The Power of Deferral: Maintaining a Constant-Competitive Steiner Tree Online

Albert Gu* Anupam Gupta† Amit Kumar‡

Abstract

In the online Steiner tree problem, a sequence of points is revealed one-by-one: when a point arrives, we only have time to add a single edge connecting this point to the previous ones, and we want to minimize the total length of edges added. Here, a tight bound has been known for two decades: the greedy algorithm maintains a tree whose cost is $O(\log n)$ times the Steiner tree cost, and this is best possible. But suppose, in addition to the new edge we add, we have time to change a single edge from the previous set of edges: can we do much better? Can we, e.g., maintain a tree that is constant-competitive?

We answer this question in the affirmative. We give a primal-dual algorithm that makes only a single swap per step (in addition to adding the edge connecting the new point to the previous ones), and such that the tree's cost is only a constant times the optimal cost. Our dual-based analysis is quite different from previous primal-only analyses. In particular, we give a correspondence between radii of dual balls and lengths of tree edges; since dual balls are associated with points and hence do not move around (in contrast to edges), we can closely monitor the edge lengths based on the dual radii. Showing that these dual radii cannot change too rapidly is the technical heart of the paper, and allows us to give a hard bound on the number of swaps per arrival, while maintaining a constant-competitive tree at all times. Previous results for this problem gave an algorithm that performed an *amortized* constant number of swaps: for each n, the number of swaps in the first n steps was $O(n)$. We also give a simpler tight analysis for this amortized case.

1 Introduction

In the online Steiner tree problem, a sequence of points $0, 1, 2, \ldots, n, \ldots$ is revealed online. When the point i arrives we are told the distances $d(i, j)$ for all $j < i$; the distances between previous points do not change, and we are guaranteed that the distances always satisfy the triangle inequality. The goal is to maintain a tree spanning all the arrivals and having a small cost (which is the sum of the lengths of the tree edges). As is usual in online algorithms, all decisions are irrevocable: once an edge is bought it cannot be removed. This naturally captures a situation where we are building a network, but only have time to add a single edge at a time. The greedy algorithm, upon arrival of the i^{th} vertex, greedily attaches it to its closest preceding point; [IW91, AA93] showed that this algorithm produces a tree that is $O(\log n)$-competitive against the best spanning tree on $\{1, 2, \ldots, n\}$, for every n. They also showed a matching lower bound of $\Omega(\log n)$ on the competitive ratio.

But what if the decisions were not irrevocable? What if, when a new vertex arrived, we were allowed to add a new edge, but also to swap a small number of previously-added edges for new ones? Given the power of hindsight, we could do better—but by how much? Imase and Waxman [IW91] showed a

*Department of Mathematical Sciences, Carnegie Mellon University, Pittsburgh, PA 15213. Supported by a Knaster-McWilliams Scholarship.

†Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213, USA. Supported in part by NSF awards CCF-0964474 and CCF-1016799, and by the CMU-MSR Center for Computational Thinking.

‡Dept. of Computer Science and Engg., IIT Delhi, India 110016.

natural greedy algorithm that maintains a 2-competitive tree and makes at most $O(n^{3/2})$ swaps over the course of the first n arrivals, for every n. Hence the *amortized budget*, the average number of swaps per arrival, is $O(n^{1/2})$. This was substantially improved upon recently, when Megow, Skutella, Verschae, and Wiese [MSVW12, Ver12] gave an algorithm with a constant amortized budget bound. Specifically, given $\varepsilon > 0$, their algorithm maintains a tree that is $(1+\varepsilon)$-competitive against the minimum spanning tree, and performs $O(n/\varepsilon \log 1/\varepsilon)$ swaps in n steps.

Note that both these prior results work in the amortized setting: what if we could only do a constant number of changes per arrival? *In fact, what if we only had time to perform a single swap per timestep: could we maintain a spanning tree that is constant competitive against the best Steiner tree?* The algorithms used in previous papers do not have this property, as there exist instances where a single arrival can cause their algorithms to do a linear number of swaps. The main result of this paper is an affirmative answer to the above question.

Theorem 1.1 (Constant Budget Algorithm) *There is an online algorithm for metric Steiner tree that performs a single edge swap upon each arrival, and maintains a spanning tree with cost at most a constant times that of the optimal Steiner tree on the current set of points.*

In fact, we can maintain a $2^{O(1/\delta)}$-approximate tree and perform only one swap per $1/\delta$ rounds (see Theorem 4.3).

We discuss the ideas behind the algorithm in Section 1.1; at a high level, the algorithm is based on the primal-dual method; our analysis is based on relating the edges in the tree to dual balls, and tracking the changes via changes in these dual values. This dual-based analysis is substantially different from the primal-only analyses used in previous works, and we feel it gives a better insight into the problem.

Our techniques also allow us to give a trade-off between the number of swaps and the competitiveness: in fact, we first show a weaker result that performs a constant number of swaps per arrival and maintains a constant competitive tree (in Section 2 and 3). In Section 4 , we show how refinements of our arguments can reduce the number of swaps and prove Theorem 1.1 and its extension mentioned above.

Our second result is a simpler and improved amortized-budget analysis of the greedy algorithm studied by [IW91, MSVW12]. For any $\varepsilon \in (0, 1]$, consider the online algorithm $\mathcal{B}_{1+\varepsilon}$ that greedily connects each new vertex to the closest previous vertex, and that also swaps a tree edge e for a non-edge f whenever $\mathsf{len}(e) \geq (1+\varepsilon)\,\mathsf{len}(f)$ and $T + f - e$ is also a spanning tree. By construction, $\mathcal{B}_{1+\varepsilon}$ maintains a tree that is $(1+\varepsilon)$-competitive against the best spanning tree.

Theorem 1.2 (Tight Amortized Budget Algorithm) *For any $\varepsilon \in (0, 1]$, the algorithm $\mathcal{B}_{1+\varepsilon}$ makes at most $n \cdot \log_{1+\varepsilon} 4 \leq 2n/\varepsilon$ swaps over the course of n arrivals.*

This result is asymptotically tight, as a lower bound of $\Omega(n/\varepsilon)$ is known [Ver12]. The previous best amortized bound was $O(n/\varepsilon \log 1/\varepsilon)$ given by [MSVW12], for a variant of $\mathcal{B}_{1+\varepsilon}$ which did not perform all possible $(1+\varepsilon)$-swaps. The proof of Theorem 1.2 appears in Section 5. In Section 5.2 we give an instance where algorithm $\mathcal{B}_2$ needs at least $1.25n$ swaps; no instances were known earlier where more than n swaps were needed for $\varepsilon = 1$.

1.1 The Constant-Budget Algorithm: Ingredients, Intuition, and Ideas

One of the main difficulties in analyzing "primal" algorithms that directly deal with edge lengths of the current tree is that swaps in the tree are not local: two close-by edges may be swapped for two edges that are far from each other, and spatio-temporal relationships between them become difficult to reason about. Instead we take a primal-dual approach that talks about duals around vertices—since vertices

do not move, we can argue about them more easily. The rest of this section outlines the steps and the intuition behind them; the algorithm itself is summarized in Section 1.1.1.

Say vertices $\{0, 1, 2, \ldots, i-1\}$ had arrived previously, and now vertex i arrives. We first run a *clustering process* on the vertices in $[i] = \{0, 1, \ldots, i\}$ (described in Section 2.1): this is similar to the moat-growing process in the Agrawal-Klein-Ravi/Goemans-Williamson primal-dual algorithm, but here we grow the clusters in discrete exponentially-sized steps. This clustering process defines integer "ranks" $\boldsymbol{\rho}_i(v)$ for vertices $v \in [i]$. Then we run a *tree-forming process* using these vertex ranks, which outputs a tree $T_i^\star$ on $[i]$ that is a constant-approximation to the optimal Steiner tree on $[i]$. This is where ranks are useful: we ensure a correspondence between lengths of tree edges and vertex ranks—a vertex of rank k corresponds to some tree edge of length $\approx \alpha^k$ (for some small constant α), and so we can shift our focus from tracking edge lengths to tracking vertex ranks. Our clustering ensures that the ranks of existing vertices never increase as future arrivals occur, and also that the total number of rank decrements for all vertices over the course of i arrivals is $O(i)$. Furthermore, the tree-formation satisfies a Lipschitz property: if s rank decrements occur due to the arrival of vertex i, then $T_i^\star$ can be obtained from $T_{i-1}^\star$ by adding an edge connecting vertex i to its closest vertex in $[i-1]$, and then performing at most s edge swaps. Putting these two facts together gives an $O(1)$-*amortized*-budget and $O(1)$-competitive algorithm.

But we had promised a constant-*worst-case*-budget algorithm; we cannot directly use the algorithm above, since some arrivals may cause the ranks of a linear number of vertices to drop. However, we can fix things, and this is where the advantages of our dual-based approach become apparent. In addition to the ranks, we also maintain *virtual ranks* $\boldsymbol{\nu}_i(v)$ for vertices $v \in [i]$, which also drop monotonically, and which are upper bounds on the ranks. And we run the tree-forming process on these virtual ranks to get the actual tree T_i. We want the virtual ranks to be close to the real ranks, but also not to change too drastically upon arrivals—so when vertex i arrives, we define $\boldsymbol{\nu}_i(\cdot)$ in such a way that

- $\boldsymbol{\nu}_i(v) \le \boldsymbol{\nu}_{i-1}(v)$ for all $v \in [i-1]$ (virtual ranks are also monotone decreasing),
- $\boldsymbol{\nu}_i(v) \ge \boldsymbol{\rho}_i(v)$ for all $v \in [i-1]$ (virtual ranks are upper bounds for actual ranks),
- $\boldsymbol{\nu}_i(i) = \boldsymbol{\rho}_i(i)$ (the virtual rank equals the actual rank for new arrivals), and
- $\|\boldsymbol{\nu}_i - \boldsymbol{\nu}_{i-1}\|_1 \le O(1)$ (the number of virtual rank changes is only a constant).

By this last property and the Lipschitz-ness of our tree-formation algorithm, the number of edge swaps to get from T_{i-1} to T_i is a constant.

But what about the competitiveness of the tree? The cost of our tree T_i is $\approx \sum_{v \in [i]} \alpha^{\boldsymbol{\nu}_i(v)}$, whereas the ideal tree T_i^* has cost $\approx \sum_{v \in [i]} \alpha^{\boldsymbol{\rho}_i(v)}$. Since we want the former sum to be close to the latter, we define the virtual ranks by decrementing it for those nodes v for which $\boldsymbol{\nu}_{i-1}(v) > \boldsymbol{\rho}_i(v)$ and the numerical value of $\boldsymbol{\nu}_{i-1}(v)$ is the largest. The technical heart of the paper lies in showing that this way of maintaining virtual ranks gives us a constant approximation tree at all times; the analysis is given in Section 3.

1.1.1 The Algorithm in a Nutshell

Most of the above description is intuition and analysis. Our algorithm is simply the following. When vertex i arrives,

(i) run clustering to get ranks $\boldsymbol{\rho}_i(v)$ for all $v \in [i]$,
(ii) define the virtual ranks $\boldsymbol{\nu}_i(v)$ using the simple greedy rule described above, and
(iii) run tree-formation on the virtual rank function $\boldsymbol{\nu}_i$ to get the tree T_i.

We emphasize that the clustering and tree-formation algorithms are just two halves of the Agrawal-Klein-Ravi/Goemans-Williamson moat-growing primal-dual algorithm, by viewing the moat-growing and edge-additions separately.

1.2 Related Work

The online Steiner tree problem was studied by Imase and Waxman [IW91], who proved $\Theta(\log n)$-competitiveness for the problem when no swaps are allowed. The proof was simplified by Alon and Azar [AA93], who also gave an $\Omega(\frac{\log n}{\log \log n})$-lower bound for planar point sets. Imase and Waxman also gave an algorithm that maintained a 2-competitive tree and used an amortized budget of $O(\sqrt{n})$ over n steps. They also considered the model where vertices could leave the system: for this problem they also gave an algorithm with the same amortized budget, but a weaker 8-competitiveness.

The primal-dual method has been a powerful tool in offline algorithm design; see, e.g., the treatment in the textbooks [Vaz01, WS11]. This technique was used in original Steiner forest papers of Agrawal, Klein, and Ravi [AKR95] and Goemans and Williamson [GW95]. The popularity and power of primal-dual in online algorithm design is more recent (see, e.g., the monograph of Buchbinder and Naor [BN07] for many successes). Some early uses of primal-dual ideas in online algorithms can be seen in the online Steiner forest analysis of Awerbuch, Azar, and Bartal [AAB04], and the algorithm and analysis of Berman and Coulston [BC97].

Apart from the results of [IW91, MSVW12], the idea of making a small number of alterations to maintain a good solution in online settings had been studied for other problems. See, e.g., the works of [Wes00, AGZ99, AAPW01, SSS09, SV10, EL11] which study alterations in the context of online scheduling and routing problems. Ashwinkumar [Var11] gives optimal results for the problem of maximizing the value of an independent set in the intersection of p matroids, where the elements of the universe arrive online; in his model, an element can get added to the set and subsequently canceled (for a price), but an element that has been dropped can never be added subsequently. However, the ideas in these papers seem to be orthogonal to ours.

Our results are also related, at a conceptual level, to those of Kirsch and Mitzenmacher [KM07], Arbitman et al. [ANS09] and related works on *de-amortizing* data structures (such as cuckoo hash-tables) by maintaining a suitable auxiliary data structure (e.g., a queue or a "stash"); this allows them to achieve constant update times with overwhelming probability. Since our goal is slightly different—to achieve low cost—our de-amortization works by delaying certain updates via a priority queue.

2 A Constant-Swaps Algorithm

It will be useful to first prove a slightly weaker version of Theorem 1.1, which will introduce the main ideas.

Theorem 2.1 *Let $\alpha \geq 6$. There is an online algorithm that makes at most $K = 2\alpha^2$ swaps upon each vertex arrival, and for every n, maintains a tree T_n with cost at most $C = \frac{2\alpha^5}{(\alpha-1)^2}$ times the cost of an optimal Steiner tree on the first n points.*

We describe the algorithm of Theorem 2.1 in this section, and give its analysis in Section 3. We then show how to modify the algorithm slightly to trade off swaps for performance, and hence get a single-swap algorithm in Section 4.

