Tight Bounds for Online Weighted Tree Augmentation

Joseph (Seffi) Naor
Technion, Haifa, Israel
naor@cs.technion.ac.il

Seeun William Umboh
The University of Sydney, Australia
william.umboh@sydney.edu.au

David P. Williamson
Cornell University, Ithaca, NY, USA
http://www.davidpwilliamson.net/work
davidpwilliamson@cornell.edu

Abstract

The Weighted Tree Augmentation problem (WTAP) is a fundamental problem in network design. In this paper, we consider this problem in the online setting. We are given an n-vertex spanning tree T and an additional set L of edges (called links) with costs. Then, terminal pairs arrive one-by-one and our task is to maintain a low-cost subset of links F such that every terminal pair that has arrived so far is 2-edge-connected in $T \cup F$. This online problem was first studied by Gupta, Krishnaswamy and Ravi (SICOMP 2012) who used it as a subroutine for the online survivable network design problem. They gave a deterministic $O(\log^2 n)$-competitive algorithm and showed an $\Omega(\log n)$ lower bound on the competitive ratio of randomized algorithms. The case when T is a path is also interesting: it is exactly the online interval set cover problem, which also captures as a special case the parking permit problem studied by Meyerson (FOCS 2005). The contribution of this paper is to give tight results for online weighted tree and path augmentation problems. The main result of this work is a deterministic $O(\log n)$-competitive algorithm for online WTAP, which is tight up to constant factors.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Online algorithms, competitive analysis, tree augmentation, network design

Digital Object Identifier 10.4230/LIPIcs.ICALP.2019.88

Category Track A: Algorithms, Complexity and Games

Related Version A full version of this paper is available at http://arxiv.org/abs/1904.11777.

Funding *Joseph (Seffi) Naor*: Supported in part by ISF grant 1585/15 and BSF grant 2014414.
Seeun William Umboh: Supported in part by NWO grant 639.022.211 and ISF grant 1817/17. Part of this work was done while a postdoc at Eindhoven University of Technology, and while visiting the Hebrew University of Jerusalem and the Technion.

Acknowledgements This work was done in part while the authors were visiting the Simons Institute for the Theory of Computing.

1 Introduction

In the *weighted tree augmentation problem* (WTAP), we are given an n-vertex spanning tree $T = (V, E)$ together with an additional set of edges L called *links*, where $L \subseteq \binom{V}{2}$. Each link $\ell \in L$ has a cost $c(\ell) \geq 0$. Terminal pairs (s_i, t_i), $i = \{1, \ldots, k\}$, are given and the goal is to compute a minimum cost subset of links $F \subseteq L$ such that each terminal pair is (edge) 2-connected in $T \cup F$. In the unweighted version, the links have unit costs and the problem

46th International Colloquium on Automata, Languages, and Programming (ICALP 2019).
Editors: Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi;
Article No. 88; pp. 88:1–88:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

is known as the *tree augmentation problem* (TAP). If the spanning tree T is a path, then the unweighted problem is called the *path augmentation problem* (PAP), while the weighted version is called *weighted path augmentation* (WPAP).

TAP and WTAP are considered to be fundamental connectivity augmentation problems, and have been studied extensively. TAP is already known to be APX-hard and the best approximation algorithms for WTAP and TAP achieve approximation factors of 2 and 1.458 respectively [6, 7]. Improving these bounds is an important open problem.

We consider these problems in the online setting. In online WTAP, we are initially given a spanning tree $T = (V, E)$, and the set of links L together with their costs. The terminal pairs (s_i, t_i) arrive online one by one. Our goal is to maintain a low-cost subset of links $F \subseteq L$ such that each terminal pair seen so far is (edge) 2-connected in $T \cup F$.

Online WTAP occurs as a subproblem in the online survivable network design algorithm of Gupta, Krishnaswamy and Ravi [8]. They observed that the online tree augmentation problem can be cast as an instance of the online set cover problem[1] in which the elements are the fundamental cuts defined by the terminal pairs and the sets are the links. Since there are only n elements and $O(n^2)$ sets, applying the results of Alon et al. [1] yields a fractional $O(\log n)$-competitive algorithm. But, then, how does one round the fractional solution online? Randomized rounding seems to be the only rounding technique we have for this problem, and it yields a randomized $O(\log^2 n)$-competitive algorithm, as observed by [1]. This competitive factor can even be achieved deterministically at no further cost [1]. We note that the loss of a logarithmic factor in the rounding step seems inherent. Interestingly, Gupta, Krishnaswamy and Ravi [8] also showed for the rooted setting ($s_i = r$ for some root r) a lower bound of $\Omega(\log n)$ against randomized algorithms. It is easy to observe that this lower bound also holds against fractional online algorithms.

There has been a long line of work on maintaining connectivity online, starting in the seminal paper of Imase and Waxman [11]. A $\Theta(\log n)$-competitive algorithm is given there for the online Steiner problem in undirected graphs. In this problem the graph with a fixed root vertex is known in advance and the terminals are given one by one, and one must ensure that all terminals that have arrived so far are connected to the root. Other polylogarithmic (in n) competitive algorithms have been given for more complex models of connectivity, including those with node costs rather than edge costs and penalties for violating connectivity constraints; see [2, 3, 13, 9, 10, 16, 14]. Gupta, Krishnaswamy, and Ravi [8] consider the online survivable network design problem, which generalizes WTAP. In this problem, a graph is fixed in advance and terminal pairs (s_i, t_i) arrive with connectivity requirements r_i; one must ensure that there are at least r_i edge-disjoint paths between s_i and t_i for all pairs that have arrived thus far. They give a randomized $\tilde{O}(r_{\max} \log^3 n)$-competitive algorithm for the problem, where $r_{\max} = \max_i r_i$. Note that this problem with uniform requirements $r_i = 2$ already generalizes WTAP.

