

A Near-tight Bound for the Online Steiner Tree Problem in Graphs of Bounded Asymmetry

Spyros Angelopoulos
Max-Planck-Institut für Informatik
sangelop@mpi-inf.mpg.de

April 4, 2008

Abstract

The edge asymmetry of a directed, edge-weighted graph is defined as the maximum ratio of the weight of antiparallel edges in the graph, and can be used as a measure of the heterogeneity of links in a data communication network. In this paper we provide a near-tight upper bound on the competitive ratio of the Online Steiner Tree problem in graphs of bounded edge asymmetry α . This problem has applications in efficient multicasting over networks with non-symmetric links. We show an improved upper bound of $O\left(\min\left\{\max\left\{\alpha\frac{\log k}{\log \alpha}, \alpha\frac{\log k}{\log \log k}\right\}, k\right\}\right)$ on the competitive ratio of a simple greedy algorithm, for any request sequence of k terminals. 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 ϵ is an arbitrarily small constant) due to Faloutsos *et al.* [8] and Angelopoulos [3].

1 Introduction

The *Steiner Tree* problem occupies a central place in the area of approximation and online algorithms. In its standard version, the problem is defined as follows. Given an undirected graph $G = (V, E)$ with a weight (cost) function $c : E \rightarrow \mathbb{R}^+$ on the edges, and a subset of vertices $K \subseteq V$ with $|K| = k$ (also called *terminals*), the goal is to find a minimum-cost tree which spans all vertices in K . When the input graph is *directed*, the input to the problem must specify, in addition to G and K , a vertex $r \in V$ called the *root*. The problem is then to find a minimum cost *arborescence* rooted at r which spans all vertices in K .

In the *online* version of the problem, the terminals in K are revealed to the algorithm as a sequence of requests. When a request for terminal $u \in V$ is issued, and assuming a directed graph, the algorithm must guarantee a directed path from r to u . The input graph G is assumed to be known to the algorithm. Using the standard framework of competitive analysis (see, e.g., [6]), the objective is to design online algorithms of small *competitive ratio*. More precisely, the competitive ratio is defined as the supremum (over all request sequences and input graphs) of the ratio of the cost of the arborescence produced by the algorithm over the optimal off-line cost assuming complete knowledge of the request set K .

Apart from its theoretical importance and its application in several combinatorial optimization problems, the Steiner tree problem is useful in modeling efficient multicast communication over a network. Indeed, multicasting involves dissemination of information from a designated source to members of a subscribing group. The online variant describes then the situation in which

subscribers to the service are not known beforehand, but rather issue dynamic requests to join to the group. The reader is referred to [10] for a study of the relation between Steiner tree problems and network multicasting.

The majority of existing research in Steiner trees and its generalizations applies to undirected graphs. In contrast, actual communication networks contain, in their majority, links asymmetric in the quality of service they offer; this situation is even more prevalent in satellite and radio networks. In [7], studies on the traffic of network backbones reveal marked asymmetry in parameters such as speed, link utilization and reliability. Another example is wireless networks, in which the upstream and downstream links can vary significantly in terms of their characteristics, due to considerations such as the transmission power, the noise difference and the mobility of the endpoints.

Ramanathan [11] introduced the problem of multicast-tree generation in the presence of asymmetric links. To this end, he considered several metrics of network asymmetry, among which the *maximum edge asymmetry* is the most intuitive and easiest to measure in a real network. Formally, the measure (to which we will refer to simply as *asymmetry* in what follows) is defined as the maximum ratio of the weights of antiparallel links. More precisely, 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

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

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 extreme cases, graphs of bounded asymmetry can be useful in modeling networks with a certain degree of link heterogeneity.

The competitive ratio of the online Steiner tree problem in graphs of either constant, or unbounded asymmetry is tightly bound. For the former class, Imase and Waxman [9] showed a bound of $\Theta(\log k)$, achieved by a simple greedy algorithm, (a result which was extended by Berman and Coulston [5] to the Generalized Steiner Problem). 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 [13]. For the online Steiner Tree in the Euclidean plane, the best known lower bound on the competitive ratio is $\Omega(\log k / \log \log k)$ due to Alon and Azar [1]. On the other hand, Westbrook and Yan [12] showed that in directed graphs (of unbounded asymmetry), the competitive ratio of any algorithm can be bad as $\Omega(k)$, a bound trivially matched by a naive algorithm which serves each request by buying a least-cost path from the root to the requested terminal.

Faloutsos *et al.* [8] were the first to study the online Steiner tree problem in graphs of bounded asymmetry. They 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. Angelopoulos [3] (see also [2] for the full version) improved the upper bound on the competitiveness of GREEDY to $O\left(\min\left\{\alpha \frac{\log k}{\log \log \alpha}, k\right\}\right)$, and showed a corresponding lower bound of $\Omega\left(\min\left\{\frac{\alpha \log k}{\log \log k}, k^{1-\epsilon}\right\}\right)$ for every constant $0 < \epsilon < 1$.

It is important to note that when $\alpha \in \Omega(k)$ the lower bound on the competitive ratio due to [8] is $\Omega(k)$, which is obviously tight (using the trivial upper bound of $O(k)$ for GREEDY). Thus the problem is interesting only when $\alpha \in o(k)$.

In this paper we provide an almost-tight upper bound on the competitiveness of GREEDY:

Theorem 1 *The competitive ratio of GREEDY for an input graph of asymmetry α and a request sequence of k terminals is $O\left(\min\left\{\max\left\{\alpha^{\frac{\log k}{\log \alpha}}, \alpha^{\frac{\log k}{\log \log k}}\right\}, k\right\}\right)$.*

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 ϵ is any arbitrarily small constant) due to [8] and [3]. In particular¹ it provides a tight bound on the competitive ratio of the problem for the case where either $\alpha \in O(k^{1-\epsilon})$ (for some constant $\epsilon \in (0, 1)$) or $\alpha \in \Omega(k)$. In contrast, [3] is not tight when α is relatively small, e.g., when α is polylogarithmic in k .

In addition, we believe that the techniques in this paper may yield a better upper bound on the competitive ratio of the greedy algorithm for the online Euclidean Steiner tree problem [1] (for which there is a $\Theta(\log \log k)$ gap between the known upper and lower bounds).

1.1 Preliminaries and notation

We denote by $e = (v, u)$ and $\bar{e} = (u, v)$ a pair of antiparallel directed edges, incident to vertices u, v . Let $T = (r', V', E')$ be an arborescence rooted at r' , we denote by \hat{T} the graph (V', E'') , with $E'' = E' \cup \{\bar{e} : e \in E'\}$. In words, \hat{T} induces all edges in T as well as all their antiparallel edges. We denote by $p_T(u, v)$ (resp. $p_{\hat{T}}(u, v)$) the simple directed path from u to v using exclusively edges in T (resp. \hat{T}). Note that such paths are uniquely defined (provided that $p_T(u, v)$ exists in T).

The cost of a directed path p will be denoted by $c(p)$. We denote by $c(T)$ the cost of arborescence T , namely the sum of the cost of the directed edges in T . We emphasize that only edges in T and none of their antiparallel edges contribute to $c(T)$. We will always use T^* to denote the optimal arborescence on input (G, K) , with $|K| = k$, and $OPT = c(T^*)$. For any $K' \subseteq K$, we let $c_{GR}(K')$ denote the cost that GREEDY pays on the subset K' of the input (in other words, the contribution of terminals in K' towards the total cost of GREEDY).

For convenience, we will be using the term “tree” to refer to a (rooted) arborescence.

2 Outline of the proof of Theorem 1 and intuition

In order to prove Theorem 1, we first show that it applies to situations in which the spanning arborescence has a fairly simple structure: in particular, to instances called *comb instances* in [3] (see Figure 1 for an illustration).

Definition 2 *Let T' denote a tree rooted at vertex $r' \in V$ and let $K' \subseteq K$, with $|K'| = k'$. We call the triplet $\mathcal{C} = (T', K', r')$ a comb instance, or simply comb if the following hold: T' consists of a directed path P from r' to a certain vertex v_1 , which visits vertices $v_{k'}, \dots, v_1$ in this order (but possibly other vertices too); there are also disjoint directed paths t_i from v_i to u_i . No other edges*

¹A more precise, albeit more cumbersome expression of the lower bound is $\Omega\left(\min\left\{\max\left\{\alpha^{\frac{\log k}{\log \alpha}}, \alpha^{\frac{\log k}{\log \log k}}\right\}, \frac{k}{f(k)}\right\}\right)$, where $f(k)$ is any function such that $f(k) \in o(k^\epsilon)$, for every constant $\epsilon < 1$, e.g. $f(k)$ polylogarithmic in k . Therefore, the small gap that remains is bounded by $f(k)$ (in the worst case) rather than by a polynomial function of k .

are in T' . Finally the set K' is precisely the set $\{u_1, \dots, u_{k'}\}$. We call P the backbone of \mathcal{C} , and the paths t_i the terminal paths² of the comb. The vertex set of \mathcal{C} is the set of vertices in T' .

