An $O(\log n)$-competitive algorithm for online constrained forest problems

Jiawei Qian[⋆] and David P. Williamson[⋆⋆]

School of Operations Research and Information Engineering,
Cornell University, Ithaca, NY 14853, USA

Abstract. In the generalized Steiner tree problem, we find a minimum-cost set of edges to connect a given set of source-sink pairs. In the online version of this problem, the source-sink pairs arrive over time. Agrawal, Klein, and Ravi [1] give a 2-approximation algorithm for the offline problem; Berman and Coulston [3] give an $O(\log n)$-competitive algorithm for the online problem. Goemans and Williamson [4] subsequently generalized the offline algorithm of Agrawal et al. to handle a large class of problems they called *constrained forest problems*, and other problems, such as the prize-collecting Steiner tree problem. In this paper, we show how to combine the ideas of Goemans and Williamson and those of Berman and Coulston to give an $O(\log n)$-competitive algorithm for online constrained forest problems, including an online version of the prize-collecting Steiner tree problem.

1 Introduction

Given an undirected graph $G = (V, E)$, edge costs $c_e \geq 0$ for all $e \in E$, and a set of l source-sink pairs s_i-t_i, the goal of the *generalized Steiner tree problem* (also known as the *Steiner forest problem*) is to find a minimum-cost set of edges $F \subseteq E$ such that for each i, s_i and t_i are connected in (V, F). This problem is (as its name implies) a generalization of the *Steiner tree problem*: in Steiner tree problem, we are given an undirected graph with edge costs as above, and also a set $R \subseteq V$ of *terminals*. The goal of the Steiner tree problem is to find a minimum-cost tree T that spans all the terminals R. If we choose one of the terminals $r \in R$ arbitrarily, and set $s_i = r$ for all i and the sink vertices t_i are the remaining vertices in R, then clearly a Steiner tree instance can be expressed as a generalized Steiner tree problem instance. In the 1990s, Agrawal, Klein, and Ravi [1] gave a 2-approximation algorithm for the generalized Steiner tree problem.

At about the same time, online algorithms were being proposed for online versions of the Steiner tree problem, and later, the generalized Steiner tree problem. In the online version of the Steiner tree problem, terminals arrive over time.

[⋆] `jq35@cornell.edu`. Supported in part by NSF Grant CCF-0830519.

[⋆⋆] `dpw@cs.cornell.edu`. Part of this work was performed while the author was on sabbatical at TU Berlin, sponsored by the Berlin Mathematical School and a von Humboldt Research Award.

At each time step we must give a set of edges F that connects all of the terminals that have arrived thus far; we are not allowed to remove any edges from F in future iterations. The quality of an online algorithm for this problem is measured in terms of its *competitive ratio*: an α-competitive algorithm is one such that at any time step, the set of edges constructed by the algorithm is within a factor of α of the cost of an optimal Steiner tree on the set of terminals that have arrived thus far. Similarly, in the online generalized Steiner tree problem, source-sink pairs arrive in each time step, and we must find a set of edges F such that each s_i-t_i pair that has arrived thus far is connected in (V, F). Imase and Waxman [7] gave a greedy $O(\log n)$-competitive algorithm for the online Steiner tree problem, where $n = |V|$; when a terminal arrives, it finds the shortest path from the terminal to the tree already constructed, and adds that set of edges to its solution. Imase and Waxman also show that the competitive ratio of any online algorithm must be $\Omega(\log n)$. Awerbuch, Azar, and Bartal [2] then showed that a similar greedy algorithm for the online generalized Steiner tree problem has competitive ratio $O(\log^2 n)$. In 1997, Berman and Coulston [3] devised a more complicated algorithm that is an $O(\log n)$-competitive algorithm for the online generalized Steiner tree problem, matching the lower bound of Imase and Waxman to within constant factors.

Also in the 1990s, Goemans and Williamson [4] extended the offline algorithm of Agrawal, Klein, and Ravi to a large class of problems they called constrained forest problems; in doing so, they cast the algorithm of Agrawal et al. as a primal-dual algorithm. A constrained forest problem is defined by a function $f : 2^V \rightarrow \{0, 1\}$; for any set $S \subseteq V$ such that $f(S) = 1$, a feasible solution must have selected at least one edge in $\delta(S)$, the set of edges with exactly one endpoint in S. The Goemans-Williamson algorithm works when the function f is *proper*: that is, when $f(S) = f(V - S)$ for all $S \subseteq V$, and for all disjoint sets $A, B \subseteq V$, $f(A \cup B) \leq \max(f(A), f(B))$. For instance, for the case of the generalized Steiner tree problem $f(S) = 1$ if and only if there exists some i such that $|S \cap \{s_i, t_i\}| = 1$, and this function is proper. Another example of constrained forest problems given in [4] is the nonfixed point-to-point connection problem, in which a subset C of the vertices are sources, a disjoint subset D of vertices are destinations, and we must find a minimum-cost set of edges such that each source is connected to a destination so that each connected component has the same number of sources and destinations; this is modelled by having $f(S) = 1$ if $|S \cap C| \neq |S \cap D|$. Yet another example given in [4] is that of partitioning specified vertices D into connected components such that the number of vertices of D in each connected component C is divisible by some parameter k. This problem is given the proper function f such that $f(S) = 1$ if $|S \cap D| \not\equiv 0 (\text{mod } k)$.

