive in the same online fashion as described above and yield a sequence of graphs $G_0, \dots, G_t, \dots$.

The objective now is to construct a sequence of TSP tours $Q_0, Q_1, \ldots, Q_t, \ldots$ for graphs $G_0, G_1, \ldots, G_t, \ldots$ with low cost for each tour, where $Q_0 = Q_1 = \emptyset$. We apply the same budget constraints as for trees.

The performance of our online algorithms is measured using classic competitive analysis. Let $\mathrm{OPT}_t$ be the cost of an MST (resp., TSP tour) of $G_t$, and for a given set of edges $E$ denote $c(E) := \sum_{e \in E} c(e)$. We say that an algorithm is $\alpha$-competitive for some $\alpha \geq 1$ if for any input sequence the algorithm computes a solution sequence $X_0, X_1, \ldots$ such that $c(X_t) \leq \alpha \cdot \mathrm{OPT}_t$ for each $t$.

**3. An online approximation scheme with constant amortized budget.** The main result of this section is a $(1+\varepsilon)$-competitive algorithm for the online MST problem with amortized recourse budget $O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$ for any $\varepsilon > 0$. This improves on a previous 2-competitive algorithm that requires a nonconstant amortized budget [11]. We first give a lower bound on the amortized budget of any $(1+\varepsilon)$-competitive algorithm which shows that our budget bound is best possible up to logarithmic factors. This result also implies that 1-competitive solutions need nonconstant amortized budget.

THEOREM 3.1. *Any $(1+\varepsilon)$-competitive algorithm for the online MST problem requires an amortized recourse budget of $\Omega(\frac{1}{\varepsilon})$.*

*Proof.* We use a construction similar to that given by Imase and Waxman [11, Figure 4]. Let us fix $\varepsilon > 0$, and let $n := \lfloor \ln(2)/\ln(1+2\varepsilon) \rfloor$. Consider an instance with $n+1$ vertices $v_0, \ldots, v_n$. The costs are chosen so that, for every iteration $t$, the cost of any edge incident to $v_t$ is a $(1+2\varepsilon)$ factor smaller than any other edge previously available. The precise definition is as follows: For each $t \in \{1, \ldots, n\}$ define $c(v_s v_t) = c_t := (1+2\varepsilon)^{n-t}$ for any $s < t$. Note that our choice of $n$ implies that $c_t \in [1, 2]$ for all $t \in \{1, \ldots, n\}$. Hence, the constructed graph is metric.

Let $T_0, \ldots, T_n$ be the output of a $(1+\varepsilon)$-competitive algorithm, and denote by $k_t$ the budget used in iteration $t$, i.e., $k_t := |T_t \setminus T_{t-1}|$. Since up to iteration $t-1$ all available edges have cost at least $c_{t-1}$, then

$$c(T_t) \geq c_t \cdot k_t + c_{t-1} \cdot (t - k_t) = (c_{t-1} - c_t) \cdot k_t + c_{t-1} \cdot t.$$

On the other hand, $T_t$ is a $(1+\varepsilon)$-approximate solution, and therefore

$$c(T_t) \leq (1+\varepsilon)\mathrm{OPT}_t = (1+\varepsilon) c_t \cdot t.$$

Combining these two inequalities and simple algebra implies that

$$k_t \geq t \cdot \frac{c_{t-1} - (1+\varepsilon)c_t}{c_t - c_{t-1}} = t \cdot \frac{(1+2\varepsilon) - (1+\varepsilon)}{(1+2\varepsilon) - 1} = \frac{t}{2},$$

where the second-to-last equality follows from the definition of $c_t$. Recalling that $n := \lfloor \ln(2)/\ln(1+2\varepsilon) \rfloor$, the theorem follows since

$$\sum_{t=1}^{n} k_t \geq \frac{1}{2} \sum_{t=1}^{n} t = \frac{n(n+1)}{4} \geq \frac{n \ln(2)}{4 \ln(1+2\varepsilon)} \in \Omega\left(\frac{n}{\varepsilon}\right). \qquad \square$$

In the remainder of this section we prove our main result.

THEOREM 3.2. *There exists a $(1+\varepsilon)$-competitive algorithm for the online MST problem with amortized recourse budget $O\left(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$.*

A natural approach to solving the online MST problem is as follows. Let $T_{t-1}$ be the tree solution in iteration $t-1$. To construct $T_t$, we first find the closest connection, edge $g_t$, between the new node $v_t$ and $V_{t-1}$, and initialize $T_t$ as $T_{t-1} \cup \{g_t\}$. We can diminish the cost of $T_t$ by subsequently inserting a low-cost edge $f$ to $T_t$ and removing the largest edge $h$ in the formed cycle. Indeed, performing this swapping operation often enough will eventually turn $T_t$ into the optimal solution, i.e., the MST. The difficulty lies in balancing the number of swaps that increases the budget and the closeness of the tree to the MST which diminishes the total cost. We find a balance by introducing two *freezing rules* that effectively avoid unnecessary swaps while maintaining low-cost solutions.

The intuition behind these freezing rules is as follows. If at iteration $t$ the optimal value $\mathrm{OPT}_t$ is much higher than $\mathrm{OPT}_s$ for some $s < t$, e.g., $\mathrm{OPT}_s \in O(\varepsilon \mathrm{OPT}_t)$, then the edges in $T_s$—whose total cost is approximately $\mathrm{OPT}_s$—are already very cheap. Thus, replacing these edges by cheaper ones would only waste rearrangements. To avoid technical difficulties we use $\mathrm{OPT}_t^{\max} := \max\{\mathrm{OPT}_s : 1 \le s \le t\}$ instead of $\mathrm{OPT}_t$ to determine whether $\mathrm{OPT}_s \in O(\varepsilon \mathrm{OPT}_t)$; note that the value of $\mathrm{OPT}_t$ can decrease as new nodes arrive; however, the triangle inequality guarantees that $\mathrm{OPT}_t \le \mathrm{OPT}_t^{\max} \le 2\mathrm{OPT}_t$.

With this in mind, we define $\ell(t)$ as the largest iteration with ignorable edges with respect to $\mathrm{OPT}_t^{\max}$; i.e., $\ell(t) \le t-1$ is the largest nonnegative integer such that $\mathrm{OPT}_{\ell(t)}^{\max} \le \varepsilon \mathrm{OPT}_t^{\max}$.

For our first freezing rule we consider sequences of edges $(g_s^0, \dots, g_s^{i(s)})$, where $g_s^0$ corresponds to the greedy edge added at iteration $s$ (that is, an edge connecting $v_s$ to one of its closest neighbors in $V_{s-1}$). At the moment when edge $g_s^0$ is removed from our solution we define $g_s^1$ as the element that replaces $g_s^0$. In general, $g_s^i$ is the edge that was swapped in for edge $g_s^{i-1}$. In this way, the only edge in the sequence that belongs to the current solution is $g_s^{i(s)}$. Note that $i(s)$ changes through the iterations, and thus it depends on $t$. Notationally, $i(s)$ will refer to the value at the iteration under consideration in the current context (unless it is stated otherwise). With this construction, we *freeze* a sequence $(g_s^0, \dots, g_s^{i(s)})$ in iteration $t$ if $s \le \ell(t)$. Note that since $\ell(\cdot)$ is nondecreasing, once the sequence is frozen by this rule, edge $g_s^{i(s)}$ will stay indefinitely in the solution.

Our second freezing rule is somewhat simpler. We skip swaps that remove edges that are too small, namely, smaller than $\varepsilon \mathrm{OPT}_t^{\max}/(t-\ell(t))$. Note that this quantity is not necessarily monotone, so a sequence that is frozen by this rule might get unfrozen in a later iteration. Combining these ideas, we propose the following algorithm.

**Algorithm Sequence-Freeze**

Define $T_0 = \emptyset$. For each iteration $t \ge 1$ do as follows.

1. Let $g_t^0$ be any minimum cost edge in $\{v_t v_s : 0 \le s \le t-1\}$.
2. Initialize $T_t := T_{t-1} \cup \{g_t^0\}$ and $i(t) := 0$.
3. While there exists a pair of edges $(f, h) \in (E_t \setminus T_t) \times T_t$ such that $(T_t \cup \{f\}) \setminus h$ is a tree, and the following three conditions are satisfied
   - (C1) $c(h) > (1+\varepsilon) \cdot c(f)$,
   - (C2) $h = g_s^{i(s)}$ for some $s \ge \ell(t)+1$, and
   - (C3) $c(h) > \varepsilon \frac{\mathrm{OPT}_t^{\max}}{t-\ell(t)}$,

   set $T_t := (T_t \cup \{f\}) \setminus \{h\}$, $i(s) := i(s)+1$, and $g_s^{i(s)} := f$.
4. Return $T_t$.

Conditions (C2) and (C3) correspond to the two freezing rules described above. In the following we show that this algorithm is $(1+O(\varepsilon))$-competitive and uses amortized budget $O\left(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon}\right)$.

**Competitive analysis.** To prove that our algorithm is $(1+O(\varepsilon))$-competitive, we first show that conditions (C1) and (C3) imply a cost increase of at most a factor $(1+3\varepsilon)$. Then we show that skipping swaps because of condition (C2) can increase the cost of the solution by at most $O(\varepsilon\mathrm{OPT}_t)$.