The following is a key theorem in the analysis of GREEDY. Essentially the lemma states that we can achieve the desired competitive ratio if we know, for instance, that (T^*, K, r) is a comb.

Theorem 3 *Given the comb $\mathcal{C} = (T', K', r')$, let $z \in K'$ denote the terminal requested the earliest among all terminals in K' . Then $c_{GR}(K') = c_{GR}(z) + O\left(\max\left\{\alpha \frac{\log k'}{\log \alpha}, \alpha \frac{\log k'}{\log \log k'}\right\}\right) c(T')$.*

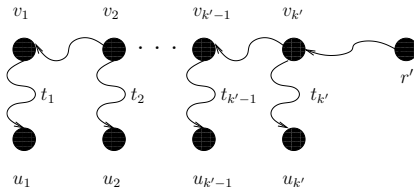


Figure 1: The structure of a comb instance.

Given Theorem 3, the main result follows by partitioning the set of all requests K into a collection of near-disjoint comb-instances. Here, by near-disjoint we require that every edge in T^* appears in at most two comb instances. We refer the reader to Appendix B for a formal proof.

In order to prove Theorem 3, let π denote a permutation of $\{1, \dots, k'\}$ such that $\sigma = u_{\pi_1}, \dots, u_{\pi_{k'}}$ is the sequence of the requests in K' in the order in which they are requested (hence $z = u_{\pi_1}$). Note that we aim towards bounding $c_{GR}(K' \setminus u_{\pi_1})$. To this end, we will determine an assignment for every terminal u_{π_i} with $2 \leq i \leq k'$ to a *specific* terminal $\bar{u}_{\pi_i} \in \{u_{\pi_1}, \dots, u_{\pi_{i-1}}\}$. We call terminal \bar{u}_{π_i} the *mate* of u_{π_i} . Let q_i denote the directed path in \hat{T} from \bar{u}_{π_i} to u_{π_i} , also called the *connection path* for u_{π_i} . Since $c_{GR}(K' \setminus u_{\pi_1}) \leq \sum_{i=2}^{k'} c(q_i)$ suffices to show that:

$$C \stackrel{\text{def}}{=} \sum_{i=2}^{k'} c(q_i) = O\left(\max\left\{\alpha \frac{\log k'}{\log \alpha}, \alpha \frac{\log k'}{\log \log k'}\right\}\right) c(T'). \quad (1)$$

Comb instances were identified in [3] as the hard instances for the problem, and for such instances, a weaker version of Theorem 3 was proved (c.f. Lemma 3.2 in [3]). More precisely, the definition of the comb in [3] requires a strict upper bound of $O(\alpha)$ on the number of terminals in the comb: this leads to an upper bound for $c_{GR}(K' \setminus u_{\pi_1})$ equal to $O\left(\alpha \frac{\log \alpha}{\log \log \alpha}\right) c(T')$. The proof of the main result in [3] proceeds then by first extending the result to all subtrees of T^* of $O(\alpha)$ terminals (not necessarily combs), and then by applying it, in a recursive manner, in a hierarchical partition of T^* in trees of $O(\alpha)$ terminals each. This process yields an additional multiplicative overhead of $\log k / \log \alpha$ compared to the cost incurred by a comb instance of size $O(\alpha)$.

In this paper we follow a different approach. We allow the combs to contain an arbitrarily large number of terminals, which may very well be in $\Omega(\alpha)$. This allows us to bypass the need for recursion, and thus to save the factor $\log k / \log \alpha$. Instead, as already mentioned, suffices to decompose T^* (and K) into a collection of near-disjoint comb instances. In this more general setting, some of the high-level proof ideas remain as in [3]: we will still partition the terminals in a

²Note that the definition allows the terminal paths t_i to be empty, in which case $v_i \equiv u_i$; in addition, the directed paths from v_i to v_{i-1} (as well as the path from r' to $v_{k'}$) may also be empty, in which case $v_i \equiv v_{i-1}$. As argued in [3] we can assume, without loss of generality that such degeneracies do not arise.

comb in appropriately defined subsets called *runs* which dictate how to select the proper mate for each terminal. However, the definition of runs and the assignment of mates in [3] is not applicable anymore when $k' \in \Omega(\alpha)$: In [3] a connection path can be as costly as the cost of the backbone (which becomes far too expensive if the number of terminals in the comb is in $\Omega(k)$). Instead, a substantially more involved assignment is required. The entire section 3 deals with finding such an assignment, and proving that it incurs small overall cost.

Definition 4 Let $\mathcal{C} = (T', K', r')$ be a comb instance. For a terminal u_i in the comb we say that its index is i . For two terminals u_i, u_j in the comb with $i < j$ we say that u_i precedes u_j in \mathcal{C} (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 $s(u_i, u_j)$. 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. (see Appendix A for an illustration of the above definition).

3 Proof of Theorem 3

3.1 Assignment of terminals to their mates

The first step towards bounding the cost of the connection paths for terminals in the comb is to assign each terminal to a unique mate. This assignment is determined by Algorithm 1. 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 $w = \min\{\alpha, x\}$, where x is the solution 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 in the comb), 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 comb, 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 comb. 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). If u is not close to either terminal then it is assigned the representative of the run it belongs to as its mate (line 22).

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 1: Assignment of terminals to runs and mates

Lemma 5 (Appendix) *The total cost of connection paths for terminals in run 0 is $O(\alpha) \cdot c(T')$.*

Since the connection cost for terminals in run 0 is small, we will focus only on terminals in

```

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  $s(u, u_h) \leq seg(r)/w$  then
5   label  $(u, u_h)$  left
6 end
7 if  $s(u_l, u) \leq seg(r)/w$  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 2: Algorithm Free(u, u_l, u_h)

runs > 0 from this point on. We need to bound the total connection cost C as expressed by (1). For this purpose, we will express C as the sum of six partial costs, denoted by C_1, \dots, C_6 (C_1, \dots, C_5 apply to terminals in runs other than run 0). In particular:

- C_1 is defined as the cost of connection paths due to edges e such that \bar{e} belongs in some terminal path t_i in the comb (i.e., the cost of edges antiparallel to edges of a terminal path).
- C_2 is defined as the cost of connection paths due to edges e such that \bar{e} belongs in the backbone P (i.e., the cost of edges antiparallel to edges in the backbone) and which are bought by connection paths established in either line 3 or line 14 of **NonFree** (i.e., when the current request is assigned its immediate predecessor as its mate).
- C_3 is defined as the cost of connection paths due to edges e such that e belongs in the backbone P , and which are bought by connection paths established in either line 8 or line 18 of **NonFree** (i.e., when the current request is assigned its immediate successor as its mate).
- C_4 is defined as the cost of connection paths due to edges e such that e belongs in the backbone P , and which are bought by connection paths established in either line 3 of **Free** or line 22 of **NonFree**.
- C_5 is defined as the cost due to edges e such that e belongs in some terminal path t_i in the comb.
- C_6 is defined as the cost of connection paths for terminals in run 0.

Since the terminal paths t_i are edge-disjoint, it follows that $C_5 \leq c(T')$. In addition, C_6 is bounded by $O(\alpha) \cdot c(T')$, from Lemma 5. Thus, it remains to bound C_j , $j \in [1, 4]$. We will denote by $C_{j,i}$ the contribution of the connection path q_i for terminal u_{π_i} to the cost C_j , which means that $C_j = \sum_{i=1}^{k'} C_{j,i}$.

Appendix A presents an example of the run decomposition. The next section presents certain important properties of runs and labellings, as determined by the assignment algorithm.

3.2 Properties of runs and labellings

Fix a terminal u . Every time during the execution of the assignment algorithm a terminal u' is requested such that u is the immediate successor of u' at the time of the request, and the interval (u', u) receives a label L by the assignment algorithm, then we say that u becomes *high- L* (in the