In this paper, we show that by melding the ideas of Goemans and Williamson with those of Berman and Coulston, we can obtain an $O(\log n)$-competitive algorithm for any *online* constrained forest problem. In an online constrained forest problem, in each time step i we are given a proper function f_i. We must choose a set of edges F such that for all $S \subseteq V$, if $\max_{j=1,\ldots,i} f_j(S) = 1$, then $|\delta(S) \cap F| \geq 1$. This yields, for example, online algorithms for online variants of the nonfixed point-to-point connection problem and the partitioning problems given above.

Our techniques also extend to give an $O(\log n)$-competitive algorithm for an online version of the prize-collecting Steiner tree problem. In the offline version of the problem, we are given an undirected graph $G = (V, E)$, edge costs $c_e \geq 0$ for all $e \in E$, a root vertex $r \in V$, and penalties $\pi_v \geq 0$ for all $v \in V$. The goal is to find a tree T spanning the root vertex that minimizes the cost of the edges in the tree plus the penalties of the vertices not spanned by the tree; that is, we want to minimize $\sum_{e \in T} c_e + \sum_{v \in V - V(T)} \pi_v$, where $V(T)$ is the set of vertices spanned by T. In the online version of the problem, initially every vertex v has penalty $\pi_v = 0$. At each step in time, for one vertex v its penalty π_v is increased from 0 to some positive value. We then must either connect the vertex to the root by adding edges to our current solution or pay the penalty π_v for each remaining time step of the algorithm even if it is connected to the root later on. The competitive ratio of the algorithm compares the cost of our solution in each step with the cost of the optimal solution of the instance at that point in time.

The basic idea of the Berman-Coulston algorithm (BC) is that it constructs many different families of nonoverlapping balls around terminals as they arrive; in the jth family, balls are limited to have radius at most 2^j. Each family of balls is a lower bound on the cost of an optimal solution to the generalized Steiner tree problem. When balls from two different terminals touch, the algorithm buys the set of edges connecting the two terminals, and balls from one of the two terminals (in some sense the 'smaller' one) can be charged for the cost of the edges, leaving the balls from the other terminal (the 'larger' one) uncharged and able to pay for future connections. One can show that the $O(\log n)$ largest families are essentially all that are relevant for the charging scheme, so that the largest of these $O(\log n)$ families is within an $O(\log n)$ factor of the cost of the constructed solution, thereby giving the competitive ratio. Our algorithm replaces each family of balls with an analogous solution to the dual of the linear programming relaxation of the constrained forest problem, as used by Goemans and Williamson. We need somewhat more complicated dual solutions than the balls used by BC. However, we can then largely follow the outline of the BC analysis to obtain our $O(\log n)$ competitive ratio.

The rest of the paper is structured as follows. In Section 2, we introduce the online constrained forest problem more precisely and define some concepts we will need for our algorithm. In Section 3, we give the algorithm and its analysis. In Section 4, we extend the algorithm and analysis to the prize-collecting Steiner tree problem. We mention some open problems in our conclusion in Section 5.

2 Preliminaries

Given an undirected graph $G = (V, E)$, edges costs $c_e \geq 0$ and a $\{0, 1\}$-proper function $f : 2^V \rightarrow \{0, 1\}$, the offine constrained forest problem studied in Goemans and Williamson [4] is to find a set of edges F of minimum cost that satisfies a connectivity requirement function $f : 2^V \rightarrow \{0, 1\}$; the function is *satisfied* if for each set $S \subseteq V$ with $f(S) = 1$, we have $|\delta_F(S)| \geq 1$, where $\delta(S)$ is the set of edges with exactly one endpoint in S, and $\delta_F(S) = \delta(S) \cap F$. In the online version of this problem, we have a sequence of connectivity functions $f_1, f_2, ..., f_i$, arriving one by one. Starting with $F = \emptyset$, for each *time step* $i \geq 1$, function f_i arrives and we need to add edges to F to satisfy function f_i. Let $g_i(S) = \max\{f_1(S), ..., f_i(S)\}$ for all $S \subseteq V$ and $i \geq 1$. Then our goal is to a find a minimum-cost set of edges F that satisfies function g_i, that is, all connectivity requirements given by $f_1, ..., f_i$ that have arrived thus far. We require that each function f_i be a proper function, as defined above. It is easy to see that function g_i is also proper.