To simplify notation, let us fix an iteration $t$ and set $\ell := \ell(t)$. We partition the tree $T_t$ into two disjoint subsets, $T_t = T_t^{\mathrm{old}} \cup T_t^{\mathrm{new}}$, where $T_t^{\mathrm{old}} := \{g_1^{i(1)}, \dots, g_\ell^{i(\ell)}\}$ and $T_t^{\mathrm{new}} := \{g_{\ell+1}^{i(\ell+1)}, \dots, g_t^{i(t)}\}$. We start by bounding the cost of $T_t^{\mathrm{new}}$. To this end we use the following fact that follows from a classic result on matroid theory; see, e.g., [18, Corollary 39.12a].

LEMMA 3.3. *Consider a connected graph $G = (V, E)$, and let $T_1$ and $T_2$ be two spanning trees for this graph. Then there exists a bijection $\Psi : T_1 \setminus T_2 \to T_2 \setminus T_1$ such that each edge $e \in T_1 \setminus T_2$ belongs to the unique cycle contained in $T_1 \cup \{\Psi(e)\}$.*

LEMMA 3.4. *For each iteration $t$ it holds that $c(T_t^{\mathrm{new}}) \le (1+3\,\varepsilon)\mathrm{OPT}_t$.*

*Proof.* To bound the cost of $T_t^{\mathrm{new}}$, we consider each edge $h \in T_t^{\mathrm{new}} \setminus T_t^*$, where $T_t^*$ is an MST in iteration $t$. Each such edge $h$ was not removed from our solutions because of condition (C1) or (C3) in the algorithm. Since $|T_t^{\mathrm{new}}| = t - \ell(t)$, the total cost of the edges not removed because of condition (C3) is at most $\varepsilon\mathrm{OPT}_t^{\max} \le 2\varepsilon\mathrm{OPT}_t$. For the edges corresponding to condition (C1), by Lemma 3.3 we can construct a bijection $\Psi : T_t \setminus T_t^* \to T_t^* \setminus T_t$ such that each $h \in T_t$ is in the unique circuit contained in $T_t \cup \{\Psi(h)\}$. This implies that $(T_t \cup \{\Psi(h)\}) \setminus \{h\}$ is a tree. Thus, if $h \in T_t^{\mathrm{new}}$ is not removed because of condition (C1), then $c(h) \le (1+\varepsilon)\cdot c(\Psi(h))$, and thus the total cost of these edges is at most $(1+\varepsilon)\mathrm{OPT}_t$. We conclude that $c(T_t^{\mathrm{new}}) \le 2\varepsilon\mathrm{OPT}_t + (1+\varepsilon)\mathrm{OPT}_t = (1+3\varepsilon)\mathrm{OPT}_t$. ∎

For bounding the cost of $T_t^{\mathrm{old}}$, we use induction over the iterations. The inductive step is given in the following lemma.

LEMMA 3.5. *Let $\varepsilon < \frac{1}{7}$. Consider an iteration $t$ and suppose that $c(T_{\ell(t)}) \le (1+7\varepsilon)\mathrm{OPT}_{\ell(t)}$. Then it holds that $c(T_t^{\mathrm{old}}) \le 4\,\varepsilon\,\mathrm{OPT}_t$.*

*Proof.* Recall that we denote $\ell = \ell(t)$. Notice that whenever the algorithm removes an edge, it is replaced by an edge of smaller cost. Thus, for each $s$ we have that $c(g_s^0) > c(g_s^1) > \cdots > c(g_s^{i(s)})$. Since each element in $T_\ell$ must belong to a sequence $g_s^0, g_s^1, \dots, g_s^{i(s)}$ for some $s \le \ell$, we conclude that $c(T_t^{\mathrm{old}}) \le c(T_\ell)$. Using our induction hypothesis and that $\varepsilon < 1/7$ we conclude that

$$\begin{aligned} c(T_t^{\mathrm{old}}) &\le c(T_\ell) \le (1+7\varepsilon)\mathrm{OPT}_\ell \\ &\le 2\mathrm{OPT}_\ell \le 2\mathrm{OPT}_\ell^{\max} \le 2\varepsilon\mathrm{OPT}_t^{\max} \le 4\varepsilon\mathrm{OPT}_t, \end{aligned}$$

where the second-to-last inequality follows from the definition of $\ell = \ell(t)$ and the last one since $\mathrm{OPT}_t^{\max} \le 2\mathrm{OPT}_t$ for all $t$. ∎

The above reasoning implies the following lemma.

LEMMA 3.6. *Algorithm* SEQUENCE-FREEZE *is $(1+7\varepsilon)$-competitive for any $\varepsilon < \frac{1}{7}$.*

*Proof.* Recall that we are considering a fixed value $\varepsilon < 1/7$, and that we want to show by induction that $c(T_t) \le (1+7\varepsilon)\cdot\mathrm{OPT}_t$ for all $t$. Clearly this is satisfied for $t \in$

$\{0,1\}$. For a given $t \geq 2$, the induction hypothesis implies that $c(T_\ell) \leq (1+7\varepsilon)\cdot \mathrm{OPT}_\ell$, and thus Lemmas 3.4 and 3.5 imply that

$$c(T_t) = c(T_t^{\mathrm{old}}) + c(T_t^{\mathrm{new}}) \leq 4\varepsilon \mathrm{OPT}_t + (1+3\varepsilon)\mathrm{OPT}_t. \qquad \square$$

**Amortized budget bound.** To bound the amortized budget of the algorithm, we define $k_q := |T_q \setminus T_{q-1}|$ and prove that for every $t \geq 1$ it holds that $\sum_{q=1}^{t} k_q \leq D_\varepsilon \cdot t$, where $D_\varepsilon \in O\left(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon}\right)$. To this end, we will first show that the total number of rearrangements needed in iterations $\ell(t)+1$ to $t$ is proportional to $t - \ell(\ell(t)+1)$. As shown in the next lemma, this implies our claim on the amortized budget by losing a factor of 2 on the guarantee.

LEMMA 3.7. *Assume that* $\sum_{q=\ell(t)+1}^{t} k_q \leq C_\varepsilon \cdot (t - \ell(\ell(t)+1))$ *for every* $t \geq 1$ *with* $C_\varepsilon \in O(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$. *Then for every* $t \geq 1$ *it holds that* $\sum_{q=1}^{t} k_q \leq 2C_\varepsilon \cdot t$.

*Proof.* We show by induction that for all $t \geq 1$,

$$\sum_{q=1}^{t} k_q \leq 2 \cdot C_\varepsilon \cdot \ell(t) + C_\varepsilon \cdot (t - \ell(t)).$$

Notice that this directly implies the lemma. Clearly the inequality holds for $t = 1$ since $k_1 = 0$. Let us fix $t \geq 2$, and assume that the inequality is valid for all $t' \leq t-1$. In particular this holds for $t' = \ell(t) \leq t-1$. By denoting $\ell(\ell(t)) = \ell^2(t)$, we have

$$\sum_{q=1}^{\ell(t)} k_q \leq 2 \cdot C_\varepsilon \cdot \ell^2(t) + C_\varepsilon \cdot (\ell(t) - \ell^2(t)).$$

Also, the assumption of this lemma implies that

$$\sum_{q=\ell(t)+1}^{t} k_q \leq C_\varepsilon(t - \ell(\ell(t)+1)) \leq C_\varepsilon(t - \ell^2(t)),$$

where the last inequality follows since $\ell(\cdot)$ is nondecreasing. Summing together the last two inequalities we obtain

$$\begin{aligned}\sum_{q=1}^{t} k_q &\leq 2C_\varepsilon \cdot \ell^2(t) + C_\varepsilon \cdot (\ell(t) - \ell^2(t)) + C_\varepsilon(t - \ell^2(t)) \\ &\leq C_\varepsilon \cdot (t + \ell(t)) = 2C_\varepsilon \cdot \ell(t) + C_\varepsilon \cdot (t - \ell(t)).\end{aligned}$$

With this we showed the induction, and thus the lemma follows. $\square$

It remains to prove that the assumption of Lemma 3.7 holds. The two freezing rules, conditions (C2) and (C3), are crucial for this purpose. Indeed, we will bound the length of the sequences $(g_s^0, \ldots, g_s^{i(s)})$, which will give a direct bound on $\sum_{q=\ell(t)+1}^{t} k_q \leq C_\varepsilon \cdot (t - \ell(\ell(t)+1))$. This can be done since by condition (C1), we swap edges only when the cost decreases by a factor of $(1+\varepsilon)$, that is, $c(g_s^j) < c(g_s^{j-1})/(1+\varepsilon)$ for each $j$. Thus, the length of this sequence is upper bounded by $\log_{1+\varepsilon} c(g_s^0) - \log_{1+\varepsilon} c(g_s^{i(s)-1}) + 1$. We can bound this quantity further by lower bounding the cost $g_s^{i(s)-1}$ with our freezing rules and by exploiting a particular cost structure of greedy edges $g_s^0$.

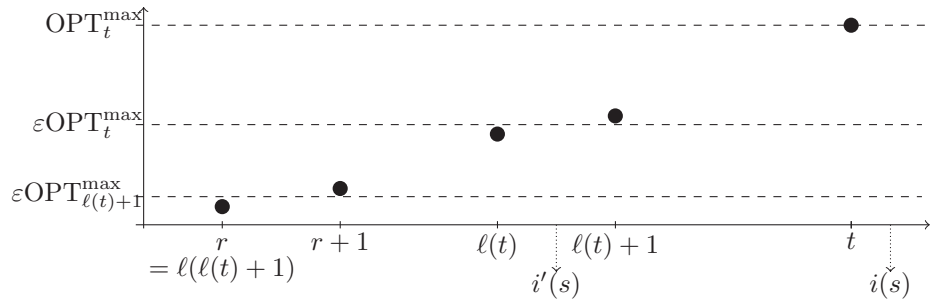



FIG. 3.1. *Sketch of notation. The abscissa denotes the iterations and the ordinate denotes the value of* $\mathrm{OPT}_{(\cdot)}^{\max}$.