```

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  $s(u_l, u) \leq seg(r)/w$  then
14     set  $mate(u) \leftarrow u_l$ 
15     label  $(u_l, u)$  right
16   end
17   else if  $s(u, u_h) \leq seg(r)/w$  then
18     set  $mate(u) \leftarrow u_h$ 
19     label  $(u, u_h)$  left
20   end
21   else
22     set  $mate(u) \leftarrow rep(r)$ 
23   end
24 end

```

Algorithm 3: Algorithm NonFree(u, u_l, u_h)

sense that u is the high-endpoint of an interval which receives a label L). Similarly, every time during the execution of the assignment algorithm a terminal u' is requested such that u is the immediate predecessor of u' and the interval (u, u') receives a label L , then we say that u becomes *low- L* . Recall that $L \in \{\text{free}, \text{left}, \text{right}, \text{blank}\}$.

Property 6 (Appendix) *Let v denote a specific terminal, and suppose that after terminal u_{π_i} is requested and processed by the assignment algorithm, v becomes high- L . Then the following hold:*

- (i) *If $L=\text{left}$, then for all subsequent requests u_{π_j} ($j > i$), v will remain high-left.*
- (ii) *If $L=\text{blank}$, then for all subsequent requests u_{π_j} ($j > i$), v will become either high-left or remain high-blank.*
- (iii) *If $L=\text{free}$ then for all subsequent requests u_{π_j} ($j > i$), v will either become high-left or high-blank, or remain high-free.*
- (iv) *If $L=\text{right}$ then for all subsequent requests u_{π_j} ($j > i$), if the high-label of v changes, it will never become high-right again.*

Corollary 7 (Appendix) *Every terminal u can be the representative of at most one run, and the left-end of at most one run.*

The following is a straightforward property which shows that all terminals of a run are between the left-end of the run and the representative of the run.

Property 8 (Appendix) For every terminal u , $l(\text{run}(u)) \prec u \prec \text{rep}(\text{run}(u))$.

We say that an interval (u, u') is *contained within* interval (v, v') (u, u', v, v' denote terminals in the comb) if each of u, u' is contained within interval (v, v') . We say that a run r is contained within interval (u, u') if $(l(r), \text{rep}(r))$ is contained within (u, u') (and hence from property 8 the same holds for all terminals in r). Last, r is contained within r' if $(l(r), \text{rep}(r))$ is contained within $(l(r'), \text{rep}(r'))$. Note that this implies that $\text{seg}(r) \leq \text{seg}(r')$.

Property 9 (Appendix) Let (u, v) be an interval labeled **free** at some point in the execution of the assignment algorithm. Then for every future request u' with $u \prec u' \prec v$, u' will become member of a run r which is contained within the interval (u, v) .

Property 10 (Appendix) Let terminal u be requested, with immediate predecessor and successor at the time of its request u_l, u_h , respectively, and suppose that the interval (u, u_h) becomes **left** as a result of line 5 of **NonFree**. Suppose also that there exists a pair of terminals u_1, u_2 , in immediate successor/predecessor relation at the time of one of their requests such that $u_1 \prec u_l \prec u_h \preceq u_2$, and (u_1, u_2) was labeled **free**. Then there exists a terminal v with the following properties:

- $u_1 \prec v \preceq u_l$;
- At the time v is requested no terminal other than u_h has been requested in the interval (v, u_h) ;
- Interval (v, u_h) becomes left as a result of the execution of either line 5 in algorithm **Free** or line 19 in algorithm **NonFree**.

Properties symmetric to Property 6 and Property 10 are given in the Appendix (see Property 18 and Property 19).

Lemma 11 (Appendix) For any given run r , at most w terminals in K' are assigned $\text{rep}(r)$ as their mate in line 22 of algorithm **NonFree**.

3.3 Bounding the cost $C_{1,i}$

Lemma 12 $C_{1,i} \leq \alpha(w + 5) \cdot c(t_i)$, where t_i is the terminal path for u_{π_i} in the comb.

Proof sketch. Fix a terminal $v = u_{\pi_i}$: we will bound the number of terminals in K' which are assigned v as their mate. There are six possible cases for which a requested terminal u is assigned v as its mate, in particular during executions of the following lines: line 3 for $\text{Free}(*, *, v)$; line 8 or line 18 for $\text{NonFree}(*, *, v)$; line 3 or line 14 for $\text{NonFree}(*, v, *)$; and last, line 22 of **NonFree**, more specifically during the call $\text{NonFree}(u, u_l, u_h)$ for some terminals u, u_h, u_l , with $\text{rep}(r) \equiv v$. Here, “*” denotes any arbitrary terminal. We will show that, for fixed v , with the exception of line 22 which will be invoked at most w times, all other invocations will occur at most once which suffices to prove the lemma. We only highlight one case, due to space limitations.

Case 1: Suppose that line 3 in algorithm **Free** is executed, in particular during an execution of the form $\text{Free}(*, *, v)$. Note that right before the first execution of this form, more precisely, the first time $\text{Free}(u, u_l, v)$ is executed for some terminals u and u_l , the interval (u_l, v) must be free (the if-condition of line 13 of the assignment algorithm must hold). In other words, v is labeled high-free, prior to this first execution, whereas at the end of this execution, it will become either high-blank

(default), or high-left (line 5). By the same reasoning, v must be labeled high-free every time line 3 in $NonFree(*, *, v)$ is executed. However, Property 6(iii) states that a high-free terminal will never become high-free twice during the execution of the assignment algorithm. Hence line 3 of algorithm **Free** is executed at most once during invocations of the form $Free(*, *, v)$, for fixed v .

All remaining cases can be shown along the same lines (with the exception of line 22 which relies on Lemma 11), and can be found in the Appendix. \square

3.4 Bounding $C_{2,i}$ and $C_{3,i}$

We will first show how to bound $C_{2,i}$, then the bound for $C_{3,i}$ will follow an almost identical proof.

Recall that u_{π_i} contributes to C_2 when it is assigned a mate as a result of one of lines 3 and 14 in **NonFree**. More precisely, we say that u_{π_i} *contributes* the directed edge e , with $\bar{e} \in P$ when the connection path q_i for u_{π_i} includes e . For the remainder of the proof for $C_{2,i}$ we will call such edges *expensive*. Also, let q'_i denote the subpath of q_i which consists of expensive edges only (i.e., the subpath of q_i which consists of edges antiparallel to edges in the backbone of the comb), then clearly $C_{2,i} = c(q'_i)$. Let X denote the subset of the set of comb terminals K' which consists of terminals with non-zero contribution to C_2 . Consider the sequence of connection paths for terminals in X , as such terminals are requested over time. More precisely, we can think of all edges in q'_i being “bought”, as the connection path between the terminal and its mate is *established*, at the precise moment $u_{\pi_i} \in X$ is requested. In this view, every time an expensive edge is contributed due to such an assignment, we say that the *depth* of the edge increases by 1 (initially, i.e., before any terminals have been requested, all expensive edges have depth zero).

Claim 13 (Appendix) *For a terminal $u_{\pi_i} \in X$, all expensive edges in q'_i have the same depth, right after q_i is established.*

Claim 13 asserts that it is meaningful to say that terminal $u_{\pi_i} \in X$ is of depth δ if right after it is assigned to its mate, and the connection path q_i is established, the depth of all expensive edges at the connection path becomes equal to δ . This implies that we can further partition X into sets $X_1, X_2 \dots$ such that X_i consists of all terminals of depth i . Note that for all i with $u_{\pi_i} \in X_j$, the paths q'_i are edge-disjoint.

The following is the main technical lemma of this section. The lemma shows that the contribution of a terminal to C_2 decreases exponentially with its depth (recall that $c(P)$ denotes the cost of the backbone P of the comb).

Lemma 14 (Appendix) *For a terminal $u_{\pi_i} \in X_j$, with $j \geq 1$, $C_{2,i} \leq \frac{\alpha c(P)}{w^{j-1}}$.*