Call a vertex v a *terminal* if $f_l(\{v\}) = 1$ for some $l \leq i$. Let $R_i = \{s \in V \mid g_i(\{s\}) = 1\}$ be the set of terminals defined by function g_i; that is, R_i is the set of all terminals that have arrived by time i. A special case of this problem is the online generalized Steiner tree problem where terminal pairs $(s_1, t_1), ..., (s_i, t_i)$ arrive one at a time. In this case, $f_i(S) = 1$ if $|S \cap \{s_i, t_i\}| = 1$ and (s_i, t_i) is the pair of terminals arrive in time step i; then $R_i = \{s_j, t_j : j \leq i\}$. Berman and Coulston [3] give an $O(\log |R_i|)$-competitive algorithm for the online generalized Steiner tree problem.

Let (IP_i) be the integer program corresponding to the online proper constrained forest problem with set of functions $f_1, ..., f_i$ that have arrived thus far and the corresponding function g_i. The integer programming formulation of (IP_i) is

$$(IP_i) \qquad \begin{aligned} &\text{Min} \sum_{e \in E} c_e x_e \\ &\sum_{e \in \delta(S)} x_e \geq g_i(S), && \forall S \subseteq V, \\ &x_e \in \{0, 1\}, && \forall e \in E. \end{aligned}$$

We let (LP_i) denote the corresponding linear programming relaxation in which the constraints $x_e \in \{0, 1\}$ are replaced with $x_e \geq 0$. The dual of this linear program, (D_i), can be described as

$$(D_i) \qquad \begin{aligned} &\text{Max} \sum_{S \subseteq V} g_i(S) y_S \\ &\sum_{S: e \in \delta(S)} y_S \leq c_e, && \forall e \in E, \\ &y_S \geq 0, && \forall S \subseteq V. \end{aligned}$$

We now define a number of terms that we will need to describe our algorithm. We will keep an infinite number of feasible dual solutions y^j to bound the cost of edges in F over all time steps; we call this the dual solution for *level* j. For each level j, we will maintain that for any terminal s that has arrived thus far, $\sum_{S \subseteq V : s \in S} y_S^j \leq 2^j$. So we say that the *limit* of the dual in level j is 2^j, and we say that a dual variable y_S^j *reaches its limit* if the inequality for level j is tight for any terminal $s \in S$. As a matter of algorithmic implementation, we don't need to maintain levels $j < -1$, or $j > \lceil \log(\max_{u,v \in V} d(u,v)) \rceil$, where $d(u,v)$ is the distance in G between u and v using edge costs c_e.

An edge $e \in E$ is *tight* in level j for dual vector y^j if the corresponding constraint in dual problem (D_i), $\sum_{S: e \in \delta(S)} y_S^j \leq c_e$, holds with equality. A path $p \subseteq E$ is *tight* in level j if every edge in the path is tight in level j.

Let $\bar{F}^j$ denote the set of edges that are tight in level j. To avoid confusion with connected components in F, we will use the term *moat* to refer to a connected component S^j in $\bar{F}^j$ and use y_S^j to refer the dual variable associated with S^j.

A set $S \subseteq V$ is a *violated* set for function g_i by edges B if $|\delta_B(S)| < g_i(S)$; that is, if $g_i(S) = 1$ but $\delta_B(S) = \emptyset$. A *minimal violated set* is a violated set with every strict subset not violated. The connectivity requirement function g_i is satisfied by edges B if every set $S \subseteq V$ is not a violated set for g_i by B.

During time step i, a terminal $s \in R_i$ is *active* if for some set S, we have $s \in S$ and S is a violated set for function g_i by current solution F. Let A be the set of active terminals at any time of the algorithm. We define the set of *active* moats as the moats S^j of the lowest level j that satisfy the following three conditions: (i) S^j contains some active terminal $s \in A$; (ii) S^j is a minimum violated set for g_i by edges $\bar{F}^j$; (iii) y_S^j has not yet reached its limit in level j. We denote the current set of active moats by $\mathcal{M}$. Last, we say a dual variable y_S^j is *active* if its corresponding moat S^j is active.

3 The Algorithm and Its Analysis

3.1 The Primal-Dual Online Algorithm

Our algorithm (see Fig. 1) is a dual ascent algorithm in which we grow active dual variables. We say two disjoint moats S_1^j and S_2^j *collide* in level j during our dual growing process if both of them have been active at some point and a path connecting two terminals $s_1 \in S_1^j$ and $s_2 \in S_2^j$ becomes tight in level j. In order for this to happen, at least one of S_1^j and S_2^j must currently be active.

