New Geometry-Inspired Relaxations and Algorithms for the Metric Steiner Tree Problem*

Deeparnab Chakrabarty† Nikhil R. Devanur ‡ Vijay V. Vazirani§

Abstract

Determining the integrality gap of the bidirected cut relaxation for the metric Steiner tree problem, and exploiting it algorithmically, is a long-standing open problem. We use geometry to define an LP whose dual is equivalent to this relaxation. This opens up the possibility of using the primal-dual schema in a geometric setting for designing an algorithm for this problem.

Using this approach, we obtain a 4/3 factor algorithm and integrality gap bound for the case of quasi-bipartite graphs; the previous best integrality gap upper bound being 3/2 [RV99]. We also obtain a factor $\sqrt{2}$ strongly polynomial algorithm for this class of graphs.

A key difficulty experienced by researchers in working with the bidirected cut relaxation was that any reasonable dual growth procedure produces extremely unwieldy dual solutions. A new algorithmic idea helps finesse this difficulty – that of reducing the cost of certain edges and constructing the dual in this altered instance – and this idea can be extracted into a new technique for running the primal-dual schema in the setting of approximation algorithms.

1 Introduction

Some of the major open problems left in approximation algorithms are centered around LP-relaxations which researchers believe have not been fully exploited algorithmically, i.e., the best known algorithmic result does not match the best known lower bound on their integrality gaps. One of them is the bidirected cut relaxation for the metric Steiner tree problem [Edm67] and this is the main focus of our paper.

The integrality gap of the bidirected cut relaxation is believed to be very close to 1; the best lower bound known on the gap is 8/7, due to Goemans [Goe96] and Skutella[KPT07]. On the other hand, the known upper bound is the same as the weaker undirected cut relaxation which is 2 by a straightforward 2-factor algorithm for it. The only algorithmic results using the bidirected relaxation that we are aware of are: a 6/5 factor algorithm for the class of graphs containing at most 3 required vertices [Goe96] and a factor 3/2 algorithm for the class of quasi-bipartite graphs, i.e., graphs that do not have edges connecting pairs of Steiner vertices [RV99].

In this paper, we use geometry to develop a new way of lower bounding the cost of the optimal Steiner tree. The best such lower bound can be captured via an LP, which we call the *simplex-embedding LP*. A short description of this LP is that it is an ℓ_1-embedding of the given metric on

*Work supported by NSF Grant CCF-0728640. Preliminary version of this work appeared in the Thirteenth Conference on Integer Programming and Combinatorial Optimization (IPCO), May 26-28, 2008

†Department of Combinatorics and Optimization, University of Waterloo. Email: deepc@math.uwaterloo.ca

‡Microsoft Research, Redmond. Email: ndevanur@gmail.com

§College of Computing, Georgia Tech Email: vazirani@cc.gatech.edu

a simplex, maximizing a linear objective function. Interestingly enough, the dual of the simplex-embedding LP is a relaxation of the metric Steiner tree problem having the same integrality gap as the bidirected cut relaxation.

We believe our geometric approach would open up new ways to use the primal-dual schema for the bidirected cut relaxation. In this paper we exhibit progress made for the class of *quasi-bipartite* graphs – graphs in which the set of Steiner vertices are stable. In particular, we describe one dual growing procedure (the EMBED algorithm in Section (4)) which helps us prove the following property about the bidirected cut relaxation in quasi-bipartite graphs: If the minimum spanning tree is the optimum Steiner tree, then the LP relaxation is exact (Theorem (4.1)). Based on a modification of this algorithm, in Section 5 we design a primal-dual 3/2-approximate algorithm for *quasi-bipartite* graphs matching the previous known bounds by [RV99, Riz03]. However, our algorithms runs in nearly linear time while the previous algorithms were weakly polynomial.

A second key feature of our paper is the new algorithmic idea of *reduced costs*. We show that modifying the problem instance by reducing costs of certain edges and then running the primal-dual schema allows us to obtain provably better upper bounds on the integrality gap of the relaxation for quasi-bipartite graphs. We use this idea first to get a simple and fast algorithm which proves an upper bound of $\sqrt{2}$ on the integrality gap for quasi-bipartite graphs. Using another way of reducing costs, we improve the upper bound to 4/3. This algorithm (similar to the algorithm in [RV99, Riz03]) doesn't run in strongly polynomial time, unlike the $\sqrt{2}$ algorithm. We believe each of our algorithms demonstrates geometric and algorithmic ideas which could be insightful for studying the bidirected cut relaxation in general graphs.