The online WPAP, when T is a path, is an interesting problem in its own right. This problem is equivalent to online interval set cover. It captures as a special case the parking permit problem introduced by Meyerson [12]. In this problem, there is a sequence of days; each day it is either sunny or it rains, and if it rains we must purchase a parking permit. Permits have various durations and costs. We can model the parking permit problem by online path augmentation by letting the edges of the path correspond to the sequence of days, the links to the permits, and the rainy days to a terminal pair request for the

[1] In the online set cover problem, elements arrive online and need to be covered upon arrival by sets from a set system known in advance. (Note that not necessarily all elements will appear.)

corresponding day. Meyerson [12] gives a deterministic $O(\log n)$-competitive algorithm for the problem and a randomized $O(\log \log n)$-competitive algorithm, and shows lower bounds on the competitive ratio of $\Omega(\log n / \log \log n)$ for deterministic algorithms and $\Omega(\log \log n)$ for randomized algorithms. Note that online WPAP is a strict generalization of the parking permit problem because the parking permit problem assumes that permits of the same duration have the same cost, whereas no such assumption is made of the links in WPAP.

1.1 Our Results

The contribution of this paper is to give tight results (within constant factors) for online tree and path augmentation problems. Our main result is that weighted online tree augmentation has a competitive ratio of $\Theta(\log n)$.

▶ **Theorem 1.** *There is a deterministic algorithm for online WTAP with competitive ratio $O(\log n)$.*

This result is tight up to constant factors because of the $\Omega(\log n)$ lower bound on randomized algorithms for WTAP given by [8]. As we mention above, [8] gives a randomized $\tilde{O}(r_{\max} \log^3 n)$-competitive algorithm for the online survivable network design problem. An intriguing open question is whether this competitive ratio can be improved, say to $O(r_{\max} \log n)$ or even $O(\log n)$. In fact, we are unaware of lower bounds that rule out the latter bound. We view our main result as a necessary stepping stone towards obtaining an $O(r_{\max} \log n)$ or $O(\log n)$ bound. Indeed, for $r_{\max} = 2$, plugging in our algorithm for online WTAP into the algorithm of [8] improves their competitive ratio from $\tilde{O}(\log^3 n)$ to $\tilde{O}(\log^2 n)$.

▶ **Corollary 2.** *For online survivable network design with $r_{\max} = 2$, there is a randomized algorithm with competitive ratio $\tilde{O}(\log^2 n)$.*

Our second result shows that the competitive ratio for deterministic algorithms for online path augmentation is also $\Theta(\log n)$. Meyerson [12] gives a lower bound of $\Omega(\log n / \log \log n)$ for deterministic algorithms for the parking permit problem, and hence for online path augmentation. We improve the analysis of his lower bound instance to show the following.

▶ **Theorem 3.** *Every deterministic algorithm for online WPAP has competitive ratio $\Omega(\log n)$.*

Since we use a parking permit instance to show the lower bound, we have the same lower bound for the parking permit problem.

Finally, we show that the fractional version of online path augmentation has competitive ratio $\Theta(\log \log n)$ for deterministic algorithms. Meyerson [12] gives a lower bound of $\Omega(\log \log n)$ for randomized algorithms for the parking permit problem, and hence for online fractional path augmentation. Our algorithm implies an exponential gap between the competitive ratios of fractional path augmentation and fractional tree augmentation. We show the following.

▶ **Theorem 4.** *There is a deterministic algorithm for online fractional WPAP with competitive ratio $O(\log \log n)$.*

Recall that online WPAP is equivalent to online interval set cover. Thus, Theorems 1 and 4 imply that restricting online set cover to interval sets allows for improved competitive ratios. Also, even though interval set cover and interval hitting set are equivalent in the offline case, the latter turns out to be exponentially more difficult than the former in the online case; in contrast to Theorem 4, Even and Smorodinsky [5] gave a lower bound of $\Omega(\log n)$ for online fractional hitting set.

1.2 Our Techniques

We now outline some of the ideas behind our algorithms.

Online WTAP

As mentioned before, there is an online *fractional* $O(\log n)$-competitive algorithm for WTAP that follows from the work of [1] on the online set cover problem. However, it is unclear how to exploit the special structure of the set system in hand in WTAP (as defined by the links) to avoid the loss of another factor of $O(\log n)$ when rounding the fractional solution into an integral one (either randomized or deterministic). Thus, our approach to proving Theorem 1 takes a completely different route. There are two key ingredients in our proof:

1. **Low-width path decomposition.** The first ingredient is a path decomposition of low "width": in particular, there is a decomposition of the tree into edge-disjoint paths such that any path in the tree intersects at most $O(\log n)$ paths of the decomposition. Such a decomposition can be obtained using the heavy-path decomposition of Sleator and Tarjan [15]. This immediately implies an $O(\log n)$-approximate black-box reduction from online tree augmentation to online path augmentation. Unfortunately, Theorem 3 gives a lower bound of $\Omega(\log n)$ for the latter problem. Since a tree may have width $\Omega(\log n)$ in the worst case (e.g., a binary tree), the best we can achieve for WTAP using a black-box reduction is a competitive ratio of $O(\log^2 n)$.
2. **Refined guarantee for path augmentation.** The second ingredient is our main technical contribution. We define a notion of *projection* for links onto paths in the path decomposition, and call the projected link *rooted* if it has, as its endpoint, the node of the path closest to the root of the tree. The key insight is that the path decomposition has a special structure: for each link, its projection is rooted for all but at most one of the paths in the decomposition. We then give a version of the path algorithm that treats rooted links differently from non-rooted links; in particular, an online path augmentation algorithm that finds a solution whose cost is within a constant factor of the rooted links of the optimal solution plus an $O(\log n)$ factor of the cost of the non-rooted links. Intuitively, then, summing the cost over all the paths in the decomposition, each link appears as a rooted link in at most $O(\log n)$ paths in the decomposition and as a non-rooted link in at most one path in the decomposition, yielding the $O(\log n)$ factor overall.