In order to describe the online algorithm, let us lay down some notation. Let $[n]$ denote the integers $\{0, 1, 2, \ldots, n\}$, and $[i \ldots j]$ denote the integers $\{i, i+1, \ldots, j\}$. We associate the arriving vertices in the metric space with the integers: we start off with the root vertex 0, and the n^{th} arriving vertex is called n; and hence the root and the first n arriving vertices are identified with the set $[n]$.

For $i \geq 1$, when the i^{th} vertex arrives, we begin *round* i. Round i involves three steps:

(a) running a clustering algorithm on the vertex set $[i]$ (itself involving several "phases") which defines the rank function $\boldsymbol{\rho}_i : [i] \to \mathbb{Z}_{\geq 0}$ (described in Section 2.1),
(b) getting a virtual rank function $\boldsymbol{\nu}_i$ from the actual rank function $\boldsymbol{\rho}_i$ (as in Section 2.3), and
(c) finally constructing the tree T_i given the virtual rank function (described in Section 2.2).

This concludes round i. We denote $\mathcal{R}_i$ to be the run of the clustering algorithm for round i; as mentioned above it has several phases. Let $\mathsf{opt}([i])$ denote the cost of the optimal Steiner tree on $[i]$. We want to relate the cost of T_i to $\mathsf{opt}([i])$.

For a vertex $x \in [n]$ and subset $S \subseteq [n]$, define $d(x, S) := \min_{s \in S} d(x, s)$. Similarly, let $d(S, T) = \min_{s \in S} d(s, T)$. For *radius* $r \geq 0$, define the *ball* $B(x, r) = \{y \in [n] \mid d(x, y) \leq r\}$. For a set $S \subseteq [n]$ and radius $r \geq 0$, let $B(S, r) = \cup_{s \in S} B(s, r) = \{y \mid d(y, S) \leq r\}$.

2.1 The Clustering Procedure

Let α be a universal constant ≥ 6. We assume that all inter-vertex distances are at least 2α, else we can scale things up. The run $\mathcal{R}_i$ on vertex set $[i]$ starts off with trivial clustering $\mathscr{C}_i(0)$ containing $i + 1$ disjoint *clusters* $\{\{0\}, \{1\}, \ldots, \{i\}\}$. At the beginning of phase t of $\mathcal{R}_i$, we have a clustering $\mathscr{C}_i(t-1)$, which is a partition of $[i]$, produced by phase $t-1$. The invariant is that two distinct clusters $C, C' \in \mathscr{C}_i(t-1)$ have $d(C, C') \geq 2\alpha^t$. (Since all distances are at least 2α, the clustering $\mathscr{C}_i(0)$ satisfies this invariant for the first phase $t = 1$.) We start with the clusters in $\mathscr{C}_i(t-1)$, and while there exist two clusters C and C' that satisfy $d(C, C') < 2\alpha^{t+1}$, we *merge* these two clusters into one (i.e., we remove C and C' and add in $C \cup C'$). Note the resulting clustering, which we call $\mathscr{C}_i(t)$, satisfies minimum inter-cluster distance $2\alpha^{t+1}$ by construction. This defines clusterings $\mathscr{C}_i(t)$ for all $t \in \mathbb{Z}_{\geq 0}$.

For each cluster $C \in \mathscr{C}_i(t)$, define C's *leader* as the vertex in C with the least index. Note that for $t = 0$ each vertex is a leader, and as t gets very large, only the vertex 0 remains the leader. Let the *rank* $\boldsymbol{\rho}_i(x)$ of vertex $x \in [1 \ldots i]$ be the largest t for which x is the leader of its cluster in $\mathscr{C}_i(t)$ (i.e., at the end of phase t); define the rank of vertex 0 as $\boldsymbol{\rho}_i(0) := \infty$. Finally, for a value $j \geq 0$ and a function $f : [j] \to \mathbb{Z}_{\geq 0}$, define its weight

$$\mathsf{Wt}_j(f) := \sum_{l=1}^{j} \alpha^{f(l)}. \tag{2.1}$$

(Note the definition of $\mathsf{Wt}_j(\cdot)$ *does not* include the index 0 in the sum.)

2.1.1 Properties of the Ranks and Clusterings

We now prove some useful properties about ranks and clusterings: these are not needed for the algorithm, only for the analysis.

Lemma 2.2 *For any $i \geq 1$, we have $\mathsf{Wt}_i(\boldsymbol{\rho}_i) \leq \frac{\mathsf{opt}([i])}{\alpha - 1}$.*

Proof. Recall the dual LP for the natural relaxation of the Steiner tree problem. We consider the graph $G = ([i], \binom{[i]}{2})$, with the length of the edge (i, j) being $d(i, j)$. The dual says:

$$\max \sum_{S} y_S \tag{DLP_{ST}}$$

$$\sum_{S : |S \cap \{j,l\}| = 1} y_S \leq d(j, l) \qquad \forall j, l \in [i] \tag{2.2}$$

$$y_S \geq 0.$$

We shall define a feasible solution y such that $\sum_S y_S \geq \mathsf{Wt}_i(\boldsymbol{\rho}_i) \cdot (\alpha - 1)$; by weak duality this will imply that $\mathsf{opt}([i]) \geq \mathsf{Wt}_i(\boldsymbol{\rho}_i) \cdot (\alpha - 1)$. Consider the clustering defined by the run $\mathcal{R}_i$. For every $t \geq 0$, and every cluster $C \in \mathscr{C}_i(t)$ such that vertex $0 \notin C$, we define $y_C = \alpha^t(\alpha - 1)$. For all other sets S, set $y_S = 0$.

To check feasibility, consider any edge (j, l), and let t be the last phase such that j and l lie in different clusters in $\mathscr{C}_i(t)$. For all phases $t' \leq t$, we contribute $2\alpha^{t'}(\alpha - 1)$ towards the left hand side of (2.2). Moreover, $d(j, l) \geq 2\alpha^{t+1}$, because they are in different clusters in $\mathscr{C}_i(t)$). Hence, the LHS of (2.2) is $\sum_{t'=0}^{t} 2\alpha^{t'}(\alpha - 1) = 2(\alpha^{t+1} - 1) \leq d(j, l)$.

Now consider the objective function value $\sum_C y_C$, which we claim is at least $\mathsf{Wt}(\boldsymbol{\rho}_i) \cdot (\alpha - 1)$. Indeed, consider the following map g from $[1 \ldots i]$ to $\cup_t \mathscr{C}_i(t)$: for vertex $j \in [1 \ldots i]$, let $g(j)$ be the cluster in $\mathscr{C}_i(\boldsymbol{\rho}_i(j))$ for which j is the leader. Since this is a 1-1 mapping,

$$\sum_C y_C \geq \sum_{j=1}^{i} y_{g(j)} = \sum_{j=1}^{i} \alpha^{\boldsymbol{\rho}_i(j)}(\alpha - 1) = \mathsf{Wt}_i(\boldsymbol{\rho}_i) \cdot (\alpha - 1).$$

This proves the lemma. ∎

The following lemma shows that if a set of vertices S is far from the rest of the vertices, then until a high enough phase t, any cluster in $\mathscr{C}_i(t)$ will be a subset of S or of $[i] \setminus S$.

Lemma 2.3 *Suppose $S \subseteq [i]$ such that $B(S, 2\alpha^t) \cap [i] = S$ for some value t. Then for any phase $k \leq t - 1$ and any cluster C in $\mathscr{C}_i(k)$, either $C \subseteq S$ or $C \cap S = \emptyset$.*

Proof. Proof by induction on the phases of run $\mathcal{R}_i$. For the base case, each cluster in $\mathscr{C}_i(0)$ is a singleton and the claim holds. Now suppose for some phase $k < t - 1$, each $C \in \mathscr{C}_i(k)$ either lies within S or is disjoint from it. Note that the assumption $B(S, 2\alpha^t) \cap [i] = S$ is same as saying $d(S, [i] \setminus S) > 2\alpha^t$. So, if $C, C' \in \mathscr{C}_i(k)$ satisfy $C \subseteq S$ and $C' \cap S = \emptyset$, then $d(C, C') > 2\alpha^t \geq 2\alpha^{k+1}$, and we will not merge these two clusters in $\mathscr{C}_i(k + 1)$. Hence, the claim holds for phase $k + 1$ as well. ∎

The clusterings produced by the runs $\mathcal{R}_{i-1}$ on the set $[i - 1]$, and $\mathcal{R}_i$ on $[i]$ are closely related: the clustering $\mathscr{C}_{i-1}(t)$ is a refinement of the clustering $\mathscr{C}_i(t)$, as the next lemma shows.

Lemma 2.4 *For a cluster $C \in \mathscr{C}_i(t)$, exactly one of the following holds:*

(a) vertex $i \notin C$: in this case C is also a cluster in $\mathscr{C}_{i-1}(t)$, or
(b) vertex $i \in C$: in this case $C = \{i\} \cup C_1 \cup C_2 \cup \ldots \cup C_p$ for some $p \geq 0$ clusters $C_1, \ldots, C_p \in \mathscr{C}_{i-1}(t)$.

(In the latter case, note that p may equal 0, in which case $C = \{i\}$.)

Proof. The proof is again by induction on t. At $t = 0$, this is true because all clusters are singleton elements. Suppose the claim of the lemma holds for some phase $t \geq 0$. Let $C_1, \ldots, C_l$ be the clusters in $\mathscr{C}_{i-1}(t)$. By the induction hypothesis we can renumber these clusters in $\mathscr{C}_{i-1}(t)$ such that the clusters in $\mathscr{C}_i(t)$ are $\{i\} \cup C_1 \cup \ldots \cup C_p, C_{p+1}, \ldots, C_l$ for some $p \geq 0$. We construct an auxiliary graph $G_{i-1}(t)$ with vertices corresponding to the clusters in $\mathscr{C}_{i-1}(t)$, and join two clusters C_i, C_j by an edge in $G_{i-1}(t)$ if $d(C_i, C_j) \leq 2\alpha^{t+2}$. By the definition of the clustering process, the clustering $\mathscr{C}_{i-1}(t + 1)$ is obtained by taking the unions of the clusters in each connected component of $G_{i-1}(t)$. We can define $G_i(t)$ similarly, and again, the clusters in $\mathscr{C}_i(t)$ correspond to connected components of $G_i(t)$.

Now observe that if $i, j > p$, we have an edge between (C_i, C_j) in $G_i(t)$ exactly when we have this edge in $G_{i-1}(t)$. And if $i \leq p, j > p$, and we have the edge $(C_i, C_j) \in G_{i-1}(t)$, then we have an edge between C_j and $\{i\} \cup C_1 \cup C_2 \cup \ldots \cup C_p$ in $G_i(t)$. The hypothesis for $t + 1$ follows from these facts. ∎

Lemma 2.5 *If $j, l \in [i-1]$ lie in a common cluster in $\mathscr{C}_i(t)$, then in run $\mathcal{R}_{i-1}$ they lie in a common cluster either in $\mathscr{C}_{i-1}(t)$ or in $\mathscr{C}_{i-1}(t+1)$.*

Proof. Suppose $j, l \in [i-1]$ lie in a common cluster in $\mathscr{C}_i(t)$, but belong to different clusters of $\mathscr{C}_{i-1}(t)$. By Lemma 2.4, there are a set of clusters $C_1, \ldots, C_p \in \mathscr{C}_{i-1}(t)$ such that $C = \{i\} \cup C_1 \cup \ldots \cup C_p$ is a cluster in $\mathscr{C}_i(t)$, and these two vertices $j, l \in C$. Consider a cluster C_q for $1 \leq q \leq p$. We claim that $d(i, C_q) \leq 2\alpha^{t+1}$. Suppose not. The fact that C_q is a cluster in $\mathscr{C}_{i-1}(t)$ implies that $d(C_q, [i-1] \setminus C_q) \geq 2\alpha^{t+1}$. Then, $d(C_q, [i] \setminus C_q) \geq 2\alpha^{t+1}$ as well. But then, by Lemma 2.3, there cannot be a cluster in $\mathscr{C}_i(t)$ containing a vertex from both C_q and $[i] \setminus C_q$, which contradicts the assumption about the cluster $C \in \mathscr{C}_i(t)$. Now applying the triangle inequality, for any $q, q' \in [1 \ldots p]$, the distance $d(C_q, C_{q'}) \leq d(i, C_q) + d(i, C_{q'}) \leq 4\alpha^{t+1} \leq 2\alpha^{t+2}$. (Recall that $\alpha \geq 2$.) Consequently, we will merge C_q and $C_{q'}$ in phase $t+1$ of the run $\mathcal{R}_{i-1}$. Indeed, all the vertices in $C_1 \cup \ldots \cup C_q$ will lie in a common cluster in $\mathscr{C}_{i-1}(t+1)$, which proves the lemma. ∎

Corollary 2.6 (Ranks are Monotone) *For $j \in [i-1]$, $\boldsymbol{\rho}_i(j) \leq \boldsymbol{\rho}_{i-1}(j) \leq \boldsymbol{\rho}_i(j) + 1$. Hence the rank of a vertex is non-increasing as a function of i.*

Proof. If j is no longer the leader of its cluster in $\mathscr{C}_{i-1}(t)$ (because some $l < j$ lies in its cluster), then since $\mathscr{C}_{i-1}(t)$ is a refinement of $\mathscr{C}_i(t)$, l lies in j's cluster in $\mathscr{C}_i(t)$ too. Hence $\boldsymbol{\rho}_i(j) \leq \boldsymbol{\rho}_{i-1}(j)$.

Now suppose j loses leadership of its cluster during phase t of $\mathcal{R}_i$, i.e., $\boldsymbol{\rho}_i(j) = t-1$. So there exists a vertex $l < j$ which lies in the cluster of $\mathscr{C}_i(t)$ containing j. Lemma 2.5 says j, l must share a cluster in $\mathscr{C}_{i-1}(t+1)$, making $\boldsymbol{\rho}_{i-1}(j) \leq t+1$. ∎

Claim 2.7 (Initial Ranks) *If the initial rank of vertex i (i.e., i's rank in run $\mathcal{R}_i$) is r, then the distance $d(i, [i-1]) \in [2\alpha^{r+1}, 2\alpha^{r+2})$.*

Proof. Let j be the closest vertex in $[i-1]$ to i. Then if $d(i,j) < 2\alpha^{r+1}$, then i, j would be part of the same cluster in $\mathscr{C}_i(r)$ and hence $\boldsymbol{\rho}_i(i) < r$. Similarly, if $d(i,j) \geq 2\alpha^{r+2}$, then Lemma 2.3 shows that i would form a singleton cluster in $\mathscr{C}_i(r+1)$ and hence $\boldsymbol{\rho}_i(i) \geq r+1$. ∎

2.2 The Tree-Building Process

In this section, we explain the second ingredient of our algorithm: given the rank function $\boldsymbol{\rho}_i$, how to build a tree $T_i = ([n], E_i)$. In fact, we give a more general process that takes a function from a wider class of "admissible" functions and produces a tree for such a function. We want the trees T_i and T_{i-1} to look similar, so our tree-building procedure assumes access to T_{i-1} when building tree T_i.