A similar upper bound can be shown for $C_{3,i}$ (see the statement of Lemma 21 in the Appendix) since terminals which contribute to $C_{3,i}$ follow assignments to mates which are symmetric to the assignments for terminals with contribution to $C_{2,i}$ (even strongly, the α factor does not appear in the upper bound since connection paths which contribute to $C_{3,i}$ follow edges in the backbone, and not their antiparallel edges).

3.5 Towards bounding cost $C_{4,i}$

In this section we establish a lemma which is instrumental in bounding $C_{4,i}$. For a given edge $e \in P$ define the r -*depth* (or for simplicity *depth* for the remainder of this section) 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). Recall from Corollary 7 that every terminal can be the representative of at most one run and the left-run of at most one run.

Claim 15 (Appendix) *All edges in $seg(r)$ have the same depth right after r is established.*

Claim 15 asserts that we can partition R into sets R_1, R_2, \dots such that R_i consists of all runs of depth i . Note that for every two runs r and r' with $r, r' \in R_j$, the segments of r and r' are disjoint.

Lemma 16 *For a run $r \in R_j$, $seg(r) \leq \frac{c(P)}{w^j-1}$.*

Proof. By induction on j . The lemma is trivially true for $j = 1$. Suppose the lemma holds for j , we will show that it holds for $j + 1$. Let r be a run in R_{j+1} . We will show that r is contained within a run r' of depth j for which it holds that $seg(r) \leq seg(r')/w$: by induction hypothesis, we will then have that $seg(r) \leq \frac{c(P)}{w^j}$, and the lemma is proved.

It is easy to see first that there exists a run r' of depth j such that r is contained within r' (similar to the proof of Claim 15); more precisely, $l(r') \prec l(r) \prec rep(r) \prec rep(r')$. Recall that at the time right before $in(r')$ is requested, the interval $(l(r'), rep(r'))$ is a free interval. Likewise, at the time right before $in(r)$ is requested, the interval $(l(r), rep(r))$ is a free interval. We will thus consider cases, depending on the actions that forced the interval $(l(r), rep(r))$ to become free (since r is of depth at least 2, this can happen only due to calls to **Free** or **NonFree**).

Case 1. $(l(r), rep(r))$ became free as a result of line 4 of **NonFree**. Then there exists a terminal u with $rep(r) \prec u \preceq rep(r')$ such that when $rep(r)$ was requested, the interval $(l(r), u)$ was a left interval. From Property 10 there must exist a terminal v such that $l(r') \prec v \preceq l(r)$ and the interval (v, u) became left as a result of either line 19 of **NonFree** or line 5 of **Free**. Let r'' denote the run that v joins, then from the two cases above, we have that the if-condition of line 17 of **NonFree**, or line 4 of **Free**, respectively, holds, hence $s(v, u) \leq seg(r'')/w$. Note that $(l(r), rep(r))$ is contained within the interval (v, u) , thus $seg(r) \leq s(v, u)$; in addition from Property 9 r'' is contained within run r' , hence $seg(r'') \leq seg(r')$. Combining the above inequalities we deduce that $seg(r) \leq seg(r')/w$.

Case 2. $(l(r), rep(r))$ became free as a result of line 9 of **NonFree**. This case is very similar to case 2, in the sense that left intervals are now “replaced” by right intervals (see details in the Appendix). \square

3.6 Adding up the individual contributions

Recall from the discussion in section 3.2 that the total connection cost C for terminals in the comb is expressed as the sum of the partial costs C_1, \dots, C_6 , and that C_5 and C_6 have only a small asymptotic contribution to C . We thus focus on costs C_1, \dots, C_4 . Recall also that w is defined as $\min\{\alpha, x\}$, x is such that $x^x = k'$ and that y is such that $\alpha^y = k'$.

From Lemma 12 we have that $C_1 \leq \sum_i \alpha(w+5) \cdot c(t_i) \leq \alpha(x+5) \cdot c(T')$ where the last inequality follows from $w \leq x$ and the fact that the terminal paths of the comb are disjoint.

We show next how to bound C_4 . Let $Z \subseteq K'$ be the set of terminals contributing to C_4 . Recall that a terminal $u \in Z$ which belongs to a run r is assigned as a mate the representative of the run r (this occurs in line 3 of **Free** and line 22 of **NonFree**); moreover, the contribution of u to C_4 is at most the segment of the run r , $seg(r)$. Using the notation introduced in section 3.5, 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 . Denote by $c_4(Z_j)$ the contribution of terminals in Z_j to Z .

From Lemma 11 we know that for any fixed run r there are at most w terminals in Z which contribute to C_4 due to line 22 of **NonFree**, and their total contribution is bounded by $w \cdot \text{seg}(r)$. On the other hand, since r has a unique initiator for a fixed r at most one terminal in Z contributes to cost C_4 due to line 3 of **Free**. In total, for a given run r at most $w + 1$ terminals in run r contribute to C_4 , and their total contribution is bounded by $(w + 1) \cdot \text{seg}(r)$. For fixed j the segments of all runs in R_j are edge-disjoint, which yields $c_4(Z_j) \leq (w + 1) \cdot c(P)$. Combining this fact with Lemma 16 we have $c_4(Z_j) \leq \min\{(w + 1) \cdot c(P), (w + 1) \frac{c(P)}{w^{j-1}} |Z_j|\}$

Since $C_4 = \sum_j c_4(Z_j)$, and there are at most as many runs as terminals in the comb, it follows that C_4 is maximized if $|Z_j| = w^{j-1}$, for all $j \geq 2$, which yields

$$C_4 \in O(w \max\{x, y\} \cdot c(P)) = O(\alpha \max\{x, y\} \cdot c(P)).$$

For costs C_2 and C_3 one can show the following bounds, using a similar argument based on the lemmas of section 3.4 (see Appendix):

$$C_2 = O(\max\{\alpha x \cdot c(P), \alpha y \cdot c(P)\}) \quad \text{and} \quad C_3 = O(\max\{x \cdot c(P), y \cdot c(P)\}).$$

Theorem 3 follows by adding C_1, \dots, C_6 and the fact $c(P) \leq c(T')$. □

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. Technical Report CS-2006-36, David R. Cheriton School of Computer Science, University of Waterloo, 2006.
- [3] 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.
- [4] B. Awerbuch, Y. Azar, and Y. Bartal. On-line generalized steiner problem. *Theor. Comp. Sci.*, 324(2–3):313–324, 2004.
- [5] P. Berman and C. Coulston. Online algorithms for Steiner tree problems. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing*, pages 344–353, 1997.
- [6] A. Borodin and R. El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.
- [7] K. Claffy, G. Polyzos, and H.W. Braun. Traffic characteristics of the t1 nsfnet backbone. In *Proceedings of INFOCOM*, 1993.
- [8] 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.
- [9] M. Imase and B. Waxman. The dynamic Steiner tree problem. *SIAM Journal on Discrete Mathematics*, 4(3):369–384, 1991.
- [10] C. A. S. Oliveira and P. M. Pardalos. A survey of combinatorial optimization problems in multicast routing. *Comput. Oper. Res.*, 32(8):1953–1981, 2005.
- [11] S. Ramanathan. Multicast tree generation in networks with asymmetric links. *IEEE/ACM Trans. Netw.*, 4(4):558–568, 1996.
- [12] J. Westbrook and D. C. K. Yan. Linear bounds for on-line steiner problems. *Information Processing Letters*, 55(2):59–63, 1995.
- [13] J. Westbrook and D. C. K. Yan. The performance of greedy algorithms for the on-line steiner tree and related problems. *Math. Syst. Theory*, 28(5):451–468, 1995.

Appendix

A An illustration of a comb instance and related definitions

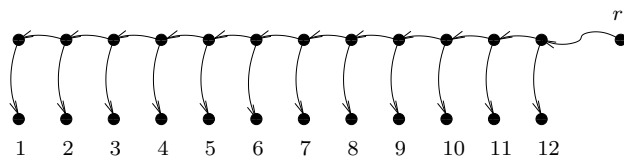


Figure 2: Illustration of some comb definitions.

Consider the comb of Figure 2, where $i \leq 12$ denotes terminal u_i . The index of u_i is equal to i . Then we have, for instance, that $u_4 \prec u_{10}$, and u_6 is between u_4 and u_{10} . All terminals $u_4 \dots u_{10}$ are in the interval $(4, 10)$.

Consider the example in which terminals arrive in the following order:

$$5, 8, 2, 3, 12, 7, 1, 6, 10, 9, 11, 4$$