We also give a second geometric LP in which Steiner vertices are not constrained to be on the simplex but are allowed to be embedded "above" the simplex. Again, the dual of this LP is a relaxation of the metric Steiner tree problem. We show that on any instance, the integrality gap of this latter LP is at most that of the bidirected relaxation. It turns out that this LP is in fact exact for Goemans' 8/7 example. However, there are examples for which the gap is 8/7 even for this relaxation. Could it be that this relaxation has a strictly smaller integrality gap than the bidirected cut relaxation?

1.1 Related Work

Historically, the idea of using an extra vertex to get a shorter tree connecting three points on the plane goes back to Torricelli and Fermat in the seventeenth century. The Euclidean Steiner tree problem, in its full generality, was first defined by Gauss in a letter to his student, Schumacher. This problem was made popular by the book of Courant and Robbins [CRS96], who attributed it to the nineteenth century geometer, Steiner. The rich combinatorial structure of this problem was explored by many researchers; e.g., see the books by Hwang, Richards and Winter [HRW92] and Ivanov and Tuzhilin [IT94].

The use of the bidirected cut relaxation for the Steiner tree problem goes back to Edmonds[Edm67] who showed the relaxation is exact for the the case of spanning trees. Wong [Won84] used this relaxation to study the Steiner arborescence problem, and Chopra and Rao[CR94a, CR94b] studied the properties of facets of the polytope defined by the relaxation. Goemans and Myung[GM93] study various undirected equivalent relaxations to the bidirected cut relaxation.

The bidirected cut relaxation has interesting structural properties which have been exploited in diverse settings. Let α denote the integrality gap of this relaxation. [JV02] use this LP for giving

a factor 2 budget-balanced group-strategy-proof cost-sharing method for the Steiner tree game; Agarwal and Charikar [AC04] prove that for the problem of multi-casting in undirected networks, the coding gain achievable using network coding is precisely equal to α. The latter result holds when restricted to quasi-bipartite networks as well. Consequently, for these networks, the previous best bound was 3/2 [RV99], and our result improves it to 4/3.

The best approximation algorithms for the Steiner tree problem is due to Robins and Zelikovsky [RZ05]. The authors prove a guarantee of 1.55 for general graphs and also show that when restricted to the quasi-bipartite case, the ratio is 1.28. However, it is not clear if these results would imply an upper bound on the integrality gap of the bidirected cut relaxation. Very recently, Konemann et.al. [KPT07] introduced another LP relaxation for the minimum Steiner tree problem which they show is as tight as the bidirected cut relaxation, and showed that the algorithm of Robins and Zelikovsky can be interpreted as a primal-dual algorithm on their LP. However, even their interpretation does not prove any upper bound on the integrality gap of their relaxation as they also compare with the optimum Steiner tree and not the optimum LP solution. Nevertheless, they show upper bounds for their relaxation for a larger class of graphs called b-quasi-bipartite graphs.[1] We stress that their LP seems quite different from ours – while we have variables for edges of the tree, they maintain variables and constraints for special sub-trees called *full components* much in the spirit of [RZ05]. It is an interesting question to study the relationship between their LP and ours.

2 Preliminaries

Let $G = (V, E)$ be an undirected graph with edge costs $c : E \to \mathbb{Q}_+$. Let $R \subseteq V$ denote the set of *required* vertices. These are also called *terminals* and we use both terms interchangeably. The vertices in $S = V \setminus R$ are called *Steiner* vertices. The Steiner tree problem is to find the minimum cost tree connecting all the required vertices and some subset of Steiner vertices. The edge costs can be extended to all pairs of vertices such that they satisfy triangle inequality (simply define the cost of (u, v) to be the cost of the shortest path from u to v). This version is called the metric Steiner tree problem. The two versions are equivalent.