More precisely, consider the values $i(s)$ at the end of iteration $t$, and let $i'(s)$ be the value of $i(s)$ at the beginning of iteration $\ell(t)+1$ (and $i'(s) := 0$ for $s \geq \ell(t)+1$); see Figure 3.1. By condition (C2), in iterations $\ell(t)+1$ to $t$ we only touch edges belonging to $\{g_s^{i'(s)}, g_s^{i'(s)+1}, \ldots, g_s^{i(s)}\}$ for some $s \in \{\ell(\ell(t)+1)+1, \ldots, t\}$. Let us denote $r := \ell(\ell(t)+1)$. Then

$$\sum_{q=\ell(t)+1}^{t} k_q \leq \sum_{s=r+1}^{t} (i(s) - i'(s) + 1) = 2(t-r) + \sum_{s=r+1}^{t} (i(s) - 1 - i'(s)). \tag{3.1}$$

We now upper bound each term $i(s)-1-i'(s)$ for $s \in \{r+1, \ldots, t\}$, which corresponds to the length of the sequence $(g_s^{i'(s)+1}, g_s^{i'(s)+2}, \ldots, g_s^{i(s)-1})$.

LEMMA 3.8. *For each* $s \in \{r, r+1, \ldots, t\}$ *it holds that*

$$i(s) - 1 - i'(s) \leq \frac{1}{\ln(1+\varepsilon)} \cdot \left(\ln c(g_s^0) - \ln c(g_s^{i(s)-1})\right). \tag{3.2}$$

*Proof.* By condition (C1) whenever we add an edge $g_s^j$ and remove $g_s^{j-1}$ it holds that $c(g_s^j) < c(g_s^{j-1})/(1+\varepsilon)$. Then,

$$\begin{aligned} i(s) - 1 - i'(s) &\leq \log_{1+\varepsilon} c(g_s^{i'(s)}) - \log_{1+\varepsilon} c(g_s^{i(s)-1}) \\ &\leq \log_{1+\varepsilon} c(g_s^0) - \log_{1+\varepsilon} c(g_s^{i(s)-1}) \\ &= \frac{1}{\ln(1+\varepsilon)} \cdot \left(\ln c(g_s^0) - \ln c(g_s^{i(s)-1})\right). \end{aligned}$$

∎

In the next claims, we lower bound $c(g_s^{i(s)-1})$ and upper bound $\sum_{s=r+1}^{t} \ln c(g_s^0)$. These bounds applied to inequality (3.2), together with (3.1), will lead to the desired bound on $\sum_{q=\ell(t)+1}^{t} k_q$.

PROPOSITION 3.9. *Due to condition* (C3)*, it holds that either* $c(g_s^{i(s)-1}) \geq \varepsilon^2 \frac{\mathrm{OPT}_t^{\max}}{(t-r)}$ *or* $i(s) - 1 - i'(s) \leq 0$.

*Proof.* Assume that $i(s) - 1 > i'(s)$; otherwise we are done. This implies that edge $g_s^{i(s)-1}$ was swapped for edge $g_s^{i(s)}$ in some iteration $q^* \in \{\ell(t)+1, \ldots, t\}$. By condition (C3) this implies that

$$c(g_s^{i(s)-1}) \geq \varepsilon \cdot \frac{\mathrm{OPT}_{q^*}^{\max}}{(q^* - \ell(q^*))} \geq \varepsilon \frac{\mathrm{OPT}_{\ell(t)+1}^{\max}}{(t-r)} \geq \varepsilon^2 \frac{\mathrm{OPT}_t^{\max}}{(t-r)},$$

where the last inequality follows by the definition of $\ell(\cdot)$ (see Figure 3.1). ∎

Recall that for any $s$, $g_s^0$ is a closest connection between $v_s$ and any element in $\{v_0, \dots, v_{s-1}\}$. Such greedy edges are known to have a special cost structure, as shown by Alon and Azar [1].

LEMMA 3.10 (see [1]). *Let $e_1, \dots, e_t$ be a relabeling of the greedy edges $g_1^0, \dots, g_t^0$ such that $c(e_1) \geq c(e_2) \geq \cdots \geq c(e_t)$. Then, $c(e_j) \leq 2\frac{\mathrm{OPT}_t}{j}$ for all $j \in \{1, \dots, t\}$.*

LEMMA 3.11. $\sum_{s=r+1}^{t} \ln c(g_s^0) \leq (t-r) \cdot (\ln(2 \cdot \mathrm{OPT}_t^{\max}) - \ln(t-r) + 1)$.

*Proof.* We rename edges $\{g_{r+1}^0, \dots, g_t^0\} = \{e_1, \dots, e_{t-r}\}$ such that $c(e_1) \geq \cdots \geq c(e_{t-r})$. Lemma 3.10 implies that $c(e_j) \leq 2\frac{\mathrm{OPT}_t}{j} \leq 2\frac{\mathrm{OPT}_t^{\max}}{j}$ for all $j$. Thus,

$$\sum_{s=r+1}^{t} \ln c(g_s^0) \leq \sum_{j=1}^{t-r} \ln c(e_j) \leq (t-r) \ln(2 \cdot \mathrm{OPT}_t^{\max}) - \sum_{j=1}^{t-r} \ln j.$$

The lemma follows since for any $n \in \mathbb{N}_{>0}$ it holds that $\sum_{j=1}^{n} \ln j \geq \int_1^n \ln(x)dx = n\ln(n) - n$. ∎

The above statement and basic arithmetic imply the desired bound.

LEMMA 3.12. *For each $t \geq 1$ it holds that $\sum_{q=1}^{t} k_q \leq D_\varepsilon \cdot t$, where $D_\varepsilon \in O\left(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$.*

*Proof.* Combining (3.2), Proposition 3.9, and Lemma 3.11 in (3.1) we conclude that

$$\begin{aligned}
\sum_{q=\ell(t)+1}^{t} k_q &\leq 2(t-r) + \frac{(t-r)\ln(2 \cdot \mathrm{OPT}_t^{\max}) + (t-r) - (t-r) \cdot \ln(\varepsilon^2 \cdot \mathrm{OPT}_t^{\max})}{\ln(1+\varepsilon)} \\
&= (t-r) \cdot \left(2 + \frac{\ln\left(\frac{2}{\varepsilon^2}\right) + 1}{\ln(1+\varepsilon)}\right) \\
&\leq (t-r) \cdot C_\varepsilon
\end{aligned}$$

for some $C_\varepsilon \in O\left(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$. Hence, the condition of Lemma 3.7 is fulfilled and the claim of the lemma follows. ∎

Our main result, Theorem 3.2, follows directly from Lemmas 3.6 and 3.12.

**4. The nonamortized scenario.** For the amortized setting we have seen that with a sufficient (but constant) budget we can obtain a competitive ratio of $1 + \varepsilon$. In the nonamortized setting, however, there can be no $(2 - \varepsilon)$-competitive algorithm with a constant budget.

PROPOSITION 4.1. *For every fixed $\varepsilon > 0$ and $k \in \mathbb{N}_0$, there is no $(2 - \varepsilon)$-competitive algorithm with budget $k$.*

*Proof.* Let us fix a value $n \geq 1$. Consider a complete graph with vertices $v_0, \dots, v_n$ such that $c(v_t v_n) = 1$ for all $t \leq n-1$. All other edges $v_s v_t$ with $s, t \leq n-1$ have cost 2. Note that this graph is metric and that the MST of $G_n$ is a star centered at $v_n$ whose total cost is $n$. However, in any sequence of trees, tree $T_{n-1}$ can have only edges of cost 2. Hence, tree $T_n$ has to contain at least $n - k$ edges of weight 2, and thus $c(T_n) \geq 2(n-k) + k = 2n - k$. We conclude that the competitive ratio of the algorithm is at least $(2n-k)/n = 2 - k/n$, which is larger than $2 - \varepsilon$ for sufficiently large $n$. ∎

Recently, Gu, Gupta, and Kumar [9] gave an algorithm with budget 2 and constant-competitive guarantee. However, the competitive guarantee is a large (unspecified) constant, and their primal-dual algorithm is relatively intricate and hard to implement. In contrast, in this section we consider a simple and natural online algorithm with budget 2, as was also suggested in [11]. In a preliminary version of this paper [15], we proved that if the sequence of optimal solutions fulfills a particular Property (P), then the algorithm is constant competitive. We conjectured that (P) holds for every instance. However, we now show that the conjecture is not true in general. This does not rule out a constant-competitive ratio that is independent of Property (P). For the sake of completeness and to encourage further investigations, we state here the algorithm and Property (P), and we give a counterexample where (P) does not hold.

**Algorithm Greedy**

*Input*: A sequence of complete graphs $G_t = (V_t, E_t)$ for $t \geq 1$ revealed online with $V_0 = \{v_0\}$ and $V_t = V_{t-1} \cup \{v_t\}$ for all $t \geq 1$. A metric cost function $c$ revealed together with the edges.

Define $T_0 := \emptyset$. For each iteration $t \geq 1$ do as follows.

1. Define $g_t$ as any edge of minimum cost connecting $v_t$ to any node in $V_{t-1}$.
2. If there exists a pair of edges $g_t, f_t$ satisfying that
   - $(T_{t-1} \cup \{g_t, f_t\}) \setminus \{h_t\}$ is a tree,
   - $f_t$ is adjacent to $v_t$, and
   - $c(f_t) \leq \frac{c(h_t)}{2}$,

   then choose such a pair maximizing $c(h_t) - c(f_t)$ and return
   $$T_t := (T_{t-1} \cup \{g_t, f_t\}) \setminus \{h_t\}.$$
3. If there is no such pair of edges, return $T_t := T_{t-1} \cup \{g_t\}$.