Online Fractional WPAP

Directly applying the online fractional set cover algorithm of [1] to online fractional WPAP only yields a competitive ratio of $O(\log n)$. However, for online set cover instances in which each element is covered by at most d sets, the algorithm of [1] is $O(\log d)$-competitive. Thus, to get a competitive ratio of $O(\log \log n)$, the basic idea is to reduce to a restricted instance in which each request can only be covered by $O(\log n)$ links. For such restricted instances, applying the algorithm of [1] gives a competitive ratio of $O(\log \log n)$.

1.3 Other Related Work

Recently, Dehghani et al. [4] studied online survivable network design, giving a bicriteria approximation algorithm, and considering several stochastic settings.

1.4 Organization of the Paper

We start with the preliminaries and describe the low-width path decomposition in Section 2. In Section 3, we present the refined guarantee needed for online path augmentation. Then, we show how to achieve the required refined guarantee in Section 4. Due to lack of space, we defer the proofs of Theorems 3 and 4 to the full version.

2 Preliminaries

We restate the formal definition of the problem. In the online weighted tree augmentation problem, we are initially given a spanning tree $T = (V, E)$, and an additional set of edges called *links* $L \subseteq \binom{V}{2}$ with costs $c(\ell) \geq 0$. Then, terminal pairs (s_i, t_i) arrive one by one. Our goal is to maintain a low-cost subset of links $F \subseteq L$ such that each terminal pair seen so far is 2-connected in $T \cup F$.

Notation

Denote by $P(u, v)$ the path between u and v in T. For a link $\ell = (u, v)$, we write $P(\ell) = P(u, v)$ and for a set S of links, we write $P(S) = \bigcup_{\ell \in S} P(\ell)$. We say that a link $\ell \in L$ *covers* an edge $e \in E$ if $e \in P(\ell)$. Define $\text{cov}(e) = \{\ell \in L : e \in P(\ell)\}$ to be the set of links covering e. Note that $\text{cov}(e)$ is exactly the set of links crossing the cut induced by the tree edge e. Let $R \subseteq E$ be a set of requests. Then, a solution F is feasible if and only if for every edge $e \in R$, we have $F \cap \text{cov}(e) \neq \emptyset$; or equivalently, if $P(F) \supseteq R$.

Simplifying assumptions

In the rest of this paper, we assume that link costs are powers of 2; this assumption is without loss of generality since we can round up all edge costs and lose only a factor of 2 in the competitive ratio. Given that link costs are powers of 2, we say that the *class* of a link ℓ is j if $c(\ell) = 2^j$ and we write $\text{class}(\ell) = j$.

Given an instance in which link costs are powers of 2, we also assume that requests are *elementary*: each request (s_i, t_i) is a tree edge $e \in E$. This is without loss of generality because an adversary can simulate a non-elementary request (s_i, t_i) by a sequence of requests, where each request is an edge along the path between s_i and t_i in T.

Path decomposition

We next define a rooted path decomposition, see Figure 1 for an example.

▶ **Definition 5** (Rooted Path Decomposition). *Let T be a tree. A* path decomposition *of T is a partition $\mathcal{P}$ of its edge set into edge-disjoint paths. We say $\mathcal{P}$ is* rooted *if there is a vertex $r \in T$ such that if we root T at r, then for each path $p \in \mathcal{P}$, the least common ancestor of the vertices on p is an endpoint of the path (we call this endpoint the* root *of p). The* width *of $\mathcal{P}$ is $\text{width}(\mathcal{P}) = \max_{u,v \in V(T)} |\{p \in \mathcal{P} : P(u, v) \cap p \neq \emptyset\}|$, the maximum number of paths $p \in \mathcal{P}$ that any path in T intersects.*

▶ **Lemma 6** (Existence of Low Width Rooted Path Decompositions). *Every tree on n vertices admits a rooted path decomposition of width $O(\log n)$.*

An $O(\log n)$-width rooted path decomposition can be obtained using the so-called *heavy path decomposition* of Sleator and Tarjan [15]. For the sake of completeness, we give a proof here. The following notion of a caterpillar decomposition will be convenient.

Figure 1 Example of a graph and its rooted path decomposition. The edge colors reflect the partition of the edges. The root of each path is the highest vertex of the path.

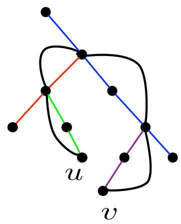

Figure 2 Illustration of the projections of the link (u, v) onto the paths of the decompositon. Only the projection onto the blue path is non-rooted.

▶ **Definition 7** (Caterpillar Decomposition). *Let T be a rooted tree on n vertices. A* caterpillar decomposition *of T is a vertex-disjoint decomposition of T into a root-to-leaf path B (called the* backbone*) and subtrees T_i that are connected to B. The decomposition is* balanced *if for each subtree T_i, we have $|V(T_i)| \leq n/2$.*

▶ **Lemma 8.** *Every tree admits a balanced caterpillar decomposition.*

Proof. The existence of a balanced caterpillar decomposition is an easy consequence of the fact that every tree T has a balanced vertex separator v, i.e. after removing v from T, each of the remaining connected components has at most $n/2$ vertices. The following is a balanced caterpillar decomposition of T: pick an arbitrary root-to-leaf path containing v to be the backbone B, and the subtrees T_i to be the connected components of T after removing B. ◀

Proof of Lemma 6. The lemma easily follows by choosing an arbitrary root vertex of T and recursively applying Lemma 8. ◀

3 Refined Guarantee for Online Path Augmentation

As already mentioned, Lemma 6 implies an $O(\log n)$-approximate black-box reduction to online path augmentation: given an α-competitive algorithm for online path augmentation, we have an $O(\alpha \log n)$-competitive algorithm for online tree augmentation. However, Theorem 3 says that $\alpha = \Omega(\log n)$ for deterministic algorithms. To get around this lower bound, a more refined guarantee for online path augmentation is needed.