Our algorithm starts with $F = \emptyset$ and $y_S^j = 0$ for all j and all $S \subseteq V$. At the beginning of each time step i, the function f_i arrives and some non-terminal nodes in V may become terminals. We will update active terminal set A and active moat set $\mathcal{M}$. In each time step i, while there are still some active terminals in A, our algorithm will grow uniformly all active dual variables until: (1) an active y_S^j with reaches its limit in level j; (2) an edge $e \in E$ becomes tight in level j; we then add e to $\bar{F}^j$; (3) two disjoint moats S_1^j and S_2^j collide in level j; we then let p be the path connecting two terminals $s_1 \in S_1^j$ and $s_2 \in S_2^j$ in level j, and we build path p in F if s_1 and s_2 are not yet connected in F, and update the set A of active terminals. At the end of each iteration, we update the set $\mathcal{M}$ of active moats. We output F as the solution for (IP_i).

3.2 The Analysis

Now, we will now state our main theorem and a few lemmas that we need to prove it.

```
Algorithm
$F = \emptyset$, $\bar{F}^j = \emptyset$ for all $j$, and $y_S^j = 0$ for all $j$ and $S \subseteq V$
For each $\{0,1\}$-proper function $f_i$ that arrives
    Update active terminals $A$, and active moats $\mathcal{M}$
    While $|A| > 0$
        Grow uniformly all active dual variables $y_S^j$ until
        1) An active $y_S^j$ with reaches limit in level $j$
        2) An edge $e \in E$ becomes tight in level $j$, then
            $\bar{F}^j = \bar{F}^j \cup \{e\}$
        3) Two disjoint moats $S_1^j$ and $S_2^j$ collide in level $j$, then
            Let $p \subseteq E$ be the corresponding path that becomes tight in level $j$
            Let $s_1$ and $s_2$ be the two terminals connected by $p$ in $\bar{F}^j$
            If $s_1$ and $s_2$ are yet connected in $F$
                $F = F \cup \{p\}$, i.e. build edges $p - F$
                Update $A$
        Update $\mathcal{M}$
```

Fig. 1. Primal-Dual Algorithm for Online Proper Constrained Forest Problem

Theorem 1. *Our algorithm gives an* $O(\log |R_i|)$ *competitive ratio for the online proper constrained forest problem* (IP_i).

Lemma 1. *At the end of time step* i *of our algorithm,* F *is a feasible solution to* (IP_i) *and each dual vector* y^j *is a feasible solution to* (D_i).

Proof. Our algorithm terminates each time step i when there are no active terminals in A. By definition of active terminals, this implies that there is no violated set for g_i for the solutions F; that is, F is a feasible solution to (IP_i).

We need to show our algorithm always terminates in each time step. Notice that if there are no active moats in $\mathcal{M}$, then there must be no active terminals in A, since there is always a level j for sufficiently large j that conditions (ii) and (iii) are satisfied in the definition of $\mathcal{M}$. Similarly, if there is still an active terminal, there must be an active moat that contains it. Then our algorithm will continue to grow duals at progressively higher levels; eventually all pairs of active terminals must be connected.

By construction of the algorithm each dual solution y^j is feasible for (D_i) since we stop growing a dual y_S^j if it would violate a dual constraint. □

In order to give a bound to the total cost of edges in F, we create an account for each connected component X in F, denoted Account(X). We will define a shadow algorithm to credit potential to accounts as dual grows and remove potential from accounts to pay for building edges. We will show that the total cost of edges in F plus the total unused potential remaining in all accounts is always equal to the sum of all dual variables over all levels, i.e. $\sum_j \sum_S y_S^j$.

We need the following lemma before we describe the shadow algorithm.

Lemma 2. *Any active dual variable y_S^j has a unique connected component X in F that contains all terminals in its corresponding moat S^j.*

Proof. It is sufficient to show that all terminals in an active moat S^j are connected in F. Suppose not; then we have terminals s_1 and s_2 both in S^j and not connected in F. Then either s_1 and s_2 have no path in $\bar{F}^j$ connecting them or at least one of s_1 and s_2 was not a terminal when the path in $\bar{F}^j$ connecting s_1 and s_2 became tight. The first case contradicts the fact that y_S^j is active so that S^j must be a moat, i.e. a connected component in $\bar{F}^j$. The second case cannot happen by the construction of our algorithm since we grow duals from lowest levels possible. When s_1 and s_2 became terminals, some level j' with $j' < j$ small enough will have s_1 and s_2 in different moats, and a path p between them becomes tight in some level between j' and $j-1$ by our structure of limits in each level (i.e. dual growth in one level can be no larger than two times of dual growth in one level below). Then path p is built in F and s_1 and s_2 will be connected in F before the algorithm grows dual in level j again. □

Now the shadow algorithm is as follows. First, whenever we grow an active dual variable y_S^j, we will credit the same amount of potential to Account(X), where X is the unique connected component in F that contains all terminals in S^j. Second, whenever the algorithm builds a path p in F connecting two terminal s_1 and s_2, it must be the case that two disjoint moats S_1^j and S_2^j collide in some level j with $s_1 \in S_1^j$ and $s_2 \in S_2^j$ not yet connected in F. Let X_k be connected component in F that contains s_k for $k = 1, 2$. As a result of building edges $p - F$, $X_3 = X_1 \cup X_2 \cup \{p - F\}$ will become a connected component in F. We will merge unused potential remaining in Account(X_1) and Account(X_2) into Account(X_3) and remove potential from Account(X_3) to pay for the cost of building edges in $p - F$.