We abuse notation and denote both the optimum Steiner tree and its cost as OPT. Also, given a set of vertices X, we denote the minimum spanning tree on X and its cost as $MST(X; c)$ or simply as $MST(X)$ when c is clear from the context.

Let $\mathcal{U} := \{U \subsetneq V : U \cap R \neq \varnothing \text{ and } U \cap R \neq R\}$ denote the subsets of V which contain at least one required vertex but not all. Let $\delta(U)$ denote the edges with exactly one end point in U. Note that any Steiner tree would use at least one edge in $\delta(U)$. This motivates the following *undirected cut relaxation* of the Steiner tree problem:

$$\min\{\sum_{e \in E} c(e)x_e : \quad x(\delta(U)) \geq 1, \ \forall U \in \mathcal{U}; \ x \geq 0\}$$

The MST on R is known (see for instance [GW95]) to be within factor 2 of the fractional optimum of this LP, so this relaxation has an integrality gap of at most 2. However the integrality gap can be arbitrarily close to 2 even when the set of Steiner vertices is empty. The example is that of an n-cycle with each edge having cost 1 where all vertices are required. The optimal Steiner

[1] A graph is b-quasi-bipartite if on deleting all required vertices, the largest size of any component is at most b

(spanning in this case) has value $n-1$ while the solution giving $x_e = 1/2$ on all edges is a valid solution of cost $n/2$.

2.1 The bidirected cut relaxation

Edmonds [Edm67] introduced the following stronger relaxation called the *bidirected cut relaxation*[2]. Fix an arbitrary required vertex r as root. Now replace each undirected edge (u, v) with two directed arcs $[u, v]$ and $[v, u]$, each of cost $c(u, v)$ (hence the name: bidirected). We will use square brackets to denote arcs and parantheses for undirected edges. Call the set of arcs $\vec{E}$. Call a subset U valid if it contains the root but not all the required vertices. Let $\mathcal{U} := \{U \subsetneq V : U \cap R \neq R \text{ and } r \in U\}$ denote the family of valid sets. The observation now is that if the edges of any Steiner tree are directed to point away from the root, then at least one arc in this directed arborescence must be in the cut set $\delta^+(U)$ of arcs going out of U. This gives the bidirected cut relaxation for the minimum Steiner tree problem.

$$\min\{ \sum_{[u,v] \in \vec{E}} c([u,v]) x_{[u,v]} : \quad x(\delta^+(U)) \geq 1, \quad \forall U \in \mathcal{U}; \quad x \geq 0\} \tag{BCR}$$

The above description of the LP seems to be dependent on the choice of the root r. However, it is known (see for instance [GM93]) that the choice of the root doesn't affect the value of the LP. We denote the optimum of the above LP on a graph G as $BCR(G)$.

Observe that the bidirected cut relaxation is stronger than the undirected cut relaxation. To see this note that any solution to LP(BCR) corresponds to a solution to the undirected cut relaxation: for every undirected edge add the variables on its two corresponding arcs. Edmonds [Edm67] showed that the bidirected cut relaxation is exact for the MST problem, i.e., the integrality gap of the relaxation is 1 for spanning tree instances. In fact, in Section 4 we will prove a stronger statement – we will show the integrality gap is 1 in quasi-bipartite graphs where the spanning tree is the optimal Steiner tree.

For Steiner trees however, no upper bound better than 2 (which is implied by the undirected cut relaxation) is known on the integrality gap of LP(BCR). Nevertheless, it is believed to be strictly better. Goemans [Goe96] showed an example where the gap is 8/7, which is the largest known lower bound on the integrality gap. The only algorithmic results using the bidirected relaxation that we are aware of prior to our work are: a 6/5 factor algorithm for the class of graphs containing at most 3 required vertices [Goe96] and a factor 3/2 algorithm by Rajagopalan and Vazirani [RV99] for the class of quasi-bipartite graphs, i.e., graphs that do not have edges connecting pairs of Steiner vertices.