Notice that this algorithm uses a recourse budget of at most 2. We showed in [15] that if a particular Property (P) holds, then the algorithm is constant competitive. To state this property we need the following definition: Given a complete graph $G = (V, E)$ and a nonnegative cost function $c$ on the edges, we say that the graph is 2-*metric* if for every cycle $C \subseteq E$ it holds that $c(e) \leq 2 \cdot c(C \setminus \{e\})$ for all $e \in C$. Moreover, for a given real number $x$ we define $x_+ := \max\{x, 0\}$ and $x_- := \max\{-x, 0\}$. Note that $x = x_+ - x_-$. Also, denote $\Delta\mathrm{OPT}_t := \mathrm{OPT}_t - \mathrm{OPT}_{t-1}$.

Property (P). *There exists a constant $\alpha \geq 1$ satisfying the following. Consider any input sequence $G_0, G_1, \ldots, G_n$ of the online MST problem with recourse, with a cost function $c'$ on the edges such that $G_t$ is 2-metric for all $t \geq 0$. If $\mathrm{OPT}_t$ denotes the optimal cost of the tree in iteration $t$ for cost function $c'$, then for all $t \geq 1$ it holds that $\sum_{s=1}^{t} (\Delta\mathrm{OPT}_s)_- \leq \alpha \cdot \mathrm{OPT}_t$.*

Theorem 4.2. *If Property* (P) *holds, then Algorithm* Greedy *is $(2 \cdot (\alpha + 1))$-competitive.*

In order to show the theorem, we considered $I \subseteq \mathbb{N}_0$ as the subset of iterations $t$ such that $T_t = T_{t-1} \cup \{g_t\}$. In particular, for all iterations $t \notin I$ we have that

$$c(T_t) = c(T_{t-1}) + c(g_t) + c(f_t) - c(h_t) \leq c(T_{t-1}) + 2c(f_t) - c(h_t) \leq c(T_{t-1}).$$

Setting $\Delta T_t := c(T_t) - c(T_{t-1})$ for each $t$, we can decompose the cost of $T_t$ by $c(T_t) =$

$\sum_{s=1}^{t} \Delta T_t$, and by our previous observation, we have

$$c(T_t) \leq \sum_{s \in I, s \leq t} \Delta T_s = \sum_{s=1}^{t} (\Delta T_s)_+. \tag{4.1}$$

By bounding the right-hand side of this inequality, Property (P) implies the previous theorem (for details, see [15]). However, the following example shows that this inequality is not enough to prove that $c(T_t)$ is within a constant factor of the optimal cost. This implies that Property (P) is not true for every graph (although the property holds for the particular input of the example). A simple modification of the construction, explained below, gives an explicit input instance that directly disproves Property (P).

*Example* 4.3. All of our nodes are points in the real line, and the costs correspond to the Euclidean distances. The iterations are divided by phases $P_0, \ldots, P_k$. Each phase $P_i$ defines a collection of $\ell_i$ intervals $I_1^i, \ldots, I_{\ell_i}^i$. In $P_0$ we get a node at 0 and at 1 which defines the interval $I_1^0 = [0, 1)$. In $P_1$ two nodes appear, the first at $1/3$ and the second at $2/3$. This breaks $I_1^0$ into three intervals $I_1^1 = [0, 1/3)$, $I_2^1 = [1/3, 2/3)$, and $I_3^1 = [2/3, 1)$. Then we operate recursively, so that the intervals of phase $P_i$ are formed by breaking each interval $I_j^{i-1} = [a, b)$ of $P_{i-1}$ into three intervals $[a, a + (b-a)/3)$, $[a + (b-a)/3, a + 2(b-a)/3)$, and $[a + 2(b-a)/3, b)$. Additionally, two new nodes in $I_j^{i-1}$ arrive in phase $P_i$, one at $a + (b-a)/3$ and the other at $a + 2(b-a)/3$. Note that the number of nodes is basically tripled in each phase, so that the number of nodes $n$ at the end of phase $k$ is in $O(3^k)$.

Now we show that if we apply our algorithm to this example, then the sum of the increments of the solution is unbounded. Since the optimal costs of the example equals 1 in each iteration, this shows that the inequality in (4.1) does not suffice to prove that the algorithm is constant competitive.

LEMMA 4.4. *Applying our algorithm to Example* 4.3 *yields a sequence of solutions* $T_0, T_1, \ldots, T_n$ *such that*

$$\sum_{t=1}^{n} (\Delta T_t)_+ \in \Theta(\log n),$$

*while* $c(\mathrm{OPT}_n) = 1$ *for all* $n$.

*Proof.* Let $\mathcal{I}_i = \{I_1^i, \ldots, I_{\ell_i}^i\}$ be the collection of intervals for phase $i$. We say that an edge $e$ *corresponds* to an interval $I_j^{i-1}$ if the endpoints of $e$ correspond to the extreme points of $I_j^{i-1}$. We show by induction that at the end of phase $P_i$ the solution given by the algorithm consists of exactly the edges corresponding to each interval in $\mathcal{I}_i$. For the base case, note that at the end of $P_0$ the solution output by the algorithm connects the node 0 with the node 1 and thus satisfies the claimed property. Let us assume inductively that the property holds at the end of phase $P_{i-1}$. Consider an iteration in phase $P_i$, where an interval $I_j^{i-1} = [a, b)$ is broken into three intervals. When the node $a + (b-a)/3$ arrives, the algorithm simply connects this node to $a$ and does nothing else. Indeed, adding the edge $[a + (b-a)/3, b)$ implies that the edge that needs to be removed is $[a, b)$, which is not a valid operation in the algorithm. Importantly, this implies that the cost of the solution is increased by $(b-a)/3$. However, when the node $a + 2(b-a)/3$ arrives, the algorithm adds the edges corresponding to $[a + (b-a)/3, a + 2(b-a)/3)$ and $[a + 2(b-a)/3, b)$ and removes the edge corresponding to $I_j^{i-1} = [a, b)$. This means that the three new

edges added by the algorithm correspond to the three new intervals $[a, a+(b-a)/3)$, $[a+(b-a)/3, a+2(b-a)/3)$, and $[a+2(b-a)/3, b)$. With this we finish the induction. Moreover, notice that for every phase $P_i$ with $\{1, \dots, k\}$, the total contribution of $P_i$ to $\sum_{t=1}^{n}(\Delta T_t)_+$ is exactly $1/3$. We conclude that $\sum_{t=1}^{n}(c(T_t) - c(T_{t-1}))_+ = 1 + k/3$. The lemma follows since $k \in \Theta(\log n)$. ∎

To give an explicit instance that disproves Property (P), it suffices to slightly modify the costs as follows: For any iteration $t$ and edge $e$ that belongs to $\mathrm{OPT}_t$ but not to $T_t$, we double the cost of $e$. It is easy to check that $T_t$ is an optimal solution for the new costs. Also the new cost function induces a 2-metric graph, and the optimal cost remains bounded by 2 in each iteration.

LEMMA 4.5. *There exist instances for which Property* (P) *does not hold.*

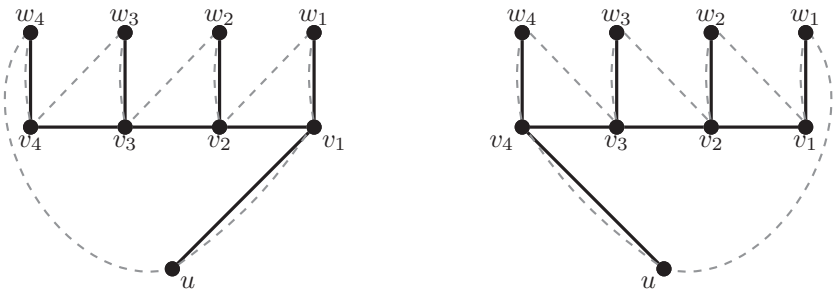
**5. Applications to TSP.** In this section we consider online TSP with recourse. In a natural approach we aim at combining our algorithms for the online MST problem with the classic *shortcutting* technique [13], which yields a sequence of tours, each with a cost at most twice the cost of the tree. This implies a $(2+\varepsilon)$-competitive algorithm. However, bounding the budget is intricate. Obviously, MSTs that differ in few edges might have quite different Eulerian walks and thus TSP tours. However, even when adapting the Eulerian walks as much as possible, the standard shortcutting might lead to very different TSP tours. In fact, we give examples in section 5.1 in which two trees $T, T'$ differ in just one edge and shortcutting *any* Eulerian walk on (doubled) $T'$ in the standard way yields a tour $Q'$, which differs from $Q$ for $T$ in an unbounded number of edges.

Our key ingredients for solving this problem in section 5.2 are as follows: In the case of an edge swap, we decompose the Eulerian walk $W$ corresponding to the tour $Q$ before the swap into 4 subwalks defined by the swapped edges, which we concatenate then in an appropriate way. Furthermore, we find a *robust* variant of the shortcutting technique: Instead of shortcutting the new Eulerian walk by visiting the *first* appearance of a node, we remember the copy of each node that we visit in $W$ to construct $Q$, and then we visit the same copy when constructing $Q'$. In general we cannot expect to obtain the same tour, but we prove that $Q$ and $Q'$ differ in at most 4 edges, which may be necessary when concatenating the subwalks. This will help us show the following key result.