At any time of the algorithm, for each connected component X, define the class of X to be the highest level j with a dual variable already grown that credits X; we denote this as as Class(X) and sometimes refer to it as the *top level* of X. Define TopGrowth(X) to be the maximum total dual growth of a terminal in X in level Class(X), i.e.

$$\text{TopGrowth}(X) = \max_{s \in X}\{\sum_{S \subseteq V : s \in S} y_S^{\text{Class}(X)} \text{ and } s \text{ is a terminal}\}.$$

We know that TopGrowth(X) $\leq 2^j$ by dual limit on level j.

Lemma 3. *At any time of the algorithm, the following two invariants hold:*

1. *Every connected component X of F has*

$$\text{Account}(X) \geq 2^{\text{Class}(X)} + \text{TopGrowth}(X);$$

2. $\sum_{e \in F} c_e + \sum_{X \in F} \text{Account}(X) = \sum_j \sum_S y_S^j$.

Proof. Invariant 1 ensures that for a component X, Account(X) stores at least 2^j total potential for each level $j <$ Class(X) plus the maximum total dual growth of a terminal in X at the top level, which gives $2^{\text{Class}(X)-1} + 2^{\text{Class}(X)-2} + ... = 2^{\text{Class}(X)}$ plus TopGrowth(X).

We prove the first invariant by induction on the algorithm. It is easy to see that this invariant holds when no edges are added to F since the algorithm grows dual variables in level j until some active dual variable reaches limit 2^j; it then grows duals in next higher level. Account(X) is credited 2^j for each level below the level Class(X) while getting TopGrowth(X) for current level.

Now, assume invariant 1 holds just before we add edges to F. The algorithm builds a path p in F connecting two terminals s_1 and s_2 only if there are two disjoint moats S_1^j and S_2^j that collide in some level j with $s_1 \in S_1^j$ and $s_2 \in S_2^j$ not yet connected in F. Let X_k be connected component in F that contains s_k for $k = 1, 2$. We know at least one of S_1^j and S_2^j must be active. Without loss of generality, let it be S_1^j. Then we know Class(X_1) $= j$ and Class(X_2) $= j' \geq j$ since we only grow active dual variables in the top level of each component in F. Then by assumption, Account(X_1) $\geq 2^j +$ TopGrowth(X_1) and Account(X_2) $\geq 2^{j'} +$ TopGrowth(X_2). After building edges $p - F$, $X_3 = X_1 \cup X_2 \cup \{p - F\}$ will be a connected component in F. Our shadow algorithm merges the unused potential remaining in Account(X_1) and Account(X_2) into Account(X_3), and removes potential from Account(X_3) to pay for the cost of building edges $p - F$. Since Class(X_3) $= \max\{j, j'\} = j'$, we need to show Account(X_3) $\geq 2^{j'} +$ TopGrowth(X_3).

If $j = j'$, we know TopGrowth(X_1)+TopGrowth(X_2) $\geq \sum_{e\in p} c_e \geq \sum_{e\in p-F} c_e$ since S_1^j and S_2^j collide in level j, and the cost of the path from s_1 to s_2 cannot be more than the total dual containing s_1 and s_2 in level j. Also, TopGrowth(X_3) $\leq 2^j$ by the dual limit on level j. So we have

$$\begin{aligned}\text{Account}(X_3) &= \text{Account}(X_1) + \text{Account}(X_2) - \textstyle\sum_{e\in p-F} c_e\\ &\geq 2^j + \text{TopGrowth}(X_1) + 2^j + \text{TopGrowth}(X_2) - \textstyle\sum_{e\in p-F} c_e\\ &\geq 2^j + 2^j \geq 2^{j'} + \text{TopGrowth}(X_3).\end{aligned}$$

If $j < j'$, we know TopGrowth(X_1) $+ 2^j \geq \sum_{e\in p} c_e \geq \sum_{e\in p-F} c_e$ since S_1^j and S_2^j collide in level j and the cost of the path from s_1 to s_2 cannot be more than the total dual containing s_1 and s_2 in level j. Also, TopGrowth(X_3) $=$ TopGrowth(X_2) since $j < j'$. So, we have

$$\begin{aligned}\text{Account}(X_3) &= \text{Account}(X_1) + \text{Account}(X_2) - \textstyle\sum_{e\in p-F} c_e\\ &\geq 2^j + \text{TopGrowth}(X_1) + 2^{j'} + \text{TopGrowth}(X_2) - \textstyle\sum_{e\in p-F} c_e\\ &\geq 2^{j'} + \text{TopGrowth}(X_3).\end{aligned}$$