Recall that for a vertex $j \leq i$, $\boldsymbol{\rho}_i(j)$ denotes its rank in the primal-dual process $\mathcal{R}_i$. Moreover, define $\texttt{Init}(j) := \boldsymbol{\rho}_j(j)$ to be the initial rank of j (when it arrived in round j); define $\texttt{Init}(0) = \infty$. We say that a function $\boldsymbol{\beta} : [i] \to \mathbb{Z}_{\geq 0}$ is *admissible* if $\boldsymbol{\beta}(j) \in [\boldsymbol{\rho}_i(j) \ldots \texttt{Init}(j)]$ for all $j \in [i]$. (Thus the rank function $\boldsymbol{\rho}_i$ is always admissible.) For a set $S \subseteq [i]$, the *head of S* with respect to the function $\boldsymbol{\beta}$ is defined to be the vertex $j \in S$ with highest $\boldsymbol{\beta}(j)$ value (in case of ties, say, choose the lowest numbered vertex among these to be the head). [1]

A tree $T = ([i], E_T)$ is defined to be *valid with respect to $\boldsymbol{\beta}$* if we can partition the edge set E_T into sets $E_T^1, E_T^2, \ldots, E_T^r$ such that the following two conditions are satisfied for each $l \in [1 \ldots r]$:

(i) Let $E_T^{\leq l}$ denote $E_T^1 \cup \cdots \cup E_T^l$. For any connected component of $E_T^{\leq l}$, let j be the head of this component. Then we require $\boldsymbol{\beta}(j) \geq l$.
(ii) Each edge in E_T^l has length at most $2\alpha^{l+1}$.

[1] Since $\boldsymbol{\beta}(0) = \infty$, the root vertex will always be the head of the component containing it.

Lemma 2.8 *Let T be any tree valid with respect to $\boldsymbol{\beta}$. Then the total cost of T is at most $2\frac{\alpha^3}{\alpha-1}\cdot \mathsf{Wt}_i(\boldsymbol{\beta})$.*

Proof. For any $l \in [1 \dots r]$, there must be at least $|E_T^l| + 1$ connected components in $E_T^{\leq(l-1)}$. The cost of each edge in E_T^l can be charged to the heads of the components of $E_T^{\leq(l-1)}$, except for the root vertex 0. Each of these head vertices have $\boldsymbol{\beta}(j)$ values at least $l-1$ by condition (i) of validity. Now any vertex $j \neq 0$ is charged by some E_T^l only if $l \leq \boldsymbol{\beta}(j)+1$, and since each edge in E_T^l has length at most $2\alpha^{l+1}$ (by condition (ii) of validity), the total charge to j is at most

$$\sum_{l=1}^{\boldsymbol{\beta}(j)+1} 2\alpha^{l+1} \leq 2\frac{\alpha^3}{\alpha-1}\cdot\alpha^{\boldsymbol{\beta}(j)}.$$

Summing over $j \neq 0$ and using the definition of $\mathsf{Wt}_i(\cdot)$ from (2.1) completes the proof. ∎

We now prove a Lipschitz property of the $\boldsymbol{\beta}$ function: namely, if we decrement some coordinates of an admissible function $\boldsymbol{\beta}$ to get another admissible function $\boldsymbol{\beta}'$ at Hamming distance $\|\boldsymbol{\beta}-\boldsymbol{\beta}'\|_H$, then we can change a tree T valid for $\boldsymbol{\beta}$ into a tree T' valid for $\boldsymbol{\beta}'$ by making at most $\|\boldsymbol{\beta}-\boldsymbol{\beta}'\|_H \leq \|\boldsymbol{\beta}-\boldsymbol{\beta}'\|_1$ swaps. (The Hamming distance $\|\boldsymbol{\beta}-\boldsymbol{\beta}'\|_H$ is the number of locations $j \in [i]$ such that $\boldsymbol{\beta}(j) \neq \boldsymbol{\beta}'(j)$.)

Lemma 2.9 *Let $\boldsymbol{\beta}$ be an admissible function and $T = ([i], E_T)$ be a valid tree with partition $(E_T^1, \dots, E_T^r)$. Let $j^\star \in [i]$, and suppose $\boldsymbol{\beta}'$ satisfies $\boldsymbol{\beta}'(j) = \boldsymbol{\beta}(j)$ if $j \neq j^\star$, and $\boldsymbol{\beta}'(j^\star) \leq \boldsymbol{\beta}(j^\star) - 1$. Assume that $\boldsymbol{\beta}'$ is also admissible (i.e., $\boldsymbol{\rho}_i(j^\star) \leq \boldsymbol{\beta}'(j^\star)$). Then there is a valid tree $T' = ([i], E_{T'})$ with respect to $\boldsymbol{\beta}'$ such that $|E_{T'} \triangle E_T| \leq 2$.*

Proof. For brevity, let $l^\star := \boldsymbol{\beta}'(j^\star) + 1$, and hence $\boldsymbol{\beta}(j^\star) \geq l^\star$. Let us define the tree T' as follows. For values $l \leq l^\star - 1$, define $E_{T'}^l := E_T^l$. Condition (ii) remains satisfied for these values of l since the edge sets are unchanged; moreover, since $l \leq l^\star - 1 \Rightarrow l \leq \boldsymbol{\beta}'(j^\star)$, even if $j^\star$ happened to be the head of a component of $E_{T'}^{\leq l}$, condition (i) would be satisfied.

Next, we initialize set $E_{T'}^{l^\star}$ to contain all the edges in $E_T^{l^\star}$. It may however happen that $j^\star$ was the head of a connected component C in $E_{T'}^{\leq l^\star}$; since $\boldsymbol{\beta}'(j^\star) < l^\star$, this component C would now violate condition (i). In this case, we claim that there must be some vertex $j \notin C$ such that $d(j, C) \leq 2\alpha^{l^\star+1}$. Suppose not; then Lemma 2.3 implies that there is a vertex $v \in C$ such that $\boldsymbol{\rho}_i(v) \geq l^\star$, and so $\boldsymbol{\beta}'(v) \geq l^\star$. But then $j^\star$ cannot be the head of C, a contradiction. We now add an edge between the claimed $j \notin C$ and its closest vertex in C, with the cost of this edge at most $2\alpha^{l^\star+1}$—this completes the description of $E_{T'}^{l^\star}$. Note that this satisfies condition (i), since the vertex $j \notin C$ belonged to a component whose head had $\boldsymbol{\beta}$ value at least $l^\star$, and adding the edge between C and that component fixes the problem for C.

For $l \geq l^\star$, we define the edge sets $E_{T'}^l$ to inductively maintain the following invariants: (a) each component of $E_{T'}^{\leq l}$ consists of union of some of the components of $E_T^{\leq l}$, and (b) the vertex $j^\star$ is not the head of any component of $E_{T'}^{\leq l}$. As a base case, this holds for $l = l^\star$. Suppose the invariants hold for some $l \geq l^\star$. We add all edges of E_T^{l+1} to $E_{T'}^{l+1}$, except for edges that connect two vertices in the same component of $E_{T'}^{\leq l}$. It is easy to check that the invariants continue to hold, and that the set of edges $E_{T'}^{l+1}$ satisfy conditions (i) and (ii) for validity with respect to $\boldsymbol{\beta}'$: the main observation is that we are not changing the connectivity structure of $E_{T'}^{\leq l+1}$ by dropping these edges. Since we added at most one new edge at level $l^\star$, we will drop at most one edge in this process. Hence the symmetric difference between T and T' is at most 2, and the theorem follows. ∎

We now show that a Lipschitz property also holds when adding a new vertex.

Lemma 2.10 *Suppose T is a valid tree on $[i]$ with respect to $\boldsymbol{\beta}$. Consider a new function $\boldsymbol{\beta}' : [i+1] \to \mathbb{Z}_{\geq 0}$ defined thus: $\boldsymbol{\beta}'(j) := \boldsymbol{\beta}(j)$ if $j \leq i$, and $\boldsymbol{\beta}'(i+1) := \texttt{Init}(i+1)$. Then, there is a valid tree T' with respect to $\boldsymbol{\beta}'$ such that $|T' \triangle T| = 1$. Moreover, if $\boldsymbol{\beta}$ was admissible, then so is $\boldsymbol{\beta}'$.*

Proof. Let $l^\star$ denote $\texttt{Init}(i+1)$. We know that if $j^\star \in [i]$ is the closest vertex to the new vertex $i+1$, then $2\alpha^{l^\star+1} < d(j^\star, i) \le 2\alpha^{l^\star+2}$ (Claim 2.7). For $l \neq l^\star + 1$, we set $E^l_{T'} := E^l_T$, and we define $E^{l^\star+1}_{T'} := E^{l^\star+1}_T \cup \{(j^\star, i+1)\}$. It is easy to verify that T' is valid with respect to $\boldsymbol{\beta}'$.

Note that if $\boldsymbol{\beta}$ was admissible, then using the facts that the ranks of the vertices can never increase, and that $\boldsymbol{\beta}'(i+1) = \boldsymbol{\rho}_{i+1}(i+1)$, we get that $\boldsymbol{\beta}'$ is admissible. ■

Observe that rank functions $\{\boldsymbol{\rho}_i\}_i$ produced by the clustering procedure upon each arrival are always admissible. Hence, starting from $T^\star_{i-1}$, we can add a single edge (from i to its closest vertex in $[i-1]$) using Lemma 2.10, and then perform at most $\|\boldsymbol{\rho}_i - \boldsymbol{\rho}_{i-1}\|_H$ edge swaps (using Lemma 2.9) to get the tree $T^\star_i$ valid with respect to $\boldsymbol{\rho}_i$. This tree is constant competitive, because of Lemma 2.8 and Lemma 2.2); moreover, the results in Section 3 will show that $\sum_{i=1}^n \|\boldsymbol{\rho}_i - \boldsymbol{\rho}_{i-1}\|_1 \le O(n)$, which gives us another constant-amortized-swaps algorithm. However, there may be rounds that perform a non-constant number of swaps, so this does not give us our final result. To get both constant-worst-case-swaps and constant competitiveness, we need another admissible function—the *virtual rank function*—which we define next.

2.3 Defining the Virtual Rank Function

We now describe how to maintain an (admissible) virtual rank function $\boldsymbol{\nu}_i$ for all rounds i. We will ensure that $\boldsymbol{\nu}_i$ and $\boldsymbol{\nu}_{i-1}$ only differ in a constant number K of coordinates—we can then use Lemmas 2.9 and 2.10 to construct a corresponding valid tree T_i which differs from T_{i-1} only in a constant number of edges. Furthermore, we need to keep the cost of T_i, which is $\approx \sum_{j>0} \alpha^{\boldsymbol{\nu}_i(j)}$, as small as possible. A natural way to obtain $\boldsymbol{\nu}_i$ from $\boldsymbol{\nu}_{i-1}$ is to decrease the virtual rank of those K vertices for which $\boldsymbol{\nu}_{i-1}(j)$ values are highest (provided $\boldsymbol{\nu}_{i-1}(j)$ is strictly larger than $\boldsymbol{\rho}_i(j)$).

Motivated by this, we define a total ordering on pairs (j,k), where v is a vertex and k is an integer: we say that $(j,k) \prec (j',k')$ if either $k < k'$, or else $k = k'$ and $j < j'$. We formally give the algorithm for maintaining virtual ranks in the figure below.

Virtual Ranks :

1. Initially, we just have the root vertex 0. Define $\boldsymbol{\nu}_0(0) = \infty$.
2. For $i = 1, 2, \ldots$
   (i) Run the clustering algorithm $\mathcal{R}_i$ to define the rank function $\boldsymbol{\rho}_i$.
   (ii) Set $\boldsymbol{\nu}_i(i)$ as $\texttt{Init}(i)$.
   (iii) Define $Q(i) = \{(j,k) \mid j \in [i-1], k \in [\boldsymbol{\rho}_i(j) \ldots (\boldsymbol{\nu}_{i-1}(j) - 1)]\}$.
   (iv) Let Q_K be the set of the K highest pairs (w.r.t. $\prec$) from Q.
   (v) Define the first $i-1$ coordinates of $\boldsymbol{\nu}_i$ as follows:

$$\boldsymbol{\nu}_i(j) := \begin{cases} \boldsymbol{\nu}_{i-1}(j) & \text{if } (j,\star) \notin Q_K \\ \min\{k \mid (j,k) \in Q_K\} & \text{if } (j,\star) \in Q_K \end{cases}$$

Figure 1: Algorithm maintaining virtual ranks; $K = 2\alpha^2$.

An important observation about the definition of $\boldsymbol{\nu}_i$: the set Q_K might contain both tuples $(j, k+1), (j,k)$ for some $j \in [i]$ and $k \ge 0$. This is wasteful, since when reducing the virtual rank of j to k, we don't need to explicitly reduce it to $k+1$ first. But it makes the analysis cleaner, and we pay a slightly larger constant in the budget. We omit such optimizations for purposes of clarity.

2.4 The Final Algorithm

The final constant-budget algorithm is the following. Initially, T_0 is just the root vertex 0. Given a valid tree T_{i-1} with respect to the admissible virtual rank function $\boldsymbol{\nu}_{i-1}$, we obtain T_i as follows. We first run the clustering algorithm to get $\boldsymbol{\rho}_i$. Then we construct the virtual rank function $\boldsymbol{\nu}_i$ as described in the previous section, and finally construct a valid tree T_i with respect to $\boldsymbol{\nu}_i$. Lemma 2.10 and Lemma 2.9 imply that we can construct T_i from T_{i-1} by adding one edge and swapping at most K edges. The algorithm outputs T_i at the end of each round i.