We need some notation to describe this refined guarantee. Suppose $\mathcal{P}$ is a rooted path decomposition of T and ℓ a link. For $Q \in \mathcal{P}$, let $\pi_Q(\ell)$ be the link whose endpoints are endpoints of the path $P(\ell) \cap Q$; we call $\pi_Q(\ell)$ the *projection* of ℓ onto Q. We say that ℓ is Q-*rooted* if one of the endpoints of $\pi_Q(\ell)$ is the root of Q, and Q-*non-rooted* otherwise. (See Fig. 2 for an illustration.) The main ingredient for the refined guarantee is the next lemma.

▶ **Lemma 9.** *Consider a tree T and link $\ell = (u, v)$. Suppose $\mathcal{P}$ is a rooted path decomposition of T. Then, there is at most one path $Q \in \mathcal{P}$ for which ℓ is Q-non-rooted.*

Proof. We claim that for any path $Q \in \mathcal{P}$ such that ℓ is a non-rooted link, the least common ancestor a of u and v must lie in Q but is not an endpoint of Q, i.e. it lies strictly in the middle of Q. Since $\mathcal{P}$ is an edge-disjoint decomposition of T, there can be at most one such path and thus the claim implies the lemma.

We proceed to prove the claim. Let u', v' be the endpoints of $\pi_Q(\ell)$. Since $\mathcal{P}$ is a rooted path decomposition, either u' is an ancestor of v' or vice versa; suppose the former. We now argue that u' is the least common ancestor of u and v. There are two cases: (1) either u' is an endpoint of ℓ; (2) or there is a vertex z of $P(u, v)$ adjacent to u' but is not on Q. In case (1), we are done. Consider case (2). Since u' is not an endpoint of Q, its parent must be on Q, and thus z is a child of u'. Therefore, u' is the least common ancestor of u and v. ◀

Motivated by Lemma 9, we define the online rooted path augmentation problem. An instance of online rooted path augmentation consists of a rooted path Q where the root r is an endpoint of Q. For such an instance, we say that a link is rooted if one of its endpoints is r. Lemma 9 suggests that we should devise an algorithm for online rooted path augmentation with the following refined guarantee.

▶ **Definition 10** (Nice Solution). *A solution F for an instance of online rooted path augmentation is* nice *if for any feasible solution F^*, we have $c(F) \leq O(1)c(R^*) + O(\log n)c(S^*)$ where R^* is the set of rooted links and S^* is the set of non-rooted links of F^*, respectively. An algorithm is nice if it always produces a nice solution.*

▶ **Lemma 11.** *Suppose that there exists a deterministic nice algorithm Path-ALG for online rooted path augmentation. Then, there exists an $O(\log n)$-competitive deterministic algorithm for online tree augmentation.*

Proof. Here is our algorithm for general instances. Consider a general instance of online weighted tree augmentation with tree $T = (V, E)$, requests $e_1, \dots, e_k \subseteq E$ and links $L = \binom{V}{2}$ with costs $c(\ell)$. Our algorithm works as follows. By Lemma 6, there exists a rooted path decomposition $\mathcal{P}$ of T with width $w = O(\log n)$. Now, each rooted path $Q \in \mathcal{P}$ defines an instance of online rooted path augmentation: the links are $L_Q = \{\pi_Q(\ell) : \ell \in L\}$ where $\pi_Q(\ell)$ has cost $c(\ell)$, and the sequence of requests is exactly the subsequence of requests that lie on Q. So, our algorithm runs in parallel $|\mathcal{P}|$ instantiations of Path-ALG, one per rooted path $Q \in \mathcal{P}$. When request e_i arrives, if $e_i \in Q$ (since e_i is an elementary request, it must lie on some path of $\mathcal{P}$), then our algorithm uses the instantiation of Path-ALG on Q to handle that request; in particular, if Path-ALG buys the projected link $\pi_Q(\ell)$, then our algorithm buys the link ℓ.

Let us now analyze the competitive ratio of this algorithm. Let F^* be a feasible solution. For $Q \in \mathcal{P}$, we denote by R^*_Q, and S^*_Q the subset of F^* which is Q-rooted, and Q-non-rooted, respectively. Since Path-ALG is nice, we have that our algorithm's solution F has cost

$$c(F) \leq \sum_{Q \in \mathcal{P}} \left[O(1)c(R^*_Q) + O(\log n)c(S^*_Q)\right] \leq O(\log n)c(F^*),$$

where the last inequality is because Lemma 9 implies that each link of F^* is in S^*_Q for at most one $Q \in \mathcal{P}$ and is in R^*_Q for at most $w = O(\log n)$ paths $Q \in \mathcal{P}$. ◀

In the next section, we construct a nice deterministic algorithm. Together with Lemma 11 this gives a deterministic $O(\log n)$-competitive algorithm for online tree augmentation, thus proving Theorem 1.

4 A Nice Algorithm for Online Path Augmentation

In this section, we devise a nice algorithm for online rooted path augmentation. In the following, we use the convention that the root of the path is the left endpoint of the path.

We begin by showing in Section 4.1 that it suffices to consider simpler instances that we call *minimal instances*. Then, we describe in Section 4.2 how to prove niceness using an LP for the problem. Finally, we describe and analyze the algorithm in Sections 4.3 and 4.4.

4.1 Minimal Instances

The first step is a preprocessing step that simplifies the structure of the link set. In particular, we prune the instance so that it is of the following type.

▶ **Definition 12** (Minimal Instances). *A set of links L and its costs c are* minimal *if they satisfy the following properties:*

1. *for each class j, there is at most one rooted link and for every edge e, there are at most two links ℓ with $e \in P(\ell)$;*
2. *for any two rooted links ℓ and ℓ', if $\mathrm{class}(\ell) > \mathrm{class}(\ell')$, then $P(\ell) \supsetneq P(\ell')$.*

An instance is minimal if its links and costs are minimal.

