

# On the Competitiveness of the Online Asymmetric and Euclidean Steiner Tree Problems

Spyros Angelopoulos

Max-Planck-Institut für Informatik

**Abstract.** This paper addresses the competitiveness of online algorithms for two Steiner Tree problems. In the online setting, requests for  $k$  terminals appear sequentially, and the algorithm must maintain a feasible, incremental solution at all times. In the first problem, the underlying graph is directed and has bounded asymmetry, namely the maximum weight of antiparallel links in the graph does not exceed a parameter  $\alpha$ . Previous work on this problem has left a gap on the competitive ratio which is as large as logarithmic in  $k$  in worst case. We present a refined analysis, both in terms of the upper and the lower bounds, that closes the gap and shows that a greedy algorithm is optimal for all values of the parameter  $\alpha$ .

The second part of the paper addresses the Euclidean Steiner tree problem on the plane. Alon and Azar [SoCG 1992, Disc. Comp. Geom. 1993] gave an elegant lower bound on the competitive ratio of any deterministic algorithm equal to  $\Omega(\log k / \log \log k)$ ; however, the best (and only) upper bound known so far is the trivial bound  $O(\log k)$ . We give the first analysis that makes progress towards closing this long-standing gap. In particular, we present an online algorithm with competitive ratio  $O(\log k / \log \log k)$ , provided that the optimal offline Steiner tree belongs in a class of trees with relatively simple structure. This class comprises (among others) not only the adversarial instances of Alon and Azar, but also all rectilinear Steiner trees which can be decomposed in a polylogarithmic (in  $k$ ) number of rectilinear full Steiner trees. Interestingly, our analysis is based on techniques developed for the online asymmetric Steiner tree problem.

## 1 Introduction

*Problem statements and motivation* The *Steiner tree* problem occupies a central place in combinatorial optimization, and has been studied extensively under many variants. The standard setting involves an underlying graph  $G = (V, E)$ , with a non-negative weight function  $c : E \rightarrow \mathbb{R}^+$  over its edges (which reflects the *cost* of the edge). Given a subset  $K \subseteq V$  of vertices also called *terminals*, the objective is to find a tree of minimum cost that spans all vertices in  $K$ . Here, the cost of the tree is defined as the sum of the cost of all the edges in the tree.

In this paper we focus on the following Steiner tree problems.

- In the *asymmetric* Steiner tree problem, the underlying graph is directed, and a specific vertex  $r \in V$  is designated as the *root*. We define the *asymmetry*  $\alpha$  of graph  $G$  as the maximum ratio of the cost of antiparallel links in  $G$ . More formally, let  $A$  denote the set of pairs of vertices in  $V$  such that if the pair  $u, v$  is in  $A$ , then either  $(v, u) \in E$  or  $(u, v) \in E$  (i.e, there is an edge from  $u$  to  $v$  or an edge from  $v$  to  $u$  or both). Then the edge asymmetry is defined as

$$\alpha = \max_{\{v,u\} \in A} \frac{c(v,u)}{c(u,v)}$$

Given a set  $K \subseteq V$  of terminals, we seek the minimum cost *arborescence* rooted at  $r$  that spans all vertices in  $K$ . In addition, our aim is to express the performance of the algorithm in terms of the parameters  $k$  and  $\alpha$ .

- In the *Euclidean* Steiner tree problem on the plane, there is no underlying graph. Instead, the input consists of  $k$  points (terminals) on the plane and the objective is to construct a connected graph that spans all terminals so as to minimize the total length (based on Euclidean distances). The case in which the distance between points of the plane is given by the rectilinear metric is known as the *Rectilinear Steiner tree* problem. Here, the tree consists only of horizontal and vertical segments. Clearly, any solution to the Euclidean Steiner tree problem can be translated to a solution to the rectilinear Steiner tree problem, at the expense of a constant-factor increase of the solution cost.

Steiner tree problems are often used in formulating multicast routing over computer networks, in that information must be disseminated from a designated source to all members of a subscribing group (namely the set of terminals  $K$ ). The asymmetric Steiner tree problem is motivated by the observation that a directed graph is a more appropriate and realistic representation of a real network. A typical communication network consists of links asymmetric in the quality of service they provide. In [10], studies on the traffic of network backbones reveal marked asymmetry in parameters such as speed, link utilization and reliability. This situation becomes even more prevalent in a wireless network, due to differences in noise levels, power of transmission, and mobility levels of its endpoints.

Such observations led Ramanathan [13] to define the concept of graph asymmetry as a measure of network homogeneity. According to this measure, undirected graphs are the class of graphs of asymmetry  $\alpha = 1$ , whereas directed graphs in which there is at least one pair of vertices  $v, u$  such that  $(v, u) \in E$ , but  $(u, v) \notin E$  are graphs with unbounded asymmetry ( $\alpha = \infty$ ). Between these extremes, graphs of relatively small asymmetry model networks which are relatively homogeneous in terms of the quality/characteristics of antiparallel links.

On the other hand, the Euclidean Steiner tree problem can be used in modeling problems where a number of facilities is given (on a certain planar region), and we want to guarantee connectivity among any given pair of these facilities (e.g., by building a network of roads, or by deploying a communications network). The objective translates into a solution that is as cheap as possible. The problem has also connections to insertion strategies for TSP (see e.g. [6]).

In this work we address the above problems from the point of view of *online* algorithms, in that the set  $K$  of terminals is not known in advance, but rather is revealed as a sequence of requests. Upon a new request for terminal  $t$ , the algorithm must guarantee a directed path from the root to  $t$ , in the case of the asymmetric Steiner tree problem, or simply that  $t$  is connected to the current solution, in the case of the Euclidean Steiner tree problem. For the former, the graph  $G$  is assumed to be known to the algorithm. Paths are bought irrevocably when serving the requests. In terms of performance analysis, we apply the standard framework of competitive analysis. More precisely, the competitive ratio of an algorithm is defined as the supremum (over all request sequences and graphs) of the ratio of the cost of the solution produced by the algorithm over the optimal off-line cost assuming complete knowledge of the request set  $K$ .

*Related Work* We review some important results that pertain to online Steiner tree problems. For graphs of either constant or unbounded asymmetry, the competitive ratio is tight. For the former class, Imase and Waxman [12] showed that a simple greedy algorithm is optimal and achieves competitive ratio  $\Theta(\log k)$ . Berman and Coulston [8] extended the result to the Generalized Steiner Problem by providing a more sophisticated algorithm. The performance of the greedy algorithm for online Steiner Trees and its generalizations has also been studied by Awerbuch *et al.* [4] and Westbrook and Yan [15]. In a different work, Westbrook and Yan [14] showed that in directed graphs (of unbounded asymmetry), the competitive ratio of any algorithm can be as bad as  $\Theta(k)$ .

The first study of the online asymmetric Steiner tree problem is due to Faloutsos *et al.* [11] who showed that a simple greedy algorithm (to which we refer to as GREEDY) has competitive ratio  $O(\min\{\alpha \log k, k\})$ . The algorithm works by connecting each requested terminal  $u$  to the current arborescence by buying the edges in a least-cost directed path from the current arborescence to  $u$ . On the negative side, they showed a lower bound of  $\Omega\left(\min\left\{\frac{\alpha \log k}{\log \alpha}, k\right\}\right)$  on the competitive ratio of every deterministic algorithm. A better analysis of GREEDY [2] provides an improved upper bound on the competitiveness of GREEDY, namely  $O\left(\min\left\{\alpha \frac{\log k}{\log \log \alpha}, k\right\}\right)$ . The same work showed a corresponding lower bound of  $\Omega\left(\min\left\{\frac{\alpha \log k}{\log \log k}, k^{1-\epsilon}\right\}\right)$  on the competitiveness of any deterministic algorithm (and for every constant  $0 < \epsilon < 1$ ). In recent work [3], a more careful analysis proves that GREEDY has a competitive ratio equal to  $O\left(\min\left\{\max\left\{\alpha \frac{\log k}{\log \alpha}, \alpha \frac{\log k}{\log \log k}\right\}, k\right\}\right)$ . The result almost matches the lower bound of  $\Omega\left(\min\left\{\max\left\{\alpha \frac{\log k}{\log \alpha}, \alpha \frac{\log k}{\log \log k}\right\}, k^{1-\epsilon}\right\}\right)$  (where  $\epsilon$  is any arbitrarily small constant) due to [11] and [2].

It is important to note that when  $\alpha \in \Omega(k)$  the lower bound on the competitive ratio due to [11] is  $\Omega(k)$ , which is obviously tight (using the trivial upper bound of  $O(k)$  for GREEDY). Thus the problem is interesting only when  $\alpha \in o(k)$ .

Even though the bound of [3] is almost-tight, from a worst-case perspective a gap still remains. Indeed, [3] is tight when either  $\alpha \in O(k^{1-\epsilon})$  (for some constant

$\epsilon \in (0,1)$ ), or  $\alpha \in \Omega(k)$ , that is, for a broad range of values of asymmetry. However, when the asymmetry is such that  $\alpha \in O(k)$ , and  $\alpha \neq O(k^{1-\epsilon})$ , for any positive constant  $\epsilon < 1$  (e.g, when  $\alpha = k/\text{polylog}(k)$ ), the gap between upper and lower bounds can be large, namely logarithmic in  $k$ .

Concerning the online Steiner Tree in the Euclidean plane, Alon and Azar [1] presented an elegant adversarial construction that guarantees a lower bound of  $\Omega(\log k / \log \log k)$ . The only known upper bound is the  $O(\log k)$  bound that applies to any distances induced by undirected graphs (and hence by extension to Euclidean distances). No better upper bounds have been known.

*Contributions of this paper* In the first part of this paper, we give a tight bound for the online asymmetric Steiner tree problem. In particular, we show that when  $\alpha$  is in the range of values for which [3] is not tight, the competitive ratio of GREEDY is  $\Theta\left(\frac{\log(k/\alpha)}{\log \log(k/\alpha)}\right)$ , and the bound is tight for any deterministic algorithm (c.f. Theorem 1 and Theorem 2). This completely resolves the problem of determining the optimal competitive ratio, for all values of asymmetry.

There are several reasons that motivate the close study of this problem. First, as argued earlier, for certain values of the asymmetry, a gap as large as logarithmic (in  $k$ ) remains. From a worst-case point of view the gap is large, even though it occurs for a relatively narrow range of values of  $\alpha$  (recall that  $\Theta(\log k)$  is the competitive ratio in undirected graphs). Moreover, since  $k$  cannot be known in advance, it is not possible to predict whether the algorithm will be optimal without a proof of optimality for all ranges of  $\alpha$ .

More importantly, algorithms for Steiner tree problems are often used as subroutines in solving more complex problems, and in many cases the competitive ratio for the complex problem is a function of the competitive ratio of the Steiner tree algorithm. A representative example is the problem of *file allocation* in general undirected networks, which is a well-studied problem in both theory and applications of distributed systems. The influential work of Bartal *et al.* [7] shows that if there exists a  $c$ -competitive algorithm for the online Steiner tree problem (in undirected graphs), then it is possible to derive a randomized  $(2 + \sqrt{3})c$ -competitive algorithm for file allocation. This yields an optimal randomized algorithm, since it is easy to argue that the online Steiner tree problem can be viewed as a special case of the more general file allocation problem. In subsequent work, Awerbuch *et al.* [5] showed that a similar reduction is possible for deterministic algorithms. Notice that, as with multicasting, file allocation is a problem motivated by distributed networks, and once again, it would only be natural to study it under directed graphs. Therefore, we expect that strict optimality results for Steiner trees will provide a useful tool in resolving other online network optimization problems, in settings which are more realistic in practice.

In the second part of the paper, we address the online Euclidean Steiner tree problem on the plain. Bridging the gap between the known lower and the upper bounds has been an outstanding open question in the area of competitive analysis for more than 15 years. Unfortunately, only a trivial upper bound is known. We make progress by showing that if the underlying optimal tree observes

certain structural properties, then there exists an algorithm that is optimal. In particular, we require that the optimal, off-line tree is “close” (w.r.t. cost) to a rectilinear tree that is union of at most a polylogarithmic number of *basic trees* (c.f. Theorem 3 and Corollary 1). A formal definition of a basic tree is given in Section 3. The definition includes, for instance, any *Full Rectilinear Steiner tree* (or FST for brevity); see, e.g., [9] for a precise definition of a FST.

We need to further motivate our assumption on optimal trees. First, the lower bound of Alon and Azar, although fairly technical, is based on an instance for which the optimal Steiner tree is a single basic tree. Thus our result states that if we aim to improve the lower bound, more complicated constructions will be required. Second, the result provides a connection between the number of basic trees and the performance of an algorithm. This is along the lines of known results on approximation algorithms, which relate the (offline) construction of rectilinear Steiner trees and FST’s (see [17] and [16] for representative examples). Third, the result follows from techniques used in the analysis of the greedy algorithm for the online asymmetric Steiner tree problem (specifically, the emergence of the  $(\log / \log \log)$ -factors in both problems arises from the use of similar techniques). We thus demonstrate some close connection between the two problems, which may lead into further improvements on the competitive ratio of the online Euclidean Steiner tree problem.