In this example, run 0 will consist of terminals 5,8,2,12 and 1. Terminals 3,7 and 10 initiate new runs (e.g. when terminal 3 is requested, $(2,5)$ is a free interval, and 3 initiates a run with representative 5 and left-end 2). All remaining terminals, namely terminals 4,6,9 and 11 will be assigned to runs depending on the cost of edges in the backbone, as described in section 3.1.

B Proof of Theorem 1

We will show how to apply the main technical result (Theorem 3) to show essentially the same bound on the competitive ratio for all instances to the problem. Let T^* denote the optimal tree, rooted at vertex r . We will partition the set K of terminals into near-disjoint comb instances. The important property we want to guarantee is that each edge in T^* will be shared by at most two comb instances in this partition.

The partition is induced by the order in which terminals in K are requested. Let σ denote the sequence of requests for the set K , and let $\sigma[i]$ denote the i -th requested terminal in K . We initialize the set of comb instances, denoted by \mathcal{C} to the empty set. For any two vertices $v, u \in T^*$, we will denote by $p(v, u)$ the path $p_{\hat{T}^*}(v, u)$, namely the path from v to u in \hat{T}^* .

The first terminal in σ , namely $\sigma[1] = z$, induces a (trivial) comb instance of the form $\mathcal{C}_1 = (p(r, z), \{z\}, r)$, with backbone $p(r, z)$, and a single, empty terminal path³. We say that we *assign* z to \mathcal{C}_1 , and we add \mathcal{C}_1 to \mathcal{C} .

Consider now terminal $t = \sigma[i]$, with $i > 1$. Focus on the sequence of vertices in the order they are visited, in the path $p(t, r)$. Let v denote the first vertex in this sequence which belongs to the vertex set of some $\mathcal{C}_l \in \mathcal{C}$, with the convention that in the case where t itself is in the vertex set of \mathcal{C}_l , we consider v to be vertex t (if v belongs to more than one combs, then we choose any of

³Note that as discussed in [3], the definition of a comb instance allows combs in which the terminal paths are empty, and such degeneracies do not affect the validity of Theorem 3.

such combs to be \mathcal{C}_l .) Note that the sequence of vertices must include r which is in \mathcal{C}_1 , so such a v always exists. We consider the following cases, and make appropriate decisions *in this order*:

- *Case 1.* v is a vertex in the backbone of \mathcal{C}_l . In this case we assign t to \mathcal{C}_l and we update \mathcal{C}_l by adding the corresponding terminal path $p(v, t)$ (possibly empty) to \mathcal{C}_l .
- *Case 2.* If Case 1 does not apply, then v must belong in one of the terminal paths of \mathcal{C}_l . In particular, there must exist some $j < i$, such that terminal $s = \sigma'[j]$ is a terminal already assigned to \mathcal{C}_l , and v is a vertex in the terminal path corresponding to s in \mathcal{C}_l such that v does not belong in the backbone of \mathcal{C}_l . Let s' denote the vertex of the terminal path for s in \mathcal{C}_l which also belongs in the backbone of \mathcal{C}_l . In this case, we create a new comb $\mathcal{C}_{\rho+1}$, where ρ is the highest current index of combs in the collection \mathcal{C} to be added to \mathcal{C} . More precisely we define $\mathcal{C}_{\rho+1}$ to be rooted at s' , with $p(s', s)$ as its backbone, and the path $p(v, t)$ as the terminal path for t . (See Figure 3 for an illustration).

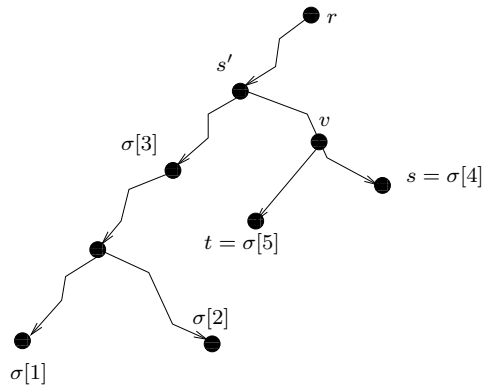


Figure 3: An example of the assignment of terminals into combs. Here, terminals $\sigma[1] \dots \sigma[4]$ are assigned to \mathcal{C}_1 . Terminal $t = \sigma[5]$ initiates \mathcal{C}_2 , with terminal $s = \sigma[4]$ paying for initiating \mathcal{C}_2 .

The following lemma can be shown along the lines of the proof of Lemma 3.1 in [3]. Define $x = \log k / \log \log k$ and $y = \log k / \log \alpha$. Then:

Lemma 17 $c_{GR}(K) = c_{GR}(z) + O(\alpha \{\max\{x, y\}\})OPT = O(\alpha \max\{x, y\})OPT$

The theorem follows then from the straightforward facts that $c_{GR}(K) \leq k \cdot OPT$ and $c_{GR}(z) \leq OPT$. \square

Proof of Lemma 17

We say that a terminal t initiates a comb $\mathcal{C}_{\rho+1} \in \mathcal{C}$ if when considering t , Case (2) applies. For s as defined in the description of Case (2), we say that s pays $\mathcal{C}_{\rho+1}$. For every comb $\mathcal{C}_i \in \mathcal{C}$ ($i > 1$) we define by t_i^1 and s_i^1 the terminals which initiate \mathcal{C}_i and pay \mathcal{C}_i , respectively. Also let $K_i \subseteq K$ denote the set of terminals assigned to comb \mathcal{C}_i , and T_i the edge-set of \mathcal{C}_i .

From the decomposition we have:

$$c_{GR}(K) = \sum_i c_{GR}(t_i^1) + \sum_i c_{GR}(K_i \setminus \{t_i^1\}). \quad (2)$$

The partition into combs has the following property of *near-disjointness*: A terminal path for any comb $\mathcal{C}_i \in \mathcal{C}$ is the backbone for at most one \mathcal{C}_j , with $j \neq i$. In addition, the terminal paths

of all combs in \mathcal{C} are edge-disjoint. Using this property in combination with Theorem 3 we derive that⁴

$$\sum_i c_{GR}(K_i \setminus \{t_i^1\}) = O(\alpha \max\{x, y\}) \sum_i c(T_i) \leq O(\alpha \max\{x, y\})2c(T), \quad (3)$$

Consider terminal $t = t_i^1$. Then $c_{GR}(t_i^1) \leq c(p(s_i^1, t_i^1)) \leq \alpha c(T_i)$. Therefore,

$$\begin{aligned} \sum_i c_{GR}(t_i^1) &= c_{GR}(t_1^1) + \sum_{i \geq 1} c_{GR}(t_i^1) \leq c_{GR}(z) + \sum_{i \geq 1} (\alpha c(T_i)) \\ &\leq c_{GR}(z) + 2\alpha c(T), \end{aligned} \quad (4)$$

where, in the last line of inequalities, we used the fact that a specific terminal will always initiate only one comb, in conjunction with near-disjointness of combs.

Using, (3) and (4), Eq (2) gives $c_{GR}(K) = c_{GR}(z) + O(\alpha \max\{x, y\})c(T)$. \square

C Omitted proofs for Theorem 3

Proof of Lemma 5. Let Q_0 denote the set of connection paths for terminals in R_0 , then every edge in \hat{T}' is shared by at most two connection paths in Q_0 , hence the total cost of paths in Q_0 is at most $2\alpha \cdot c(T')$ (in fact it is easy to show that it is at most $(\alpha + 1) \cdot c(T')$). \square

Proof of Property 6. First of all, note that once v becomes high-L, then by definition v can modify its label only when a terminal u is requested such that v is the immediate successor of u , at the time the latter is requested. With this observation into account, we can argue in a case-by-case basis:

- (i) If L=left, then for a terminal u defined above, lines 2–6 of **NonFree** will be executed. Line 4 dictates that v will remain high-left.
- (ii) If L=blank, then for a terminal u defined above, lines 13-23 of **NonFree** will be executed. This means that either v retains the default high-blank label or becomes high-left (line 19), and these are the only possibilities. From property 6(i) if v becomes high-left it will remain high-left. We conclude that for all requests subsequent to u_{π_i} the only possible high-labels for v are high-left and high-blank.
- (iii) If L=free then for a terminal u defined as above, algorithm $Free(u, u_l, v)$ for some u_l will be executed, which implies that either v becomes high-left (line 5) or becomes high-blank. From properties 6(i) and 6(ii) v can only become either high-left or high-blank in subsequent requests.
- (iv) If L=right, then for a terminal u defined as above, lines 7–11 of **NonFree** will be executed, thus v will become high-free (line 9). From property 6(iii), v will never become high-right again.

⁴Note that the function $\log x / \log \log x$ is increasing in x .

□

The following is a property symmetric to Property 6, concerning low-labels.

Property 18 *Let v denote a specific terminal, and suppose that after terminal u_{π_i} is processed by the assignment algorithm, v becomes low-L. Then the following hold:*

- (i) *If L=right, then for all subsequent requests u_{π_j} ($j > i$), v will remain low-right.*
- (ii) *If L=blank, then for all subsequent requests u_{π_j} ($j > i$), v will become either low-right or remain low-blank.*
- (iii) *If L=free then for all subsequent requests u_{π_j} ($j > i$), v will either become low-blank or low-right, or remain low-free.*