Given a set of links L and its costs c, we prune L to get a minimal subset of links $L' \subseteq L$ as follows. We begin by pruning the rooted links: while there exists a rooted link ℓ and a rooted link ℓ' of the same or lower class such that $P(\ell') \supseteq P(\ell)$, remove ℓ. Then we prune the non-rooted links for each class j: let L_j be the set of class j links and L'_j be a minimum-size subset of L_j that covers L_j, i.e. $P(L'_j) \supseteq P(L_j)$; then, remove the links $L_j \setminus L'_j$. Such a minimum cover may be computed efficiently using an algorithm for minimum interval cover. By minimality, we have that for any edge e, there are at most two links $\ell, \ell' \in L'_j$ such that $e \in P(\ell) \cap P(\ell')$. The following claim shows that any link $\ell \in L_j$ that was pruned away can be replaced with at most three links of L'_j and so restricting to L' only causes the value of the optimal solution to increase by at most a factor of 3. We defer the proof to the full version.

▷ Claim 13. For every link $\ell \in L_j \setminus L'_j$, there exists (at most) three links $\ell_1, \ell_2, \ell_3 \in L'_j$ with $P(\ell) \subseteq P(\ell_1) \cup P(\ell_2) \cup P(\ell_3)$.

Given a subset of links $L' \subseteq L$, we say that a solution $F \subseteq L'$ is *nice for L'* if for any feasible solution $F' \subseteq L'$, we have $c(F) \leq O(1)c(R') + O(\log n)c(S')$ where R' is the set of rooted links and S' is the set of non-rooted links of F', respectively. The following lemma says that it suffices to have a solution that is nice for a pruning of L and thus it suffices to devise a nice algorithm for minimal instances. We defer the proof of the lemma to the full version.

▶ **Lemma 14.** *Let $L' \subseteq L$ be a pruning of L. Then, a solution that is nice for L' is also nice for L.*

Henceforth, we will focus on devising a nice algorithm for minimal instances.

4.2 Proving Niceness via the Dual LP

Our algorithm uses the standard LP formulation of the problem. Let $\mathcal{R}$ be the set of requests. The following are the primal and dual LPs, respectively.

$$\boxed{\begin{array}{lll} \text{minimize} & \displaystyle\sum_{\ell \in L} x(\ell)c(\ell) & \\ \text{subject to} & \displaystyle\sum_{\ell \in \mathrm{cov}(e)} x(\ell) \geq 1 & \forall e \in \mathcal{R} \end{array}} \tag{1}$$

$$\boxed{\begin{array}{lll} \text{maximize} & \displaystyle\sum_{e \in \mathcal{R}} y(e) & \\ \text{subject to} & \displaystyle\sum_{e \in P(\ell)} y(e) \leq c(\ell) & \forall \ell \in L \end{array}} \tag{2}$$

We say that a link ℓ is *tight* with respect to a dual solution y if $\sum_{e \in P(\ell)} y(e) = c(\ell)$.

The following lemma tells us how to use the dual to prove niceness.

▶ **Lemma 15.** *Let F be a solution. Suppose y is a dual solution such that*
1. $c(F) \leq O(1) \sum_e y(e)$*,*
2. $\sum_{e \in P(\ell)} y(e) \leq O(\log n)c(\ell)$ *for every non-rooted link ℓ, and*
3. $\sum_{e \in P(\ell)} y(e) \leq O(1)c(\ell)$ *for every rooted link ℓ.*

Then, F is a nice solution.

Proof. Let F^* be a feasible solution, R^* be the subset of F^* that is rooted and S^* the subset that is non-rooted. We now show that $\sum_e y(e) \leq O(1)c(R^*) + O(\log n)c(S^*)$, which then implies that $c(F) \leq O(1)c(R^*) + O(\log n)c(S^*)$. Since we have a dual variable $y(e)$ for each request e and F^* is feasible, we have that

$$\sum_e y(e) \leq \sum_{e \in P(R^*)} y(e) + \sum_{e \in P(S^*)} y(e).$$

Using the fact that $\sum_{e \in P(\ell)} y(e) \leq O(1)c(\ell)$ for every rooted link ℓ, we also have

$$\sum_{e \in P(R^*)} y(e) \leq \sum_{\ell \in R^*} \sum_{e \in P(\ell)} y(e) \leq O(1)c(R^*).$$

Similarly, we get that $\sum_{e \in P(S^*)} y(e) \leq O(\log n)c(S^*)$. Putting all of these together, we conclude that $\sum_e y(e) \leq O(1)c(R^*) + O(\log n)c(S^*)$, as desired. ◀

4.3 Algorithm

We now give some of the ideas behind our algorithm.

An $O(\log n)$-competitive algorithm

First, we describe a simple algorithm that constructs a solution F and a dual solution y that satisfies $c(F) \leq O(1) \sum_e y(e)$ and $\sum_{e \in P(\ell)} y(e) \leq O(\log n)c(\ell)$ for *every* link ℓ. The algorithm maintains a maximal feasible dual solution y and is as follows: when a request e_i arrives, raise its dual variable $y(e_i)$ until some link ℓ with $e_i \in P(\ell)$ goes tight; add this link to F. There are two parts to the analysis. First, let $\widehat{F}$ be the set of links in F that cost at least $\max_{\ell \in F} c(\ell)/n^2$. Since $|F| \leq n^2$, we get that $c(F) \leq 2c(\widehat{F})$ so it suffices to bound $c(\widehat{F})$. The second part of the analysis uses the following charging argument to bound $c(\widehat{F})$: whenever we add a tight link ℓ to $\widehat{F}$, we charge its cost to the dual variables $y(e)$ for $e \in P(\ell)$. Let $\lambda(e)$ be the total number of links charged to $y(e)$ and $\widehat{y}$ be the dual solution where $\widehat{y}(e) = \lambda(e)y(e)$. We have $c(\widehat{F}) \leq O(1)\sum_e \lambda(e)y(e)$. Now observe that $\lambda(e) \leq O(\log n)$ because Property 1 of minimal instance implies that there can be at most 2 links $\ell \in \widehat{F}$ with $e \in P(\ell)$ for a single cost class, and, by definition, $\widehat{F}$ can have at most $O(\log n)$ cost classes. So, for each link ℓ, we have

$$\sum_{e \in P(\ell)} \lambda(e)y(e) \leq O(\log n) \sum_{e \in P(\ell)} y(e) \leq O(\log n)c(\ell)$$

where the last inequality follows from the fact that y is feasible.