## 2 A tight bound for online asymmetric Steiner trees

### 2.1 The lower bound

The main result of this section is the following lower bound:

**Theorem 1.** *The competitive ratio of every deterministic algorithm is  $\Omega\left(\alpha \frac{\log(k/\alpha)}{\log \log(k/\alpha)}\right)$ , assuming that  $\alpha \in o(k)$ .*

For the proof of Theorem 1 we will construct an adversarial input, namely a graph  $G$  and an appropriate sequence  $\sigma$  of terminal requests. The graph  $G$  will be defined using  $\widehat{G}$  as a building block, where  $\widehat{G}$  is a parameterized version of the adversarial graph used in the construction of the lower bound in [2].

**Construction of the adversarial graph  $G$ .** We will begin by defining some auxiliary constructions. Let  $v, u$  be two vertices in a graph, and let the directed edge  $(v, u)$  have cost  $c$ , whereas the antiparallel edge  $\bar{e} = (u, v)$  has cost  $\alpha c$ , where  $\alpha$  is the asymmetry of the graph in which  $u, v$  belong. We say that we *insert a vertex  $w$  at height  $h$  in  $e$*  if we introduce a new vertex  $w$  and replace  $e, \bar{e}$  with new edges of costs  $c(v, w) = c - h$ ,  $c(w, u) = h$ ,  $c(u, w) = \alpha h$ , and  $c(w, v) = \alpha(c - h)$  (for the sake of visualisation, we should think of  $v$  as located higher than  $u$ ). Note that the insertion maintains the asymmetry of the graph.

Second, let  $T_1 = \{v_1, \dots, v_l\}$  and  $T_2 = \{v'_1, \dots, v'_l\}$  denote two disjoint sets of  $l$  vertices each (again, we should think of vertices in  $T_1$  as being located “higher” than vertices in  $T_2$ ). In addition, we require that  $T_1$  and  $T_2$  have the

property that all edges of the form  $e_i = (v_i, v'_i)$  have the same cost, say  $c$ , and all antiparallel edges have cost  $\alpha c$ . Informally, the “downwards” direction is cheap, whereas the “upwards” direction is expensive, and this is a property that will be maintained throughout the construction. Let  $E$  denote the set  $\{e_i : i \in [1, l]\}$ . We call index  $i \in [1, l]$  the  $i$ -th column, and require that  $l$  is a power of 2, and that it is large compared to  $k$  (e.g., at least  $2^k$ ). On the collection of columns  $E$  we will define a construction called *block* which we denote by  $B(E, m, h)$ . Here  $m$  and  $h$  are parameters used in the construction, with  $h \leq c$  and  $m < k$ , and  $E$  is the set of columns induced by  $T_1$  and  $T_2$ .

In a nutshell,  $B(E, m, h)$  is built by inserting appropriate vertices (and edges) in  $E$ , in a layered fashion. There are  $m$  layers inserted in total. Layer  $i$  consists of  $l$  vertices inserted, one for each column, at height equal to  $\Theta(h/m)$ . Let  $w^{i,1}, \dots, w^{i,l}$  denote these  $l$  vertices. We group the vertices of layer  $i$  into appropriate disjoint groups of the same size, denoted by  $S^{i,1}, \dots, S^{i,s_i}$ , which we call  $s$ -sets. Also, for each group of layer  $i$ , say  $S^{i,j}$ , we add a vertex  $u^{i,j}$ , as well as edges from every  $w$ -vertex in  $S^{i,j}$  to  $u^{i,j}$ , of cost  $c^{i,j}$ , whereas the antiparallel edges from  $(u^{i,j}$  to every  $w$ -vertex in  $S^{i,j}$ ) have cost  $\alpha c^{i,j}$ . The partition of vertices of each layer into  $s$ -sets, as well as the precise values of  $c^{i,j}$  are functions of the parameters  $m, c, h$  and  $\alpha$ . Figure 1 (Appendix A.1) gives an example of a block. The precise construction of a block is fairly complicated (see Appendix A.2); we will focus mostly on important properties.

We say that an  $s$ -set *crosses* a certain set of columns if and only if the set of columns in which the vertices of  $S$  lie intersects the set of columns in question. Two  $s$ -sets cross each other iff the intersection of the sets of columns crossed by each one is non-empty. Note that there is a 1-1 correspondence between the sets  $S^{i,j}$  and vertices  $u^{i,j}$ , which means that several properties/definitions pertaining to  $s$ -sets carry over to the corresponding  $u$  vertices (e.g., we will say that two  $u$  vertices cross if their  $s$ -sets cross).

Let  $T_1 = \{v_1, \dots, v_l\}$  and  $T_2 = \{v'_1, \dots, v'_l\}$  denote two sets of  $l$  vertices each (here  $l$  has to be large compared to  $k$ , although it suffices to set  $l > 2^k$ ), with  $c(v_i, u_i) = 1$  and  $c(u_i, v_i) = \alpha$ . Let  $E$  denote the set of columns induced by  $T_1$  and  $T_2$ . There is also a root  $r$  and edges from  $r$  to each  $v_i$  of infinitesimally small cost, and the same holds for the antiparallel edges from  $v_i$  to  $r$ ). In [2], graph  $\widehat{G}$  then is derived by adding a single block  $B(E, k, 1)$ , where  $k$  is the total number of terminals requested by the adversary (see also Appendix A.2 for details).

We proceed with the description of the adversarial graph  $G$ . At a high level, the construction is based on *phases* of insertions of appropriately defined blocks, unlike  $\widehat{G}$  which consists of a single block. We begin with  $T_1, T_2, E$  and  $r$  defined as above. Given the set of columns  $E$ , in the first phase we add the block  $B_{1,1} = (E, m, m/k)$ , where  $m \leq k$  is a function of  $k$  and  $\alpha$  that will be specified later. Inductively, let  $B_{i,1}, B_{i,2}, \dots$  denote the blocks added during the  $i$ -th phase in the construction of  $G$ . Consider the highest layer added so far, namely the highest layer added at phase  $i$ , and let  $\{S_{i,1}, S_{i,2}, \dots\}$  denote the collection of  $s$ -sets at this layer. For a set  $S_{i,j}$ , in the above collection of highest  $s$ -sets, let  $T_{i,j} \subset T_1$  denote the set of vertices in  $T_1$  of the same column index as vertices in  $S_{i,j}$ . The

pairs  $(T_{i,j}, S_{i,j})$  induce a partition of the columns of  $E$  into disjoint subsets of columns (of equal size). More precisely, let  $E_{i,j}$  denote the set of edges  $(v_p, w)$ , with  $p \in [1, l]$  such that  $w \in S_{i,l}$ . Then, phase  $i + 1$  consists of adding the blocks  $B_{(i+1),j} = B(E_{i,j}, m, m/k)$ , for all  $j$  for which the column sets  $E_{i,j}$  are defined.

We want to ensure that  $\Theta(k)$  layers are added in total, since the adversarial request sequence on  $G$  will request one  $u$ -vertex per layer. Thus the above process is comprised of  $\Theta(k/m)$  phases. An illustration is given in Appendix A.1.

**The algorithm–adversary game.** We will describe how to construct an adversarial sequence of requests  $\sigma$  in  $G$ . We begin by describing, in high-level, the algorithm/adversary game in  $\widehat{G}$ , and the corresponding request sequence  $\widehat{\sigma}$ . Later on, we will build upon this game in order to define  $\sigma$ . A detailed definition of  $\widehat{\sigma}$  is given in Appendix A.3.

• **The adversarial game on  $\widehat{G}$ , and the sequence  $\widehat{\sigma}$**

Consider  $\widehat{G}$  with  $k$  levels in total, and let  $x$  be the solution of  $x^x = k$ , hence  $x = \Theta(\log k / \log \log k)$ . The adversarial request sequence  $\widehat{\sigma}$  consists of  $u$ -vertices exclusively, and is comprised of  $x$  rounds: In particular, in round  $i \leq x$ ,  $\Theta((x + 1)^i)$  terminals are requested.

Every time a new terminal, say  $u$ , is requested in  $\widehat{\sigma}$ , the online algorithm must guarantee the existence of a directed path from the root to  $u$ , possibly buying new edges. Among the many such paths the algorithm may establish, the adversary will fix one such path, to which we refer as the *connection path for  $u$* , denoted by  $p(u)$ . For the lower-bound analysis, we charge parts (edges) of a connection path, then so long as we guarantee that each edge is charged at most once, the total cost of all charged edges is a lower bound on the algorithm’s cost. We need to summarize some properties of the sequence  $\widehat{\sigma}$ , and how the edges of the connection paths are charged.

As argued in [2], for every requested terminal  $u$ , the connection path  $p(u)$  can be of one of the following two forms:

- Connection *from  $r$* : A connection path which originates from  $r$ .
- Connection path *from above/below*. A connection path which originates from a previously requested vertex that lies higher (resp. lower) than the requested vertex  $u$ .

A main design aim in the construction of  $\widehat{G}$  and  $\widehat{\sigma}$  is the presence of consecutive pairs of requests in a *parent/child* relation. The precise definition is as follows: Consider the three  $s$ -sets  $S, S_l, S_r$  in  $\widehat{G}$  with the property that  $S$  crosses columns  $i, \dots, j$  (with  $j - i + 1$  even number) whereas  $S_l$  and  $S_r$  cross columns  $i, \dots, (i + j - 1)/2$  and  $(i + j + 1)/2 + 1, \dots, j$ , respectively. In addition,  $S_l$  and  $S_r$  are at the same layer in  $\widehat{G}$  and both higher than  $S$ . We call  $S_l$  and  $S_r$  the *left* and *right* children of  $S$ , respectively, and  $S$  their *parent*. The definition extends to the corresponding  $u$ -vertices of these  $s$ -sets in the natural way: two  $u$ -vertices are in parent/child relation if their corresponding  $s$ -sets are in such relation.

The sequence  $\widehat{\sigma}$  is defined in such a way that almost all consecutive pairs of requested terminals in  $\widehat{\sigma}$  are in parent/child relation. For such terminals, the

adversary takes advantage of the structure of  $\widehat{G}$ : Suppose that  $u$  is the last requested terminal and that the children of  $u$  (say  $u_l, u_r$ ) are present in  $\widehat{G}$ . Then it is easy to argue that no matter what the choice of the connection path  $p(u)$ , the path will cross<sup>1</sup> exactly one of  $u_l, u_r$ : if it crosses  $u_l$ , then the next terminal to be requested is  $u_r$ , and vice versa.

We summarize the important properties concerning  $\widehat{\sigma}$ . We say that a terminal  $u$  requested in the  $i$ -th round of the game is *typical*, if the next requested terminal in  $\widehat{\sigma}$  is a child of  $u$  in  $\widehat{G}$ . It turns out that there are  $\Theta((x+1)^i)$  typical terminals in round  $i$  (recall that round  $i$  consists of  $\Theta((x+1)^i)$  terminals), thus almost all terminals in a round are typical. We also define the *depth* of  $u$  in  $\widehat{G}$  as the total cost of the directed path from the root  $r$  to the  $s$ -set to which  $u$  corresponds. The following are important properties of  $\widehat{\sigma}$  (for details see Appendix A.4).

*Property 1 (cost property in  $\widehat{\sigma}$ ).* Let  $u$  be a typical request of round  $i$ , which is at depth  $d$  in  $\widehat{G}$ . Then:

- (i) If  $p(u)$  is from the root, then the algorithm is charged cost  $\Omega(d)$ .
- (ii) If  $p(u)$  is from above/below, then the algorithm is charged cost  $\Omega\left(\frac{a}{(x+1)^i}\right)$ .

The following set of properties will be instrumental in deriving the adversarial sequence  $\sigma$  in  $G$ . The first property implies that an offline algorithm suffices to buy edges of a single column (in the “downwards” direction), namely a column that crosses the last terminal in  $\widehat{\sigma}$ , as well as the corresponding directed edges from  $s$ -sets to requested terminals. The second property will be useful in applying iteratively the game in  $\widehat{G}$  to the design of the game in  $G$ , one time per phase.

*Property 2 (structural property in  $\widehat{\sigma}$ ).* Let  $u$  be the current request in  $\widehat{\sigma}$ , then:

- (i) Every column that crosses  $u$  crosses all previously requested terminals in  $\widehat{\sigma}$ .
- (ii) No connection path of any previously requested terminal crosses any columns of  $u$ , and no edges in columns that cross  $u$  are charged by the adversary (with the exception of  $u$  itself).