THEOREM 5.1. *Consider a sequence of complete metric graphs $G_0, \dots, G_t, \dots$ where $G_t = (V_t, E_t)$ and $V_t = \{v_0, \dots, v_t\}$. Assume that in each iteration $t$ we are given a spanning tree $T_t$ for graph $G_t$. Then there exists an online algorithm that computes a sequence of tours $Q_0, Q_1, \dots, Q_t, \dots$ such that $c(Q_t) \le 2 \cdot c(T_t)$ and $|Q_t \setminus Q_{t-1}| \le 4 \cdot |T_t \setminus T_{t-1}|$ for all $t \ge 1$.*

This theorem immediately implies that most of our results for the online MST problem translate directly into the TSP setting by only increasing the competitive ratio and the budget (resp., amortized budget) by constant factors. In particular, online TSP admits an online $(2+\varepsilon)$-competitive algorithm with amortized budget $O(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$.

**5.1. Critical example and intuition for solution method.** We first explain the intuition to our approach by considering two spanning trees $R$ and $R'$, where $R' = (R \cup \{f\}) \setminus \{g\}$ for some edges $f \notin R$ and $g \in R$. Tree $R'$ is obtained by a single edge swap, which is arguably the simplest possible way of modifying $R$ to obtain a new spanning tree. Thus, for Theorem 5.1 to be true we must at least be able to update a tour $Q$ obtained from $R$ to a tour $Q'$ obtained from $R'$ such that $|Q' \setminus Q| \le 4$. We




(a) A tree $R$ (solid lines) and a possible tour obtained by taking shortcuts (dashed lines).

(b) A tree $R' = (R \cup \{uv_4\}) \setminus \{uv_1\}$ (solid lines) and a possible tour obtained by taking shortcuts (dashed lines).

FIG. 5.1. *Example of trees $R$ and $R'$ and possible tours obtained by taking shortcuts. A trivial generalization of this example shows that visiting the first copy of each node in the walk might yield arbitrarily different tours.*

begin with the insight that the standard double-tree and shortcutting technique does not guarantee this.

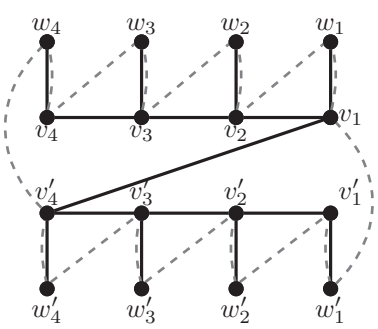
Given any tree $T$, we denote by $2 \cdot T$ the multigraph obtained by duplicating each edge in $T$. It is clear that the graph induced by $2 \cdot R$ is Eulerian (each node has even degree). Therefore, we can construct an Eulerian walk $W$ for it, that is, a sequence of nodes $W = x_1, x_2, \ldots, x_r$, where $x_1 = x_r$ and each edge in $2 \cdot R$ is traversed exactly once by the walk. Based on this walk, we construct a tour by skipping some nodes in the list such that each node is visited exactly once (except for the starting node $u$, which we visit twice). The standard technique for this is to visit a node when it appears in the list for the first time. We say that we *visit the first copy of a node in $W$*, which yields a tour $Q$. However, applying the same strategy to tree $R'$ might yield a tour $Q'$, where $|Q' \setminus Q|$ is unbounded. This might happen even if the walk $W'$ for $2 \cdot R'$ is constructed to be as similar as possible to $W$.

Consider the example shown in Figure 5.1. On the left-hand side we depict a tree $R$ (solid lines) and a walk $W$ (dashed lines). To distinguish different *copies of a node*, i.e., the different times that a node appears in the list of an Eulerian walk, we use bars; e.g., $v$, $\bar{v}$, and $\bar{\bar{v}}$ are copies of the same node. We remark, however, that we do not make distinctions on edges; e.g., we consider $vw$ to be equal to $v\bar{\bar{w}}$.

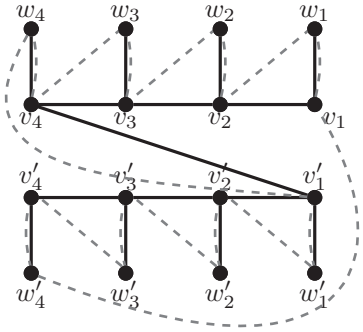
Consider one possible Eulerian walk for $2 \cdot R$, e.g.,

$$W = u, v_1, w_1, \overline{v_1}, v_2, w_2, \overline{v_2}, v_3, w_3, \overline{v_3}, v_4, w_4, \overline{v_4}, \overline{\overline{v_3}}, \overline{\overline{v_2}}, \overline{\overline{v_1}}, \overline{u}.$$

Visiting the first appearance of each node in $W$ yields the tour depicted with dashed lines in Figure 5.1(a). In Figure 5.1(b) we show a tree $R'$ (solid lines) that is of the form $(R \cup \{f\}) \setminus \{g\}$. Again, we can pick any Eulerian walk $W'$ for $2 \cdot R'$ and visit the first appearance of each node, obtaining a tour $Q'$. However, it is easy to observe that for any $W'$ this strategy yields a tour $Q'$ that does not contain any of the edges of the form $w_i v_{i+1} \in Q$. This means that $|Q' \setminus Q| \geq 3$. Generalizing this example by adding more pairs $(v_i, w_i)$, we obtain an instance for which $|Q' \setminus Q|$ is unbounded. It is worth noting that in this example, changing the starting vertex of the Eulerian tour from $u$ to $v_1$ would yield a tour $Q'$ that differs from $Q$ in only one edge. However, a simple extension of this instance, depicted in Figure 5.2, shows that changing the first node of the Eulerian tour, even choosing any Eulerian tour, is in general not enough.




(a) A tree $R$ (solid lines) and a possible tour obtained by taking shortcuts (dashed lines) to an Eulerian tour of the form $v'_4, v_1, w_1, v_1, v_2, \ldots$.




(b) A tree $R' = (R \cup \{v'_1 v_4\}) \setminus \{v'_4 v_1\}$ (solid lines) and a possible tour (dashed lines) obtained by taking shortcuts to an Eulerian tour of the form $v_1, w_1, v_1, v_2, \ldots$.

FIG. 5.2. *Example of two trees differing in one edge such that standard shortcutting of a (resp., any) walk in the double-tree yields tours differing in $\Omega(n)$ edges.*

*Observation* 5.2. There are examples of trees differing in one edge such that standard shortcutting of any walk in the double-tree yields tours that differ in arbitrarily many edges.

Our first step to address this problem is to choose a walk $W'$ for $2 \cdot R'$ that is "similar" to $W$. To explain the idea we keep working on the instance of Figure 5.1. Let us decompose $W$ as

$$W = u, v_1, \underbrace{w_1, \overline{v_1}, v_2, w_2, \overline{v_2}, v_3, w_3, \overline{v_3}, v_4, w_4}_{W_1}, \underbrace{\overline{v_4}, \overline{\overline{v_3}}, \overline{\overline{v_2}}, \overline{\overline{v_1}}}_{W_2}, \overline{u},$$

so that $W = u, v_1, W_1, W_2, \overline{u}$ (we use a comma to denote concatenation of walks). Based on this decomposition we define $W' := u, W_2, W_1, \overline{\overline{v_4}}, \overline{u}$, where $\overline{\overline{v_4}}$ is a new copy of $v_4$. Clearly $W'$ is an Eulerian walk for $2 \cdot R'$. The tour obtained by visiting the first appearance of each node in $W'$ is depicted with dashed lines in Figure 5.1(b). As argued before, visiting the first copy of a node in $W$ and $W'$ gives very different tours. We fix this problem by not visiting the first copy of each node in $W'$. Rather, we remember the copy of each node that we visit in $W$ to construct $Q$, and then visit the same copy when constructing $Q'$. In the following we write in boldface the copies of nodes that we visit when constructing tour $Q$:

$$W = \mathbf{u}, \mathbf{v_1}, \underbrace{\mathbf{w_1}, \overline{v_1}, \mathbf{v_2}, \mathbf{w_2}, \overline{v_2}, \mathbf{v_3}, \mathbf{w_3}, \overline{v_3}, \mathbf{v_4}, \mathbf{w_4}}_{W_1}, \underbrace{\overline{v_4}, \overline{\overline{v_3}}, \overline{\overline{v_2}}, \overline{\overline{v_1}}}_{W_2}, \overline{u}. \tag{5.1}$$

If, when traversing $W'$, we visit the same copies of nodes as in $W$, that is, we also choose to visit the nodes in boldface,

$$W' = \mathbf{u}, \underbrace{\overline{v_4}, \overline{\overline{v_3}}, \overline{\overline{v_2}}, \overline{\overline{v_1}}}_{W_2}, \underbrace{\mathbf{w_1}, \overline{\mathbf{v_1}}, \mathbf{v_2}, \mathbf{w_2}, \overline{v_2}, \mathbf{v_3}, \mathbf{w_3}, \overline{v_3}, \mathbf{v_4}, \mathbf{w_4}}_{W_1}, \overline{\overline{v_4}}, \overline{u}, \tag{5.2}$$

then we also obtain a Hamiltonian tour $Q'$. We remark that the copy of node $v_1$ in $W$ visited when constructing $Q$ does not appear in $W'$. Thus, we had chosen to visit

one of the remaining copies, namely $\overline{v_1}$. In this simple example, this strategy yields a tour $Q'$ equal to $Q$. We remark that this is not going to be true in general, but rather $Q$ and $Q'$ will differ in only a few edges. Additionally, since we take shortcuts to construct $Q'$ we have that $c(Q') \leq c(2 \cdot R') = 2 \cdot c(R')$.