Saving the rooted links

A natural idea to ensure that $\sum_{e \in P(\ell)} \lambda(e)y(e) \leq O(1)c(\ell)$ for each rooted link ℓ is to modify the above algorithm to explicitly take into account the charging method as follows: after buying the tight link (we call this a type-1 link), if there is a rooted link ℓ' such that $\sum_{e \in P(\ell')} \lambda(e)y(e) > c(\ell')$, buy the one of highest class among such links (we call this a type-2 link). Moreover, we also modify the charging method to only charge each type-1 link ℓ to the dual variables $y(e)$ for $e \notin P(\ell')$ where ℓ' is the last type-2 link bought.

As we will see later, these modifications allow us to argue that $\sum_{e \in P(\ell)} \lambda(e)y(e) \leq O(1)c(\ell)$ for each rooted link ℓ. However, it also introduces a complication: it might be possible that for some type-1 link ℓ, most of the dual variables $y(e)$ paying towards its cost have $e \in P(\ell')$ where ℓ' is the last type-2 link bought. Since the charging method only charges to dual variables $y(e)$ for $e \notin P(\ell')$, this would mean that it might charge an amount that is much less than the cost of ℓ'.

Fixing the complication

To fix the above issue, whenever we buy a type-2 link ℓ', we also buy all links ℓ'' of class at most class(ℓ') that crosses ℓ', i.e. $\emptyset \subsetneq P(\ell'') \cap P(\ell') \subsetneq P(\ell')$. Property 1 implies that the total cost of these links is at most $O(1)c(\ell')$. We call these links type-3 links. This ensures that later on, when we buy a type-1 link ℓ, if $P(\ell) \cap P(\ell') \neq \emptyset$, then ℓ must be of higher class than ℓ' and thus most of its cost is paid for by dual variables $y(e)$ for $e \notin P(\ell')$.

We describe the complete algorithm formally in Algorithm 1. In Algorithm 1, we use Z to keep track of $P(\ell)$ where ℓ is the last type-2 link bought so far ($Z = \emptyset$ if no type-2 link is bought yet). The links bought in Step 4, 9, 11, are type-1, type-2, and type-3 links, respectively.

4.4 Analysis of Algorithm

We now prove that Algorithm 1 is nice. Let $F_1, F_2, F_3 \subseteq F$ be the sets of type-1, type-2 and type-3 links, respectively. The proof consists of three steps. First, we show that $c(F) \leq O(1)c(F_1)$ (Lemma 17) and thus it suffices to bound the cost of type-1 links. Then, we construct a dual solution $\widehat{y}$ that accounts for the cost of type-1 links (Lemma 18). This shows that $\widehat{y}$ satisfies the first condition of Lemma 15. Finally, Lemmas 20 and 19 show that $\widehat{y}$ satisfies the remaining conditions of Lemma 15.

For each type-1 link $\ell \in F_1$, define $C(\ell)$ to be the set of edges e such that $\lambda(e)$ was incremented during the iteration that ℓ was assigned to F_1, i.e. each dual variable $y(e)$ for $e \in C(\ell)$ contributes towards paying $c(\ell)$. Observe that $\lambda(e) = |\{\ell : e \in C(\ell)\}|$ and $C(\ell) \subseteq P(\ell)$.

Algorithm 1 Nice algorithm for online rooted path augmentation.

$F \leftarrow \emptyset; y \leftarrow 0; \lambda \leftarrow 0; Z \leftarrow \emptyset$
for each unsatisfied request e_i **do**
  Increase $y(e_i)$ until some link ℓ with $e_i \in P(\ell)$ goes tight
  Add such a link ℓ to F
  for each $e \in P(\ell) \setminus Z$ such that $y(e) > 0$ **do**
    $\lambda(e) \leftarrow \lambda(e) + 1$
  end for
  if there exists a rooted link $\ell \notin F$ such that $\sum_{e \in P(\ell)} \lambda(e)y(e) \geq c(\ell)$ **then**
    Among such links, add to F the link ℓ of highest class
    for $j \leq \text{class}(\ell)$ **do**
      Add to F all class-j links ℓ' that cross ℓ, i.e. $\emptyset \subsetneq P(\ell') \cap P(\ell) \subsetneq P(\ell)$
    end for
    $Z \leftarrow P(\ell)$
  end if
end for

▶ **Proposition 16.** *Algorithm 1 satisfies the following properties. Let Z_i and λ_i denote Z and λ at the end of the i-th iteration. Then, for every iteration i, we have*

1. $Z_i \supseteq Z_{i-1}$;
2. *if* $y(e_i) > 0$, *then* $\lambda_i(e_i) > 0$.

Proof. The first follows from Property 2 of minimal instances. The second follows from the fact that in the iteration that e_i arrives, since it is unsatisfied, it must not be contained in Z. Let ℓ be the link added to F in that iteration. Since $e_i \in P(\ell) \setminus Z$ and $y(e_i) > 0$, we have that $\lambda(e_i)$ is increased by 1 during the iteration and thus $\lambda_i(e_i) > 0$. ◀

▶ **Lemma 17.** $c(F) \leq O(1)c(F_1)$.