### • The adversarial game on $G$ , and the sequence $\sigma$

We now show how to use  $\widehat{G}$  and  $\widehat{\sigma}$  in order to construct an adversarial sequence  $\sigma$  for  $G$ . At a high level, the game between the algorithm and the adversary in  $G$  proceeds in  $\Theta(k/m)$  *phases*: the  $i$ -th phase requests  $u$ -vertices which belong in a block in  $G$  that was added in the  $i$ -th phase of its construction. Within this specific block, terminals are requested in a manner similar to requests in  $\widehat{\sigma}$ : this is because  $\widehat{G}$  essentially consists of a single block.

We describe how  $\sigma$  is derived, as well as the actions of the adversary. In the first phase, the adversary requests a total of  $m$   $u$ -vertices in the lowest block, namely  $B_{1,1}$ . This is essentially identical to the game played in  $\widehat{G}$ , i.e., vertices

<sup>1</sup> We say that the connection path for a terminal crosses an  $s$ -set (or its corresponding  $u$ -vertex) if it contains (vertical) edges in a column that is crossed by the  $s$ -set.

are requested in rounds. The only difference lies in the cost charged for the connection paths (informally, in  $G$ , the vertices of  $B_{1,1}$  are in a higher depth than vertices in  $\widehat{G}$ ). However this does not affect the decisions of the adversary, since we can still classify connection paths (from the root, or from above/below), and we can still classify terminals as typical, in the same manner as in  $\widehat{\sigma}$ . In particular, the last requested terminal of the first phase satisfies Property 2.

Inductively, suppose that the adversary has requested terminals in phase  $i$ , and suppose that every terminal requested in phases up to and including phase  $i$  satisfies Property 2. We will describe the requests of phase  $i+1$ . Let  $u$  denote the last terminal requested in the  $i$ -th phase, and  $S$  its corresponding  $s$ -set. From construction of  $G$ , there is a unique block, say  $B_{i+1,j}$ , for some index  $j$ , which was added in iteration  $i+1$  and which is defined over the set of columns which are crossed by  $S$ . From the structural property, and in particular, Property 2(ii), right before  $u$  is requested no columns crossed by  $u$  are charged by the adversary. This implies that, once again, the adversary can play the same game in  $B_{i+1,j}$  as the game played in  $\widehat{G}$ , and charge columns in identical manner, without ever charging an edge more than once. Phase  $i+1$  then is comprised by the  $u$ -vertices requested *in rounds* within block  $B_{i+1,j}$ . It also follows from the construction of  $G$ , as well as the induction hypothesis, that any terminal requested during this phase satisfies the structural property. The sequence  $\sigma$  is the sequence derived by combining the requests of all phases: there are  $\Theta(k/m)$  phases in total, and  $\Theta(m)$  requests per phase, hence there are  $\Theta(k)$  requests in  $\sigma$  in total.

Because of the correspondence between phases and blocks, we say that a terminal requested in  $\sigma$  is typical if it is typical in the corresponding block in which it belongs. Based on this, we derive the following property concerning the cost of typical terminals in  $\sigma$ . Here,  $x$  is set to be  $\Theta(\log m / \log \log m)$ .

*Property 3.* Let  $u$  be a typical request of phase  $j$  and round  $i$ , at depth  $d$  in  $G$ .

- (i) If  $p(u)$  is from the root, then the algorithm is charged cost  $\Omega(d)$ .
- (ii) If  $p(u)$  is from above/below, then the algorithm is charged cost  $\Omega\left(\frac{\alpha}{(x+1)^i} \frac{m}{k}\right)$ .

Note that the cost scale-down in Property 3(ii) (compared to Property 1(ii)) is due to the fact that a phase in the game in  $G$  takes place in a block whose columns are a factor  $(k/m)$  times cheaper than the columns of  $\widehat{G}$ . Also note that  $x$  is a log / log log function of  $m$ , not  $k$ , since each block contains  $m$  layers.

## 2.2 Analysis

For the purpose of the analysis, we need the following lemma:

**Lemma 1.** *For the sequence of requests  $\sigma$  on graph  $G$ , the following hold:*

- The cost of the optimal offline algorithm is bounded by a constant.
- The cost of any deterministic algorithm is  $\Omega\left(\min\left\{\frac{k}{m}, \alpha \cdot \frac{\log m}{\log \log m}\right\}\right)$

Once we prove Lemma 1, then Theorem 1 follows by selecting a value  $m$ , so as to minimize the cost expression. In particular, we choose  $m = \Theta\left(\frac{k \log \log(k/\alpha)}{\alpha \log(k/\alpha)}\right)$ . Substituting this value in the cost expression, it is easy to see that both  $k/m$  and  $\alpha \cdot \frac{\log m}{\log \log m}$  are in  $\Omega\left(\alpha \cdot \frac{\log(k/\alpha)}{\log \log(k/\alpha)}\right)$ , and the theorem is proved.

*Proof sketch of Lemma 1.* The upper bound on the cost of the optimal algorithm follows from the fact that there exists a column that crosses all terminals in  $\sigma$ : an offline algorithm will buy all downwards edges in this column, as well as edges from the  $s$ -sets crossed by the column in question to all  $u$ -vertices in  $\sigma$ . The cost of the latter edges is such that their total contribution is small.

For the lower bound on the cost of any algorithm, the rough idea is that we classify each phase into one of two possible groups, depending on whether the algorithm pays its cost mostly due to rule (i) or rule (ii) of Property 3. We can prove that the first group contributes a total cost of  $\Omega(k/m)$ , whereas the second group contributes  $\Omega(\alpha \log m / \log \log m)$ . The overall cost will be at least the minimum of these two expressions. See Appendix A.5 for details.  $\square$

### 2.3 The upper bound

**Theorem 2.** *Suppose that  $\alpha$  is such that  $\alpha \in \omega(k^{1-\epsilon})$ , for every constant  $\epsilon \in (0, 1)$ , and also  $\alpha \in o(k)$ . Then for every request sequence  $\sigma$ , the cost paid by GREEDY is  $c_{GR}(\sigma) = O\left(\alpha \cdot \frac{\log(k/\alpha)}{\log \log(k/\alpha)}\right) c(T^*)$ , where  $k = |\sigma|$ , and  $T^*$  is the optimal arborescence for  $\sigma$ .*

*Proof sketch.* Partition  $T^*$  into  $\Theta(\alpha)$  edge-disjoint trees  $T_1, T_2, \dots$  each containing  $\Theta(k/\alpha)$  terminals. Then the total cost for serving the *first* request in each of these subtrees is small. On the other hand, we can show that the cost for serving all remaining terminals within  $T_i$  is  $O\left(\alpha \frac{\log(k/\alpha)}{\log \log(k/\alpha)} \cdot c(T_i)\right)$ . The lemma follows from the edge-disjointness of the  $T_i$ 's. See Appendix A.5 for details.  $\square$

## 3 A non-trivial bound for online Euclidean Steiner tree on the plane

In this section we present an algorithm for the online Steiner tree problem in the Euclidean plane, and prove that achieves better competitive ratio than  $O(\log k)$ , assuming the optimal Steiner tree has certain structural properties. The algorithm is as follows: Suppose that terminal  $u$  is requested, and let  $T_{curr}$  denote the current Steiner tree built so far. The algorithm first finds a path of minimum cost from  $u$  to the current tree, say  $p_u$ , and buys this path. Let  $c(p_u)$  denote the cost of the path in question. In addition, the algorithm buys horizontal and vertical segments of size  $2c(p_u)$  each, such that  $u$  is at the center of each segment<sup>2</sup>. Hence in serving request  $u$ , the algorithm pays a total cost equal to  $5c(p_u)$ .

<sup>2</sup> The resulting graph may not be a tree, however this is not an issue since we only require a connected graph that spans all terminals.

Before proceeding with the main result, we will show that the algorithm is optimal when the underlying optimal Steiner tree has a relatively simple structure, in particular, when it is a *basic* tree with respect to the set of requested terminals. We begin with the definition of a basic tree.

**Definition 1.** Let  $K' = \{u_1, \dots, u_{k'}\}$  denote a set of  $k'$  terminals, and let  $T'$  be a tree that spans  $K'$  on the Euclidean plane. We call the tree  $T'$  basic (with respect to  $K'$ ) if the following conditions are met:  $T'$  consists of a segment  $P$ , which is either horizontal or vertical, and which we call the backbone of  $T'$ , as well as  $k'$  disjoint paths  $t_1, \dots, t_{k'}$ , one for each terminal in  $K'$ , which we call the terminal paths. Here, each terminal path  $t_i$  is such that  $u_i$  is one of the end-points of path  $t_i$ ; the other end-point must be part of the backbone  $P$ .

Figure 6 (Appendix B) illustrates examples of basic trees.

The following is the main technical result, which states that if the optimal Steiner tree for a set of terminals  $K$  is “close” to a basic tree wrt  $K$ , then the online algorithm we proposed achieves the best-possible competitive ratio.

**Theorem 3.** Let  $K$  be a set of  $k$  terminals, and let  $T^*$  denote the optimal Steiner tree for  $K$ . If there exists a basic tree  $T$  such that  $c(T) = O(c(T^*))$ , then the cost of the online algorithm for requests in  $K$  is  $O(\log k / \log \log k) \cdot c(T^*)$ .

The proof of Theorem 3 is based on techniques developed for the analysis of the greedy algorithm for the online asymmetric Steiner tree in [3]. More precisely, the central technical result in [3] is first derived for a class of directed graphs (called *combs*) whose structure is very similar to the structure of a basic tree (see the statement of Theorem 3 in [3]). However, there are some important technical difficulties, in that the analysis in [3] yields a high cost with respect to the backbone of a comb (which would translate to a high cost with respect to the backbone of a basic tree for our algorithm). Instead, we need a refined analysis, that takes into account the geometric nature of the problem.

Specifically, we will rely on the following property (Lemma 2). In informal terms, the lemma states that if a tree  $T'$  is basic wrt a set of terminals  $K'$ , then the online algorithm (on requests drawn from the set  $K'$ ) incurs cost which has only a linear dependency in the backbone cost. That is, if the backbone cost dominates the cost of  $T'$  (in comparison to the total cost of terminal paths), then the cost of the algorithm is linear in the cost of the basic tree  $T'$ .

**Lemma 2 (Appendix B).** Let  $K' = \{u_1, \dots, u_{k'}\}$  be a set of  $k'$  terminals, and let  $T'$  be a basic tree wrt  $K'$ , with backbone  $P$  of cost  $c(P)$ , and  $k'$  terminal paths  $t_1, \dots, t_{k'}$ . Suppose that terminals in  $K'$  arrive online. Then the cost of the algorithm is  $O(c(P) + k' \cdot c_{\max})$ , where  $c_{\max}$  is the cost of the longest path among terminal paths  $t_1, \dots, t_{k'}$ .

A complete proof of Theorem 3 is given in Appendix B.

It is worth pointing out that the construction for the lower bound of Alon and Azar [1] is such that the optimal Steiner tree is a basic rectilinear tree. Note also that any full (rectilinear) Steiner tree (FST) is trivially a basic tree. Motivated

by the decomposition of rectilinear Steiner trees to edge-disjoint FST's , we can also decompose every rectilinear Steiner tree into a collection of edge-disjoint basic trees. We can thus extend our result to all instances whose optimal Steiner tree consists of a small number of basic Steiner trees, in this decomposition.