A graph is called *quasi-bipartite* if there are no edges between any two Steiner vertices[3]. The class of quasi-bipartite graphs is a non-trivial class for the bidirected cut relaxation. In fact, recently Skutella (as reported by Konemann et.al.[KPT07]) exhibited a quasi-bipartite graph on 15 vertices for which the integrality gap of the bidirected cut relaxation is 8/7. Moreover, the best known hardness results for the Steiner tree problem in this class of graphs is quite close to that known in general graphs ($\frac{128}{127}$ versus $\frac{96}{95}$)[CC02].

[2]Actually, Edmonds was concerned only with the spanning tree problem, however the generalization to the Steiner tree problem is natural

[3]In the original graph G and not in its metric completion.

The bidirected cut relaxation has interesting structural properties which have been exploited in diverse settings. [JV02] use this LP for giving a factor 2 budget-balanced group-strategy-proof cost-sharing method for the Steiner tree game. Let α denote the integrality gap of this relaxation. Agarwal and Charikar [AC04] prove that for the problem of multi-casting in undirected networks, the coding gain achievable using network coding is precisely equal to α. The latter result holds when restricted to quasi-bipartite networks as well. Consequently, for these networks, the previous best bound was 3/2 [RV99], and our result improves it to 4/3.

Organization: In Section (3) we show the geometric theorem giving the lower bound, and other results relevant to it. In Section(4), we demonstrate a dual-growing embedding procedure for quasi-bipartite graphs. In Sections (5), (6) and (7) we give our 3/2, $\sqrt{2}$ and $\frac{4}{3}$ factor primal-dual approximation algorithms respectively.

3 A Geometric Lower Bound and its Consequences

We first present a special case of the geometric theorem, for the sake of ease of presentation. Let Δ_k be the unit simplex in $\mathbb{R}^k_+$, that is, $\Delta_k := \{x \in \mathbb{R}^k_+ : \sum_{i\in[k]} x(i) = 1\}$, where $x(i)$ is the ith coordinate of x. The corners of Δ_k are the unit vectors in $\mathbb{R}^k_+$ and we denote the set of k points as R. Let T be a tree whose vertices are R and a finite set S of points from Δ_k which are not the corners. For any vertex $v \in R \cup S$ of T, let $deg_T(v)$ denote the degree of the vertex in the tree T. Define the distance between two points to be half the ℓ_1-distance, also called the *variational distance*; for any two points $x, y \in \Delta_k$, $d(x, y) := \frac{1}{2}\sum_{i=1}^{k} |x(i) - y(i)|$. (The half is so that two corners are at a distance of 1). For any edge $e = (x, y)$, let $d(e) := d(x, y)$. Let $d(T) := \sum_{e\in T} d(e)$. Then

Theorem 3.1 $d(T) \geq k - 1$.

Note that if the set S were empty, that is T was simply a spanning tree of R, then the above relation holds with equality since any two corners are at a distance of 1. One way to interpret the above theorem is that Steiner points from the simplex don't improve upon the MST with respect to the variational distance of the corners of the simplex. This we believe is somewhat counter-intuitive, since in most geometric spaces, the Steiner points do improve upon the MST. What is special here is the ℓ_1-distance, and the location of the required vertices (they are the corners) on the simplex.

Proof: Let $R = \{e_1, e_2, \ldots, e_k\}$ be the unit vectors in $\mathbb{R}^k$. The proof follows by a counting argument. For every edge (x, y) in T, and $i \in [k]$, note that either x is on the unique path from y to e_i in T or y is on the unique path from x to e_i. In the first case we say x is *nearer* to e_i than y. If x is nearer to e_i than y, then we lower bound $|x(i) - y(i)|$ by $x(i) - y(i)$, otherwise we lower bound by $y(i) - x(i)$. Thus,

$$d(T) = \frac{1}{2}\sum_{(x,y)\in E}\sum_{i\in[k]} |x(i) - y(i)| \quad \geq \frac{1}{2}\sum_{(x,y)\in E}\sum_{i\in[k]} sgn_i(x, y) \cdot (x(i) - y(i)),$$

where $sgn_i(x, y)$ is 1 if x is nearer to e_i than y in T, and -1 otherwise.

Note that in the summation, the coefficient of $x(i)$ gets a contribution $+1$ for all of its neighbors in T except possibly the unique neighbor on the path from x to e_i, from which it gets a contribution

of -1. (We say possibly because if x were e_i itself, then there is no neighbor on the path from x to e_i). Thus, the coefficient of $x(i)$ is $\frac{1}{2}\left(deg_T(x) - 2\right)$ if $x \neq e_i$ and $\frac{1}{2}(deg_T(x))$ if $x = e_i$.