Proof. We will show that $c(F_3) \leq O(1)c(F_2)$, that $c(F_2) \leq O(1)\sum_e \lambda(e)y(e)$ and that $\sum_e \lambda(e)y(e) \leq c(F_1)$. Let ℓ_r be the last type-2 link bought. We have that $c(\ell_r) \leq \sum_{e \in P(\ell_r)} \lambda(e)y(e)$ by construction. Moreover, since $c(\ell_r) \geq c(\ell)$ for every $\ell \in F_2$ and there is at most one rooted link of each class, we get that $c(F_2) \leq 2c(\ell_r)$. Thus, we get that $c(F_2) \leq 2\sum_{e \in P(\ell_r)} \lambda(e)y(e)$. For each type-2 link ℓ bought, we buy at most two type-3 links per class $j \leq \text{class}(\ell)$ because of Property 1 of minimal instances. Therefore, we have $c(F_3) \leq 2c(F_2) \leq 4\sum_{e \in P(\ell_r)} \lambda(e)y(e)$.

Finally, we show that $\sum_e \lambda(e)y(e) \leq c(F_1)$. Since $\lambda(e) = |\{\ell : e \in C(\ell)\}|$, we have $\sum_e \lambda(e)y(e) = \sum_{\ell \in F_1} \left(\sum_{e \in C(\ell)} y(e)\right)$. Now, since $C(\ell) \subseteq P(\ell)$ and y is feasible, we get $\sum_{e \in C(\ell)} y(e) \leq \sum_{e \in P(\ell)} y(e) \leq c(\ell)$. Combining the previous two inequalities gives us that $\sum_e \lambda(e)y(e) \leq c(F_1)$. ◀

Let $c_{\max} = \max_{\ell \in F_1} c(\ell)$. Define $\widehat{F_1} = \{\ell \in F_1 : c(\ell) \geq c_{\max}/n^2\}$ and $\widehat{\lambda}(e) = |\{\ell \in \widehat{F_1} : e \in C(\ell)\}|$. We now show that F and the dual solution $\widehat{y}$ where $\widehat{y}(e) = \widehat{\lambda}(e)y(e)$ satisfies the conditions of Lemma 15.

▶ **Lemma 18.** $c(F_1) \leq O(1)\sum_e \widehat{\lambda}(e)y(e)$.

Proof. Observe that $c(F_1) \leq 2c(\widehat{F_1})$ so it suffices to prove that

$$c(\widehat{F_1}) \leq O(1)\sum_e \widehat{\lambda}(e)y(e). \tag{3}$$

We now show that this inequality holds at the end of each iteration of the algorithm. Consider an iteration in which the current request e_i is not already covered and suppose $\ell \in \widehat{F}_1$ is the type-1 link bought in this iteration. The LHS of Inequality (3) increases by $c(\ell)$ in this iteration. We now show that $\sum_e \widehat{\lambda}(e)y(e)$ increases by at least $c(\ell)/2$. In this iteration, $\widehat{\lambda}(e)$ increases by 1 for every $e \in P(\ell) \setminus Z$ and $y(e) > 0$, and so $\sum_e \widehat{\lambda}(e)y(e)$ increases by exactly $\sum_{e \in P(\ell)\setminus Z} y(e)$.

In the remainder of the proof, we show that $\sum_{e\in P(\ell)\setminus Z} y(e) \geq c(\ell)/2$. If $P(\ell) \cap Z = \emptyset$, then $\sum_{e\in P(\ell)\setminus Z} y(e) = \sum_{e\in P(\ell)} y(e) = c(\ell)$ since ℓ is tight. Now suppose $P(\ell) \cap Z \neq \emptyset$. Let ℓ' be the type-2 link such that $Z = P(\ell')$. Since $P(\ell) \cap P(\ell') \neq \emptyset$, it must be the case that ℓ is of type higher than class(ℓ'). This is because otherwise, ℓ would have been bought earlier as a type-3 link in the same iteration as ℓ'. But then since $e_i \in P(\ell)$, it would contradict the assumption that e_i is not already covered at the start of the current iteration. Thus, class(ℓ) > class(ℓ') and so $c(\ell) \geq 2c(\ell')$. So, we now have

$$\sum_{e\in P(\ell)\setminus Z} y(e) \geq \sum_{e\in P(\ell)} y(e) - \sum_{e\in P(\ell')} y(e) \geq c(\ell) - c(\ell') \geq c(\ell)/2,$$

where the second last inequality follows from the fact that y is a feasible dual and ℓ is tight. Therefore, Inequality (3) holds at the end of each iteration, as desired. ◀

Lemmas 17 and 18 imply that $c(F) \leq O(1)\sum_e \widehat{y}(e)$.

▶ **Lemma 19.** *For each non-rooted link ℓ, we have $\sum_{e\in P(\ell)} \widehat{\lambda}(e)y(e) \leq O(\log n)c(\ell)$.*

Proof. Property 1 of minimal instances implies that for each j, there are at most two links $\ell' \in \widehat{F}_1$ of class j with $e \in C(\ell')$. Since each link in $\widehat{F}_1$ has cost between $c_{\max}/n^2$ and $c_{\max}$ and link costs are powers of 2, we have that $\widehat{\lambda}(e) \leq O(\log n)$. Thus we get that $\sum_{e\in P(\ell)} \widehat{\lambda}(e)y(e) \leq O(\log n)\sum_{e\in P(\ell)} y(e) \leq O(\log n)c(\ell)$, where the last inequality follows from the fact that y is a feasible dual. ◀

▶ **Lemma 20.** *For each rooted link ℓ, we have $\sum_{e\in P(\ell)} \widehat{\lambda}(e)y(e) \leq O(1)c(\ell)$.*

Proof. We will in fact show that $\sum_{e\in P(\ell)} \lambda(e)y(e) \leq O(1)c(\ell)$. Suppose, at the end of some iteration, that we have $\sum_{e\in P(\ell)} \lambda(e)y(e) > c(\ell)$. Consider the earliest iteration that this happens. We now show that $\sum_{e\in P(\ell)} \lambda(e)y(e) \leq O(1)c(\ell)$ at the end of the iteration and later show that the LHS cannot increase in future iterations.