**5.2. Robust TSP tours.** To show formally how to construct robust tours and prove Theorem 5.1, we first argue on consecutive trees $T_{t-1}$ and $T_t$ and then show the theorem for the entire sequence. We can assume that tree $T_t$ is obtained from $T_{t-1}$ by a series of local changes. Here, a local change means an operation that adds an edge and removes another if it is necessary. This observation can be formalized as follows.

LEMMA 5.3. *Consider two spanning trees $T_{t-1}$ and $T_t$ for graphs $G_{t-1}$ and $G_t$, respectively. Let us also denote $\ell := |T_t \setminus T_{t-1}|$. Then there exists a sequence of trees $R^1, R^2, \ldots, R^\ell$ satisfying the following:*

- *Tree $R^1$ equals $T_{t-1} \cup \{f_1\}$ for some $f_1 \in T_t \setminus T_{t-1}$ adjacent to $v_t$.*
- *For all $i \in \{2, \ldots, \ell\}$ there exist elements $f_i \in T_t \setminus T_{t-1}$ and $g_i \in T_{t-1} \setminus T_t$ such that*
$$R^i = (R^{i-1} \cup \{f_i\}) \setminus \{g_i\}.$$
- *Tree $R^\ell$ equals $T_t$.*

*Proof.* Assume that $T_t \setminus T_{t-1} = \{f_1, \ldots, f_\ell\}$, where $f_1$ is any edge adjacent to node $v_t$. We construct trees $R^1, \ldots, R^\ell$ with the following procedure: Set $R^1 = T_{t-1} \cup \{f_1\}$; for each $i \in \{2, \ldots, \ell\}$ let $g_i$ be any edge in $C(R^{i-1}, f_i) \cap (T_{t-1} \setminus T_t)$, where $C(R^{i-1}, f_i)$ is the unique cycle in set $R^{i-1} \cup \{f_i\}$; set $R^i := (R^{i-1} \cup \{f_i\}) \setminus \{g_i\}$. We remark that this procedure is well defined and that edge $g_i$ must exist for every $i$; otherwise tree $T_t$ would contain a cycle. Also, note that $R^\ell = T_t$, since in every iteration we add an edge in $T_t \setminus T_{t-1}$ and remove an edge in $T_{t-1} \setminus T_t$. ∎

With this lemma, it is enough to find an algorithm that is robust against a local change of the tree. More precisely, assume that we have constructed a tour $Q_{t-1}$ based on $T_{t-1}$. To construct tour $Q_t$ we consider trees $R^1, \ldots, R^\ell$ as in the previous lemma. Then we derive a procedure for updating tour $Q_{t-1}$ to a tour $Q^1$, where $Q^1$ is a tour constructed based on tree $R^1$ such that $|Q^1 \setminus Q_{t-1}| \leq 4$. After, for each $i \in \{2, \ldots, \ell\}$ we must update tour $Q^{i-1}$ to a tour $Q^i$ such that $|Q^i \setminus Q^{i-1}| \leq 4$. In each of these steps we construct tour $Q^i$ by an appropriate way of shortcutting tour $R^i$ so that $c(Q^i) \leq 2 \cdot c(R^i)$. By defining $Q_t := Q^\ell$ we will obtain that $|Q_t \setminus Q_{t-1}| \leq 4 \cdot \ell = 4 \cdot |T_t \setminus T_{t-1}|$, implying Theorem 5.1.

Based on this we consider two cases. The first corresponds to updating tour $Q_{t-1}$ to $Q^1$. This is a somewhat easier case since $R^1$ and $T_{t-1}$ differ by only one edge, namely edge $f_1$. The second case corresponds to updating tour $Q^{i-1}$ to tour $Q^i$, which is a more involved operation since $R^i$ is obtained from $R^{i-1}$ by swapping a pair of edges. We now focus on the second case. The first case then follows easily by a similar argument.

Given a graph $G = (V, E)$, let us consider two spanning trees $R$ and $R'$, where $R' = (R \cup \{f\}) \setminus \{g\}$ for some edges $f \notin R$ and $g \in R$. Assume that we have already computed a walk $W = x_1, x_2, \ldots, x_r$ corresponding to $2 \cdot R$. Consider the set $\{x_1, \ldots, x_r\}$, and recall that we consider all of the copies of nodes in $W$ as distinct, so that there are no repeated elements in this set. Additionally, assume that we have computed a function $I$ that assigns to each element in $\{x_1, \ldots, x_r\}$ a number in $\{0, 1\}$ such that for each node $v \in V$ exactly one copy $x_i$ of $v$ satisfies $I(x_i) = 1$. In the case that $I(x_i) = 1$, we say that $I$ *selects* $x_i$. This function indicates whether a copy of a node is visited by our Hamiltonian tour or not (with our visual representation in (5.1)

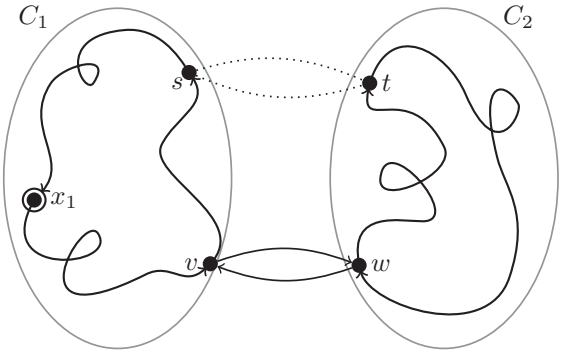



FIG. 5.3. *Sketch of the walk $W$ and its decomposition.*

and (5.2), we would write $x$ in boldface if and only if $I(x) = 1$ for any copy of a node $x$ appearing in $W$). The computed tour $Q$ is defined with the following algorithm.

**Algorithm Robust-Tour-Shortcut**

*Input*: A tree $R$, an Eulerian walk $W = x_1, \dots, x_r$ for $2 \cdot R$, and a function $I$ that assigns each element in the set $\{x_1, \dots, x_r\}$ a number in $\{0, 1\}$ such that exactly one copy $x_i$ of $v$ in $\{x_1, \dots, x_r\}$ satisfies that $I(x_i) = 1$.

1. Create a walk of the form $x_{\ell_1}, x_{\ell_2}, \dots, x_{\ell_{|V|}}$, where $\ell_i < \ell_j$ for all $i < j$ and $I(x_{\ell_i}) = 1$ for all $i$.
2. Return
$$Q := \{x_{\ell_1}x_{\ell_2}, x_{\ell_2}x_{\ell_3}, \dots, x_{\ell_{|V|-1}}x_{\ell_{|V|}}, x_{\ell_{|V|}}x_{\ell_1}\}.$$

We first observe that, independently of the chosen function $I$, the constructed tour has cost at most twice the cost of the original tree $R$. This follows from the same classic argument as shown by Christofides [7].

*Observation* 5.4. By using any valid function $I$ as input of Algorithm ROBUST-TOUR-SHORTCUT, the returned output $Q$ satisfies that $c(Q) \leq 2 \cdot c(R)$.

We now study how to update the Eulerian walk $W$ to obtain an appropriate walk $W'$ for the new tree $R'$. Recall that $R' = (R \cup \{f\}) \setminus \{g\}$ for some edges $f, g$, and assume that $f = st$ and $g = vw$ for some nodes $v, w, s, t$. Also, note that the set $R \setminus \{g\}$ induces two connected components, which we denote by $C_1$ and $C_2$. Assume without loss of generality that $x_1, v, s \in C_1$ and $t, w \in C_2$. Therefore, the walk $W$ starts in $C_1$, travels to $C_2$ by using a copy of edge $vw$, traverses all edges in $2 \cdot R_2$ restricted to $C_2$, and then returns to $C_1$ by using the second copy of $vw$ (see Figure 5.3). We conclude that $W$ is of the form

$$W = W_1, v, w, W_2, \overline{w}, \overline{v}, W_1',$$

where $W_1$ and $W_1'$ only touch nodes in $C_1$ and $W_2$ is a closed Eulerian walk for $2 \cdot R$ restricted to $C_2$. This implies that $W_2$ must visit $t$. Also, notice that either $W_1$ or $W_1'$ visits (or both visit) node $s$. We can assume without loss of generality that $W_1'$ visits $s$; otherwise we redefine $W$ as $x_r, x_{r-1}, \dots, x_1$ (notice that inverting the order of $W$ but maintaining function $I$ yields the same Hamiltonian tour). Therefore, we

can decompose $W$ as

$$W = W(x_1, v), v, W(w, t), W(t, w), \overline{w}, W(v, s), W(s, x_1),$$

where walk $W(x, y) = x_\ell, x_{\ell+1}, \ldots, x_u$ satisfies that $x_\ell$ is a copy of $x$ and $x_{u+1}$ is a copy of $y$. Based on this decomposition we construct an Eulerian tour for $2 \cdot R'$,

$$W' := W(x_1, v), W(v, s), \overline{s}, W(t, w), W(w, t), \overline{t}, W(s, x_1).$$

We remark that $\overline{s}$ and $\overline{t}$ are new copies of $s$ and $t$, respectively, which are different from any copy of these nodes appearing in $W$. For these copies we define $I(\overline{s}) = I(\overline{t}) = 0$. Also, notice that $W$ contains two elements (i.e., copies of nodes) that are not appearing in $W'$, namely $v$ and $\overline{w}$. Thus, if function $I$ originally selects one of these elements, then, for constructing $Q'$, function $I$ must be updated so that it selects another copy of $v$ (resp., $w$). In this case we set $I(x) = 1$, where $x$ is the first element appearing in $W(v, s)$ (which is a copy of $v$) and $W(w, t)$ (which is a copy of $w$), respectively.