Therefore,

$$\begin{aligned} d(T) &\geq \frac{1}{2} \sum_{x \in T, i \in [k]} \left(deg_T(x) - 2\right) x(i) + \frac{1}{2} \sum_{i \in [k]} 2e_i(i) \\ &= \frac{1}{2} \sum_{x \in T} \left(deg_T(x) - 2\right) + \sum_{i \in [k]} 1 \\ &= k - 1. \end{aligned}$$

where the equality in the second line holds because $\sum_{i \in [k]} x(i) = 1$ and the last equality follows from the fact that in a tree $\sum_{x \in T} deg_T(x) = 2|V(T)| - 2$. $\square$

The general theorem allows two concessions on the location of the points: first, the points need not be on the *unit* simplex but rather on a λ-simplex; second, the set R need not be the corners of the λ-simplex.

The λ-simplex in k dimensions denoted as $\Delta_k^{(\lambda)}$ is defined as the set of points in $\mathbb{R}_+^k$ whose coordinates sum up to λ, for some parameter $\lambda > 0$. That is, $\Delta_k^{(\lambda)} := \{x \in \mathbb{R}^k : \sum_{i \in [k]} x(i) = \lambda\}$. Let $R = \{z_1, z_2, \cdots, z_k\}$ be a (ordered) set of k points in $\Delta_k^{(\lambda)}$. Define $\gamma(R) := (\sum_{i \in [k]} z_i(i) - \lambda)$. Let T be any tree whose vertices are R and a finite set S of other points in $\Delta_k^{(\lambda)}$. Let $d()$ be the variational distance defined above. Then,

Theorem 3.2 $d(T) \geq \gamma(R)$.

Note that when R is the set of corners of a unit simplex, the above implies Theorem (3.1).

Proof: Let $R' := \{e_1, \cdots, e_k\}$ be the corners of the λ-simplex. From T, construct the tree T' with vertices $v(T') = V(T) \cup R'$ and edges $E(T') = E(T) \cup \{(e_i, z_i) : \ i = 1 \cdots k\}$.

Note that the distance between z_i and e_i is $d(e_i, z_i) = \lambda - z_i(i)$. By scaling. Theorem (3.1) can be easily modified to give $d(T') \geq (k-1) \cdot \lambda$. Thus,

$$d(T) = d(T') - \sum_{i \in [k]} (\lambda - z_i(i)) \geq (k-1) \cdot \lambda - \sum_{i \in [k]} (\lambda - z_i(i)) = \gamma(R)$$

$\square$

3.1 A Lower Bound on OPT

Theorem (3.2) can be used to get a lower bound on the minimum Steiner tree of any graph $G = (R \cup S, E)$. Given G, suppose $|R| = k$ and $R = [k]$. Embed the vertices of the graph onto the λ-simplex in k dimensions using $z : V \to \Delta_k^{(\lambda)}$. Call an embedding z *valid* if for all edges $(u, v) \in E$, $c(u, v) \geq d(z_u, z_v)$. Henceforth, we will abuse notation and $d(u, v)$ to denote $d(z_u, z_v)$ and $d(T)$ to denote $\sum_{(u,v) \in E(T)} d(z_u, z_v)$. Let $z(R) := \{z_1, \ldots, z_k\}$ be the embedding of the vertices in R. Henceforth, we will use $\gamma(z)$ to denote $\gamma(z(R))$.