3 Analysis

The constant number of swaps is enforced by the very definition of the virtual rank function, so it remains show that for each i, the cost of the tree T_i is close to the cost of the optimal Steiner tree at the end of round i, i.e., $\mathsf{Wt}_i(\boldsymbol{\nu}_i) \approx \mathsf{Wt}_i(\boldsymbol{\rho}_i)$. One approach is to ensure the functions $\boldsymbol{\nu}_i$ and $\boldsymbol{\rho}_i$ remain close *coordinate-wise close*. We do not know how to ensure this, but we do achieve closeness in cost.

Let us first give an overview of the proof. The main problem is that on arrival of vertex $(i+1)$, there may be many vertices in $[i]$ whose ranks decrease, i.e., $\boldsymbol{\rho}_{i+1}(j) = \boldsymbol{\rho}_i(j) - 1$. Since $\boldsymbol{\nu}_i$ and $\boldsymbol{\nu}_{i+1}$ can differ in only a constant number of locations, the virtual ranks now trail the actual ranks in many places. And this could potentially happen repeatedly. Our first technical result is that such bad events cannot happen in quick succession. We show that if rank of a vertex j decreases by at least 2 between two rounds i and i'—i.e., $\boldsymbol{\rho}_{i'}(j) \leq \boldsymbol{\rho}_i(j) - 2$—then many arrivals must have happened *close* to j after round i. We charge the rank decrease of j to one such arrival after round i, and prove that this charging can be done so that any arrival gets charged only a constant number of times. More formally, for every pair (j, k) which denotes that the rank of vertex $j \in [n]$ has fallen to $k \in [(\boldsymbol{\rho}_n(j) - 2) \ldots (\texttt{Init}(j) - 1)]$, we *charge* a vertex l that arrives after the rank of j drops to k; moreover, each vertex l gets charged only K times. The proof of this charging lemma appears in Section 3.1.

How can we use this charging argument to define the $\boldsymbol{\nu}_i$ function for each round i? As a thought experiment, suppose we were allowed a small amount of look-ahead. Then we could proceed as follows : suppose $\boldsymbol{\rho}_i(j)$ decreases by 1 in round $i+1$, i.e., $\boldsymbol{\rho}_{i+1}(j) = \boldsymbol{\rho}_i(j) - 1$, and we *know* that the rank of j will decrease by at least 1 more in future. If $\boldsymbol{\rho}_i(j) < \boldsymbol{\nu}_i(j)$, we add $(j, \boldsymbol{\rho}_i(j))$ to a queue. In round $i+1$, we pick any K pairs from the queue; if we pick (j, k), we decrease the virtual rank of j to k. Using the above charging argument, we can show that following such a strategy means the functions $\boldsymbol{\nu}_n$ and $\boldsymbol{\rho}_n$ differ by at most two (additively) in each coordinate. And Lemma 2.8 then implies that the cost of our tree is within a constant of the optimal Steiner tree on $[n]$.

Unfortunately, we do not have the luxury of this look-ahead, and so we instead follow a greedy strategy: in any round, among all the pairs (j, k) in the queue, we pick the ones with highest k (thereby decreasing the cost of the tree by the maximum possible). By a careful matching-based argument given in Section 3.2, we show that this strategy indeed works. We look at an arbitrary round n for rest of the analysis—our goal is to compare the cost of the tree T_n constructed by our algorithm at the end of round n, and the optimal cost $\mathsf{opt}([n])$.

3.1 The Charging Argument

In this section, we describe the charging scheme, which charges every rank decrease of a vertex (except the two most recent rank decreases for each vertex) to one of the subsequent arrivals, such that each arrival is charged at most $K = 2\alpha^2$ times. Formally, we will prove the following result.

Theorem 3.1 *Let L be the set of the ("not so recent") rank decreases until this point, and defined as*

follows:

$$L := \bigcup_{j \in [n]} \{ j^{(k)} \mid k \in [\, (\boldsymbol{\rho}_n(j) + 2) \ldots (\texttt{Init}(j) - 1)\,] \}$$

Then there is a map $F : L \to [n]$ assigning the rank changes to rounds such that

(a) (constant budget) at most $K = 2\alpha^2$ rank changes from L map to any round $i \in [n]$,
(b) (feasibility) if $F(j^{(k)}) = i$, then j's rank dropped to k at or before round i (i.e., $\boldsymbol{\rho}_i(j) \le k$), and
(c) (monotonicity) if $j^{(k)}, j^{(k-1)}$ both lie in L, then $F(j^{(k)}) \le F(j^{(k-1)})$.

(Note that $j^{(k)}$ is a syntactic object, not j raised to the power of k.) The proof of this theorem is by constructing a (b-)matching in a suitable bipartite graph. In Section 3.1.1, we give some technical results which show that when the rank of a vertex decreases by at least two, then many new arrivals will happen close to this vertex. In Section 3.1.2, we describe the bipartite graph, and use the technical results to prove the existence of a fractional b-matching, and hence an integral b-matching, in this graph.

3.1.1 Disjoint Balls and Witnessing Rank Decreases

Consider a round $i \in [n]$ and integer $k \ge 0$. Let A_{ik} be the set of vertices $j \in [n]$ which satisfy one of the following two conditions: either (i) $j \in [i]$ and $\boldsymbol{\rho}_i(j) \ge k$, or (ii) $j \in [n] \setminus [i]$ and $\texttt{Init}(j) \ge k$. I.e., A_{ik} has all those nodes that have arrived by round i and have rank at least k in that round, or will have an initial rank at least k when they arrive in the future.

For vertex $v \in [i]$, define $\kappa_{i,t}(v)$ to be the cluster in $\mathscr{C}_i(t)$ containing v. We extend this definition to the nodes arriving after round i by defining $\kappa_{i,t}(v)$ for such a node $v \in [n] \setminus [i]$ to be the singleton set $\{v\}$.

Lemma 3.2 *For round i and rank k, the balls $B(\kappa_{i,k}(j), \alpha^{k+1})$ for $j \in A_{ik}$ are disjoint.*

Proof. Let $j, l \in A_{ik}$. First assume that $j, l \in [i]$. Then the clusters in $\mathscr{C}_i(k)$ containing j and l respectively are disjoint (because they both have rank at least k, by the definition of A_{ik}), and hence the distance between them is at least $2\alpha^{k+1}$.

Now suppose at least one of j, l does not lie in $[i]$, and say l arrives after j. The result follows directly from Claim 2.7. ∎

The next lemma says that for a vertex j with "high" rank k in some round such that its rank subsequently falls below k, this decrease in rank is witnessed by arrivals close to j's cluster, whose initial ranks (collectively) are large.

Lemma 3.3 *For round i and rank k, suppose $j \in A_{ik}$. Moreover, suppose $\boldsymbol{\rho}_n(j) \le k - 2$. Let $X := \{i+1, \ldots, n\} \setminus \{j\}$, and let $Y := X \cap B(\kappa_{i,k}(j), \alpha^{k+1})$ be the points in X that are within distance α^{k+1} of j's component in $\mathscr{C}_i(k)$. Then,*

$$\sum_{l \in Y} \alpha^{\texttt{Init}(l)} \ge \alpha^{k-2}. \tag{3.3}$$

Furthermore, for any vertex $l \in Y$, $\texttt{Init}(l) \le k$.

Proof. First consider the case when $j \in [i]$, and hence $X = [n] \setminus [i]$. Let C represent j's component $\kappa_{i,k}(j)$ in $\mathscr{C}_i(k)$. Since $\boldsymbol{\rho}_i(j) \ge k$, j is the leader of C, and hence it arrived before all other vertices in C. For a set S and parameters b, b', define the *annulus* $B(S, b, b')$ to be $\{l \mid d(l, S) \in (b, b']\}$—note the half-open interval in the definition. In case $b \ge b'$, note that the annulus $B(S, b, b')$ is empty.

Observe that $B(C, 0, 2\alpha^{k+1}) \cap [i] = \emptyset$, just because if there were any other cluster within distance $2\alpha^{k+1}$ of C, we would have merged C with this cluster during phase k of $\mathcal{R}_i$. Let the vertices in Y—i.e., those

which arrive after round i in $B(C, \alpha^{k+1})$ be $l_1, l_2, \ldots, l_s$. Let us use l_0 to denote the vertex i. For an index $u \leq s$, let Δ_u denote the cumulative value $\sum_{u'=1}^{u} \alpha^{\texttt{Init}(l_{u'})}$ (we define Δ_0 as 0). Let $l_{[u]}$ denote the set of vertices $\{l_0, l_1, \ldots, l_u\}$.

Claim 3.4 *For all $u \in [0 \ldots s]$, the annulus $B(C, 2\alpha^2\Delta_u, \alpha^{k+1} - 2\alpha^2\Delta_u)$ does not contain any vertex from $l_{[u]}$.*

Proof. The proof is by induction on $u \in [0 \ldots s]$. The base case is when $u = 0$, where the claim follows from $B(C, 0, 2\alpha^{k+1}) \cap [i] = \emptyset$. Now suppose the claim is true for some $u < s$. The next vertex to arrive after l_u in $B(C, 0, \alpha^{k+1})$ is l_{u+1}. By induction hypothesis, at the beginning of round l_{u+1}, the annulus $B(C, 2\alpha^2\Delta_u, \alpha^{k+1} - 2\alpha^2\Delta_u)$ is still empty. For a contradiction, suppose l_{u+1} lies in the smaller annulus $B(C, 2\alpha^2\Delta_{u+1}, \alpha^{k+1} - 2\alpha^2\Delta_{u+1})$, then the ball of radius $2\alpha^2(\Delta_{u+1} - \Delta_u) = 2\alpha^{2+\texttt{Init}(l_{u+1})}$ around l_{u+1} would be empty in round l_{u+1}. But this contradicts Claim 2.7. This proves the claim for $u + 1$, and hence for all $u \in [0 \ldots s]$. ∎

Observe that the inequality (3.3) asks us to show that $\Delta_s \geq \alpha^{k-2}$. For the sake of contradiction, suppose $\Delta_s < \alpha^{k-2}$. In this case Claim 3.4 says that the annulus $B(C, 2\alpha^k, \alpha^{k+1} - 2\alpha^k)$ is empty at the end of round l_s; since there are no further arrivals in $B(C, \alpha^{k+1})$, the annulus $B(C, 2\alpha^k, \alpha^{k+1} - 2\alpha^k)$ is also empty after round n. Since $\alpha \geq 6$, this means that $B(C, 2\alpha^k, 4\alpha^k)$ is empty. If C' denotes the ball $B(C, 2\alpha^k)$, the following two properties hold:

- The set C' does not contain any vertex from $[i] \setminus C$. This is because $B(C, 2\alpha^{k+1})$ does not contain any vertex from $[i] \setminus C$, proved in the base case of Claim 3.4. Moreover, since j was the leader of the cluster C in round i, j is the earliest vertex in C' as well.
- $B(C', 2\alpha^k) = C'$, just because the annulus $B(C, 2\alpha^k, 4\alpha^k)$ was empty.

Using the latter property and applying Lemma 2.3 with set S set to C', and round i set to n, we infer that j's cluster $\kappa_{n,k-1}(j)$ must be contained within C', and j is the leader of this cluster $\kappa_{n,k-1}(j)$. But this contradicts the assumption that $\boldsymbol{\rho}_n(j) \leq k - 2$.

To show that the initial ranks of $l_1, \ldots, l_s$ are at most k, observe that $d(l_j, C) \leq \alpha^{k+1}$ by the definition of Y. Consequently, in $\mathcal{R}_{l_u}$, l_u must share a cluster with at least one vertex of C in the clustering $\mathscr{C}_{l_u}(k)$. Since all nodes in C are from $[i]$ and arrive before l_u, l_u cannot be the leader of its component. This completes the proof for the case $j \in [i]$.

The other case is when $j \notin [i]$. Since $j \in A_{ik}$, its initial rank $\texttt{Init}(j) \geq k$. This means $B(j, 2\alpha^{k+1})$ does not contain any vertex from $[j]$ other than j itself—in other words, $\kappa_{j,k}(j) = \{j\}$. Now we can use the same arguments as above, just starting from round j (since there are no arrivals in $B(\{j\}, 2\alpha^{k+1})$ during rounds i to j). ∎

We can extend Lemma 3.3 to subsets of A_{ik} as follows.

Corollary 3.5 *For round i and integer k, let $S \subseteq A_{ik}$. Moreover, for each $j \in S$, assume $\boldsymbol{\rho}_n(j) \leq k-2$. Let $X = [i+1 \ldots n] \setminus S$, and let $Y := X \cap (\cup_{j \in S} B(\kappa_{i,k}(j), \alpha^{k+1}))$. Then,*

$$\sum_{l \in Y} \alpha^{\texttt{Init}(l)} \geq |S| \cdot \alpha^{k-2}. \tag{3.4}$$

Furthermore, for any vertex $l \in Y$, $\texttt{Init}(l) \leq k$.

Proof. By Lemma 3.2, the balls $B(\kappa_{i,k}(j), \alpha^{k+1})$ are disjoint, so we can define Y_j as $X \cap B(\kappa_{i,k}(j), \alpha^{k+1})$ for each $j \in S$, and apply Lemma 3.3 to each one of them separately. ∎

3.1.2 Constructing the Mapping F via a Matching

We construct a bipartite graph $H = (L, R = [n], E)$. Here the set L is as described in the statement of Theorem 3.1; i.e., they are of the form $j^{(k)}$ indicating that the rank of j fell to k at some round in the past, and has subsequently fallen to $k-2$ or lower by the end of round n. The nodes in R simply represent arrivals $[n]$. The edge set E is constructed as follows: we have edge $(j^{(k)}, i)$ if $\boldsymbol{\rho}_i(j) \le k$. (Note that this condition is same as the feasibility condition of the map F in Theorem 3.1). We say that edge $(j^{(k)}, i)$'s rank is k.

Some notation: let $\Gamma(v)$ denote the set of neighbors of a node v in $L \cup R$, and $E(v)$ denote the set of edges incident to v. The main result of this section is the following:

Theorem 3.6 *There exists an assignment of non-negative values $\{x_e\}_{e \in E}$ to the edges of H such that*

(a) for any node $j^{(k)} \in L$, $\sum_{e \in E(j^{(k)})} x_e = 1$, and
(b) for any node $i \in R$, $\sum_{e \in E(i)} x_e \le 2\alpha^2$.