Therefore, the invariant 1 holds at any time of the algorithm. Furthermore, since accounts get credited for dual growth and debited exactly the cost of edges in F, we also have that invariant 2 holds at any time of the algorithm. □

Lemma 4. *Let the dual vector y^j with the maximum total dual $\sum_S y_S^j$ be $y^{\max}$. At the end of time step i, we have $\sum_{e\in F} c_e \leq (\log |R_i| + 2) \sum_S y_S^{max}$.*

Proof. By invariant 2 of Lemma 3, $\sum_{e\in F} c_e = \sum_j \sum_S y_S^j - \sum_{X\in F} \text{Account}(X)$. So it suffices to show $\sum_j \sum_S y_S^j - \sum_{X\in F} \text{Account}(X) \leq (\log|R_i| + 2) \sum_S y_S^{max}$.

At the end of time step i, let X^* be a connected component in F of highest class and let $m = \text{Class}(X^*)$. We know $\text{Account}(X^*) \geq 2^m$ by invariant 1 of Lemma 3. So the total unused potential remaining in all accounts (that is, $\sum_{X\in F} \text{Account}(X)$) is at least 2^m.

Let $\lambda = \lceil \log_2 |R_i| \rceil$. Each terminal s has total dual $\sum_{S\subseteq V: s\in S} y_S^{-\lambda} \leq 2^{-\lambda} \leq 1/|R_i|$ in level $-\lambda$ by the dual limit, so that $\sum_S y_S^{-\lambda} \leq 1$. Similarly, we have $\sum_S y_S^{m-\lambda-j-1} \leq 2^{m-j-1}$. Consider all dual vectors of the form $y^{m-\lambda-j-1}$ with $j \geq 0$; then we have $\sum_{j\geq 0} \sum_S y_S^{m-\lambda-j-1} \leq 2^{m-1} + 2^{m-2} + 2^{m-3} + ... = 2^m \leq \sum_{X\in F} \text{Account}(X)$.

Then, if we consider the dual solutions $y^{m-\lambda}, \ldots, y^m$, we have

$$\sum_j \sum_S y_S^j - \sum_{X\in F} \text{Account}(X) \leq \sum_{k=0}^{\lambda} \sum_S y_S^{m-\lambda+k} \leq (\log|R_i| + 2) \sum_S y_S^{max}.$$

The lemma follows. □

Now, we are ready to prove Theorem 1.

Proof of Theorem 1: By Lemma 1, at the end of time step i of the algorithm, F is a feasible solution to (IP_i). We have

$$\begin{aligned} \textstyle\sum_{e\in F} c_e &= \textstyle\sum_j \sum_S y_S^j - \sum_{X\in F} \text{Account}(X) && \text{by Lemma 3} \\ &\leq (\log|R_i| + 2) \textstyle\sum_S y_S^{\max} && \text{by Lemma 4} \\ &\leq (\log|R_i| + 2) OPT_i && \text{by Lemma 1} \end{aligned}$$

where OPT_i is the optimal value of (IP_i) and the last inequality follows by the fact that the cost of a feasible dual solution to (D_i) is a lower bound on OPT_i. Therefore, our algorithms is an $O(\log|R_i|)$-competitive algorithm for the online proper constrained forest problem. Note that we have $|R_i| = O(n)$, where is n is the number of nodes in G. □

4 The Online Prize-Collecting Steiner Tree Problem

Define the online prize-collecting Steiner tree problem as follows: we are given a root node r in G, and a penalty of zero for each non-root node. In each time step i, a terminal $s_i \neq r$ arrives with a new penalty $\pi_i > 0$. We have a choice to either connect s_i to root r or pay a penalty π_i for not connecting it (for time step i and each future time step); in the latter case, we mark the terminal. Our goal is to find a set of edges F that minimizes the sum of edge costs in F plus sum of penalties for all marked terminals.