Summarizing, we use the following algorithm to construct a walk $W'$ based on $W$ and $I$. This algorithm takes as an input the original function $I$ (that we assumed was used to construct $W$ by Algorithm Robust-Tour-Shortcut) and updates it accordingly.

**Algorithm Robust-Walk-Update**

*Input*: A tree $R$ of graph $G = (V, E)$; an Eulerian tour $W = x_1, \ldots, x_r$ (with $x_1 = x_r$) for graph $(V, 2 \cdot R)$, where $2 \cdot R$ is a set of edges obtained by duplicating every edge in $R$; a function $I$ that assigns to each element in the multiset $\{x_1, \ldots, x_r\}$ a number in $\{0, 1\}$ such that for each $v \in V$ exactly one copy $x_i$ of $v$ satisfies that $I(x_i) = 1$; a tree $R' = (R \cup \{f\}) \setminus \{h\}$ for some edges $f = st \notin R$ and $g = vw \in R$.

1. Decompose $W$ into walks such that

   $$W = W(x_1, v), v, W(w, t), W(t, w), \overline{w}, W(v, s), W(s, x_1).$$

   Each sequence of nodes $W(x, y) = x_\ell, x_{\ell+1}, \ldots, x_u$ satisfies that $x_\ell$ is a copy of $x$ and $x_{u+1}$ is a copy of $y$. If the decomposition is not possible, then redefine $W := x_r, x_{r-1}, \ldots, x_2, x_1$ and repeat the step.
2. Return a new walk $W'$ of the form

   $$W' := W(x_1, v), W(v, s), \overline{s}, W(t, w), W(w, t), \overline{t}, W(s, x_1),$$

   where $\overline{t}$ and $\overline{s}$ are new copies of nodes $t$ and $s$, respectively.
3. Set $I(\overline{t}) = I(\overline{s}) = 0$. If $I(v) = 1$, then set $I(x) = 1$, where $x$ is the first node visited by $W(v, s)$ (and thus is a copy of $v$). Similarly, if $I(\overline{w}) = 1$, then set $I(x) = 1$, where $x$ is the first node visited by $W(w, t)$ (and thus is a copy of $w$).

Now that we have constructed a walk $W'$, we derive $Q'$ by taking shortcuts in the same way as when constructing $Q$. That is, we define $Q'$ as the output of Algorithm Robust-Tour-Shortcut on input $W'$ and $I$ (where $I$ has been updated by Algorithm Robust-Walk-Update when constructing $W'$).

We remark that our construction ensures that when constructing $Q$ and $Q'$, the same copies of nodes are taken inside each walk $W(x_1, v), W(v, s), W(t, w), W(w, t)$, and $W(s, x_r)$ (except maybe for the first node in $W(v, s)$ and $W(w, t)$). Therefore,

the edges in $Q$ and $Q'$ picked when traversing each of these five walks are (mostly) the same. This observation is the key to show the following main lemma, which summarizes our previous discussion.

LEMMA 5.5. *Consider two spanning trees $R$ and $R'$, where $R' = (R \cup \{f\}) \setminus \{g\}$ for some edges $f \notin R$ and $g \in R$. Assume that $Q$ is the output of Algorithm* ROBUST-TOUR-SHORTCUT *on input $R$, $W$, and $I$, where $W$ is a walk for $2 \cdot R$ and $I$ is chosen arbitrarily. Let $W'$ be the output of Algorithm* ROBUST-WALK-UPDATE. *Then, if $Q'$ is the output of Algorithm* ROBUST-TOUR-SHORTCUT *on input $R'$, $W'$, and $I$ (where $I$ was updated by Algorithm* ROBUST-WALK-UPDATE*), then*

$$c(Q) \leq 2 \cdot c(R),\ c(Q') \leq 2 \cdot c(R'),\ \text{and } |Q' \setminus Q| \leq 4.$$

*Proof.* We just need to show that $|Q' \setminus Q| \leq 4$. Consider any walk

$$X \in \{W(x_1, v), W(v, s), W(t, w), W(w, t), W(s, x_1)\},$$

where $X = x_h, x_{h+1}, \ldots, x_u$ for some positive integers $h, u$. Consider the copies of nodes visited by $X$ that are selected by function $I$, that is, the set $\{x_{\ell_1}, \ldots, x_{\ell_q}\}$ such that

$$h \leq \ell_1 < \ell_2 < \cdots < \ell_q \leq u,$$

and given $i \in \{h, \ldots, u\}$ we have that $I(x_i) = 1$ if and only if $i = \ell_j$ for some $j$. Notice that by construction of $Q$ and $Q'$, for each $i \in \{h+1, \ldots, u\}$ each edge of the form $x_{\ell_i} x_{\ell_{i+1}}$ belongs to $Q$ and $Q'$. If $X$ equals $W(v, s)$ or $W(w, t)$, this might not happen for $i = h$ if the value of $I(x)$, for $x$ being the first element in $X$, was modified to 1 in Algorithm ROBUST-WALK-UPDATE (step 3). Therefore, we consider these cases separately. With this observation we simply visit the vertices in the order given by $W'$ and see which edges belong to $Q' \setminus Q$. A simple case distinction shows that $|Q' \setminus Q| \leq 4$. For the sake of completeness we give the details.

We have four different cases, corresponding to the moments in which walk $W'$ traverses from $W(x_1, v)$ to $W(v, s)$, from $W(v, s)$ to $W(t, w)$, from $W(t, w)$ to $W(w, t)$, and from $W(w, t)$ to $W(s, x_1)$.

1. The first case is further divided into two subcases. Let $x$ be the first element in $W(v, s)$ (that is, a copy of $v$). For the first subcase, assume that $I(x)$ is left unchanged by Algorithm ROBUST-WALK-UPDATE. Then there can be an edge in $Q' \setminus Q$ connecting the last node in $W(x_1, v)$ selected by $I$ and the first node in $W(v, s)$ selected by $I$. For the second subcase we assume that the algorithm updates $I$ by setting $I(x) = 1$. Recall that $W$ is of the form

   $$W = W(x_1, v), v, W(w, t), W(t, w), \overline{w}, W(v, s), W(s, x_1)$$

   and that $I(x)$ is updated if and only if $v$ (the element after $W(x_1, v)$ in $W$) was selected by $I$. Since

   $$W' := W(x_1, v), W(v, s), \overline{s}, W(t, w), W(w, t), \overline{t}, W(s, x_1),$$

   and the algorithm sets $I(x) = 1$, $Q$ and $Q'$ contain the edge that connects the last element in $W(x_1, v)$ selected by $I$ and $v$. Thus, in this subcase we must only account for an edge in $Q' \setminus Q$ that connects the first node in $W(v, s)$ selected by $I$ (which is a copy of $v$) and the second node in $W(v, s)$ selected by $I$. Since these subcases cannot happen simultaneously they account for at most one edge in $Q' \setminus Q$.

2. By the discussion before, the second occurrence (if any) of an edge in $Q' \setminus Q$ can only be the edge that connects the last node in $W(v,s)$ selected by $I$ and the first node in $W(t,w)$ selected by $I$ (note that $\overline{s}$ is by definition not selected by $I$).
3. Analogously to the first case, we also distinguish two subcases depending on the value of $I(x)$ for $x$ being the first element in $W(w,t)$. By the same argument, the two subcases together account for one edge in $Q' \setminus Q$.
4. Finally, the last possible occurrence of an edge in $Q' \setminus Q$ corresponds to the edge that connects the last node in $W(w,t)$ selected by $I$ and the first node in $W(s,x_1)$ selected by $I$ (again, $\overline{t}$ is not selected by $I$).

We conclude that $|Q' \setminus Q| \leq 4$. ∎

To show Theorem 5.1, we can iterate the algorithmic ideas just presented for each pair of trees $R^i$ and $R^{i+1}$ obtained in Lemma 5.3. As mentioned before, we still need to determine how to construct a tour based on $R^1 = T_{t-1} \cup \{f_1\}$ given a tour $Q_{t-1}$ for tree $T_{t-1}$. We briefly explain how to deal with this case. Assume that we have a tour $Q_{t-1}$ constructed as the output of Algorithm Robust-Tour-Shortcut on input $T_{t-1}$, $W_{t-1}$, and $I$. We use a similar (but simpler) approach as before. Recall that $f_1 = vv_t$, where $v \in V_{t-1}$, and thus $v_t$ is a leaf of $R^1$. Thus, if $W$ is of the form $W = \overline{W}, v, \widehat{W}$, we can define an Eulerian walk $W^1 := \overline{W}, v, v_t, \overline{v}, \widehat{W}$ for $2 \cdot R^1$, where $\overline{v}$ is a new copy of $v$. Function $I$ is updated so that $I(\overline{v}) = 0$ and $I(v_t) = 1$. With this construction it is easy to observe that applying Algorithm Robust-Tour-Shortcut on input $R^1$ and the updated function $I$ yields a tour $Q^1$ such that $|Q^1 \setminus Q_{t-1}| \leq 2$.

We have presented all arguments for proving Theorem 5.1. For the sake of completeness we present in detail the algorithm claimed in this theorem. In the algorithm below we use lower indices to indicate iterations in our input sequence (e.g., for trees $T_1, \ldots, T_t$) and upper indices to indicate iterations corresponding to the sequence of trees $R^1, \ldots, R^\ell$ as given by Lemma 5.3.