**Corollary 1 (Appendix B).** *Let  $K$  denote a set of  $k$  terminals, and let  $T^*$  denote the optimal rectilinear Steiner tree for  $K$ . If  $T^*$  can be decomposed into a polylogarithmic (in  $k$ ) number of basic trees, then the cost of the online algorithm for requests in  $K$  is  $O(\log k / \log \log k) \cdot C(T^*)$ .*

## References

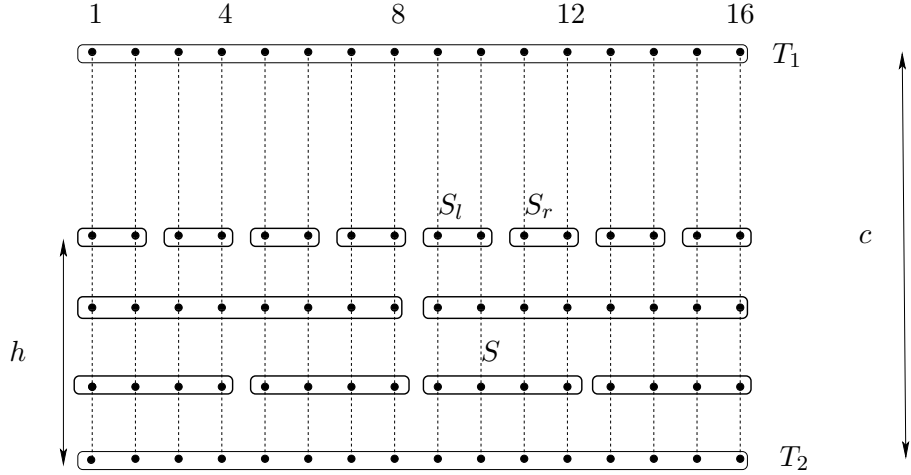
- [1] N. Alon and Y. Azar. On-line Steiner trees in the Euclidean plane. *Discrete and Computational Geometry*, 10:113–121, 1993.
- [2] S. Angelopoulos. Improved bounds for the online Steiner tree problem in graphs of bounded edge-asymmetry. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 248–257, 2007.
- [3] S. Angelopoulos. A near-tight bound for the online Steiner tree problem in graphs of bounded asymmetry. In *Proceedings of the 16th Annual European Symposium on Algorithms*, pages 76–87, 2008.
- [4] B. Awerbuch, Y. Azar, and Y. Bartal. On-line generalized Steiner problem. *Theor. Comp. Sci.*, 324(2–3):313–324, 2004.
- [5] B. Awerbuch, Y. Bartal, and A. Fiat. Competitive distributed file allocation. *Information and Computation*, 185(1):1–40, 2003.
- [6] Y. Azar. Lower bounds for insertion methods for TSP. *Combinatorics, Probability and Computing*, 3:285–292, 1994.
- [7] Y. Bartal, A. Fiat, and Y. Rabani. Competitive algorithms for distributed data management. *Journal of Computer and System Sciences*, 51(3):341–358, 1995.
- [8] P. Berman and C. Coulston. Online algorithms for Steiner tree problems. In *Proceedings of the 29th Symp. on the Theory of Computing*, pages 344–353, 1997.
- [9] M. Bern and D. Eppstein. *Chapter 8: Approximation Algorithms for Geometric Problems. In Approximation Algorithms for NP-hard problems (edited by D. S. Hochbaum)*. PWS Publishing Company, 1997.
- [10] K. Claffy, G. Polyzos, and H.W. Braun. Traffic characteristics of the T1 NSFNET backbone. In *Proceedings of INFOCOM*, pages 885–892, 1993.
- [11] M. Faloutsos, R.Pankaj, and K. C. Sevcik. The effect of asymmetry on the on-line multicast routing problem. *Int. J. Found. Comput. Sci.*, 13(6):889–910, 2002.
- [12] M. Imase and B. Waxman. The dynamic Steiner tree problem. *SIAM Journal on Discrete Mathematics*, 4(3):369–384, 1991.
- [13] S. Ramanathan. Multicast tree generation in networks with asymmetric links. *IEEE/ACM Transactions on Networking*, 4(4):558–568, 1996.
- [14] J. Westbrook and D. C. K. Yan. Linear bounds for on-line Steiner problems. *Information Processing Letters*, 55(2):59–63, 1995.
- [15] J. Westbrook and D. C. K. Yan. The performance of greedy algorithms for the on-line Steiner tree and related problems. *Math. Syst. Thr.*, 28(5):451–468, 1995.
- [16] P. Winter and M. Zachariasen. Euclidean Steiner minimum trees: An improved exact algorithm. *Networks*, 30(3):149–166, 1997.
- [17] M. Zachariasen. Rectilinear full Steiner tree generation. *Networks*, 2(33), 1999.

# Appendix

## A Details for Section 2

### A.1 Examples

The following figure illustrates an example of a block:



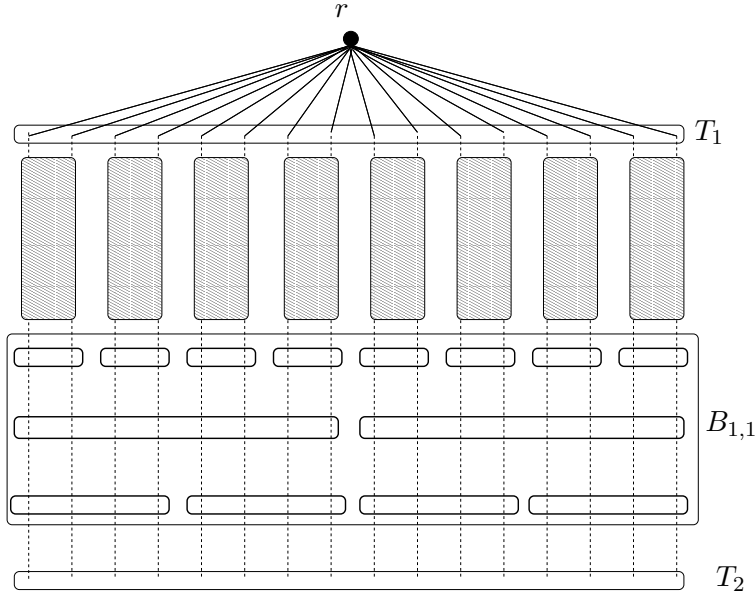
**Fig. 1.** An example of a block construction. Here  $l = 16$  (i.e., there are 16 columns), and the block consists of three layers (the first layer consists of 4  $s$ -sets, the second of 2  $s$ -sets and the third of 8  $s$ -sets. For simplicity, the figure illustrates only  $s$ -sets: in reality, each  $s$ -set is associated with a  $u$ -vertex. Note that according to the definition of Section 2.1 the two  $s$ -sets  $S_l$  and  $S_r$  are both children of the  $s$ -set denoted by  $S$ .

Figure 2 illustrates the example of an adversarial graph  $G$ , for the case in which the construction requires two phases.

### A.2 A formal definition of a *block* construction, and of the graph $\hat{g}$

In this section we present the formal details behind the construction of a block  $B(E, m, h)$ , as well as the construction of the graph  $\hat{G}$ .

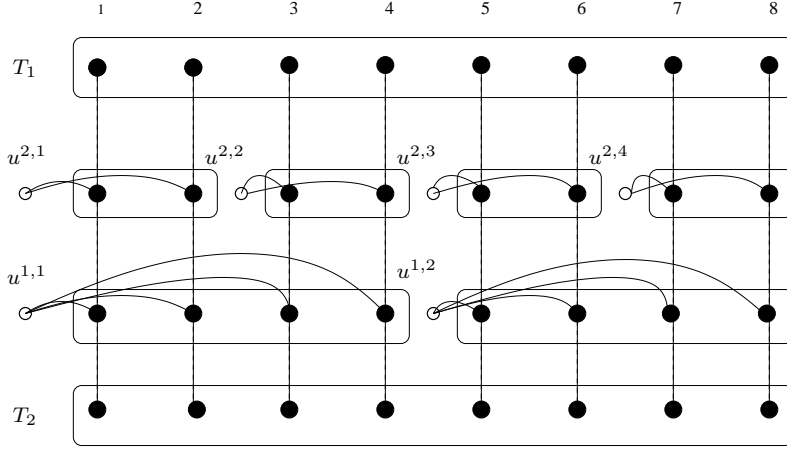
Let  $T_1 = \{v_1, \dots, v_l\}$  and  $T_2 = \{v'_1, \dots, v'_l\}$  be two disjoint sets of  $l$  vertices each (again, we may think of vertices of  $T_1$  as located higher than vertices of  $T_2$ ). We require that  $T_1$  and  $T_2$  have the property that all edges of the form  $e_i = (v_i, v'_i)$  have the same cost, say  $c$ , while all antiparallel edges  $\bar{e}_i$  have cost  $\alpha c$ . Let  $E$  denote the set  $\{e_i | i \in [1, l]\}$ . On the collection of edges  $E$  we define a construction denoted by  $(E, q, g, h)$  which we call *layered component* or simply *component* and which, informally, adds vertices in edges in  $E$  in *layers*. Here,  $q$



**Fig. 2.** An example of the construction of  $G$ , for the case in which the process requires two phases. Here, each phase will insert a block as depicted in Figure 1, as evident in the shape of  $B_{1,1}$ . In particular, the eight shaded regions corresponds to the eight blocks of the form  $B_{2,1} \dots B_{2,8}$ .

and  $g$  are parameters used in the construction (and will be given precise values later), and  $h$  is the argument in  $B(E, m, h)$ . The construction is as follows (see also Figure 3). Layer 1 consists of  $l$  vertices inserted at height  $h/q$ , one for each edge in  $E$ . Call those vertices  $w^{1,1}, \dots, w^{1,l}$ . We group these vertices into  $2^1 = 2$  groups, namely  $S^{1,1} = \{w^{1,1}, \dots, w^{1,l/2}\}$  and  $S^{1,2} = \{w^{1,l/2+1}, \dots, w^{1,l}\}$ . We also add two new vertices,  $u^{1,1}$  and  $u^{1,2}$ , such that for every vertex  $w \in S^{1,i}$ , with  $i \in \{1, 2\}$ , there is an edge of cost  $g$  from  $w^{1,i}$  to  $u^{1,i}$  and an edge of cost  $\alpha g$  from  $u^{1,i}$  to  $w^{1,i}$ . Recursively, suppose that layers  $1, 2, \dots, j-1 \leq q-2$  have been defined, we show how to derive layer  $j$ . Let  $E'$  be the collection of edges of the form  $(v_i, w^{j-1,i})$ , after layers  $1, \dots, j-1$  have been created. By construction, all of them have the same cost, which we denote by  $c'$ . Again, we insert  $l$  new vertices  $w^{j,1}, \dots, w^{j,l}$  at height  $h/q$  for each edge in  $E'$  which we then partition (left-to-right) in  $2^j$  groups  $S^{j,1}, \dots, S^{j,2^j}$ , all of the same size; namely, group  $S^{j,i}$  consists of vertices  $\{w^{j,i(2^{l-j})+1}, \dots, w^{j,(i+1)(2^{l-j})}\}$ . We also add  $2^j$  new vertices  $u^{j,1}, \dots, u^{j,2^j}$  such that for every vertex  $w \in S^{j,i}$ , there is an edge from  $w$  to  $u^{j,i}$  of cost  $g$ , and an edge from  $u^{j,i}$  to  $w$  of cost  $\alpha g$ . We stop when  $q-1$  layers in total have been defined.

We will call a set of the form  $S^{i,j}$  an  $s$ -set. We will also say that a set of vertices  $S$  crosses or intersects a certain set of columns if and only if the set of columns in which the vertices of  $S$  lie intersect the set of columns in question.



**Fig. 3.** The structure of a component for the case  $q = 3$ , and  $l = 8$ . For simplicity, we assume  $c = h = 1$ .

Two sets of vertices *cross each other* iff the intersection of the columns each one crosses is non-empty. Similar definitions apply for a set of edges: a directed edge which lies on column  $i$  is said to cross column  $i$ . Note also that there is a 1-1 correspondence between sets  $S^{i,j}$  and vertices  $u^{i,j}$ , which means that several properties/definitions pertaining to  $S$ -sets carry over to their corresponding  $u$ -vertices: we will use this correspondence to make the discussion more accessible (at a slight abuse of notation).

Note that in the layered component, a set  $S^{j,i}$  at layer  $j$  crosses a set of columns  $J = J_1 \cup J_2$  (where  $J_1$  and  $J_2$  are disjoint sets of columns) such that one  $s$ -set at layer  $j+1$  crosses  $J_1$  (and only  $J_1$ ) and another  $s$ -set in layer  $j+1$  crosses  $J_2$  (and only  $J_2$ ). We call these two  $s$ -sets the *children* of  $S^{j,i}$  (or we say that  $S^{j,i}$  is their *parent*). We extend this definition to the  $u$ -vertices to which the  $s$ -sets described above correspond. Namely, the children of  $u^{j,i}$  are the  $u$ -vertices corresponding to the children of set  $S^{j,i}$ . By convention, we will also say that the children of  $T_2$  are  $S^{1,1}$  and  $S^{1,2}$ .

We now proceed with the description of  $B(E, m, h)$ . At a high level, the construction is based on a recursive series of insertions of appropriately defined layered components. In particular, we will show how to construct a family of graphs  $B_0, B_1, \dots, B_\rho$ :  $B(E, m, h)$  will then be defined as  $B_\rho$  for a suitable choice of the value  $\rho$ . First, denote by  $B_0$  the graph which consists of two sets of vertices  $T_1 = \{v_1, \dots, v_l\}$  and  $T_2 = \{v'_1, \dots, v'_l\}$ , as well as the edges induced by these two sets.

For the sake of clarity, we first present the construction of  $B_1$ , the construction of  $B_i$  for larger values of  $i$  will follow next. Suppose that  $x'$  is such that  $(x')^{(x')} = m$ , which means that  $x' = \Theta(\frac{\log m}{\log \log m})$  (recall that  $m$  is the argument in  $B(E, m, h)$ ). We let  $x$  denote  $\lfloor x' \rfloor$ .  $B_1$  is derived by inserting only one component, namely  $C_1 = (E, x, h/(c \cdot x^2), h)$ . The *height*  $H_1$  of  $G_1$  is defined as the

total number of layers added by the construction, plus one (to account for  $T_1$  and  $T_2$  which we may think of as layers on their own; we define the height of  $T_2$  to be zero). After the insertion of the component, we partition the set of all  $l$  columns into a collection of  $\frac{l}{2^x}$  disjoint sets of columns  $R = \{R_1, \dots, R_{\frac{l}{2^x}}\}$ , with the property that column  $y \in [1, l]$  is in class  $R_i$  if and only if it crosses  $S_1^{x-1, i}$ . In other words, there is a class in  $R$  for each of the highest  $s$ -sets in the component  $C_1$ .