Moreover, if α is integer, these x_e can be chosen to be in $\{0, 1\}$.

We first note how Theorem 3.6 implies Theorem 3.1.

Proof of Theorem 3.1: If $x_e = 1$ for some edge $e = (j^{(k)}, i)$, we define $F(j^{(k)}) = i$. It is easy to check that the mapping F satisfies the first two requirements of Theorem 3.1. It remains to ensure that F satisfies the monotonicity property. Suppose $F(j^{(k)}) > F(j^{(k-1)})$ for some pair j, k. Then by swapping the values of $F(j^{(k)})$ and $F(j^{(k-1)})$, we ensure that $F(j^{(k)}) \le F(j^{(k-1)})$; moreover, this swap preserves the first two properties of F. We iteratively fix all such violations of the monotonicity property this way. ■

To prove Theorem 3.6, we partition the edges of H into subgraphs depending on their rank, and set the x_e values for each of these subgraphs independently. Specifically, for $k \ge 0$, define the bipartite graphs $H_k = (L_k, R_k, E_k)$ where $L_k = \{j \in [n] \mid j^{(k)} \in L\}$, and $R_k = \{i \in [n] \mid \texttt{Init}(i) \le k\}$. For an edge $(j^{(k)}, i) \in E$: if $\texttt{Init}(i) \le k$ then we get a corresponding edge $(j, i) \in E_k$, else this edge is simply dropped. We now prove the following stronger lemma about each H_k.

Lemma 3.7 *For each k, there exists an assignment of non-negative values $\{x_e\}_{e \in E_k}$ to the edges of H_k such that*

(a) for any node $j \in L_k$, if $E_k(j)$ is the set of edges incident to j, $\sum_{e \in E_k(j)} x_e = 1$, and
(b) for any node $i \in R_k$,

$$\sum_{e \in E_k(i)} x_e \le \frac{\alpha^{2+\texttt{Init}(i)}}{\alpha^k}. \tag{3.5}$$

Proof. The proof is constructive. We start with $x_e = 0$ for all $e \in E_k$. For each right node $i \in R_k$, define its initial potential $\Phi_k(i) = \frac{\alpha^{2+\texttt{Init}(i)}}{\alpha^k}$; this potential will measure how much x_e value can be assigned in the future to edges in $E_k(i)$. Moreover, recall that $i \in R_k \iff \texttt{Init}(i) \le k$.

For a vertex $j \in L_k$, let $\texttt{Round}_k(j)$ be the first round in which rank of j becomes k—i.e., $\texttt{Round}_k(j) = \min\{i \mid \boldsymbol{\rho}_i(j) = k\}$. Order the vertices in L_k in *non-increasing* order of their $\texttt{Round}_k(j)$ values—let this ordering be $j_1, \ldots, j_s$. Hence j_1 is the last vertex to achieve rank k, and j_s is the first vertex to do so. The algorithm in Figure 3.1.2 greedily sets the x_e values of the edges incident to the vertices in this order.

Observe that if the algorithm terminates successfully, we have an assignment of x_e values satisfying properties (a) and (b). Indeed, property (a) is guaranteed by property (i) of the algorithm, and the

Algorithm Fractional-Matching(H_k) :

Let $j_1, j_2, \ldots, j_s$ be the vertices in L_k in non-increasing order of $\texttt{Round}(\cdot)$.
For $u = 1, \ldots, s$
  Find values of x_e for edges $e \in E_k(j_u)$ such that
    (i) $\sum_{e \in E(j_u)} x_e = 1$, and
    (ii) if $e \in E_k(j_u)$ has right end-point i in R_k, then $x_e \leq \Phi_k(i)$.
  For each $(j_u, i) \in E_k(j_u)$, decrease $\Phi_k(i)$ by x_e.

Figure 2: The fractional matching algorithm for a fixed value of k

inequality (3.5) follows from the fact that the potential $\Phi_k(i)$ captures exactly how much the x_e values can be decreased without violating (3.5), and these potentials never become negative. So it suffices to show that for any vertex $j_u \in L_k$, the algorithm can find values $\{x_e\}_{e \in E_k(j_u)}$ satisfying properties (i) and (ii) during iteration u.

The proof is by induction on u. Suppose the algorithm has successfully completed the steps for $j_1, \ldots, j_{u-1}$, and we are considering j_u. Consider $\texttt{Round}_k(j_u)$, the round in which rank of j_u first becomes k. For the sake of brevity, let $r^\star := \texttt{Round}_k(j_u)$. Consider any node j_p with $p \leq u$: such a node is either j_u itself, or it has already been processed by the algorithm. There are two cases:

- Case I: $j_p \leq r^\star$; i.e., j_p arrived at or before the round in which j_u attained rank k. Since $p \leq u$, our choice of the ordering on nodes of L_k ensures that j_p itself attains rank k in or after this round $r^\star$. Hence its rank in round $r^\star$ must be at least k, and hence $j_p \in A_{r^\star k}$. [2]
- Case II: $j_p > r^\star$; i.e., in the round where j_u attained rank k, j_p has not arrived at all. However, we know j_p eventually attains rank k, so its initial rank $\texttt{Init}(j_p)$ is at least k. Consequently, $j_p \in A_{r^\star k}$ in this case as well.

Look at the set $S = \{j_1, j_2, \ldots, j_u\} \subseteq A_{r^\star k}$. By the construction of the graph H, the final rank of each node in L_k is at most $k-2$. Now Corollary 3.5 implies the existence of the set Y of vertices with $\sum_{l \in Y} \alpha^{\texttt{Init}(l)} \geq u \cdot \alpha^{k-2}$. Moreover, it ensures that each vertex $l \in Y$ has $\texttt{Init}(l) \leq k$, and hence belongs to R_k. Finally, this set $Y \subseteq [r^\star + 1 \ldots n]$, and so j_u has edges to all these nodes in $Y \subseteq R_k$—this follows from the observation that j_u achieved rank k in round $r^\star$, and hence edges $(j_u^{(k)}, i)$ were added to E for all $i \geq r^\star$. Combining all these facts, we infer that the initial potential of nodes in the neighborhood of j_u in the graph H_k is at least

$$\sum_{l \in Y} \frac{\alpha^{2+\texttt{Init}(l)}}{\alpha^k} \geq \frac{u \cdot \alpha^{k-2} \cdot \alpha^2}{\alpha^k} \geq u.$$

Since each preceding j_p results in a unit decrease in potential, the total decrease in the potential of these nodes in Y in previous steps is at most $u-1$. Hence, in iteration u, the remaining potential of the neighbors of j_u must be at least 1, which means the algorithm can always define the x_e values satisfying properties (i) and (ii). ∎

Finally, we use Lemma 3.7 to complete the proof of Theorem 3.6.

Proof of Theorem 3.6: Given the graph $H = (L, R, E)$, each rank-k edge $(j^{(k)}, i) \in E$ either gives rise to an edge $(j, i) \in H_k$, or is ignored. Independently apply Lemma 3.7 to each H_k to get an assignment of x_e to each edge in $\cup E_k$, and let each edge in E inherit the x_e value of its corresponding edge in $\cup E_k$. (If there is no such corresponding edge, set $x_e = 0$.) Since each node $j^{(k)}$ corresponds to exactly one node $j \in L_k$, Lemma 3.7(a) implies Theorem 3.6(a).

[2] Recall that A_{ik} was defined immediately after Lemma 2.3.

Now for the fractional degrees of nodes on the right. For each node $i \in R$, note that this node $i \in R_k$ only if $\texttt{Init}(i) \le k$. All edges incident to i that do not belong to any of these graphs have $x_e = 0$. Hence, adding (3.5) for all values $k \ge \texttt{Init}(i)$, we get

$$\sum_{e \in E(i)} x_e = \sum_{k \ge \texttt{Init}(i)} \sum_{e \in E_k(i)} x_e \le \alpha^2(1 + 1/\alpha + 1/\alpha^2 + \cdots) = \frac{\alpha^3}{\alpha - 1} \le 2\alpha^2.$$

The last inequality uses the fact that $\alpha \ge 2$. Finally, the statement about $\{0,1\}$ x_e values follows from the integrality of the b-matching polytope. ■

3.2 Bounding the Cost

We now show that for any $n \ge 0$, the cost of a valid tree with respect to the virtual rank function $\boldsymbol{\nu}_n$ is within a constant of $\mathsf{opt}([n])$. Recall the weight function $\mathsf{Wt}_n(\cdot)$ defined in (2.1).

Theorem 3.8 *For any round $n \ge 0$, $\mathsf{Wt}_n(\boldsymbol{\nu}_n) \le \alpha^2 \cdot \mathsf{Wt}(\boldsymbol{\rho}_n)$.*

Before we prove this theorem, we use it to prove Theorem 2.1

Proof of Theorem 2.1: Let T_n be the valid tree with respect to $\boldsymbol{\nu}_n$ constructed by our algorithm. The result follows from the following inequalities :

$$\mathsf{cost}(T_n) \overset{\text{Lemma 2.8}}{\le} \frac{2\alpha^3}{\alpha - 1} \cdot \mathsf{Wt}_n(\boldsymbol{\nu}_n) \overset{\text{Theorem 3.8}}{\le} \frac{2\alpha^5}{\alpha - 1} \cdot \mathsf{Wt}_n(\boldsymbol{\rho}_n) \overset{\text{Lemma 2.2}}{\le} \frac{2\alpha^5}{(\alpha - 1)^2} \cdot \mathsf{opt}([n]).$$

■

We now complete the proof of Theorem 3.8.

Proof of Theorem 3.8: Consider the last round $i^\star \le n$ such that, at the end of round $i^\star$, we had $\boldsymbol{\nu}_{i^\star}(j) = \boldsymbol{\rho}_{i^\star}(j)$ for all $j \in [i^\star]$—the most recent round after which we had no more pending rank reductions. (There exists such an $i^\star$, since this property is satisfied at the end of rounds 0 and 1.) This means that at the end of round $i^\star$, the tree $T_{i^\star}$ was valid with respect to the rank function $\boldsymbol{\rho}_{i^\star}$, and not just the virtual rank function $\boldsymbol{\nu}_{i^\star}$. Moreover, in every round $i \in [(i^\star + 1), \ldots, n]$, the algorithm must have done K rank reductions. Indeed, if $\boldsymbol{\nu}_{i-1}(j) > \boldsymbol{\rho}_i(j)$ for some round i and vertex $j \in [i-1]$, then we add the pair $(j, \boldsymbol{\rho}_i(j))$ to the set $Q(i)$ in the algorithm for defining virtual ranks (Figure 2.3). Hence, if the algorithm does less than K rank reductions in some round $i > i^\star$, then $|Q(i)| \le K$, and $\boldsymbol{\nu}_i(j) = \boldsymbol{\rho}_i(j)$ at the end of round i. But this would contradict the definition of $i^\star$.

What rank reductions was the algorithm doing (or trying to do)? These are represented by the set

$$X := \cup_{j \in [n]} \{ j^{(k)} \mid k \in [\boldsymbol{\rho}_n(j) \ldots (\boldsymbol{\rho}_{\max(i^\star, j)}(j) - 1)] \} \quad .$$

Indeed, at each round $i \in [(i^\star + 1) \ldots n]$, the algorithm does K of the swaps in this set. So now consider a bipartite graph H', where the vertices on the left are X, and there are K vertices $i_1, i_2, \ldots, i_K$ on the right for every round $i \in [(i^\star + 1) \ldots n]$. Put an edge between $j^{(k)}$ and a unique copy of i if the algorithm reduced j's virtual rank to k in round i. This gives us a matching M_A; since the algorithm does K swaps each round, each node on the right is matched.

Recall Theorem 3.1, and the definitions of set L and the map $F : L \to [n]$. Note that $X \not\subseteq L$ in general, since L does not contain $j^{(\boldsymbol{\rho}_n(j))}$ and $j^{(\boldsymbol{\rho}_n(j)+1)}$ for each j. However, $X \subseteq L \cup \{ j^{(\boldsymbol{\rho}_n(j))}, j^{(\boldsymbol{\rho}_n(j)+1)} \mid j \in [n] \}$. The map F clearly maps $X \cap L$ to $[n]$; we claim F maps $X \cap L$ to $[(i^\star + 1) \ldots n]$. To see this, consider $j^{(k)} \in X \cap L$: j achieves rank $\boldsymbol{\rho}_{i^\star}(j) - 1$ only in round $i^\star + 1$ or later, and hence the feasibility property of F says that $F(j) \ge i^\star + 1$, which proves our claim. Hence, we can think of F as giving another

matching M_F on the bipartite graph H' defined above: if $F(j^{(k)}) = i$, then we add an edge between $j^{(k)}$ and a distinct one of the K copies of i; we need at most K copies due to the "constant budget" property of F. Furthermore, M_F matches every vertex in $X \cap L$ to some vertex on the right.

Given these two matchings, M_A capturing the algorithm's behavior, M_F encoding the "suggested schedule" given by the mapping F, it is natural to consider their symmetric difference $M_A \triangle M_F$, which consists of paths and cycles and argue about these. It will be convenient to introduce one last piece of notation. For an edge $e = (j^{(k)}, i_s) \in M_F \cup M_A$, where i_s is one of the K copies of i, we associate the quantity $\texttt{deficit}(e) := \alpha^{k+1} - \alpha^k$. Since we think of $(j^{(k)}, i_s)$ as reducing the rank of j from $k+1$ to k in round i, and so $\texttt{deficit}(e)$ intuitively denotes the reduction in the total cost by this rank reduction in round i. The following easy-to-prove claims make this intuition formal.