The integer programming formulation of (IP_i) is

$$(IP_i) \qquad \begin{array}{lll} \text{Min} \sum_{e \in E} c_e x_e + \sum_{s_l \in R_i} \pi_l z_l & & \\ \sum_{e \in \delta(S)} x_e + z_l \geq 1, & & \forall S \subseteq V - \{r\}, s_l \in S \cap R_i, \\ x_e \in \{0,1\}, & & \forall e \in E, \\ z_l \in \{0,1\}, & & \forall s_l \in R_i. \end{array}$$

Let (LP_i) denote the corresponding linear programming relaxation in which the constraints $x_e \in \{0,1\}$ and $z_l \in \{0,1\}$ are replaced with $x_e \geq 0$ and $z_l \geq 0$. The dual of this linear program, (D_i), is

$$(D_i) \qquad \begin{array}{lll} \text{Max} \sum_{S \subseteq V - \{r\}} y_S & & \\ \sum_{S : e \in \delta(S)} y_S \leq c_e, & & \forall e \in E, \\ \sum_{S \subseteq U} y_S \leq \sum_{s_l \in U \cap R_i} \pi_l, & & \forall U \subset V, r \notin U, \\ y_S \geq 0, & & \forall S \subseteq V - \{r\}. \end{array}$$

For the each dual problem (D_i), call the constraints of type $\sum_{S:e\in\delta(S)} y_S \leq c_e$ the edge cost constraints and call the constraints of type $\sum_{S\subseteq U} y_S \leq \sum_{s_l \in U \cap R_i} \pi_l$ the penalty constraints. A penalty constraint corresponding to a set U^j is *tight* in level j if the left-hand side of the inequality is equal to the right-hand side.

A terminal is *active* during time step i if it is *unmarked* and it is not yet connected to root r in current solution F. A terminal is *marked* by our algorithm if we decide to pay its penalty. A moat S^j is *active* during time step i, if it is on the lowest level j that satisfies the following three conditions: (i) S^j contains some active terminal $s \in A$; (ii) y^j_S has not yet reached its limit in level j; (iii) S^j does not contain root r. We denote the current set of active moats by $\mathcal{M}$.

The rest of the definitions follow as in the main algorithm.

We extend our main algorithm to give an $O(\log |R_i|)$-competitive algorithm for the prize-collecting Steiner tree problem. For each time step i, a new terminal s_i with penalty π_i arrives. With the modified definitions of active terminals and actives moats, we follow the same lines of main algorithm to grow dual variables, with the same conditions (1)-(3) in that algorithm, but additionally: (4) When a path p connecting a terminal s to root r becomes tight in level j, we buy the path if s and r are not yet connected in F and update active terminal set A; (5) when a penalty constraint corresponding to a set U^j becomes tight in level j, in which case we mark all terminals in U^j to pay their penalties and update active terminal set A. We let a set Q be all the vertices marked by our algorithm in current time step and in previous time steps. At the end of time step i, our algorithm outputs F and the set of terminals Q marked to pay penalties.

Theorem 2. *Our extended algorithm gives an* $O(\log |R_i|)$*-competitive algorithm for the online prize-collecting Steiner tree problem* (IP_i)*.*

Proof. To bound total edge costs and penalties, we need to bound the cost of edges built by conditions (3), (4) and penalties paid by condition (5).

Consider edges in F. Let P be the set of paths we built in F by condition (4) of the extended algorithm, i.e. paths that connect a component in $F-P$ to root r. Then $F-P$ is the set of edges built by condition (3) of the extended algorithm. By invariant 2 of Lemma 3, we have $\sum_{e\in F-P} c_e = \sum_j \sum_S y_S^j - \sum_{X\in F-P} \text{Account}(X)$. By condition (4) of the algorithm, for each path p that connects a terminal s to root r, there must be a moat S^j such that $\sum_{e\in p} c_e \le \sum_{S'\subseteq S^j : s\in S'} y_{S'}^j$. Let X_p be the component in $F-P$ that connected to root by p. All $S' \subseteq S^j$ with $s\in S'$ must credit Account(X_p). Also, j must be equal to Class(X_p) since every terminal in X_p is connected to root r after building p in F; after X_p connects to r, our algorithm does not grow any dual that contains a terminal in X_p. By definition of TopGrowth, we know $\sum_{e\in p} c_e \le$ TopGrowth(X_p). Therefore, we have $\sum_{e\in P} c_e \le \sum_{X_p : p\in P} \text{TopGrowth}(X_p)$ so that

$$\begin{aligned}\sum_{e\in F} c_e &= \sum_{e\in F-P} c_e + \sum_{e\in P} c_e\\ &\le \sum_j \sum_S y_S^j - \sum_{X\in F-P} \text{Account}(X) + \sum_{X_p : p\in P} \text{TopGrowth}(X_p)\\ &\le \sum_j \sum_S y_S^j - \sum_{X\in F-P : X = X_p, p\in P} (\text{Account}(X_p) - \text{TopGrowth}(X_p))\\ &\qquad - \sum_{X\in F-P : X\ne X_p \forall p\in P} \text{Account}(X)\\ &\le (\log |R_i| + 2) \sum_S y_S^{max}.\end{aligned}$$

The last inequality follows by same argument as Lemma 4 since for the component X^* in $F-P$ with highest class, whether it has a path that connects it to root or not, Account(X^*) − TopGrowth$(X^*) \ge 2^{\text{Class}(X^*)}$ by invariant 1 of Lemma 3.