We will also use the notation  $L_1^{j, m}$  to denote the  $m$ -th (from left to right)  $s$ -set in  $G_1$  at height  $j$ . By convention we consider  $L_1^{0, 1} \equiv T_2$ , while  $L_1^{x, 1} = T_1$ , to reflect that  $T_1$  and  $T_2$  constitute the bottom and the top level, respectively.

We now describe how to define  $B_{i+1}$ . The construction is presented in pseudocode. Similar to the case of  $i = 1$ , we use the notation  $L_i^{j, m}$  to denote the  $m$ -th (from left to right)  $s$ -set in  $B_i$  at height  $j$ .  $L_i^j$  is then defined simply as the union of all  $L_i^{j, m}$ .

```

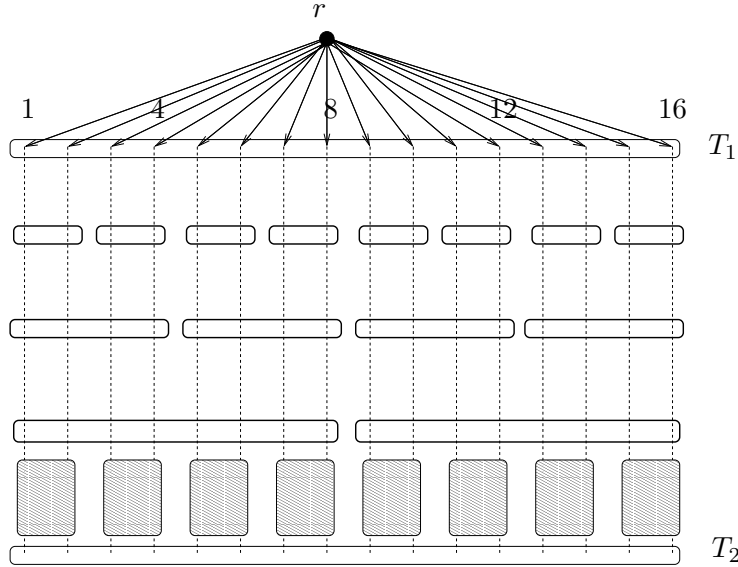
1 for  $j = 0$  to  $H_i - 1$  do
2   for every  $m$  such that  $L \equiv L_i^{j, m} \in L_i^j$  do
3     Let  $C(L) \subseteq L_i^{j+1}$  denote the collection of  $s$ -sets in  $L_i^{j+1}$  with the
     additional property that each  $s$ -set in  $C(L)$  is consecutive to  $L$  in  $B_i$  ;
4     for every  $L' \in C(L)$  do
5       Partition the set of columns crossed by both  $L'$  and  $L$  into a
       collection of classes  $P = \{P_1, \dots\}$  such that  $P_j$  consists of columns in
       one of the classes in  $R$  and only that class;
6       for each class  $P_i$  do
7         Let  $E'$  be the set of edges from  $L'$  to  $L$  which are on the same
         column with class  $P_i$  ;
8         Perform the layered insertion  $(E', x, h/(cx^{i+2}), h)$ . ;
9       end
10    end
11  end
12  Update the partition of the column space  $R$ : Redefine  $R$  such that all
  columns crossed by the same highest  $s$ -set inserted in this iteration(i.e., for
  the current value of  $j$ ) are placed in the same partition ;
13 end

```

**Algorithm 1:** Pseudocode for the creation of  $B_{i+1}$

An example of the construction is given in Figure 4

Having defined  $B_\rho$ , it is now very easy to define the adversarial graph  $\widehat{G}$ . We start with  $T_1, T_2$  as above, and  $c = 1$  (i.e, the downwards cost of any edge is 1). We also use the parameter  $h = 1$ . Finally, we set  $m = k$ . Let  $\widehat{G}_i$  denote the graph which consists of  $B_i$ , with the addition of a *root vertex*  $r$ . There are edges from  $r$  to each vertex in  $T_1$  of infinitesimally small cost (which for the purposes of the proof can be thought to be 0). Then the adversarial graph  $\widehat{G}$  is defined as



**Fig. 4.** Example of the construction of  $\widehat{G}$ . Bold rectangles depict the layers of the only component of  $\widehat{G}_1$ . The eight  $s$ -sets of the highest layer of this component induce a partition of the column space into 8 classes. Shaded rectangles correspond to components of the form  $C_{2,0,id}$ , with  $id = 1 \dots 8$ .

graph  $\widehat{G}_\rho$ . Since  $B_i$  is derived from  $B_{i-1}$  (with the addition of the root  $r$ ), we can also say that  $\widehat{G}_i$  is derived by  $\widehat{G}_{i-1}$ .

The following lemma is easy to show (proof omitted). Here  $x$  is the solution of  $x^x = k$ .

**Lemma 3.** *Let  $l_i$  denote the total number of levels in  $\widehat{G}_i$  (including  $T_1, T_2$ ). Also let  $n_i$  denote the number of “new” levels added when obtaining  $\widehat{G}_i$  from  $\widehat{G}_{i-1}$  (i.e.,  $n_i = l_i - l_{i-1}$ ). Then  $l_i, n_i \in \Theta((x+1)^i)$ . In addition for  $\rho = x$ , there are  $\Theta(k)$  levels in  $\widehat{G}_\rho$ .*

For the remainder of this section,  $x$  has a value of  $\Theta(\log k / \log \log k)$ .

In the next section we show how to derive  $\widehat{\sigma}$  based on graph  $\widehat{G}$ . We say that a  $u$ -vertex (resp.  $S$ -set) is a *vertex/set* of  $\widehat{G}_i$  if it is present in  $\widehat{G}_i$  but not in  $\widehat{G}_{i-1}$ . Note that such a vertex is present in all graphs  $\widehat{G}_{j'}$  with  $j' > i$ . For simplicity we will assume wlog that  $\widehat{G}_\rho$  has exactly  $m$  levels. (the proof is only slightly more complicated if this is not the case, e.g. we must set  $l = 2^{\Theta(m)}$  instead of  $2^m$ .)

Having defined the construction of  $\widehat{G}_i$ , we need a more convenient way of identifying all components present in graph  $\widehat{G}_i$ . In particular we will use the notation  $C_{i,l,id}$  to refer to the component inserted during the construction of graph  $\widehat{G}_i$  between levels  $l$  and  $l+1$  of  $\widehat{G}_{i-1}$  (which means that  $l$  ranges between 0 and  $H_{i-1} - 1$ ) and has a certain component  $id$ , which is simply determined

by numbering the components from left to right for each such fixed value of  $l$ . Within a specific component  $C_{i,l,j}$ , we use the notation  $S_{i,l,j}^{m_1,m_2}$  to define the  $m_1$ -th  $s$ -set from bottom to the top of the component (i.e., located at the  $m_1$ -th layer of the component) which is also the  $m_2$ -th  $s$ -set from left to right among all  $s$ -sets in that layer). The corresponding  $u$  vertices will be identified in the natural way, i.e,  $u_{i,l,j}^{m_1,m_2}$  is the  $u$ -vertex corresponding to  $S_{i,l,j}^{m_1,m_2}$ .

### A.3 The algorithm/adversary game

At a high level, the game in  $\widehat{G}$  proceeds in rounds. Only  $u$  vertices are requested. In round  $i$ , the algorithm will request vertices of  $\widehat{G}_i$  only in a bottom-up fashion, i.e., from lowest to highest height. The important property that the adversary will guarantee is that all requested  $u$ -vertices have corresponding  $s$ -sets which all lie on the same unique path of down-edges from  $r$ . This will ensure that the optimal cost is bounded by a constant. On the other side, we will show that the algorithm will have to pay a high cost per round (roughly  $\Theta(\min(\{k_i, x\}))$ , where  $k_i$  is the number of vertices requested at round  $i$ . This means that the algorithm will pay (roughly) either a constant cost per request, or a cost proportional to  $x$  per round.

We begin with certain preliminary definitions and conventions. Each time the adversary presents a new request, namely a certain  $u$ -vertex, the algorithm must guarantee there is a (directed) path from a previously requested vertex to  $u$ , possibly buying some new edges. Among all possible such paths the adversary will fix/choose *one* such path, which we call a *connection path*. For this path, our analysis will *charge* only parts of it (i.e. specific edges) to the algorithm. If we ensure that each edge bought by the algorithm is charged at most once, then the total cost of all charged edges cannot exceed the actual cost paid by the online algorithm. Without loss of generality we will assume that the connection path is acyclic (otherwise we simply bypass all cycles).

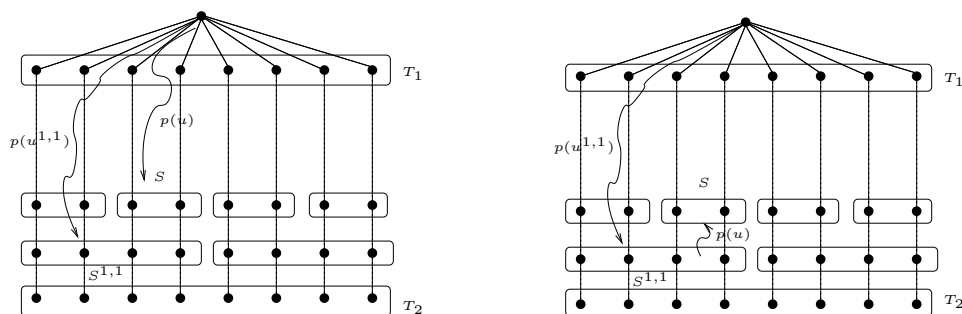
Consider now a  $u$ -vertex requested in round  $i$ , and its connection path  $p(u)$  chosen by the adversary. We observe that  $p(u)$  can be chosen so that it is one of the following three types:

- Connection path *from*  $r$ : A connection path which originates from  $r$  and does not visit any previously requested vertex.
- Connection from above: A connection path which originates from a previously requested vertex which lies *higher* than  $u$  in  $\widehat{G}_i$  (and visits no other previously requested vertices).
- Connection from below: A connection path which originates from a previously requested vertex which lies *lower* than  $u$  in  $\widehat{G}_i$  (and visits no other previously requested vertices).

To illustrate the main ideas of the game, we describe the intuition behind the actions of the adversary during the first round of the game, namely when the adversary requests  $u$ -vertices in  $\widehat{G}_1$ . Later we will describe the game in a more formal way. Since there is only one component in  $\widehat{G}_1$ , we omit subscripts for simplicity, in this example.

The first round begins with request  $u^{1,1}$  which corresponds to  $S^{1,1}$ . The algorithm will buy a connection path from  $r$ , say  $p(u^{1,1})$ . The adversary will then charge “down”-edges in this path (but no other edges). Observe that no matter what the connection path is, there is a child of  $S^{1,1}$  in the second layer of the component, which does not cross any columns of the charged edges. The adversary will choose this child as its second request; call this child  $u$ , and its  $s$ -set  $S$ . At this point, there are two choices concerning the connection path  $p(u)$ :

- *Case 1.*  $p(u)$  is of the form shown in Figure 5: namely, it consists of down-edges, from  $r$  down to (at least) the level of  $u^{1,1}$ . In this case we charge the algorithm with the cost of one down-edge per level, from  $r$  down to the level of  $u^{1,1}$  in  $\widehat{G}_1$ .
- *Case 2.* The connection path is as in Figure 5: namely the connection path consists of “up”-edges between the level of  $u^{1,1}$  up to (at least) the level of  $u$ . We charge the algorithm with the cost of one up-edge per level from the level of  $u^{1,1}$  up to the level of  $u$ .