Claim 3.9 *For the matching M_A,*

$$\mathsf{Wt}_n(\boldsymbol{\nu}_n) \leq \mathsf{Wt}_n(\boldsymbol{\nu}_{i^\star}) + \sum_{i=i^\star+1}^{n} \alpha^{\texttt{Init}(i)} - \sum_{e \in M_A} \texttt{deficit}(e),$$

whereas for the matching M_F,

$$\alpha^2 \cdot \mathsf{Wt}_n(\boldsymbol{\rho}_n) \geq \mathsf{Wt}_n(\boldsymbol{\rho}_{i^\star}) + \sum_{i=i^\star+1}^{n} \alpha^{\texttt{Init}(i)} - \sum_{e \in M_F} \texttt{deficit}(e).$$

Proof. We argue about M_A first. Consider a round $i \geq i^\star + 1$, and let the neighbors of the K copies of i (in the matching M_A) be $j_1^{(r_1)}, \ldots, j_K^{(r_K)}$. Then, $\boldsymbol{\nu}_i(j_u) = r_u$, and we must have had $\boldsymbol{\nu}_{i-1}(j_u) \geq r_u + 1$, for all $u = 1, \ldots, K$. In the matching M_A, each of the vertices j_u will be matched to a unique copy of i, say it is i_u. Furthermore, one new arrival (vertex i) happens during round i. Therefore,

$$\begin{aligned} \mathsf{Wt}_{i-1}(\boldsymbol{\nu}_{i-1}) - \mathsf{Wt}_i(\boldsymbol{\nu}_i) &= \sum_{u=1}^{K} \left(\alpha^{\boldsymbol{\nu}_{i-1}(j_u)} - \alpha^{\boldsymbol{\nu}_i(j_u)} \right) - \alpha^{\texttt{Init}(i)} \\ &\geq \sum_{u=1}^{K} \texttt{deficit}((j_u^{(r_u)}, i_u)) - \alpha^{\texttt{Init}(i)} \end{aligned}$$

Summing the above for all $i = i^\star+1, \ldots, n$ gives us the result for M_A. Now consider M_F. For any vertex j, let i_0 denote $\max(j, i^\star)$. Then matching M_F matches all the vertices in $\{j^{(\boldsymbol{\rho}_{i_0}(j)-1)}, \ldots, j^{(\boldsymbol{\rho}_n(j)+2)}\}$ (which is empty if $\boldsymbol{\rho}_{i_0}(j) - 1 < \boldsymbol{\rho}_n(j) + 2$). The sum of the deficit of all these edges is exactly $\alpha^{\boldsymbol{\rho}_{i_0}(j)} - \alpha^{\boldsymbol{\rho}_n(j)+2}$. Summing this over all j gives us the second part of the lemma. ∎

Lemma 3.10 $\sum_{e \in M_A} \texttt{deficit}(e) \geq \sum_{e \in M_F} \texttt{deficit}(e)$.

Proof. Recall that the bipartite graph H' has vertices from X on the left and $[(i^\star+1) \ldots n] \times K$ on the right. Also M_A completely matches the vertices on the right of H'. The symmetric difference $M_A \triangle M_F$ of the two matchings consists of paths and cycles. Any cycle means that the total deficit of the edges from M_A and M_F in it are equal. What about a path? Since M_A matches every vertex on the right, we know that any path is either of odd length (ending with M_A-edges), or of even length with both the end-points being on the left. In the former case, the total deficit of edges from M_A in this path is at least that of edges from M_F. So the remaining case is when the path P consists of an equal number of (alternating) edges from M_F and M_A, and the end-points of P lie on the left side of H'.

Let the vertices from the left side of the bipartite graph in this even path P be $x_0, x_1, x_2, \ldots, x_s$ (in order of their appearance in P). Each x_u is of the form $j_u^{(k_u)}$, and let us define $\texttt{val}(x_j) := k_j$. Assume

w.l.o.g. that x_0 is matched by M_F (and unmatched in M_A) and x_s is matched by M_A (and unmatched in M_F). Since $x_1, \ldots, x_{s-1}$ are matched in both M_A and M_F,

$$\sum_{e \in P \cap M_A} \texttt{deficit}(e) - \sum_{e \in P \cap M_F} \texttt{deficit}(e) = \left(\alpha^{k_s+1} - \alpha^{k_s}\right) - \left(\alpha^{k_0+1} - \alpha^{k_0}\right). \tag{3.6}$$

Thus, we will be done if we show that $k_s \geq k_0$; i.e., $\texttt{val}(x_s) \geq \texttt{val}(x_0)$. We prove this next.

For any matching M and node u that is matched in M, let $M(u)$ be the other endpoint of the matching edge containing u. Let I be the right vertices (arrivals) on the path P, say $I = \{i_1 \leq i_2 \leq \ldots \leq i_s\}$. They may appear in any order on the path, this ordering is just based on arrivals. For $u \in \{0, 1, \ldots, s\}$, let $I_u = \{i_1, i_2, \ldots, i_u\}$; hence $I_0 = \emptyset$ and $I_s = I$. Define an auxiliary graph Q_u on the vertex set $\{x_0, x_1, \ldots, x_s\}$, by adding an edge between $M_F(i)$ and $M_A(i)$ for each $i \in I_u$. Hence Q_0 has no edges, and Q_s is the path $\langle x_0, x_1, \ldots, x_s \rangle$. It is to see that each component of Q_u is a path with a "head" vertex which is the least indexed one and which is not yet matched to anyone in I_u by M_A, and a "tail" which has the highest index and is not matched to anyone in I_u by M_F. As a sanity check, in Q_0, each node is both head and tail of its component. In Q_s, there is a single path with x_0 as the head and x_s as the tail. We now prove the following lemma by induction on u.

Claim 3.11 *For each $u \in \{0, 1, \ldots, s\}$ and each component of Q_u, value of the tail of this component is at least that of its head.*

Proof. The base case is trivially true. Now suppose the claim is true for Q_{u-1}, and we get Q_u by adding an edge between $M_F(i_u) = x_a$ and $M_A(i_u) = x_{a+1}$. It must be that x_a is the tail of its component with, say, x_p as the head. And x_{a+1} is the head of its component, say x_q is the tail. By the I.H.,

$$\texttt{val}(x_q) \geq \texttt{val}(x_{a+1}). \tag{3.7}$$

Moreover, when the algorithm was choosing the rank reductions to perform at round i_u, both x_p and x_{a+1} were candidates to be matched. Since we chose x_{a+1} greedily to have maximum value, we have $\texttt{val}(x_{a+1}) \geq \texttt{val}(x_p)$. Combining with (3.7), we get $\texttt{val}(x_q) \geq \texttt{val}(x_p)$. Since the new component has head x_p and tail x_q, this proves the inductive step and hence Claim 3.11. ■

Applying Claim 3.11 for $u = s$, we get $\texttt{val}(x_s) \geq \texttt{val}(x_0)$, and hence the total deficit of edges in $P \cap M_A$ is at least that of edges in $P \cap M_F$ by (3.6). Summing this over all alternating paths in $M_A \triangle M_f$ completes the proof of Lemma 3.10. ■

By the definition of $i^\star$, we know that $\mathsf{Wt}_n(\boldsymbol{\nu}_{i^\star}) = \mathsf{Wt}_n(\boldsymbol{\rho}_{i^\star})$. Moreover, we have $\sum_{e \in M_A} \texttt{deficit}(e) \geq \sum_{e \in M_F} \texttt{deficit}(e)$ by Lemma 3.10. Plugging these into Claim 3.9, we get that

$$\begin{aligned}
\mathsf{Wt}_n(\boldsymbol{\nu}_n) &\leq \mathsf{Wt}_n(\boldsymbol{\nu}_{i^\star}) + \sum_{i=i^\star+1}^{n} \alpha^{\texttt{Init}(i)} - \sum_{e \in M_A} \texttt{deficit}(e) \\
&\leq \mathsf{Wt}_n(\boldsymbol{\rho}_{i^\star}) + \sum_{i=i^\star+1}^{n} \alpha^{\texttt{Init}(i)} - \sum_{e \in M_F} \texttt{deficit}(e) \\
&\leq \alpha^2 \cdot \mathsf{Wt}_n(\boldsymbol{\rho}_n).
\end{aligned}$$

This completes the proof of Theorem 3.8 (and of Theorem 2.1). ■

4 Just One Swap

In the previous section, we proved a weaker version (Theorem 2.1) of our main theorem (Theorem 1.1): using a constant number $K = 2\alpha^2$ of swaps per arrival, we could maintain a tree T_n with cost at most some other constant $C = \frac{2\alpha^5}{\alpha-1}$ times the optimum tree $\mathsf{opt}([n])$. We now show how to trade off the number of swaps for the approximation guarantee, and get a constant-factor approximation while performing at most a single swap per iteration. It is unclear how to convert a generic algorithm that performs K swaps and maintains a C-approximate tree into one that performs a single swap and maintains a $f(C, K)$-approximation—this is another place where our dual-based proof strategy comes handy, since it allows us to perform such a conversion.

To understand the new ideas, recall the previous algorithm/analysis used the following conceptual steps:

1. We show that all but the two most recent rank decreases for each vertex can be scheduled so that at most K changes are performed at each step. (Theorem 3.1.)
2. The algorithm takes the set of all the rank changes that have not yet been performed (i.e., the set $Q(i) = \{(j, k) \mid j \in [i-1], k \in [\boldsymbol{\rho}_i(j) \dots (\boldsymbol{\nu}_{i-1}(j) - 1)]\}$ where the virtual rank function $\boldsymbol{\nu}_{i-1}$ lags behind the real rank function $\boldsymbol{\rho}_i$, and greedily chooses K of the most beneficial changes to perform. (This defines $\boldsymbol{\nu}_i$, and is described in Section 2.3.)
3. Finally, we show (in Theorem 3.8) that this greedy process ensures the potential function $\mathsf{Wt}(\boldsymbol{\nu}_i) \leq \alpha^2\, \mathsf{Wt}(\boldsymbol{\rho}_i)$, and hence the cost of our tree is not much more than that of the optimal tree.

The main change to get a single-swap algorithm is this: suppose we don't try to schedule all the not-so-recent rank changes (as in Step 1 above), but only some subset of the rank changes, such that two successive rank changes in this set for any vertex differ by approximately K. Since we are then scheduling approximately $1/K$ as many rank decreases as in Theorem 3.1, we can get a version of that theorem with at most one rank change being mapped to each time step. Now we can change the algorithm (in Step 2 above) to greedily choose the *single most beneficial rank change* and perform it. The rest of the argument would follow pretty much unchanged. Since the virtual rank functions lags the real rank functions on average by an additive K, the corresponding tree is now $\alpha^{O(K)}$-approximate instead of being $\alpha^{O(1)}$-approximate. Since K is a constant, we prove Theorem 1.1.

In the rest of the section, we first prove an analog of Theorem 3.1, describe the modified algorithm to define the virtual ranks $\boldsymbol{\nu}_i()$, and finally describe the changes in the rest of the arguments due to these changes.

4.1 A Modified Charging Theorem

Our new algorithm will be interested in those values of ranks of a particular vertex which are separated by multiples of K. Motivated by this, we define, for a vertex j, the set $\mathbb{Z}(j, K)$ as $\{\mathtt{Init}(j) - lK \mid l \in \mathbb{Z}_{\geq 0}\}$.

Recall the set L from Theorem 3.1, and define L', the sparsified version of L, as follows:

$$L' := \bigcup_{j \in [n]} \{j^{(k)} \mid k \in \mathbb{Z}(j, K) \cap [\,(\boldsymbol{\rho}_n(j) + K + 1) \dots (\mathtt{Init}(j) - K)\,]\} \tag{4.8}$$

(Again, note that $j^{(k)}$ is a syntactic object, not j raised to the power of k.)

Theorem 4.1 *There is a map $F' : L' \to [n]$ assigning the rank changes to rounds such that*

(a) (unit budget) at most one rank change from L' maps to any round $i \in [n]$,

(b) (feasibility) if $F'(j^{(k)}) = i$, then j's rank dropped to k at or before round i (i.e., $\boldsymbol{\rho}_i(j) \leq k$), and
(c) (monotonicity) if $j^{(k)}, j^{(k-K)}$ both lie in L', then $F'(j^{(k)}) \leq F'(j^{(k-K)})$.

Proof. For each element $j^{(k)} \in L'$, consider the set $S(j^{(k)}) := \{j^{(k)}, j^{(k-1)}, \ldots, j^{(k-K+1)}\}$ of size K. By the definition of L', these sets for different elements of L' are disjoint; moreover, each set $S(j^{(k)})$ is a subset of the set L (as defined in Theorem 3.1). Now consider the map $F : L \to [n]$ given by Theorem 3.1: this can be viewed as a bipartite graph between L and $[n]$ where all nodes in L have unit degree and nodes in $[n]$ have degree at most K. We delete the nodes in L not belonging to $\cup_{j^{(k)} \in L'} S(j^{(k)})$, and contract nodes in each $S(j^{(k)})$ for $j^{(k)} \in L'$ into a single "supernode". The resulting bipartite graph has left-degree exactly K, and the right degree at most K—and by Hall's theorem, has a matching where every supernode on the left is matched. This immediately gives us the map F': if the edge out of the supernode for $S(j^{(k)})$ goes to $i \in [n]$, we set $F'(j^{(k)}) := i$.

Property (a) follows from construction. For property (b), observe that the edge from $S(j^{(k)})$ to i in the contracted graph is inherited from the fact that $F(j^{(k-c)}) = i$ for some $c \in [0 \ldots K-1]$; by Theorem 3.1(b), this means $\rho_i(j) \leq k - c$ and hence at most k. Finally, the monotonicity of F' follows from that of F. ∎

4.2 Modified Procedure to define Virtual Ranks Function

We now describe the modified algorithm to maintain the virtual ranks. This will be similar to the algorithm in Figure 2.3, except that the virtual ranks, for a vertex j, will take values in $\mathbb{Z}(j, K)$ only. The modified algorithm is described in Figure 3. It is similar to our earlier algorithm, except that we improve the virtual rank values in multiples of K only. It is easy to check that for any vertex $j \in [i]$, $\boldsymbol{\nu}_i(j)$ lies in $\mathbb{Z}(j, K)$.

Virtual Modified-Ranks :

1. Initially, we just have the root vertex 0. Define $\boldsymbol{\nu}_0(0) = \infty$.
2. For $i = 1, 2, \ldots$
   (i) Run the clustering algorithm $\mathcal{R}_i$ to define the rank function $\boldsymbol{\rho}_i$.
   (ii) Set $\boldsymbol{\nu}_i(i)$ as $\texttt{Init}(i)$.
   (iii) Define $Q(i) = \{(j, k) \mid j \in [i-1], k \in [\boldsymbol{\rho}_i(j) \ldots (\boldsymbol{\nu}_{i-1}(j) - 1)] \cap \mathbb{Z}(j, K)\}$.
   (iv) Let Q_K be the set of the K highest pairs (w.r.t. $\prec$) from Q.
   (v) Define the first $i-1$ coordinates of $\boldsymbol{\nu}_i$ as follows:

$$\boldsymbol{\nu}_i(j) := \begin{cases} \boldsymbol{\nu}_{i-1}(j) & \text{if } (j, \star) \notin Q_K \\ \min\{k \mid (j,k) \in Q_K\} & \text{if } (j, \star) \in Q_K \end{cases}$$

Figure 3: Modified algo. for virtual ranks; $K = 2\alpha^2$.