Next, we use a new copy of dual variables to bound penalties of terminals in Q, i.e. $\sum_{s_l\in Q} \pi_l$. For each dual solution y^j, let $\mathcal{S}^j$ be the set of moats in $\bar{F}^j$ that correspond to a tight penalty constraint. It must the case that $\sum_{s_l\in S^j\cap Q} \pi_l = \sum_{S'\subseteq S^j} y_{S'}^j$ for any $S^j \in \mathcal{S}^j$ by condition (5) of the algorithm. By construction, moats in $\mathcal{S}^j$ cannot overlap, so each dual variable is charged to pay a penalty at most once. Also, since we keep growing same set of dual variables in all time steps, our algorithm continues to pay penalties corresponding to tight penalty constraints over all time steps. To bound the total penalty, we know that a terminal can be marked to pay a penalty only by condition (5), so that

$$\sum_{s_l\in Q} \pi_l \le \sum_j \sum_{S^j\in\mathcal{S}^j} \sum_{S'\subseteq S^j} y_{S'}^j \le \sum_j \sum_S y_S^j.$$

Let X^* be a component in F of highest class and let $m = \text{Class}(X^*)$. Consider $\sum_j \sum_S y_S^j \le 2\sum_j \sum_S y_S^j - \sum_j \sum_S y_S^j$, by invariant 1 of Lemma 3, we know $\sum_j \sum_S y_S^j \ge \text{Account}(X^*) \ge 2^m$. By similar technique in Lemma 4, let $\lambda = \lceil \log_2 |R_i| \rceil$, we know $\sum_S y_S^{m-\lambda-k-2} \le 2^{m-k-2}$. Consider all dual vectors of the form $y^{m-\lambda-k-2}$ with $k \ge 0$, we have $2\sum_{k\ge 0} \sum_S y_S^{m-\lambda-k-2} \le 2^m$. Then, for dual solutions $y^{m-\lambda-1}, \ldots, y^{m}$, we have

$$2 \sum_{k=0}^{k=\lambda+1} \sum_{S} y_S^{m-\lambda-1+k} \leq 2(\log |R_i| + 3) \sum_{S} y_S^{max}.$$

Therefore,

$$\begin{aligned}\textstyle\sum_{e \in F} c_e + \sum_{s_l \in Q} \pi_l &\leq [(\log |R_i| + 2) + 2(\log |R_i| + 3)] \textstyle\sum_S y_S^{max} \\ &\leq O(\log |R_i|) OPT_i.\end{aligned}$$

□

5 Conclusion

In the online survivable network design problem, we are given as input an undirected graph and nonnegative edge costs, and in the ith time step, a pair of terminals (s_i,t_i) arrives with a connectivity requirement r_i. One must then augment the current solution so that there are at least r_i edge-disjoint paths between s_i and t_i. It is an interesting open question whether primal-dual algorithms for the offline survivable network design problem (such as those in [8, 5]) can be adapted to the online case as we did here. If $r_{\max} = \max_i r_i$, Gupta, Krishnaswamy, and Ravi [6] have recently given an $O(r_{\max} \log^3 n)$-competitive algorithm for the online survivable network design problem, so such an adaptation might be possible.

References

1. Agrawal, A., Klein, P., Ravi, R.: When trees collide: An approximation algorithm for the generalized Steiner problem on networks. SIAM Journal on Computing 24, 440–456 (1995)
2. Awerbuch, B., Azar, Y., Bartal, Y.: On-line generalized Steiner problem. Theoretical Computer Science 324, 313–324 (2004)
3. Berman, P., Coulston, C.: On-line algorithms for Steiner tree problems. In: Proceedings of the 29th Annual ACM Symposium on the Theory of Computing. pp. 344–353 (1997)
4. Goemans, M.X., Williamson, D.P.: A general approximation technique for constrained forest problems. SIAM Journal on Computing 24, 296–317 (1995)
5. Goemans, M., Goldberg, A., Plotkin, S., Shmoys, D., Tardos, E., Williamson, D.: Improved approximation algorithms for network design problems. In: Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 223–232 (1994)
6. Gupta, A., Krishnaswamy, R., Ravi, R.: Online and stochastic survivable network design. In: Proceedings of the 41st Annual ACM Symposium on the Theory of Computing. pp. 685–694 (2009)
7. Imase, M., Waxman, B.M.: Dynamic Steiner tree problem. SIAM Journal on Discrete Mathematics 4, 369–384 (1991)
8. Williamson, D.P., Goemans, M.X., Mihail, M., Vazirani, V.V.: A primal-dual approximation algorithm for generalized Steiner network problems. Combinatorica 15, 435–454 (1995)