**Fig. 5.** Depictions of Case 1 (left) and Case 2 (right).

In either of the above cases, let  $q(p)$  denote the part of the connection path  $p(u)$  (i.e., collection of edges) which is charged. Hence depending on which case applies, different edges will be charged, and different costs will be incurred. The critical observation (along the lines of the earlier observation concerning  $u^{1,1}$ ) is that exactly one of the children of  $u$  in  $C$  will correspond to an  $s$ -set which does not cross columns with the columns of  $q(p)$ ; call this child  $u'$ . This provides a good strategy for the adversary to choose its next request; in fact  $u'$  will be the next request. In particular, once again, the connection path to  $u'$  will either result in a charge of “down”-edges from  $r$  down to the level of  $u'$ , or a charge of up-edges from the level of  $u$  up to the level of  $u'$ . We continue the game until we reach a vertex at the top layer (layer  $x - 1$ ) of the component, at which point the first round concludes<sup>3</sup> The algorithm yields a charge for (almost) all the layers

<sup>3</sup> In order to ensure that no edge is charged more than once, we will not charge any cost incurred at the first and top  $(x - 1)$ -th layers of each component. Since this only

in the component: if the majority of such requests were charged as in case (1), then the total charge is  $\Omega(k_1)$ , where  $k_1$  is the number of requests in graph  $\widehat{G}_1$ ; otherwise the total charge due to requests charged as in case (2) will clearly be  $\Omega(\alpha)$ . Therefore, the charge incurred during round 1 is  $\Omega(\min\{k_1, \alpha\})$ .

For round 2, the adversary will play a strategy defined along the same lines; however, instead of requesting vertices within a single component only, the adversary will request vertices in  $H_1$  components in total, where  $H_1$  is the height of  $\widehat{G}_1$ . More precisely, round 2 begins with identifying the set of *critical columns*, namely those columns which cross the highest  $s$ -vertex requested in round 1 (which was also the last vertex requested in round 1); then the adversary identifies the unique component of the form  $C_{2,0,id}$  for some  $id$  which crosses exclusively the critical set of columns. For this component, the algorithm plays a game very similar to the one corresponding to round 1 (which, recall, takes place within a single component). Once “done” with this component, the critical set of columns is updated to reflect the set of columns crossed by the last request (or more precisely, the  $s$ -set of the last request) so far; the algorithm then proceeds with the next unique lowest component in  $\widehat{G}_2$  which crosses only edges of the critical set (a component of the form  $C_{2,1,id'}$  for some  $id'$ ). This repeats until the algorithm has completed requests within  $H_1$  components, in which case round 3 begins. The game continues for  $\rho$  rounds.

In the rest of this section we give a more formal description of the game. Consider the input graph  $\widehat{G}$  and any two  $s$ -sets  $s_1$  and  $s_2$  such that  $s_1$  is higher than  $s_2$  and the two  $s$ -sets cross each other. We say that a directed path  $p$  *includes all down-edges between  $s_1$  and  $s_2$*  (resp. all up-edges) if the path includes one down-edge (resp. up edge) per level, for all levels between  $s_1$  and  $s_2$ .

The following is a basic property concerning the connection paths which can be derived by observing the structure of the request graph (proof omitted).

*Property 4.* Let  $C$  be a certain component in  $\widehat{G}$  and let  $S^{i,j}$ , with  $i < x - 1$  be an  $s$ -set in layer  $i$  of  $C$  with children  $S^{i+1,j_1}$ ,  $S^{i+1,j_2}$  such that: the corresponding vertex  $u^{i,j}$  has already been requested, and no other  $u$ -vertex in  $C$  located between the level of  $S^{i,j}$  and the level of its children in  $\widehat{G}$  has been requested. Suppose that the next request, is a child of  $u^{i,j}$ , say  $u^{i+1,j_1}$ . Let  $p = p(u^{i+1,j_1})$  be the connection path for this request. Then:

- If  $p$  is a path from  $r$ , then the connection path will always include all down-edges between  $r$  and  $S^{i+1,j_1}$ .
- If  $p$  is a path from below, then then the connection path will always include all upward edges between  $S^{i,j}$  and  $S^{i+1,j_1}$ .
- If  $p$  is a path from above, say from a previously requested vertex  $u'$ , then the connection path will always include the edge  $(u', s')$  from  $u'$  to its corresponding  $s$ -set  $s'$  such that  $S^{i+1,j_1}$  crosses  $s'$ .

In addition, if  $q(p)$  denote either i) the set of up-edges between  $S^{i,j}$  and  $S^{i+1,j_1}$ ; or ii) the set of down-edges between  $r$  and  $S^{i+1,j_1}$ ; or ii) the edge  $(u', s')$  (depending

---

happens for 2 our of the  $x - 1$  layers, this technical point will not affect the analysis. We call all charged terminals *typical*.

on which case applies) then exactly one of the children of  $S^{i+1,j}$  will not cross columns with the columns of  $q(p)$ .

The first part of the property suggests that the connection paths have a certain structure. The second part of the property suggests a way for the adversary to request the “appropriate” child of the last request as the next request to be presented. In particular, let us denote by  $\bar{u}$  the last requested vertex, and  $p(\bar{u})$  the corresponding connection path for  $\bar{u}$ . Then, the second part of the property shows that given  $q(p(\bar{u}))$ , as defined in the statement of the claim, there is a unique child of  $\bar{u}$  (within the component to which  $\bar{u}$  belongs) which does not cross the columns of  $q(p(\bar{u}))$ . We will use the notation  $ch(q(p(\bar{u})), \bar{u})$  to denote this unique vertex.

To make the above more precise, the adversary will play the game as shown in the following algorithm

```

1 Initialize the critical set to the set of all columns  $\{1, \dots, l\}$  ;
2 for  $i = 1$  to  $\rho$  do
3   REQUESTS( $\widehat{G}_i$ )
4 end

```

**Algorithm 2:** Pseudocode for creating  $\widehat{\sigma}$ .

where REQUESTS( $\widehat{G}_i$ ) is the strategy for requesting vertices which belong in  $G_i$ , described in what follows:

From the structure of the graph, one can easily prove the following property:

*Property 5.* Let  $M_i$  denote the number of columns with the property that they cross the  $s$ -sets of all vertices requested up to and including the  $i$ -th request. Then  $M_i = \frac{l}{2^i}$ .

*Proof sketch.* First, observe that the set of columns crossed by the  $s$ -set which corresponds to the  $i$ -th request crosses all  $s$ -sets for vertices requested in iterations  $1 \dots i - 1$ . In addition, this set crosses, by construction, exactly  $l/2^i$  columns.  $\square$

Property 5 implies that the adversary/algorithm game is feasible, in the sense that it can go on for  $k$  iterations, namely as many as the number of terminals requested by the adversary.

#### A.4 Properties of the adversarial sequence $\widehat{\sigma}$

We will now summarize some important properties concerning  $\widehat{\sigma}$ .

Let  $u$  be a vertex requested at round  $i$  (i.e., a vertex which belongs in graph  $\widehat{G}_i$ ), and  $p(u)$  its corresponding connection path. Recall that Property 4 provides a classification of the connection paths. Our charging scheme is based on this

```

1 for  $l = 0$  to  $H_i - 1$  do
2   Find the unique component  $C_{i,l,y}$  (for some index  $y$ ) which crosses all
   columns of the critical set ;
3   for  $j = 1$  to  $x - 1$  do
4     if  $j = 1$  then
5       request  $u = u_{i,l,y}^{1,1}$  ;
6       update  $\bar{u} \leftarrow u$  ;
7     end
8     if  $j = x - 1$  then
9        $u = ch(q(p(\bar{u})), \bar{u})$  ;
10      update the critical set to the set of columns crossed by the  $s$ -set to
      which  $u$  corresponds. ;
11    end
12    else
13      request  $u = ch(q(p(\bar{u})), \bar{u})$  ;
14      for the connection path  $p(u)$  chosen update  $\bar{u} \leftarrow u$  and  $p(\bar{u}) \leftarrow p(u)$  ;
15    end
16  end
17 end
18 .

```

**Algorithm 3:** Pseudocode for  $\text{REQUESTS}(\widehat{G}_i)$

classification. In particular, we will charge certain edges in  $q(p)$ , and only such edges (with the exception of the lowest and highest layers in a component as argued earlier). There are three cases:

- *Case 1:*  $q(p)$  describes up-edges. In this case, the cost charged to the algorithm is  $\alpha \cdot \frac{1}{(x+1)^i}$  (since  $q(p)$  includes all up-edges between two consecutive levels at graph  $\widehat{G}_i$ ).
- *Case 2:*  $q(p)$  describes down-edges from  $s$ . In this case, the cost charged to the algorithm is the cost of all such down-edges, down to the level of  $u$ .
- *Case 3.*  $q(p)$  describes an edge of the form  $(u', s')$  such that  $u'$  is a previously requested vertex. Since  $u'$  belongs to some  $\widehat{G}_j$ , with  $j < i$ , the charge in this case is at least  $\frac{\alpha}{x^{j+1}} \geq \frac{\alpha}{x^i}$ .

The three cases above yield the statement of Property 1. The structural property (Property 2) follows directly from the construction of the adversarial sequence  $\widehat{\sigma}$ .

## A.5 Details for the proofs of Section 2

**Proof of Lemma 1.** Given the adversarial graph  $G$  and request sequence  $\sigma$ , we need to upper-bound the cost of the optimal off-line algorithm and lower-bound the cost of any deterministic offline algorithm. For the former, we use the property that any column that crosses the last requested terminal in  $\sigma$  also

crosses all terminals in  $\sigma$ . Hence we can construct an offline solution by buying i) a path from  $r$  that consists of all downwards edges in any one of these columns, and crosses all  $s$ -sets of terminals requested in  $\sigma$ , at a cost equal to 1; and ii) all directed edges from the  $s$ -sets in question to the corresponding requested  $u$ -vertices. Recall that there in the  $i$ -th round of any fixed phase,  $\Theta((x+1)^i)$  terminals are requested. Here,  $x$  is such that  $x = \Theta(\log m / \log \log m)$ . In addition, for such terminals, the cost of the directed edge from their corresponding  $s$ -set is equal to  $\frac{1}{x^{i+1}} \frac{m}{k}$  (this is set in line 8 of Algorithm 1). Since there are  $k/m$  phases, the overall cost of such edges in the offline solution is upper bounded by

$$\Theta \left( \frac{k}{m} \cdot \sum_{i=1}^x \frac{1}{x^{i+1}} \frac{m}{k} ((x+1)^i) \right) = \Theta \left( \frac{1}{x} \sum_{i=1}^x \left(1 + \frac{1}{x}\right)^i \right),$$

and  $\frac{1}{x} \sum_{i=1}^x \left(1 + \frac{1}{x}\right)^i \leq e$  where  $e$  denotes the base of the natural logarithm. Hence the optimal cost is upper-bounded by a constant.

To lower-bound the cost of any deterministic online algorithm, we use the cost property (Property 3). Let  $r_{i,j}$  denote the  $i$ -th round within the  $j$ -th phase of  $\sigma$ , and  $k_{i,j}$  the number of typical terminals within round  $r_{i,j}$ . Let  $k_{i,j}^1$  and  $k_{i,j}^2$  denote the number of typical terminals requested in round  $r_{i,j}$  which are charged according to Property 3(i) and Property 3(ii), respectively. If  $k_{i,j}^1 \geq k_{i,j}^2/2$ , then we say that  $r_{i,j}$  is of type 1, otherwise we say that  $r_{i,j}$  is of type 2, and with a slight abuse of notation we say that  $r_{i,j} = 1$  or  $r_{i,j} = 2$ , depending on which case applies. For a given  $j \in [1, k/m]$  we also define  $r_j = 1$ , if there are at least  $x/2$  values for  $i$  for which  $r_{i,j} = 1$ , otherwise  $r_j$  is set to be equal to 2 (to simplify the analysis, we assume wlog that there are exactly  $k/m$  phases). Note that every phase  $j$  for which  $r_j = 1$ ,  $\Omega(x^{x/2}) = \Omega(\sqrt{m})$  terminals are charged according to rule (i) (Property 3). We now distinguish two cases:

**Case 1:**  $r_j = 1$  for at least  $k/(2m)$  values of  $j$ . In this case, there are at least  $k/(2m)$  phases, with the property that at least  $\Omega(\sqrt{m})$  terminals in each phase are charged according to rule (i). Hence  $\Omega(\frac{k}{m}\sqrt{m}) = \Omega(\frac{k}{\sqrt{m}})$  terminals are charged according to this rule, and their contribution to the cost of the algorithm is at least

$$\Omega \left( \sum_{l=1}^{k/\sqrt{m}} \frac{l}{k} \right) = \Omega \left( \frac{k}{m} \right).$$

This follows from the fact that there exists at most one requested terminal per level, and the  $l$ -th smallest possible depths are thus  $1/k, \dots, l/k$ .

**Case 2:**  $r_j = 2$  for at least  $k/(2m)$  values of  $j$ . For any phase  $j$  with  $r_j = 2$ , at least  $x/2$  rounds of the phase are such that at least half the typical terminals in the round are charged according to rule (ii) (Property 3). The cost of any such round  $i$  in phase  $j$  is  $\Omega(\frac{\alpha}{(x+1)^i} \frac{m}{k} (x+1)^i) = \Omega(\alpha \frac{m}{k})$ , thus the cost of phase  $j$  is  $\Omega(\alpha x \frac{m}{k})$ , and the overall cost of all phases in this case is

$$\Omega \left( \frac{k}{m} \cdot \alpha x \cdot \frac{m}{k} \right) = \Omega(\alpha x).$$

Combining Case (1) and Case (2), and since the optimal offline cost is bounded by a constant, it follows that the competitive ratio of the algorithm is at least

$$\Omega \left( \min \left\{ \frac{k}{m}, \alpha \frac{\log m}{\log \log m} \right\} \right)$$

which concludes the proof of the lemma.  $\square$

**Proof of Theorem 2.** We will use the following lemma that provides a useful partition of  $T^*$  into subtrees of roughly the same number of terminals.