4.3 Modified Version of Theorem 3.8

We now prove the analogue of Theorem 3.8.

Theorem 4.2 *Using the new definition of $\boldsymbol{\nu}_n$, for any round $n \geq 0$, $\mathsf{Wt}_n(\boldsymbol{\nu}_n) \leq \alpha^{2K+1} \cdot \mathsf{Wt}(\boldsymbol{\rho}_n)$.*

Proof. The proof proceeds along the same lines as that of Theorem 3.8. We point out the main modifications to the proof of Theorem 3.8. For a vertex j and non-negative integer $k \leq \texttt{Init}(j)$, define $\lceil k \rceil_{j,K}$ as the smallest element of $\mathbb{Z}(j, K)$ which is at least k. For a round i and rank vector $\boldsymbol{\rho}_i$, define

the rounded rank vector $\lceil \boldsymbol{\rho}_i \rceil$ as follows : for each $j \in [i]$, $\lceil \boldsymbol{\rho}_i \rceil(j) := \lceil \boldsymbol{\rho}_i(j) \rceil_{j,K}$. Since $\nu_i(j)$ values lie in $\mathbb{Z}(j, K)$, it is easy to check that for any round i, $\boldsymbol{\nu}_i$ is component-wise at least $\lceil \boldsymbol{\rho}_i \rceil$. The round $i^\star$ is defined as the last round i in which $\boldsymbol{\nu}_i = \lceil \boldsymbol{\rho}_i \rceil$. Again, it is easy to check that we will do one rank update in every round after $i^\star$.

The set X is now defined as

$$X := \cup_{j \in [n]} \{ j^{(k)} \mid k \in \mathbb{Z}(j,k) \cap [\boldsymbol{\rho}_n(j) \ldots (\boldsymbol{\rho}_{\max(i^\star, j)}(j) - 1)] \} \quad .$$

In the bipartite graph H', we need to keep just one copy for each round. In the matching $M_{A'}$, we have an edge between $j^{(k)}$ and i if our algorithm set $\boldsymbol{\nu}_i(j)$ to k in round i. As before, $M_{A'}$ matches all vertices on the right of H' (which represent rounds $i^\star, \ldots, n$). We can define the matching $M_{F'}$ using the mapping F' given by Theorem 4.1. One can again check that F' maps $X \cap L'$ to $[(i^\star + 1) \ldots n]$. We have an edge $(j^{(k)}, i)$ in the matching $M_{F'}$ if $F'(j^{(k)}) = i$.

We look at the symmetric difference of the two matchings : $M_{F'} \triangle M_{A'}$. For an edge $e = (j^{(k)}, i) \in M_{F'} \cup M_{A'}$, define $\texttt{deficit}(e)$ as $\alpha^{k+K} - \alpha^k$. We can now show the following analogue of Claim 3.9 holds. For the matching $M_{A'}$,

$$\mathsf{Wt}_n(\boldsymbol{\nu}_n) \leq \mathsf{Wt}_n(\boldsymbol{\nu}_{i^\star}) + \sum_{i=i^\star+1}^{n} \alpha^{\texttt{Init}(i)} - \sum_{e \in M_A} \texttt{deficit}(e), \tag{4.9}$$

whereas for the matching $M_{F'}$,

$$\alpha^{K+1} \cdot \mathsf{Wt}_n(\lceil \boldsymbol{\rho}_n \rceil) \geq \mathsf{Wt}_n(\lceil \boldsymbol{\rho}_{i^\star} \rceil) + \sum_{i=i^\star+1}^{n} \alpha^{\texttt{Init}(i)} - \sum_{e \in M_F} \texttt{deficit}(e). \tag{4.10}$$

The proof of Lemma 3.10 carries over without any changes (using the modified definition of $\texttt{deficit}(e)$). So, combining inequalities (4.9) and (4.10), we get

$$\mathsf{Wt}_n(\boldsymbol{\nu}_n) \leq \alpha^{K+1} \cdot \mathsf{Wt}_n(\lceil \boldsymbol{\rho}_n \rceil).$$

But the vectors $\boldsymbol{\rho}_n$ and $\lceil \boldsymbol{\rho}_n \rceil$ differ by at most K in each coordinate. Hence, $\mathsf{Wt}_n(\lceil \boldsymbol{\rho}_n \rceil) \leq \alpha^K \mathsf{Wt}_n(\boldsymbol{\rho}_n)$. This proves the theorem. ∎

Proceeding as in the proof of Theorem 2.1, we get

$$\mathsf{cost}(T_n) \leq \frac{2\alpha^{2K+4}}{(\alpha - 1)^2} \cdot \mathsf{opt}([n]).$$

This proves Theorem 1.1.

In fact, we can prove a stronger version of Theorem 1.1.

Theorem 4.3 *Given a parameter δ, $0 < \delta \leq 1$, there is an online $2^{O(\frac{1}{\delta})}$-competitive algorithm for metric Steiner tree which performs at most one swap upon each arrival, and at most δ swaps on each arrival in the amortized sense.*

Proof. We give a sketch of the proof. Assume without loss of generality that $\frac{1}{\delta}$ is an integer. The main idea is again to strengthen Theorem 3.1. We were able to get a stronger version of this theorem, i.e., Theorem 4.1, by grouping vertices of L into groups of K.

Let K' denote $\frac{K}{\delta}$. Define

$$L'' := \bigcup_{j \in [n]} \{ j^{(k)} \mid k \in \mathbb{Z}(j, K') \cap [\, (\boldsymbol{\rho}_n(j) + K' + 1) \ \ldots \ (\texttt{Init}(j) - K')\,] \}.$$

Let $[n]_\delta$ denote those elements of $[n]$ which are multiples of $\frac{1}{\delta}$. We can now generalize Theorem 4.1 even further to show that there exists a map $F'' : L'' \to [n]_\delta$ such that

(a) (unit budget) at most one rank change from L'' maps to any round $i \in [n]_\delta$,
(b) (feasibility) if $F''(j^{(k)}) = i$, then j's rank dropped to k at or before round i (i.e., $\boldsymbol{\rho}_i(j) \le k$),
(c) (monotonicity) if $j^{(k)}, j^{(k-K')}$ both lie in L', then $F''(j^{(k)}) \le F''(j^{(k-K')})$.

The proof again follows that of Theorem 4.1, where we now group vertices of L into groups of size K' and those of $[n]$ into groups of size $\frac{1}{\delta}$. Our online algorithm is same as that in Figure 3, with K replaced by K'. Moreover, we perform the steps of this algorithm only for those rounds i which are multiples of $\frac{1}{\delta}$ (i.e., in Step 2, if i is not a multiple of $\frac{1}{\delta}$, then we just perform steps 2(i) and 2(ii)). The proof now proceeds as in that of Theorem 1.1. ∎

5 A Tight Amortized Analysis

In this section, we analyze the following greedy algorithm of Imase and Waxman [IW91]. Given a parameter $\varepsilon > 0$, their algorithm, which we call $\mathcal{B}_{1+\varepsilon}$, works as follows. It maintains a tree connecting all the demands which have arrived so far. Let T_i be the constructed by the algorithm for vertices in $[i]$. When the vertex $i+1$ arrives, it first connects $i+1$ to the closest vertex in $[i]$. Moreover, whenever there is an edge e in the current tree and a non-tree edge f such that $\mathsf{len}(e) > (1+\varepsilon)\,\mathsf{len}(f)$ and $T+f-e$ is also a (spanning) tree, we swap the edges e and f, i.e., we add f and remove e from the current tree. We get the tree T_{i+1} when this swapping process ends. It is immediate from the construction that the spanning tree maintained has weight within a factor $(1+\varepsilon)$ of the best spanning tree. The goal is to show that for any n and constant ε, the number of swaps made in the first n steps is $O(n)$. Clearly, we cannot hope for a better result, because there are simple examples showing that the arrival of a single vertex might cause $\Omega(n)$ swaps.

Recently, Megow et al. [MSVW12] showed that a close variant of this algorithm (which "froze" edges when they had a very small length and did not perform any swaps with them) performed at most $O(n/\varepsilon \log 1/\varepsilon)$ swaps. In this section, we prove Theorem 1.2 and show that the algorithm $\mathcal{B}_{1+\varepsilon}$ (without additional freezing operations) performs only $O(n/\varepsilon)$ swaps (and at most $2n$ swaps for $\varepsilon = 1$).

5.1 An Improved Bound for All-Swaps

Let $T^\star = \{e_1, \ldots, e_n\}$ be a minimal spanning tree on $[n]$. Suppose the greedy edges that we add for vertices $1, \ldots, n$ are respectively $g_1^\circ, \ldots, g_n^\circ$. An edge in the final tree T_n is obtained by a sequence of swaps starting from one of the greedy edges g_r°, for some unique r. Thus, we can define a bijection between the edges in the final tree T_n, denoted by $g_1^f, \ldots, g_n^f$, such that for any r, g_r^f is obtained by a sequence of swaps starting from g_r°. In this section, we denote the length of an edge e by $c(e)$. Since each swap replaces an edge by another that is a factor $(1+\varepsilon)$ shorter, an upper bound on the total number of swaps performed is

$$\log_{1+\varepsilon} \frac{c(g_1^\circ)}{c(g_1^f)} + \cdots + \log_{1+\varepsilon} \frac{c(g_n^\circ)}{c(g_n^f)} = \log_{1+\varepsilon} \frac{\prod_{i=1}^n c(g_i^\circ)}{\prod_{i=1}^n c(g_i^f)} \tag{5.11}$$

Theorem 5.1 *The quantity $\prod_{i=1}^n c(g_i^\circ) / \prod_{i=1}^n c(g_i^f)$ is bounded by 4^n. Hence the algorithm $\mathcal{B}_{1+\varepsilon}$ performs at most $n(\log_{1+\varepsilon} 4) \in O(n/\varepsilon)$, and $\mathcal{B}_2$ performs at most $2n$ swaps.*

This result improves on the result of [MSVW12] who gave a bound of $O(\frac{n}{\varepsilon} \log \frac{1}{\varepsilon})$ on the number of swaps for their freezing-based variant of $\mathcal{B}_{1+\varepsilon}$. In Section 5.2, we will show an example for which one needs at least $1.25n$ swaps. Now, to prove Theorem 5.1, let us give a lower bound on $\prod_{i=1}^n c(g_i^f)$.

Lemma 5.2 $\prod_{i=1}^n c(g_i^f) \geq \prod_{i=1}^n c(e_i)$.

Proof. By well-known properties of spanning trees (and matroids), there exists a bijection ψ between the edges of $T^\star$ and T_n such that for all edges $e \in T^\star$, the graph $\{T^\star \cup \psi(e)\} \setminus e$ is a tree [Sch03, Corollary 39.12a]. Since $T^\star$ was chosen to have minimal total cost, $c(e) \leq c(\psi(e))$. Therefore $\prod_{i=1}^n c(e_i) \leq \prod_{i=1}^n c(\psi(e_i)) = \prod_{i=1}^n c(g_i^f)$. ∎

In light of this claim, proving Theorem 5.1 reduces to showing the following lemma.

Lemma 5.3 $\prod_{i=1}^n c(g_i^\circ) \leq 4^n \cdot \prod_{i=1}^n c(e_i)$.

The proof of this lemma will occupy most of the rest of this section. It is based on a few useful but simple facts, which we prove next.

Lemma 5.4 *There exists a function $f : \mathbb{N} \to \mathbb{R}$ such that*

(i) $f(1) = 1$, and for all $\ell \in \mathbb{N}$, $f(\ell) \geq 1$

(ii) For all $\ell \in \mathbb{N}$,

$$\sum_{i=1}^{\ell-1} \frac{f(\ell)}{f(i) \cdot f(\ell - i)} \leq 4.$$

The proof of this lemma is based on an unedifying calculation, and is deferred to Section 5.1.1. Note that the constant 4 in Lemma 5.4 is the same constant 4 that appears in Lemma 5.3; if one is satisfied with a worse constant, one could use $f(\ell) = \ell^2$, for which a bound of $\frac{4}{3}\pi^2$ is easy to prove.

Lemma 5.5 *Consider a tree T with ℓ nodes, and a path P on this tree consisting of edges $h_1, \ldots, h_k$ in order. For any edge $e \in T$, let ℓ_e and ℓ'_e denote the number of vertices in the two trees formed by deleting e. Then there exists some edge $h \in P$ such that*

$$\frac{c(P)}{c(h)} \leq 4 \frac{f(\ell_h) \cdot f(\ell'_h)}{f(\ell)}, \tag{5.12}$$

where $f(\cdot)$ is the function from Lemma 5.4.

Proof. Suppose otherwise; then for all i,

$$c(h_i) < \frac{c(P)}{4} \frac{f(\ell)}{f(\ell_{h_i}) f(\ell'_{h_i})}.$$

Hence,

$$c(P) = \sum_{i=1}^k c(h_i) < \sum_{i=1}^k \frac{c(P)}{4} \cdot \frac{f(\ell)}{f(\ell_{h_i}) \cdot f(\ell'_{h_i})} \leq \frac{c(P)}{4} \sum_{j=1}^{\ell-1} \frac{f(\ell)}{f(j) \cdot f(\ell - j)} \leq c(P),$$

which is a contradiction. (The third inequality just contains more non-negative terms than the second one, and the last inequality used Lemma 5.4.) ∎

Definition 5.6 *For $1 \leq i \leq n$, let Δ_i be the smallest number such that there exists a partition of $[n]$ into i parts, such that the induced subgraph for each part has diameter at most Δ_i.*

Lemma 5.7 *For $1 \leq i \leq n$, suppose that h_i is the i-th largest greedy edge, so that $h_1, \ldots, h_n$ is a permutation of $g_1^\circ, \ldots, g_n^\circ$ and $h_1 \geq \cdots \geq h_n$. Then $c(h_i) \leq \Delta_i$.*

Proof. For all $1 \le i \le n$, let x_i be the vertex associated with h_i's arrival, and define $x_0 = 0$. Then any edge in the subgraph induced by $\{x_0, x_1, \ldots, x_i\}$ has cost at least h_i. By definition, there exists a partition of V into i components all of diameter at most Δ_i. But two of the $i+1$ vertices $\{x_0, \ldots, x_i\}$ must lie in the same component of the partition, so that their distance is at most Δ_i, implying that $h_i \le \Delta_i$. ∎

Moreover, note that $\prod_{i=1}^n c(g_i^\circ) = \prod_{i=1}^n c(h_i)$, so it suffices to bound the latter.