Let $\lambda^{\text{old}}(e)$ and $y^{\text{old}}(e)$ denote the values of $\lambda(e)$ and $y(e)$ at the start of the iteration and $\lambda^{\text{new}}(e)$ and $y^{\text{new}}(e)$ denote their values at the end. We have that $\sum_{e\in P(\ell)} \lambda^{\text{old}}(e)y^{\text{old}}(e) < c(\ell)$. We now show that $\sum_{e\in P(\ell)} \lambda^{\text{new}}(e)y^{\text{new}}(e) \leq 3c(\ell)$. Let e_i be the request of the current iteration. During this iteration, we only increase $y(e)$ for $e = e_i$ and we set $\lambda(e_i) = 1$ so $\lambda^{\text{new}}(e_i)y^{\text{new}}(e_i) = y(e_i)$. So, we have

$$\sum_{e\in P(\ell)} \lambda^{\text{new}}(e)y^{\text{new}}(e) = \sum_{e\in P(\ell)\setminus\{e_i\}} \lambda^{\text{new}}(e)y^{\text{old}}(e) + y(e_i).$$

Since y is a feasible dual, we have that $y(e_i) \leq c(\ell)$. Now, Proposition 16 implies that $\lambda^{\text{old}}(e) \geq 1$ if $y^{\text{old}}(e) > 0$. Together with the fact that $\lambda^{\text{new}}(e) \leq \lambda^{\text{old}}(e) + 1$, we get that $\lambda^{\text{new}}(e)y^{\text{old}}(e) \leq 2\lambda^{\text{old}}(e)y^{\text{old}}(e)$ and so

$$\sum_{e\in P(\ell)\setminus\{e_i\}} \lambda^{\text{new}}(e)y^{\text{new}}(e) \leq 2\sum_{e\in P(\ell)\setminus\{e_i\}} \lambda^{\text{old}}(e)y^{\text{old}}(e) < 2c(\ell).$$

Thus, $\sum_{e\in P(\ell)} \lambda^{\text{new}}(e)y^{\text{new}}(e) \leq 3c(\ell)$ at the end of the current iteration.

Finally, we show that $\sum_{e\in P(\ell)} \lambda(e)y(e)$ does not increase in future iterations. At the end of the current iteration, ℓ is a candidate to be added to F. Among all candidates, the one with highest class is added, so either ℓ is added to F or a rooted link ℓ' of higher class is added to F. In the second case, by Proposition 16, we have $P(\ell') \supseteq P(\ell)$. Thus, in either case, we have that $Z \supseteq P(\ell)$ at the end of the current iteration. Moreover, in future iterations, we still have $Z \supseteq P(\ell)$ by Proposition 16. Therefore, $\sum_{e\in P(\ell)} \lambda(e)y(e)$ does not increase in future iterations. Thus, we conclude that $\sum_{e\in P(\ell)} \lambda(e)y(e) \leq 3c(\ell)$ at the end of the algorithm. ◀

Therefore, we conclude that Algorithm 1 is nice. Together with Lemma 11, we get Theorem 1.

References

1 Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Naor. The Online Set Cover Problem. *SIAM J. Comput.*, 39(2):361–370, 2009. doi:10.1137/060661946.

2 Baruch Awerbuch, Yossi Azar, and Yair Bartal. On-line generalized Steiner problem. *Theoretical Computer Science*, 324:313–324, 2004.

3 Piotr Berman and Chris Coulston. On-Line Algorithms for Steiner Tree Problems. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 344–353, 1997.

4 Sina Dehghani, Soheil Ehsani, MohammadTaghi Hajiaghayi, Vahid Liaghat, and Saeed Seddighin. Greedy Algorithms for Online Survivable Network Design. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 152:1–152:14, 2018. doi:10.4230/LIPIcs.ICALP.2018.152.

5 Guy Even and Shakhar Smorodinsky. Hitting sets online and unique-max coloring. *Discrete Applied Mathematics*, 178:71–82, 2014. doi:10.1016/j.dam.2014.06.019.

6 Greg N. Frederickson and Joseph JáJá. Approximation Algorithms for Several Graph Augmentation Problems. *SIAM J. Comput.*, 10(2):270–283, 1981. doi:10.1137/0210019.

7 Fabrizio Grandoni, Christos Kalaitzis, and Rico Zenklusen. Improved approximation for tree augmentation: saving by rewiring. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 632–645, 2018. doi:10.1145/3188745.3188898.

8 Anupam Gupta, Ravishankar Krishnaswamy, and R. Ravi. Online and Stochastic Survivable Network Design. *SIAM J. Comput.*, 41(6):1649–1672, 2012. doi:10.1137/09076725X.

9 MohammadTaghi Hajiaghayi, Vahid Liaghat, and Debmalya Panigrahi. Online Node-weighted Steiner Forest and Extensions via Disk Paintings. In *Proceedings of the 54th Annual Symposium on Foundations of Computer Science*, pages 558–567, 2013.

10 MohammadTaghi Hajiaghayi, Vahid Liaghat, and Debmalya Panigrahi. Near-Optimal Online Algorithms for Prize-Collecting Steiner Problems. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming, 41st International Colloquium, ICALP 2014*, volume 8572 of *Lecture Notes in Computer Science*, pages 576–587. Springer, 2014.

11 Makoto Imase and Bernard M. Waxman. Dynamic Steiner Tree Problem. *SIAM Journal on Discrete Mathematics*, 4:369–384, 1991.

12 Adam Meyerson. The Parking Permit Problem. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*, pages 274–282, 2005.

13 Joseph Naor, Debmalya Panigrahi, and Mohit Singh. Online Node-Weighted Steiner Tree and Related Problems. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 210–219, 2011. doi:10.1109/FOCS.2011.65.

14 Jiawei Qian, Seeun William Umboh, and David P. Williamson. Online Constrained Forest and Prize-Collecting Network Design. *Algorithmica*, 80(11):3335–3364, 2018.

15 Daniel Dominic Sleator and Robert Endre Tarjan. A Data Structure for Dynamic Trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983. doi:10.1016/0022-0000(83)90006-5.

16 Seeun Umboh. Online Network Design Algorithms via Hierarchical Decompositions. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1373–1387, 2015.