**Lemma 4 ([2]).** *There exists a partition of  $T^*$  into a collection of  $l = \Theta(\alpha)$  edge-disjoint trees  $T_1, \dots, T_l$  such that each such that each tree in the partition contains  $\Theta(k/\alpha)$  terminals.*

We note that the partition need not be vertex disjoint, in the sense that a terminal may belong to more than one trees, however this is not critical, since we can think of each terminal as being assigned to exactly one tree in the partition (that is, we can choose arbitrarily to which tree the terminal is assigned).

For each tree  $T_i$ , let  $K_i$  denote the set of terminals in  $\sigma$  assigned to  $T_i$ , and  $t_i$  denote the first terminal in the request sequence  $\sigma$  which belongs in  $T_i$ . For any  $K' \subseteq \{\sigma\}$ , let  $c_{GR}(K')$  denote the cost paid by GREEDY for requests in  $K'$ . We have that

$$c_{GR}(K) = \sum_{i=1}^l c_{GR}(t_i) + \sum_{i=1}^l c_{GR}(K_i \setminus \{t_i\}) \quad (1)$$

Cost  $\sum_{i=1}^l c_{GR}(t_i)$  is bounded by  $O(\alpha \cdot c(T^*))$ , since  $l \in \Theta(\alpha)$ . On the other hand, using the result of [3]

$$c_{GR}(K_i \setminus \{t_i\}) = O \left( \max \left\{ \alpha \frac{\log(k/\alpha)}{\log \alpha}, \alpha \frac{\log(k/\alpha)}{\log \log(k/\alpha)} \right\} \cdot c(T_i) \right)$$

We emphasize that the above bound applies even though  $t_i$  need not be the root of  $T_i$  (this follows from the fact that the proof in [3] bounds, essentially, the cost paid by terminals other than the first one in the sequence).

Since  $\alpha \in \omega(k^{1-\epsilon})$ , it is clear that  $\log \log(k/\alpha) < \log \alpha$ , hence we have that  $c_{GR}(K_i \setminus \{t_i\}) = O \left( \alpha \frac{\log(k/\alpha)}{\log \log(k/\alpha)} \right)$ , and the theorem follows from the edge-disjointness of the  $T_i$ 's.  $\square$

## B Details for the proofs of Section 3

### B.1 Examples of basic trees

Figure 6 illustrates examples of basic trees.

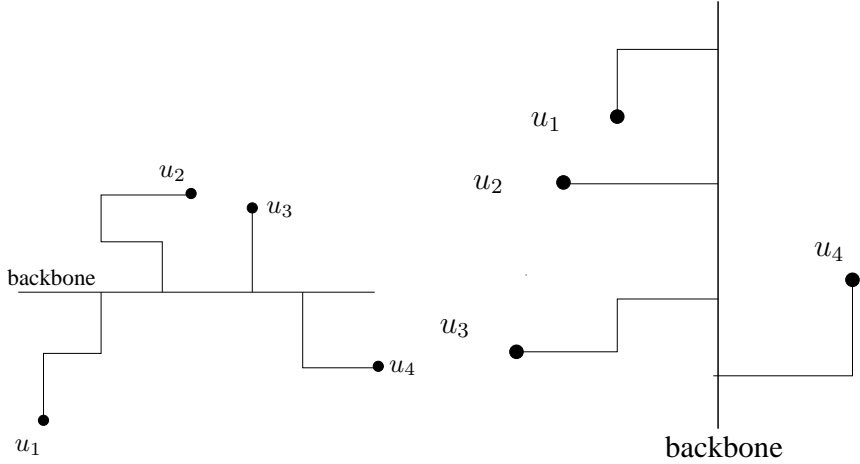


Fig. 6. Examples of basic trees with horizontal (left) and vertical (right) backbone.

## B.2 Preliminaries

We begin with some preliminary discussions and definitions. First, we observe that every basic tree  $T$  (wrt a set of terminals  $K$ ) can be transformed to a tree  $T'$  that spans  $K$ , such that every terminal in  $K$  is connected to the backbone of  $T'$  by means of a straight segment that is perpendicular to the backbone; moreover this transformation does not affect, within constant factors the cost of the basic tree, in the sense that  $c(T') = O(c(T))$ . We will thus assume, from this point onwards, that this is the structure of every basic tree.

The transformation is very simple: for every terminal  $u$ , replace the terminal path with its projection to the backbone of the basic tree. If this projection falls “outside” the backbone, we simply extend the backbone accordingly. This substitution does not affect the overall cost of the basic tree, and can be repeated for every terminal.

**Definition 2.** Let  $T'$  denote a basic tree wrt a set of terminals  $K'$ . Let  $u_1, \dots, u_{k'}$  denote the  $k'$  terminals in  $K'$ , in increasing order of their  $x$ -coordinates (if the backbone is a horizontal segment), or in increasing order of their  $y$ -coordinates (if the backbone is a vertical segment). Let also  $v_i$  denote the point at the intersection of the terminal path for terminal  $u_i$  and the backbone  $P$ . For a terminal  $u_i$  in  $T'$  we say that its index is  $i$ . For two terminals  $u_i, u_j$  in  $T'$  with  $i < j$  we say that  $u_i$  precedes  $u_j$  in  $T'$  (denoted by  $u_i \prec u_j$ ). We say that  $u_j$  is between  $u_i$  and  $u_{i'}$  iff  $u_i \prec u_j \prec u_{i'}$ . For  $u_i \prec u_j$  we call the path  $p_{T'}(v_j, v_i)$  the segment of  $u_i, u_j$  and we denote it by  $s(u_i, u_j)$ . Here,  $p_{T'}(\cdot, \cdot)$  denotes a path in  $T'$  that follows segments (edges) of  $T'$ . The interval  $(u_i, u_j)$  is simply the pair of indices of the two terminals, namely the pair  $(i, j)$ . A terminal  $u_l$  is in the interval  $(i, j)$  if  $u_i \preceq u_l \preceq u_j$  (here  $u_i \preceq u_l$  means either  $u_i \prec u_l$  or  $u_i$  is identical to  $u_l$ ).

With a slight abuse of notation, we use the term “segment” to refer to both a path and its cost, when this is clear from context.

### B.3 Proof of Lemma 2 and Corollary 1

**Proof of Lemma 2.** Partition the set  $K'$  into two subsets, say  $K'_1$  and  $K'_2$ . Here,  $K'_1$  is defined as the set of all terminals  $u \in K'$  such that at the time  $u$  is requested, the shortest path to the current tree has cost at most  $2c_{\max}$ , and  $K'_2$  is simply defined as  $K' \setminus K'_1$ . Since the algorithm pays a cost of at most  $2c_{\max} + 4 \cdot 2c_{\max} = 10c_{\max}$ , for each terminal in  $K'_1$ , it suffices to focus on the set  $K'_2$  only.

Without loss of generality, suppose that the basic tree is such that its backbone consists of a horizontal segment (the case of a vertical segment follows a similar argument). For terminal  $u_i$  in  $K'_2$ , let the cost of the shortest path from  $u_i$  to the tree be equal to  $s_i = 2c_{\max} + \lambda_i$ . We will argue that  $\sum_i \lambda_i \leq c(P)$ , which then completes the proof of the lemma. When serving  $u_i$ , the algorithm will buy a horizontal segment of length equal to  $2s_i \geq \lambda_i$ ; let  $x_i^1, x_i^2$  denote the  $x$ -coordinates of this segment. The main observation here is that no future request  $u_j \in K'_2$  can have  $x$ -coordinate in  $[x_i^1, x_i^2]$ : if this was the case, then the shortest path connecting  $u_j$  to the current tree cannot exceed  $2c_{\max}$ , hence  $u_j$  should belong to  $K'_1$  instead. This implies that for every terminal  $u_i \in K'_2$ , at least a length of  $\lambda_i$  of the horizontal segment bought by the algorithm (when serving  $u_i$ ) is such that it does not share  $x$ -coordinates with the horizontal segment bought when serving any other terminal  $u_j \in K'_2$ . But then these lengths cannot exceed the total length of the backbone, in other words  $\sum_i \lambda_i \leq c(P)$ , which concludes the proof.  $\square$

**Proof of Corollary 1.** We partition all terminals in  $K$  into two sets: Set  $K_1$  consists of all terminals  $u$  such that at the time  $u$  is requested, no other terminal in the basic tree to which  $u$  belongs has been requested. Set  $K_2$  is then defined as the set  $K \setminus K_1$ . We will bound the cost incurred by the algorithm on sets  $K_1$  and  $K_2$ .

First, recall that the online algorithm will always buy a shortest path to the current tree, and pay a total cost proportional to the cost of this shortest path. This implies that we can apply the analysis of the greedy algorithm for the online Steiner tree in general (undirected) graphs (see [12]). In particular, it follows that the cost of the algorithm for serving all requests in  $K_1$  is  $O(\log |K_1|)$ , and since there are as many terminals in  $K_1$  as basic trees in the decomposition of  $T^*$ , it follows that this cost is bounded by  $O(\log \log k)$ .

Second, consider all terminals in  $K_2$ . We can write  $K_2$  as  $\cup_j K_2^j$ , where  $K_2^j$  is the set of all terminals that belong to the  $j$ -th basic tree, say  $T_j$  in the decomposition of  $T^*$ . From Theorem 3, we know that the cost for serving all terminals in  $K_2^j$  is  $O\left(\frac{\log |K_2^j|}{\log \log |K_2^j|}\right) \cdot c(T_j)$ , where  $c(T_j)$  is the cost of tree  $T_j$ . Since all basic trees are edge-disjoint, we obtain that the cost for serving all terminals in  $K_2$  is  $O\left(\frac{\log k}{\log \log k}\right) \cdot c(T^*)$ .

The Theorem follows from combining the cost contributions due to sets  $K_1$  and  $K_2$ .  $\square$

#### B.4 Proof of Theorem 3

The proof applies the techniques used in the proof of the main result in [3]. The main idea is as follows: We need to determine a rule for assigning each terminal  $u \in K'$  to another terminal in  $K'$ , which we call the *mate* of  $u$  and we denote it by  $\bar{u}$ , such that  $\bar{u}$  has been requested earlier than  $u$ . Let  $p_{T'}(u, \bar{u})$  denote the path between  $u$  and  $\bar{u}$  that follows edges of  $T'$ ; we call this path the *connection path for  $u$* . From the statement of the algorithm, it then follows that the overall cost incurred by the algorithm in order to serve all requests in  $K'$  is bounded by

$$c(K') \leq 5 \sum_{u \in K'} c(p_{T'}(u, \bar{u})). \quad (2)$$

The objective is to provide a careful assignment of terminals to mates such that the RHS of (2) is small compared to  $c(T')$ .

Unlike the analysis in [3], in this proof we do not provide explicit connection paths for all terminals in  $K'$ . Instead, for a certain subset of terminals (to be described in detail later), we will bound the cost incurred by the algorithm using Lemma 2.

The first step towards bounding the cost of the connection paths for the terminals is to assign each terminal to a unique mate. This assignment is determined by Algorithm 4. We also seek a partition of terminals as they are being requested, in particular, every terminal becomes the member of a unique *run* (we can think of each run as being assigned a unique integer id, starting with 0 and increasing by 1 every time a new run is initiated). For a terminal  $u$  we denote by  $run(u)$  the run to which  $u$  is assigned. Define  $x$  as the solution to the equation to  $x^x = k'$ , hence  $x = \Theta(\frac{\log k'}{\log \log k'})$ . Without loss of generality we will assume that  $x$  is integral.

Let  $u = u_{\pi_{i+1}}$  denote the current request (i.e., the  $(i+1)$ -th requested terminal among terminals in  $K'$ ), and  $U_i$  denote the set of the  $i$  previously requested terminals. Every terminal  $u$  (with the exception of terminals in run 0) is characterized by two unique terminals in the set  $U_i$ , say terminals  $u_l, u_h \in U_i$  such that  $u_l \prec u \prec u_h$ , and no other terminal in  $U_i$  is in the interval  $(u_l, u_h)$ . We call  $u_l$  and  $u_h$  the *immediate successor and predecessor of  $u$* , respectively, *at the time of the request to  $u$* . After  $u$  is revealed, the algorithm assigns a *label* to each of the resulting intervals  $(u_l, u)$  and  $(u, u_h)$ . There are four types of labels an interval can be assigned, and their semantics is related to the action at the time  $u$  is requested:

- If the interval  $(u_l, u_h)$  has been labeled **free**, then  $u$  will initiate a new run, say  $r$ . We call  $u$  the *initiator* of  $r$  (denoted by  $in(r) = u$ ).
- If the interval  $(u_l, u_h)$  has been labeled **left** then  $u$  will be assigned  $u_l$  as its mate (i.e its immediate predecessor at the time of request).