Proof of Lemma 5.3: Consider a permutation $e_1, \ldots, e_n$ such that for all k, e_k lies on the longest path P_k of the forest of k trees formed by deleting $e_1, \ldots, e_{k-1}$ from T, and such that e_k, P_k satisfy the condition (5.12) in Lemma 5.5. Note that since P_k is the longest path in the forest, hence the diameter of every component is at most P_k. By Definition 5.6 and the fact that the forest is a partition of $[n]$ into k parts, we get that $\Delta_k \le P_k$. Consequently,

$$\frac{\Delta_k}{e_k} \le \frac{P_k}{e_k} \le 4\frac{f(\ell_e) \cdot f(\ell'_e)}{f(\ell)},$$

where ℓ is the size of the component that contains e_k, etc. Multiplying over all k, the right side telescopes to $\frac{4^n \cdot f(1)^{n+1}}{f(n+1)} \le 4^n$. Here we used that $f(n+1) \ge 1$ and $f(1) = 1$. Finally, putting everything together, we get

$$\frac{\prod_{i=1}^n c(g_i^\circ)}{\prod_{i=1}^n c(g_i^f)} \le \frac{\prod_{i=1}^n c(g_i^\circ)}{\prod_{i=1}^n c(\ell_i)} = \frac{\prod_{i=1}^n c(h_i)}{\prod_{i=1}^n c(\ell_i)} \le \frac{\prod_{i=1}^n \Delta_i}{\prod_{i=1}^n c(\ell_i)} \le \frac{4^n \cdot f(1)^{n+1}}{f(n+1)} \le 4^n,$$

where the first two inequalities above follow from Lemmas 5.2, 5.7, and the remaining two from the preceding discussion. ∎

5.1.1 Proof of Lemma 5.4, and its Tightness

Proof of Lemma 5.4: We claim the function

$$f(\ell) = \frac{(-1)^{\ell+1}}{2\binom{\frac{1}{2}}{\ell}} = \frac{2^{2\ell-1}\,(2\ell-1)}{\binom{2\ell}{\ell}}$$

satisfies the desired properties.

(i) Expanding the second formula out gives

$$\begin{aligned}\frac{\left(2^\ell \ell!\right)\left(2^{\ell-1}\ell!\right)(2\ell-1)}{(2\ell)!} &= \frac{(2 \cdot 4 \cdot \cdots \cdot 2\ell)(2 \cdot 4 \cdot \cdots \cdot (2\ell-2) \cdot \ell)(2\ell-1)}{2\ell!} \\ &= \frac{2 \cdot 4 \cdot \cdots \cdot (2\ell-2) \cdot \ell}{1 \cdot 3 \cdot \cdots \cdot (2\ell-3)}\end{aligned}$$

which is greater than $\ell \ge 1$.

(ii) We give two proofs of this fact. The first is via generating functions. Consider the formal power series $A(x) = \sum_{\ell=1}^{\infty} \frac{x^\ell}{f(\ell)}$.

Then

$$A(x)^2 = \sum_{\ell=2}^{\infty} x^\ell \sum_{i=1}^{\ell-1} \frac{1}{f(i)f(\ell-i)} \tag{5.13}$$

But $A(x) = \sum_{\ell=1}^{\infty} -2\binom{\frac{1}{2}}{\ell}(-x)^\ell = 2 - 2(1-x)^{\frac{1}{2}}$ by the binomial theorem, so

$$A(x)^2 = 4(A(x) - x) = \sum_{\ell=2}^{\infty} \frac{4x^\ell}{f(\ell)} \tag{5.14}$$

Equating coefficients of 5.13 and 5.14, we see that

$$\sum_{i=1}^{\ell-1} \frac{f(\ell)}{f(i) \cdot f(\ell - i)} = 4.$$

Proof II: Here is a purely algebraic proof. Fix an ℓ. First note that

$$\sum_{0}^{n} \binom{x}{i}\binom{x}{\ell - i} = \binom{2x}{\ell}$$

for all real x; this identity is a polynomial in x and it holds for all integral $x \geq \ell$, since both sides count the number of ways to choose a subset of ℓ people from a room of x boys and x girls.

It follows that

$$\begin{aligned}
\sum_{1}^{\ell-1} \frac{f(\ell)}{f(i)f(\ell-i)} &= \sum_{1}^{\ell-1} -2\frac{\binom{\frac{1}{2}}{i}\binom{\frac{1}{2}}{\ell-i}}{\binom{\frac{1}{2}}{\ell}} \\
&= \frac{-2}{\binom{\frac{1}{2}}{\ell}}\left[\sum_{0}^{\ell}\binom{\frac{1}{2}}{i}\binom{\frac{1}{2}}{\ell-i} - \binom{\frac{1}{2}}{0}\binom{\frac{1}{2}}{\ell} - \binom{\frac{1}{2}}{\ell}\binom{\frac{1}{2}}{0}\right] \\
&= \frac{-2}{\binom{\frac{1}{2}}{\ell}}\left[\binom{1}{\ell} - 2\binom{\frac{1}{2}}{\ell}\right] \\
&= \frac{-2}{\binom{\frac{1}{2}}{\ell}}\left[-2\binom{\frac{1}{2}}{\ell}\right] = 4
\end{aligned}$$

■

We can also show tightness of our technique: there is no function which can be used to get a constant better than 4.

Lemma 5.8 *There does not exist a function $g : \mathbb{N} \to \mathbb{R}$ and a constant $0 < C < 4$ such that $g(1) = 1$ and for all $\ell \in \mathbb{N}$,*

(i) $g(\ell) \geq 1$

(ii) $\sum_{i=1}^{\ell-1} \frac{g(\ell)}{g(i)g(\ell-i)} \leq C$

Proof. Suppose there existed such a function. Let $A(x) = \sum_{1}^{\infty} \frac{x^i}{g(i)}$. Choose $a \in (\frac{C}{4}, 1)$. By condition (i), $A(a) \leq \sum_{1}^{\infty} a^i = \frac{a}{1-a}$ is a positive real number. By condition (ii),

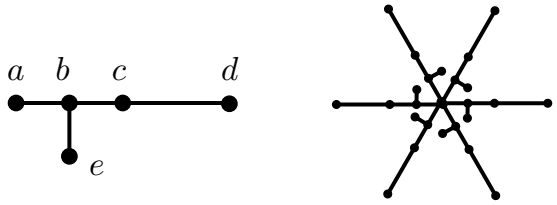
$$A(a)^2 = \left(\sum_{1}^{\infty} \frac{a^i}{g(i)}\right)^2$$

$$= \sum_{\ell=2}^{\infty} a^\ell \sum_{i=1}^{\ell-1} \frac{1}{g(i)g(\ell-i)}$$
$$\leq \sum_{\ell=2}^{\infty} C \frac{a^\ell}{g(\ell)}$$

Therefore $A(a)^2 + C\frac{a}{g(1)} \leq CA(a)$, or $A(a)^2 - CA(a) + Ca \leq 0$. But the quadratic $y^2 - Cy + Ca$ has discriminant $C^2 - 4Ca < 0$ and hence no real solutions in y. This is a contradiction, therefore no such function g exists. ■

5.2 A Lower Bound on the Potential Function, and on All-Swaps

In this section, we show that algorithm $\mathcal{B}_2$ performs asymptotically more than n swaps. Previously known examples only showed that $\mathcal{B}_2$ might need to perform at least $n-1$ swaps; the following example shows that the correct (worst-case) number lies between $1.25n$ and $2n$.



Consider the tree on the left where the edges have length 1, except cd has length 2. We take $k = m/4$ copies of it and identify the vertex a in all k copies to get the tree T on the right. (The figure shows an example $k = 6$.) There are $4k = m$ edges and $m + 1$ nodes in this tree, and the final metric will be the metric closure of this tree.

Suppose we give the vertices in the following order: we give all the copies of d, then copies of e, then of c, then the vertex a, and finally the copies of b. Each copy of d (aside from the first) adds a greedy edge of length 8. Each copy of e adds a greedy edge of length 4. The vertex a and each copy of c adds a greedy edge of length 2. Finally, each copy of B adds a greedy edge of length 1. This means $\prod_i \text{greedy}_i = 8^{k-1}4^k2^{k+1}1^k = 2^{6k-2}$. Moreover, after our algorithm finishes, the final tree is just the tree T, the product of whose edge lengths is $2^k1^{3k} = 2^k$. This gives us a ratio of $2^{5k-2} = 2^{1.25\,m-2}$. Hence we get that $\prod_i c(g_i) \geq 2^{1.25\,m-2} \prod_i c(\ell_i)$ for this instance. Moreover, it is easy to check that the number of swaps performed by our algorithm is also $1.25m - O(1)$, which proves the claim.

6 Conclusions

This paper considers maintaining an $O(1)$-competitive Steiner tree in an online environment. In this model, when a new vertex arrives the distances to previous vertices is revealed, and must form a metric space. The algorithm is allowed to add an edge connecting this new vertex to previous vertices, and also to add/delete a constant number of existing edges. It was previously known that a natural greedy algorithm makes a total of $O(n)$ additions/deletions and maintains a constant-competitive tree, which implies that the number of changes per arrival is constant *on average*. In this paper we give an algorithm that makes a single change per arrival *in the worst case*. Our idea is to use a new constant-amortized-swaps algorithm, which is then de-amortized by carefully delaying some of the swaps, and showing that these delays do not result in a significant blowup in cost. We also give a tight bound and a simpler proof of the natural greedy constant-average-swaps algorithm.

Several problems remain open: can we show that a primal-only greedy-like algorithm swap upon each arrival suffices to give $O(1)$-competitiveness? (See [Ver12] for a related conjecture.) We have not optimized the constants in our result, aiming for simplicity of exposition, but it would be useful to get a smaller constant factor that would put it in the realm of practicality. Moreover, can we extend our algorithm to the case where vertices are allowed to arrive and depart—the "fully-dynamic" case—and get even a constant amortized bound? Finally, for which other problems can we improve results by allowing a small number of changes in hindsight? And in what situations can we use similar de-amortization techniques?

References

[AA93] Noga Alon and Yossi Azar. On-line Steiner trees in the Euclidean plane. *Discrete Comput. Geom.*, 10(2):113–121, 1993.

[AAB04] Baruch Awerbuch, Yossi Azar, and Yair Bartal. On-line generalized Steiner problem. *Theoret. Comput. Sci.*, 324(2-3):313–324, 2004.

[AAPW01] Baruch Awerbuch, Yossi Azar, Serge A. Plotkin, and Orli Waarts. Competitive routing of virtual circuits with unknown duration. *J. Comput. Syst. Sci.*, 62(3):385–397, 2001.

[AGZ99] M. Andrews, M. X. Goemans, and L. Zhang. Improved bounds for on-line load balancing. *Algorithmica*, 23(4):278–301, 1999.

[AKR95] Ajit Agrawal, Philip Klein, and R. Ravi. When trees collide: an approximation algorithm for the generalized Steiner problem on networks. *SIAM J. Comput.*, 24(3):440–456, 1995.

[ANS09] Yuriy Arbitman, Moni Naor, and Gil Segev. De-amortized cuckoo hashing: provable worst-case performance and experimental results. In *ICALP (I)*, volume 5555 of *LNCS*, pages 107–118. 2009.

[BC97] Piotr Berman and Chris Coulston. On-line algorithms for Steiner tree problems. In *STOC*, pages 344–353, 1997.

[BN07] Niv Buchbinder and Joseph Naor. The design of competitive online algorithms via a primal-dual approach. *Found. Trends Theor. Comput. Sci.*, 3(2-3):front matter, 93–263 (2009), 2007.

[EL11] Leah Epstein and Asaf Levin. Robust algorithms for preemptive scheduling. In *ESA*, volume 6942 of *Lecture Notes in Comput. Sci.*, pages 567–578. Springer, Heidelberg, 2011.

[GW95] Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24(2):296–317, 1995.

[IW91] Makoto Imase and Bernard M. Waxman. Dynamic Steiner tree problem. *SIAM J. Discrete Math.*, 4(3):369–384, 1991.

[KM07] Adam Kirsch and Michael Mitzenmacher. Using a queue to de-amortize cuckoo hashing in hardware. In *Allerton*, pages 751–758, 2007.

[MSVW12] Nicole Megow, Martin Skutella, José Verschae, and Andreas Wiese. The power of recourse for online MST and TSP. In *ICALP (1)*, pages 689–700, 2012.

[Sch03] Alexander Schrijver. *Combinatorial optimization. Polyhedra and efficiency.*, volume 24 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin, 2003.

[SSS09] Peter Sanders, Naveen Sivadasan, and Martin Skutella. Online scheduling with bounded migration. *Math. Oper. Res.*, 34(2):481–498, 2009.

[SV10] Martin Skutella and José Verschae. A robust PTAS for machine covering and packing. In *ESA (I)*, volume 6346 of *LNCS*, pages 36–47. Springer, Berlin, 2010.

[Var11] Ashwinkumar Badanidiyuru Varadaraja. Buyback problem - approximate matroid intersection with cancellation costs. In *ICALP (1)*, pages 379–390, 2011.

[Vaz01] Vijay V. Vazirani. *Approximation algorithms*. Springer-Verlag, Berlin, 2001.

[Ver12] José Claudio Verschae. *The Power of Recourse in Online Optimization*. PhD thesis, Technischen Universität Berlin, 2012.

[Wes00] Jeffery Westbrook. Load balancing for response time. *J. Algorithms*, 35(1):1–16, 2000.

[WS11] David P. Williamson and David B. Shmoys. *The design of approximation algorithms*. Cambridge University Press, Cambridge, 2011.