- (iv) *If L=left then for all subsequent requests u_{π_j} ($j > i$), if v changes its low-label, it will never become low-left again.*

Proof. First of all, note that once v becomes high-L, then by definition v can modify its label only when a terminal u is requested such that v is the immediate predecessor of u , at the time the latter is requested. With this observation into account, we can argue in a case-by-case basis:

- (i) If L=right, then for a terminal u defined as above lines 7–11 of **NonFree** will be executed and v will become low-right (line 10). Hence v will not switch low-label and remain low-right.
- (ii) If L=blank, then for a terminal u defined as above lines 13-23 of algorithm **NonFree** will be executed, which implies that either v becomes low-right (line 15) or retains the default low-blank label. The property follows then from property 18(i).
- (iii) If L=free then for a terminal u defined as above, **Free** is executed, and either v becomes low-right (line 8) or becomes low-blank. Properties 18(i) and 18(ii) guarantee that v can only become either low-right or low-blank in subsequent requests, or remain low-free.
- (iv) If L=left then then for a terminal u defined as above algorithm **NonFree** will be executed (in particular lines 2–6). In this case v becomes low-free (line 4), and from Property 18(iii) v will never become low-left for any terminal requested subsequently.

□

Proof of Corollary 7. Right before a new run with u as representative is initiated, u must be labeled high-free. From algorithm **Free**, u will become either high-blank or high-left after the run is initiated. Then Properties 6(i) and 6(ii) guarantee that u will never be labeled high-free again, and hence it cannot be a representative of more than a single run. Similarly, right before a new run with u as left-end is initiated, u must be labeled low-free. From algorithm **Free**, u will become either high-blank or high-right after the run is initiated. Then Properties 18(i) and 18(ii) guarantee that u will never be labeled low-free again, and hence it cannot be the left end of more than a single run. □

Proof of Property 8. If terminal u is the initiator of r , the property follows straightforwardly, since $l(r) \prec in(r) \prec rep(r)$. Otherwise, u is assigned to run r , where r is determined by the value of $R(u)$. The latter is in turn set during a call to algorithm **Free**, in particular during a call of the form $Free(u', u_l, u_h)$ in which run r was initiated, with $l(r) = u_l$ and $rep(r) = u_h$. Then u must be between $l(r)$ and $rep(r)$, by definition of $R(u)$ and the property follows. \square

Proof of Property 9. Let u_l, u_h denote the its immediate predecessor and successor of u' , at the time of its request. We distinguish between two cases: u' is such that it initiates a new run, or u' joins an existing run. In the former case, u' initiates a run r with $l(r) = u_l$ and $rep(r) = u_h$, and since $u \preceq u_l$ and $u_h \preceq v$, the property follows. In the latter case, observe first that the first terminal requested within interval (u, v) will initiate a run, say r' , at which point $R(u') = r'$. This value may change as a result of more terminals being requested in (u, v) , however, it is clear that every time $R(u')$ is modified, say to the value r'' , r'' will be contained within the interval $(l(r'), rep(r'))$, and hence within the interval (u, v) as well, which concludes the proof. \square

Proof of Property 10.

First, note that both u_1 and u_2 are requested prior to u , otherwise they would not be in an immediate successor/predecessor relation. Since the interval (u, u_h) becomes left as a result of line 5 of **NonFree**, then at the time u is requested, (u_l, u_h) is a left interval (line 2 of **NonFree**). If (u_l, u_h) became left as a result of either line 5 in algorithm **Free** or line 19 in algorithm **NonFree**, then the property holds by picking $v \equiv u_l$. Otherwise, (u_l, u_h) became left as a result of line 5 of **NonFree**, hence there must exist a terminal u' such that at the time u_l was requested, u' and u_h were its immediate predecessor and successor, respectively and (u', u_h) was labeled left. Once again, if (u', u_h) became left as a result of either line 5 in algorithm **Free** or line 19 in algorithm **NonFree**, then the property holds by picking $v \equiv u'$. We can then repeat the same argument, and eventually we can find a terminal $v \prec u'$ such that the last two properties of the statement of the lemma hold, since there are only three ways in which an interval can become left.

We then need to guarantee that terminal v has the additional property $u_1 \prec v$. Suppose, by way of contradiction, that such v did not exist. Then, from the discussion above, there should exist a sequence of requests $v_1 \prec v_2 \prec \dots v_m \preceq u_l$ (for some m) such that $v_1 \preceq u_1$ and (v_1, u_h) became left at the time v_1 is requested. Note that v_1 cannot be identical to u_1 : this would imply that either u_1 is not an immediate predecessor of u_2 when u_1 is requested (if u_h is not identical to u_2) or the interval (u_1, u_2) becomes left (if it happens that $u_h \equiv u_2$) which contradicts the statement that (u_1, u_2) is labeled free. For the same reasons, we can argue that v_2 cannot be identical to u_1 . However, this would imply that at the time v_2 is requested, no terminals have been requested in (v_1, u_2) , which means that when u_1 is requested, an earlier request between u_1 and u_2 has occurred, a contradiction to the hypothesis that u_1 and u_2 are in an immediate successor/predecessor relation. \square

The following is a property “symmetric” to Property 10 (left intervals are replaced by right intervals) which can be shown in an almost identical manner.

Property 19 *Let terminal u be requested, with immediate predecessor and successor at the time of its request u_l, u_h , respectively, and suppose that the interval (u_l, u) becomes right as a result of line 10 of **NonFree**. Suppose also that there exists a pair of terminals u_1, u_2 , in immediate successor/predecessor relation at the time of one of their requests such that $u_1 \preceq u_l \prec u_h \prec u_2$, and*

(u_1, u_2) was labeled **free**. Then there exists a terminal v with the following properties:

- $u_h \preceq v \prec u_2$;
- At the time v is requested no other item has been requested in the interval (u_l, v) ;
- Interval (u_l, v) becomes right as a result of either line 8 of **Free** or line 15 of **NonFree**.

Proof of Lemma 11. With a slight abuse of notation, let u_1, \dots, u_m denote the terminals in K' which are assigned $rep(r)$ as their mate in line 22, and suppose wlog that $u_1 \prec u_2 \dots \prec u_m$ in the comb. We will argue that $s(u_i, u_{i+1}) > seg(r)/w$. Suppose, by way of contradiction, that $s(u_i, u_{i+1}) \leq seg(r)/w$ and suppose for definitiveness that u_{i+1} was requested later than u_i , then at the point u_{i+1} is requested and is assigned a mate, the condition of the if-statement of line 13 in **NonFree** would hold, which means that line 22 would not be executed and u_{i+1} would not be assigned $rep(r)$ as its mate, a contradiction (the same argument holds if u_{i+1} was requested earlier than u_i , since the if-condition of line 17 would hold). Since for every $i, j \in [1, m-1]$, with $i \neq j$, the segments $s(u_i, u_{i+1})$ and $s(u_j, u_{j+1})$ are edge-disjoint (by definition of the sequence), we conclude that if it were that $m > w$, then at least one of the u_i would not be contained within the segment of r , a contradiction to Lemma 8. \square

Proof of Lemma 12 (remaining cases).

Fix a terminal $v = u_{\pi_i}$: we will bound the number of terminals in K' which are assigned v as their mate. There are five remaining possible cases in which a requested terminal u is assigned v as its mate:

Case 2: Line 8 of **NonFree** is executed, in particular during an execution of the form $NonFree(*, *, v)$. Note that right before the first execution of line 8 that involves terminal v (more precisely, the first time $NonFree(u, u_l, v)$ is executed for some u, u_l) the interval (u_l, v) must be labeled right (the if-condition of line 7 must hold). In other words, v is labeled high-right, prior to this first execution, and becomes high-free once line 9 is executed. By the same reasoning, v must be labeled high-right prior to any execution of line 3 in $NonFree(*, *, v)$. However, property 6(iv) states that a high-right terminal will never become high-right twice during the execution of the assignment algorithm. Hence line 8 of **NonFree** is executed at most once during invocations of the form $NonFree(*, *, v)$, for fixed v .

Case 3: Line 3 of **NonFree** is executed, in particular during a call of the form $NonFree(*, v, *)$. Right before the first execution of line 3 that involves terminal v (more precisely, the first time $NonFree(u, v, u_h)$ is executed for some u, u_h) the interval (v, u_h) must be labeled left (the if-condition of line 2 must hold). In other words, v is labeled low-left, prior to this first execution, but becomes low-free due to the subsequent execution of line 4. By the same reasoning, v must be labeled low-left prior to every execution of line 3 in $NonFree(*, v, *)$. However, property 18(iv) states that a low-left terminal will never become low-left twice during the execution of the assignment algorithm. Hence line 3 of algorithm **NonFree** is executed at most once during invocations of the type $NonFree(*, v, *)$, for fixed v .

Case 4: Line 14 of **NonFree** is executed, in particular during an execution of the form $NonFree(*, v, *)$. Note that right before the first execution of line 14 that involves terminal v (more precisely, the

first time $NonFree(u, v, u_h)$ is executed for some u, u_h) the interval (v, u_h) must be labeled blank: the interval cannot be free (otherwise algorithm **Free** would be invoked) and cannot be left/right either (otherwise either one of the if conditions in lines 2 and 7 would hold). Line 15 labels the interval (v, u) right, or equivalently, v becomes low-right. By property 18(i), v will never switch low-label from that point on (will remain labeled as low-right), and hence no subsequent execution of line 14 of the form $NonFree(*, v, *)$ will take place, since this would require that v becomes again low-blank at some point later on.

Case 5: Line 18 of algorithm **NonFree** is executed, in particular during an execution of the form $NonFree(*, *, v)$. Note that right before the first execution of line 18 that involves terminal v (more precisely, the first time $NonFree(u, u_l, v)$ is executed for some u, u_l) the interval (u_l, v) must be labeled blank: the interval cannot be free (otherwise algorithm **Free** would be invoked) and cannot be left/right either (otherwise one of the if conditions in lines 2 and 7 would hold). Line 19 labels the interval (u_l, v) left, or equivalently, v becomes high-left. By property 6(i), v will never switch high-label from that point on (will remain labeled as high-left), and hence no subsequent executions of line 18 of the form $NonFree(u, *, v)$ will take place, since this would require that v becomes again high-left at some point in time, as argued above.

Case 6: Line 22 of algorithm **NonFree** is executed, in particular during the call $NonFree(u, u_l, u_h)$ for some terminals u, u_h, u_l , with $rep(r) \equiv v$. From Lemma 11 and Corollary 7 it follows that v is assigned as a mate to at most w terminals throughout executions of this line.

Summarizing, a given terminal v becomes the mate of at most $w + 5$ requested terminals, hence for every i , $C_{1,i} \leq w + 5$. \square

Proof of Claim 13 By way of contradiction. Consider the first terminal $u_{\pi_i} \in X$, in the order in which terminals are requested, which does not have the required property. We claim that there would exist then a terminal $u' \neq \bar{u}_{\pi_i}$ such that u' is requested earlier than u_{π_i} , and for which $\bar{u}_{\pi_i} \prec u' \prec u_{\pi_i}$ in the comb. Indeed, if this was not the case, then for all pairs $(\bar{u}_{\pi_j}, u_{\pi_j})$ such that u_{π_j} is requested before u_{π_i} and $u_{\pi_j} \in X$, we would have that either i) $\bar{u}_{\pi_j} \preceq \bar{u}_{\pi_i}$ and $u_{\pi_i} \preceq u_{\pi_j}$; or ii) $\bar{u}_{\pi_j} \prec u_{\pi_j} \preceq \bar{u}_{\pi_i}$ or $u_{\pi_i} \preceq \bar{u}_{\pi_j} \prec u_{\pi_j}$. This would imply that either q'_i is a subpath of q'_j (case (i)), or q'_j and q'_i are edge-disjoint (case (ii)). But since all edges in q'_j have the same depth after q'_j is established (by the choice of u_{π_i}), the same would be true for q'_i , a contradiction. Hence there is a terminal u' between \bar{u}_{π_i} and u_{π_i} in the comb which is requested earlier than u_{π_i} . This is a contradiction, because it implies that u_{π_i} is not assigned \bar{u}_{π_i} as its mate. \square

Proof of Lemma 14. By induction on j . The claim is trivially true for $j = 1$ from the disjointness of all q'_i 's for terminals in X_1 . Suppose the claim holds for j , we will show it holds for $j + 1$. Consider the set of terminals X_{j+1} . Recall that every terminal in X_{j+1} will buy expensive edges of current depth exactly j prior to the assignment of the said terminal to its mate, then right after the assignment the depth of such edges increases by one. Let u_{π_i} be a terminal in X_{j+1} , q_i its connection path, and $r > 1$ the run to which it belongs. Note that $C_{2,i} = c(q'_i) = \alpha \cdot s(\bar{u}_{\pi_i}, u_{\pi_i})$. Thus, suffices to show equivalently that $s(\bar{u}_{\pi_i}, u_{\pi_i}) \leq c(P)/w^j$, whereas, by induction hypothesis we know that for every $u_{\pi_l} \in X_j$, $s(\bar{u}_{\pi_l}, u_{\pi_l}) \leq c(P)/w^{j-1}$.

We begin with a simple observation. Consider the set Q of paths of the form q'_l such that $u_{\pi_l} \in X_j$. As noted earlier, any two paths in Q are edge disjoint. We claim that q'_i is a subpath of one of the paths in Q . Note first that every edge $e \in q'_i$ must belong in some path $q \in Q$, since the

depth of all edges in q'_i become $j + 1$ once q'_i is established. Then we can use an argument along the lines of the proof of Claim 13: If q'_i was not a subpath of a path in Q , then there would exist a terminal $u' \in X_j$ other than \bar{u}_{π_i} which is requested earlier than u_{π_i} and such that $\bar{u}_{\pi_i} \prec u' \prec u_{\pi_i}$ in the comb, a contradiction, since that would mean that u_{π_i} would not be assigned \bar{u}_{π_i} as its mate.

We know, therefore, that there exists a terminal $u_{\pi_l} \in X_j$ for which q'_i is a subpath of q'_l . More precisely, $\bar{u}_{\pi_l} \preceq \bar{u}_{\pi_i} \prec u_{\pi_i} \prec u_{\pi_l}$. The path q'_l is established as a result of either line 3 or line 14 in **NonFree**. Suppose, first, that the path q'_l was established as a result of the execution of line 3 (this is the complicated case—we will later return to consider the latter case). This implies then that right after q'_l is established, the interval $(\bar{u}_{\pi_l}, u_{\pi_l})$ becomes free. The following is a very useful observation which follows directly from property 9:

Claim 20 *Any terminal u for which $\bar{u}_{\pi_l} \prec u \prec u_{\pi_l}$ becomes member of a run which is contained within the interval $(\bar{u}_{\pi_l}, u_{\pi_l})$.*

We consider two cases concerning the establishment of q'_i :

- *Case 1:* q'_i was established as the result of line 14 of **NonFree**. Let r denote the run to which u_{π_i} is assigned (line 1). The if-condition of line 13 must then hold, hence $s((\bar{u}_{\pi_i}, u_{\pi_i})) \leq \text{seg}(r)/w$. From Claim 20, $\text{seg}(r) \leq s(\bar{u}_{\pi_l}, u_{\pi_l})$. Hence $s(\bar{u}_{\pi_i}, u_{\pi_i}) \leq s(\bar{u}_{\pi_l}, u_{\pi_l})/w \leq c(P)/w^j$.
- *Case 2:* q'_i was established as the result of execution of line 3 of **NonFree**. First, note that this implies that $\bar{u}_{\pi_l} \neq \bar{u}_{\pi_i}$. This is because after request u_{π_l} the terminal \bar{u}_{π_l} becomes low-free: from Property 18(iii) \bar{u}_{π_l} will never become low-left again. Since q'_i is established because of line 3 in **NonFree**, \bar{u}_{π_i} must be low-left at the time u_{π_i} is requested, which means that \bar{u}_{π_i} and \bar{u}_{π_l} cannot coincide, thus $\bar{u}_{\pi_l} \prec \bar{u}_{\pi_i}$.