- If the interval  $(u_l, u_h)$  has been labeled **right** then  $u$  will be assigned  $u_h$  as its mate (i.e. its immediate successor at the time of request).
- A **blank** label is the default labeling for an interval, and is implied if the assignment algorithm does not explicitly assign a label in the set  $\{\mathbf{free}, \mathbf{left}, \mathbf{right}\}$ .

At a high-level, the assignment algorithm works as follows: In the event  $u$  is not between two terminals in  $U_i$  (i.e., it does not have either a successor or a predecessor in  $U_i$ ) it will become part of run 0 (lines 1-9): this is a set-up phase for all remaining runs. Otherwise, let  $u_l$  and  $u_h$  denote the immediate predecessor/successor of  $u$  among terminals in  $U_i$ , at the time  $u$  is requested. (Note that  $u_{max}$  is defined as the terminal of highest index in the basic tree, among terminals in  $U_i$ , whereas  $u_{min}$  is the terminal of smallest such index).

If the interval  $(u_l, u_h)$  is free, then the assignment algorithm invokes algorithm **Free** which initiates a new run, say  $r$ : the run is associated with a *representative*, defined as  $rep(r) \equiv u_h$ , the *left-end of the run*, defined as  $l(r) = u_l$  and a *segment*, defined by  $seg(r) = s(u_l, u_h) = s(l(r), rep(r))$ . The representative of the run is assigned to be the mate of  $u$ . Last, we set the parameter  $R(u')$  to be equal to  $r$ , for all  $u'$  between  $u_l$  and  $u_h$  in the basic tree. The meaning of this assignment is that future requests for terminals within  $(u_l, u_h)$  should become members of the run  $r$  (unless their  $R()$  value changes, in the meantime, due to subsequent requests).

In any other case the assignment algorithm invokes algorithm **NonFree** which assigns  $u$  to the run  $r = R(u)$ , and follows a more complicated rule for assigning a mate and labels: More specifically, if the interval  $(u_l, u_h)$  is left (resp right) then the assignment and labeling is performed in lines 2–6 (resp 7–11), and  $u$  is assigned its immediate predecessor (resp. successor) as its mate. The only remaining possibility is for  $(u_l, u_h)$  to be a blank interval (lines 13–23). In this case, if  $u$  is “close” to  $u_l$  (resp.  $u_h$ ) wrt the cost  $s(u_l, u)$  (resp.  $s(u, u_h)$ ), then  $u$  is assigned  $u_l$  as its mate in lines 13–16 (resp.  $u$  is assigned  $u_h$  as its mate in lines 17–20). The last case is when  $u$  is not close to either terminal (line 22). We will treat this case separately, by relying on Lemma 2.

It is easy to show that the connection cost for terminals in run 0 is small (i.e., at most  $O(c(T'))$ ), we need to focus only on terminals in runs  $> 0$ . We will express the total cost paid by the algorithm on all terminals in  $K'$ , as the sum of four partial costs, denoted by  $C_1, \dots, C_4$  ( $C_1, C_2$ , and  $C_3$  apply to terminals in runs other than run 0). In particular:

- $C_1$  is defined as the cost of connection paths due to terminal paths (i.e., the cost of all terminal paths that are included to some connection path).
- $C_2$  is defined as the cost of connection paths due to edges in the backbone of  $T'$  (here, by “edge” we mean a segment of consecutive  $v$ -vertices in the backbone). More precisely, connection paths established in line 3 or line 14 of **NonFree** (i.e., when the current request is assigned its immediate predecessor as its mate), or in line 8 or line 18 of **NonFree** (i.e., when the current request is assigned its immediate successor as its mate).
- $C_3$  includes two contributions: First, the cost of connection paths due to edges in the backbone  $P$ , and which are bought by connection paths estab-

```

Input : Request  $u$  and the existing assignment of terminals in  $U_i$ 
Output: Assignment of  $u$  to an appropriate run and an appropriate mate
1 if  $u \succ u_{max}$  then
2   assign  $u$  to run 0
3    $mate(u) \leftarrow u_{max}$ 
4   label interval  $(u_{max}, u)$  free
5 end
6 if  $u \prec u_{min}$  then
7   assign  $u$  to run 0
8    $mate(u) \leftarrow u_{min}$ 
9   label interval  $(u, u_{min})$  free
10 end
11 else
12   Let  $u_l$  and  $u_h$  be the immediate successor and predecessor of  $u$ , among
      terminals in  $U_i$ 
13   if interval  $(u_l, u_h)$  is free then
14     Free( $u, u_l, u_h$ )
15   end
16   else
17     Non-free( $u, u_l, u_h$ )
18   end
19 end

```

**Algorithm 4:** Assignment of terminals to runs and mates

```

1 Initiate a run  $r$  with  $seg(r) = s(u_l, u_h)$ ; set  $rep(r) \leftarrow u_h$  and  $l(r) \leftarrow u_l$ 
2 label  $(u_l, u)$  and  $(u, u_h)$  blank
3 Set  $mate(u) \leftarrow u_h$ 
4 if  $s(u, u_h) \leq seg(r)/x$  then
5   label  $(u, u_h)$  left
6 end
7 if  $s(u_l, u) \leq seg(r)/x$  then
8   label  $(u_l, u)$  right
9 end
10 For all  $u' \in \mathcal{C}$ , with  $u_l \prec u' \prec u_h$  set  $R(u') \leftarrow r$ 

```

**Algorithm 5:** Algorithm Free( $u, u_l, u_h$ )

lished in line 3 of **Free**. Second, the contribution to the algorithm's cost due to line 22 of **Non-Free**. Recall that these terminals will require a special treatment.

- $C_4$  is defined as the cost of connection paths for terminals in run 0 (whose overall contribution is very small, as stated earlier).

Since  $C_4 \in O(c(T'))$ , we only need to bound the contributions of  $C_1, C_2$ , and  $C_3$ . Of these costs, the analysis of [3] implies that  $C_1 \in O(xc(T'))$  (follows by section 3.4 in [3]) and the same also holds for  $C_2$  (follows by section 3.4 in [3]).

```

1 Assign  $u$  to run  $r = R(u)$ . Label  $(u_l, u)$ ,  $(u, u_h)$  blank
2 if  $(u_l, u_h)$  is left then
3   set  $mate(u) \leftarrow u_l$ 
4   label  $(u_l, u)$  free
5   label  $(u, u_h)$  left
6 end
7 if  $(u_l, u_h)$  is right then
8   set  $mate(u) \leftarrow u_h$ 
9   label  $(u, u_h)$  free
10  label  $(u_l, u)$  right
11 end
12 else
13   if  $s(u_l, u) \leq seg(r)/x$  then
14     set  $mate(u) \leftarrow u_l$ 
15     label  $(u_l, u)$  right
16   end
17   else if  $s(u, u_h) \leq seg(r)/x$  then
18     set  $mate(u) \leftarrow u_h$ 
19     label  $(u, u_h)$  left
20   end
21   else
22     No assignment to a mate; treat this case separately using Lemma 2
23   end
24 end

```

**Algorithm 6:** Algorithm NonFree( $u, u_l, u_h$ )

Essentially, these two bounds follow easily by ignoring the factor  $\alpha$ , since in the Euclidean Steiner tree problem there is no concept of asymmetry.

Remains then to bound  $C_3$ ; in particular, our objective is to show that  $C_3 \in O(xc(T'))$ . The following lemma (which is proved along the lines of Lemma 1 in [3]) will be useful.

**Lemma 5.** *For any given run  $r$ , at most  $x$  terminals in run  $K'$  will execute line 22 of NonFree.*

For a given edge  $e$  in the backbone  $P$  of  $T'$ , we define the *depth* of  $e$  as the total number of runs  $r \neq 0$ , (i.e., excluding run 0) whose segment  $seg(r)$  includes edge  $e$ . Let  $R$  denote the set of all runs (again, excluding run 0) established by the assignment algorithm. We say that every time a new run  $r$  is initiated (line 1 of Free), the depth of every edge in  $seg(r)$  increases by 1 (initially, before any terminal is requested, all edges in  $P$  have zero depth). Thus the depth of a run  $r$  is the depth of any of its edges. Let  $R_j \subseteq R$  denote the set of all runs of depth  $j$

The following lemma shows that the cost of the segment of a run decreases exponentially with its depth, and is derived similar to Lemma 6 in [3].

**Lemma 6.** *For a run  $r \in R_j$ ,  $seg(r) \leq \frac{c(P)}{x^{j-1}}$ .*

We now show how to bound cost  $C_3$ . Let  $Z \subseteq K'$  be the set of terminals contributing to  $C_3$ . We say that a terminal  $u \in Z$  belongs in class  $Z_j \subseteq Z$  if and only if its corresponding run belongs in the class  $R_j$ . Consider a terminal  $u$  which belongs in run  $r \in R_j$ , and denote by  $C_3(u)$  its contribution to  $C_3$ . Denote by  $C_3(Z_j)$  the contribution of all terminals in  $Z_j$  to  $C_3$ .

We distinguish between the following two cases: If  $u$  is assigned a mate in line 3 of **Free**, then we can bound its contribution by

$$C_3(u) = O(seg(r) + c(t_u) + c(t_{rep(r)})),$$

where  $c(t_u)$  and  $c(t_{rep(r)})$  denote the cost of the terminal paths for  $u$  and  $rep(r)$ , respectively.

If, on the other hand,  $u$  executes line 22 of **Non-Free**, then we can bound the contribution of *all terminals in  $r$*  by the following quantity

$$O(seg(r) + x \cdot c_{\max}^r)$$

where  $c_{\max}^r$  is defined as the maximum cost of all terminal paths, among all terminals in  $r$  that contribute to  $C_3$ . The reason is as follows: We know from Lemma 5 that for a given run  $r$  at most  $x$  terminals will execute line 22 of **NonFree**. These terminals are all members of run  $r$ , hence their terminal paths intersect the segment  $seg(r)$ . In other words, these terminals form a basic tree with a backbone of cost at most  $seg(r)$ . We can thus apply Lemma 2.

Summarizing, we obtain that

$$C_3(Z_j) = O \left( \sum_{r \in R_j} seg(r) + \sum_{r \in R_j} x \cdot c_{\max}^r + \sum_{r \in R_j} c(t_{rep(r)}) \right).$$

hence the total contribution to cost  $C_3$  is bounded by

$$C_3 = O \left( \sum_j \sum_{r \in R_j} seg(r) + \sum_j \sum_{r \in R_j} x \cdot c_{\max}^r + \sum_j \sum_{r \in R_j} c(t_{rep(r)}) \right). \quad (3)$$

We will treat each term that appears in the RHS of (3) separately. First, note that since a terminal belongs in at most one run, we have

$$\sum_j \sum_{r \in R_j} x \cdot c_{\max}^r \leq x \sum_{u \in K'} c(t_u) \leq x \cdot c(T') \quad (4)$$

Similarly, since every terminal is the representative of at most one run, we obtain

$$\sum_j \sum_{r \in R_j} c(t_{rep(r)}) \leq c(T'). \quad (5)$$

It remains then to bound the contribution due to the term  $\sum_j \sum_{r \in R_j} seg(r)$ . The crucial observation here is that for fixed  $j$ , the segments of all runs in  $R_j$

are edge-disjoint, which yields that  $\sum_{r \in R_j} \text{seg}(r) \leq c(P)$ . Combining this fact with Lemma 6 we have  $\sum_j \sum_{r \in R_j} \text{seg}(r) \leq \min \left\{ c(P), \frac{c(P)}{x^j - 1} |Z_j| \right\}$

Since there are at most as many runs as terminals in  $K'$ , it follows that the RHS of the last inequality is maximized if  $|Z_j| = x^j - 1$ , for all  $j \geq 2$ , which yields

$$\sum_j \sum_{r \in R_j} \text{seg}(r) = O(x \cdot c(P)). \quad (6)$$

Putting everything together, using (4), (5) and (6), (3) implies

$$C_3 = O(x \cdot c(T')) = O\left(\frac{\log k'}{\log \log k'}\right) \cdot c(T').$$

Last, recall that the total cost incurred by the online algorithm on requests from the set  $K'$  is bounded by the sum of  $C_1, C_2, C_3$  and  $C_4$  and we showed that each of these contributions is bounded by  $O\left(\frac{\log k'}{\log \log k'}\right) \cdot c(T')$ , which concludes the proof of Theorem 3.