**Algorithm Robust-Tour-Sequence**

*Input*: A sequence of complete metric graphs $G_0, \ldots, G_t, \ldots$, where $G_t = (V_t, E_t)$ and $V_t = \{v_0, \ldots, v_t\}$. A spanning tree $T_t$ of graph $G_t$ for each $t \geq 0$.

1. Set $W_1 := v_0, v_1, \overline{v_0}$; $I(v_0) := 1$, $I(v_1) := 1$, and $I(\overline{v_0}) := 0$; and $T_1 := \{v_0v_1\}$.
2. For each $t \geq 2$ do the following:
   (a) Use the procedure from Lemma 5.3 to create a sequence of spanning trees $R^1, \ldots, R^\ell$ for graph $G_t$ satisfying the following properties:
   - Tree $R^1$ equals $T_{t-1} \cup \{f_1\}$ for some $f_1 = vv_t \in T_t \setminus T_{t-1}$ with $v \in V_{t-1}$.
   - Tree $R^\ell$ equals $T_t$.
   - For all $i \in \{2, \ldots, \ell\}$ there exist elements $f_i \in T_t \setminus T_{t-1}$ and $g_i \in T_{t-1} \setminus T_t$ such that $R^i = (R^{i-1} \cup \{f_i\}) \setminus \{g_i\}$.

   (b) Define an Eulerian walk $W^1$ for $2 \cdot R^1$ as follows.
   - Assume that $f_1 = vv_t$ for some $v \in V_{t-1}$. Then, there exists two walks $\overline{W}$ and $\widehat{W}$ such that $W_{t-1} = \overline{W}, v, \widehat{W}$.
   - Set $W^1 := \overline{W}, v, v_t, \overline{v}, \widehat{W}$, where $\overline{v}$ is a new copy of $v$.

   (c) Set $I(v_t) := 1$ and $I(\overline{v}) := 0$.
   (d) Define tour $Q^1$ as the output of Algorithm Robust-Tour-Shortcut on input $W^1$ and $I$.
   (e) For all $i \in \{2, \ldots, \ell\}$ do:
      i. Define $W^i$ as the output of Algorithm Robust-Walk-Update on input $R = R^{i-1}$, $W = W^{i-1}$, $I$, and $R' = R^i$ (*note that here $I$ gets updated by the algorithm*).
      ii. Define $Q^i$ as the output of Algorithm Robust-Tour-Shortcut on input $W^i$ and $I$.
   (f) Set $W_t := W^\ell$ and $Q_t := Q^\ell$.

We now show that this algorithm fulfills the claim in Theorem 5.1.

*Proof of Theorem* 5.1. Fix an iteration $t \geq 2$ of Algorithm Robust-Tour-Sequence. Notice that $Q_t = Q^\ell$ is the output of Algorithm Robust-Tour-Shortcut on input $W^\ell$, where $W^\ell$ is an Eulerian tour of $2 \cdot R^\ell = 2 \cdot T_t$. Thus, by Observation 5.4, we have that $c(Q_t) \leq 2 \cdot c(T_t)$.

Recall that $|T_{t-1} \setminus T_t| = \ell$. To bound $|Q_t \setminus Q_{t-1}|$, first notice that $|Q^i \setminus Q^{i-1}| \leq 4$ for all $i \in \{2, \ldots, \ell\}$. This follows from Lemma 5.5. Also, we have that $|Q^1 \setminus Q_{t-1}| \leq 2$. To see this, consider walk $W_{t-1} = \overline{W}, v, \widehat{W}$. Notice that walks $W^1$ and $W_{t-1}$ visit nodes in $\overline{W}$ and $\widehat{W}$ in the same order. Also, the tours $Q_{t-1}$ for $W_{t-1}$ and $Q^1$ for $W^1$ visit the same copies of nodes inside $\overline{W}$ and $\widehat{W}$. This implies that $|Q^1 \setminus Q_{t-1}| = 2$. With this we conclude that

$$|Q_t \setminus Q_{t-1}| = |Q^1 \setminus Q_{t-1}| + \sum_{i=2}^{\ell} |Q^i \setminus Q^{i-1}| \leq 4 \cdot \ell = 4 \cdot |T_t \setminus T_{t-1}|. \qquad \square$$

The theorem just shown, together with Theorem 3.2, directly implies the following result.

Theorem 5.6. *The online TSP on metric graphs admits a* $(2 + \varepsilon)$*-competitive algorithm with amortized budget* $O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$ *for any* $\varepsilon > 0$.

**6. Conclusion.** For the standard online minimum spanning tree (MST) problem without recourse the best achievable competitive ratio is $\Theta(\log n)$. It has been a longstanding open question whether one can obtain constant-competitive solutions by allowing a constant budget of recourse actions [4, 11]. In this paper, we affirmatively answered this question by showing that given a sufficiently large but still constant budget, one can reduce the competitive ratio from $\Theta(\log n)$ to $1 + \varepsilon$ in the amortized setting.

Since the original appearance of this work, many of the original questions by Imase and Waxman [11] have been answered: It is now known that a nonamortized budget of 2 (and even less) suffices to maintain a constant-competitive algorithm [9], and the same result holds for the fully dynamic case (with node deletions) if amortization is allowed [10]. Despite this great advancement many interesting questions remain open. The question most directly related to this work asks whether a simple primal algorithm with a small budget, such as Algorithm Greedy in section 4, has a constant-competitive guarantee. For the fully dynamic case, it is still open whether there exists a constant-competitive algorithm with a constant budget. Finally, it is natural to consider the online Steiner forest problem in a setting with recourse. As in the Steiner tree problem, an online algorithm without recourse can achieve $O(\log n)$-competitiveness [5], and it is possible that a small budget increase may help to reduce the competitive guarantee to a constant.

## REFERENCES

[1] N. Alon and Y. Azar, *On-line Steiner trees in the Euclidean plane*, Discrete Comput. Geom., 10 (1993), pp. 113–121.

[2] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook, *The Traveling Salesman Problem: A Computational Study*, Princeton University Press, Princeton, NJ, 2007.

[3] G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, and M. Talamo, *Algorithms for the on-line travelling salesman*, Algorithmica, 29 (2001), pp. 560–581.

[4] V. Bafna, B. Kalyanasundaram, and K. Pruhs, *Not all insertion methods yield constant approximate tours in the Euclidean plane*, Theoret. Comput. Sci., 125 (1994), pp. 345–360.

[5] P. Berman and C. Coulston, *On-line algorithms for Steiner tree problems*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC 1997), ACM, New York, 1997, pp. 344–353.

[6] J. R. Birge and F. Louveaux, *Introduction to Stochastic Programming*, Springer Ser. Oper. Res., Springer, New York, 1997.

[7] N. Christofides, *Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem*, Tech. report 388, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA, 1976.

[8] M. Dynia, M. Korzeniowski, and J. Kutylowski, *Competitive maintenance of minimum spanning trees in dynamic graphs*, in Proceedings of the 33rd Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2007), Lecture Notes in Comput. Sci. 4362, J. van Leeuwen, G. F. Italiano, W. van der Hoek, C. Meinel, H. Sack, and F. Plasil, eds., Springer, Berlin, 2007, pp. 260–271.

[9] A. Gu, A. Gupta, and A. Kumar, *The power of deferral: Maintaining a constant-competitive Steiner tree online*, in Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC 2013), ACM, New York, 2013, pp. 525–534.

[10] A. Gupta and A. Kumar, *Online Steiner tree with deletions*, in Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2014), SIAM, Philadelphia, ACM, New York, 2014, pp. 455–467. doi:10.1137/1.9781611973402.34.

[11] M. Imase and B. M. Waxman, *Dynamic Steiner tree problem*, SIAM J. Discrete Math., 4 (1991), pp. 369–384. doi:10.1137/0404033.

[12] B. Kalyanasundaram and K. Pruhs, *Constructing competitive tours from local information*, Theoret. Comput. Sci., 130 (1994), pp. 125–138.

[13] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, *The Traveling Salesman Problem*, John Wiley and Sons, New York, 1985.

[14] N. Megow, K. Mehlhorn, and P. Schweitzer, *Online graph exploration: New results on old and new algorithms*, Theoret. Comput. Sci., 463 (2012), pp. 62–72.

[15] N. Megow, M. Skutella, J. Verschae, and A. Wiese, *The power of recourse for online MST and TSP*, in Proceedings of the 39th International Colloquium on Automata, Languages, and Programming (ICALP 2012), Lecture Notes in Comput. Sci. 7391, A. Czumaj, K. Mehlhorn, A. M. Pitts, and R. Wattenhofer, eds., Springer, Heidelberg, 2012, pp. 689–700.

[16] C. A. S. Oliveira and P. M. Pardalos, *A survey of combinatorial optimization problems in multicast routing*, Comput. Oper. Res., 32 (2005), pp. 1953–1981.

[17] J.-J. Pansiot and D. Grad, *On routes and multicast trees in the Internet*, ACM SIGCOMM Comput. Comm. Rev., 28 (1998), pp. 41–50.

[18] A. Schrijver, *Combinatorial Optimization. Polyhedra and Efficiency,* Vol. B, Algorithms Combin. 24, Springer-Verlag, Berlin, 2003.

[19] N. Subramanian and S. Liu, *Centralized multi-point routing in wide area networks*, in Proceedings of the Symposium on Applied Computing (SAC 1991), IEEE Press, Piscataway, NJ, 1991, pp. 46–52.

[20] B. M. Waxman, *Routing of multipoint connections*, IEEE J. Sel. Area Comm., 6 (1988), pp. 1617–1622.