In addition, from the if-condition of line 2, it must be the case then that there exists a terminal u such that $u_{\pi_i} \prec u \preceq u_{\pi_l}$ and at the time right before u_{π_i} is requested, the interval (\bar{u}_{π_i}, u) is a left interval. We will consider further cases about how (\bar{u}_{π_i}, u) came to become a left interval.

Subcase 2(i): (\bar{u}_{π_i}, u) became left as a result of line 19 of **NonFree** or a result of line 5 of **Free**, namely at request \bar{u}_{π_i} . Then from the condition of line 17 of **NonFree** (resp. line 4 of **Free**) we have that $s(\bar{u}_{\pi_i}, u) \leq \text{seg}(r)/w$, where r is the run to which \bar{u}_{π_i} was assigned. From Claim 20 and the induction hypothesis it follows that

$$s(\bar{u}_{\pi_i}, u_{\pi_i}) \leq s(\bar{u}_{\pi_i}, u) \leq \text{seg}(r)/w \leq s(\bar{u}_{\pi_l}, u_{\pi_l})/w \leq \frac{c(P)}{w^j}.$$

Subcase 2(ii) (\bar{u}_{π_i}, u) became a left interval as a result of line 5 of **NonFree**. In this case, from Property 10 there exists a terminal v for which the following hold:

- $\bar{u}_{\pi_l} \prec v \preceq \bar{u}_{\pi_i}$;
- At the time v is requested no other terminal except u has been requested in the interval (v, u) ;
- Interval (v, u) becomes left as a result of execution of line 5 in algorithm **Free** or line 19 in algorithm **NonFree**.

If (v, u) became left as a result of line 5 of algorithm **Free**, then the if-condition of line 4 holds, hence if r denotes the run that v joins, we have that $s(v, u) \leq \text{seg}(r)/w$, and run r is contained within the interval $(\bar{u}_{\pi_l}, u_{\pi_l})$ (from Claim 20), hence

$$s(\bar{u}_{\pi_i}, u_{\pi_i}) \leq s(v, u) \leq s(\bar{u}_{\pi_l}, u_{\pi_l})/w. \tag{5}$$

If (v, u) became left as a result of line 19 of **NonFree**, then the if-condition of line 17 holds, hence if r denotes the run whose member v becomes, we have that $s(v, u) \leq \text{seg}(r)/w$. Once again, run r is contained within the interval $(\bar{u}_{\pi_l}, u_{\pi_l})$ (from Claim 20). It follows that (5) holds, and from the induction hypothesis we have that $s(\bar{u}_{\pi_i}, u_{\pi_i}) \leq \frac{c(P)}{w^3}$.

We now return to consider the case in which $(\bar{u}_{\pi_l}, u_{\pi_l})$ was established as a result of the execution of line 14 of algorithm **NonFree**. In this case, $(\bar{u}_{\pi_l}, u_{\pi_l})$ becomes right after q'_l is established. It follows easily (from lines 7–11 of **NonFree**) that there exist two terminals v, v' with the following properties:

- $\bar{u}_{\pi_l} \prec v \prec v' \preceq u_{\pi_l}$;
- At the time v is requested, (v, v') becomes a free interval;
- the interval $(\bar{u}_{\pi_i}, u_{\pi_i})$ is contained in (v, v') .

The above shows that $(\bar{u}_{\pi_i}, u_{\pi_i})$ is contained within a free interval (v, v') which is in turn contained in $(\bar{u}_{\pi_l}, u_{\pi_l})$. We can then apply the same arguments as above (namely, repeat cases (1) and (2)) essentially replacing the interval $(\bar{u}_{\pi_l}, u_{\pi_l})$ with the “shorter” (i.e., less costly wrt the cost of their segment) interval (v, v') . \square

Proof of Claim 15. By way of contradiction. Consider the first run r , in the order in which the runs are established, which does not have the required property. This implies that there would exist a run r' , established earlier than r and for which one of the following hold: (i) $\text{rep}(r') \prec \text{rep}(r)$ and $l(r) \prec \text{rep}(r')$; or (ii) $l(r) \prec l(r') \prec \text{rep}(r') \prec \text{rep}(r)$. Indeed, if neither of the above held, then either r would be contained within r' or r and r' would consist of disjoint segments, which means that all edges in $\text{seg}(r)$ would have the same depth after r is established, a contradiction. If (i) holds, then when r is established, $\text{rep}(r')$ has been requested earlier, hence $\text{seg}(r)$ and $\text{seg}(r')$ would be disjoint, which contradicts the statement of (i). If, on the other case, (ii) holds, then when r is established, $l(r')$ has already been requested, hence r would be contained within r' , which again contradicts the statement of (ii). Since both cases leads to a contradiction, we conclude that all edges in $\text{seg}(r)$ have the same depth, after r is established. \square

Details in the proof of Lemma 16. Remains to consider the case that $(l(r), \text{rep}(r))$ became free as a result of line 9 of **NonFree**. Then there must exist a terminal u with $l(r') \preceq u \prec l(r)$ such that when $l(r)$ was requested, the interval $(u, \text{rep}(r))$ was a right interval. From property 19 we know that there must exist a terminal v such that $\text{rep}(r) \preceq v \prec \text{rep}(r')$ and the interval (u, v) became right as a result of either line 15 of **NonFree** or line 8 of **Free**. Let r'' denote the run whose member v becomes, then from the if-conditions of line 14 of **NonFree** and line 7 of **Free** we have that $s(u, v) \leq \text{seg}(r'')/w$. Once again $(l(r), \text{rep}(r))$ is contained within the interval (u, v) , thus $\text{seg}(r) \leq s(u, v)$; in addition, from property 9 r'' is contained within run r' , hence $\text{seg}(r'') \leq \text{seg}(r')$. Combining the above inequalities we deduce that $\text{seg}(r) \leq \text{seg}(r')/w$.

Bounding the cost $C_{3,i}$.

We can bound $C_{3,i}$ using arguments similar to the bound for $C_{2,i}$. We say that a terminal u_{π_i} contributes edge $e \in P$ (i.e., an edge in the backbone of the comb) if the connection path q_i for u_{π_i} includes edge e . We let q''_i denote the subpath of q_i which consists of edges in the backbone,

then $C_{3,i} = c(q_i'')$. Let Y denote the subset of the set of terminals in the comb K' which consists of terminals with non-zero contribution to C_2 . Again, we can think of edges in q_i'' being “bought”, as the connection path between the terminal and its mate is *established*, at the precise moment $u_{\pi_i} \in Y$ is requested. In this view, every time an edge in P is contributed due to such an assignment, we say that the *depth* of the edge increases by 1. Recall that u_{π_i} contributes to C_3 when it is assigned a mate as a result of one of lines 8 and 18 in `NonFree`.

Similar to Claim 13 we can argue that we can partition Y into sets Y_1, Y_2, \dots such that Y_i consists of terminals of depth i . The following lemma follows along the lines of Lemma 14, using properties symmetric to properties used in the proof of Lemma 14 (more precisely, we rely on Property 19 instead of Property 10 and Property 19 instead of Property 10).

Lemma 21 *For a terminal $u_{\pi_i} \in Y_j$, $C_{3,i} \leq \frac{c(P)}{w^{j-1}}$.*

Bounding C_2 and C_3

Concerning C_2 , denote by $c_2(X_j)$ the contribution of the set of terminals X_j to C_2 . Recall that paths of the form q_l' for all $u_{\pi_l} \in X_j$ are all edge-disjoint, for a fixed j . Hence $c_2(X_j) \leq \alpha c(P)$. Combining this with Lemma 14 we have

$$c_2(X_j) = \min\{\alpha c(P), \frac{\alpha c(P)}{w^{j-1}} |X_j|\}$$

therefore,

$$C_2 = \sum_j c_2(X_j) = \sum_j \min\{\alpha c(P), \frac{\alpha c(P)}{w^{j-1}} |X_j|\} \quad (6)$$

Note that (6) is maximized when $|X_j| = w^{j-1}$, for all $j \geq 2$. We then consider two cases: if $w = x$, then $C_2 \in O(\alpha x \cdot c(P))$. If $w = \alpha$, then $C_2 = O(\alpha \cdot \frac{\log k'}{\log \alpha})$. Therefore

$$C_2 = O(\max\{\alpha \cdot xc(P), \alpha y \cdot c(P)\}). \quad (7)$$

We can apply a very similar argument with the analysis of cost C_2 to bound C_3 . Denote by $c_3(Y_j)$ the contribution of Y_j to C_3 . Since the paths of the form q_l'' for all $u_{\pi_l} \in Y_j$ are edge-disjoint, for fixed j , we have that $c_3(Y_j) \leq c(P)$. In combination with Lemma 21 we have

$$c_3(Y_j) = \min\{c(P), \frac{c(P)}{w^{j-1}} |Y_j|\}$$

which in turn gives

$$C_3 = O(\max\{xc(P), yc(P)\}). \quad (8)$$