Now given any valid embedding z, Theorem 3.2 gives a lower bound on the cost of any Steiner tree T of G since $c(T) \geq d(T) \geq \gamma(z)$. In particular we have the following minimax inequality lower bounding the optimum Steiner tree.

$$OPT \geq \max\{\gamma(z) : z \text{ is a valid embedding } \}.$$

The above maximum can be obtained by solving the following linear program which we call the *simplex-embedding LP*.

$$\begin{aligned} \max \ \{\gamma(z) = \sum_{i\in[k]} z_i(i) - \lambda : \qquad & \text{(SimpEmb)} \\ \sum_{i\in k} z_v(i) = \lambda, \ \forall v \in V; & \\ z_v(i) - z_u(i) \leq d_i(u,v) \ \forall i \in [k], (u,v) \in E; & \\ z_u(i) - z_v(i) \leq d_i(u,v), \ \forall i \in [k], (uv) \in E; & \\ \frac{1}{2}\sum_{i\in[k]} d_i(u,v) \leq c(u,v), \ \forall (u,v) \in E; & \\ z_v(i), d_i(u,v) \geq 0, \ \forall v \in V, i \in [k], (u,v) \in E\} & \end{aligned}$$

Given input graph G, let $SE(G)$ denote the maximum of the above LP. The above discussion implies the following theorem.

Theorem 3.3 *Given any graph G, $OPT \geq SE(G)$.*

Remark 1: We should remark that the idea of embedding vertices of a graph onto a simplex to get a relaxation is not new. Calinescu et.al. [CKR98] use a similar LP relaxation for the multiway cut problem. In the multiway cut problem we are given a graph G and a set of terminals R with costs on edges and the goal is to find a minimum cost set of edges whose removal separates each pair of terminals. If $|R| = k$ then an equivalent formulation would be to embed the vertices onto the *corners* of a k-dimensional unit simplex so that the total weighted (weighted by costs of edges) distance (half ℓ_1 distance) between endpoints of edges is minimized. The relaxation is obtained by letting the points embed anywhere on the simplex. [CKR98] obtain a $3/2 - 1/k$ factor approximation algorithm for the problem which was improved to $1.3438 - o(k)$ by Karger et.al [KKS+04].

On the face of it the [CKR98] relaxation seems similar to the one described above. However there are key differences. Firstly, the relaxation of [CKR98] is a primal LP relaxation while ours is a dual relaxation. Moreover, in the [CKR98] relaxation the terminals always embed to the corners of the simplex; we need to be able to embed the terminals away from the corners. Arguably, with the idea of vertices being embedded to simplex being given, theirs seems to be a natural relaxation while ours crucially depends on Theorem 3.2 which we believe is non-trivial. Nevertheless, there could be a duality between the two LP relaxations, however it is not clear how one would try to achieve it. We also mention that [CGK05] prove that the [CKR98] relaxation is strictly stronger than the bidirected relaxation for the multiway cut, while in the next section we show that the above relaxation is equivalent to the bidirected cut relaxation for Steiner trees.

We end this remark by stating that the special case of Theorem 3.3 where only one Steiner vertex is allowed in the tree was essentially proved by [CKR98]. We point out that this follows simply from by writing out the expression for the half ℓ_1 distance for the k terminals and the single Steiner point and generalizing to more than one Steiner point needs the counting argument.

3.2 Connections to the bidirected cut relaxation

We now show that for any graph G and cost vectors c, the optimum of the simplex embedding LP equals the optimum of the bidirected cut LP described in Section 2. The proof uses the following dual of the simplex-embedding LP. We have a variable x_{uv} for every edge (u, v), variables $f_i([u, v])$ and $f_i([v, u])$ for each $i \in [k]$ and $(u, v) \in E$; and variable α_v for every vertex v. Apart from the free α_v's, every other variable is non-negative. Below, we multiply each of the constraints and the objective value by 2.

$$SE(G) = \min \ \{ \sum_{(u,v)\in E} c(u,v)x_{uv} : \qquad \text{(SE-Dual)}$$

$$x_{uv} \geq f_i([u,v]) + f_i([v,u]), \ \forall i \in [k], \ (u,v) \in E;$$

$$\sum_{v:(u,v)\in E} (f_i([u,v]) - f_i([v,u])) \geq \alpha(u), \ \forall i \in [k], u \in V \setminus i;$$

$$\sum_{v:(i,v)\in E} (f_i([i,v]) - f_i([v,i])) \geq \alpha(i) + 2, \qquad \forall i \in [k];$$

$$\sum_{v} \alpha(v) = -2;$$

$$f_i([u,v]), f_i([v,u]), x_{uv} \geq 0, \quad \forall (u,v) \ \in E, i \in [k]\}$$

To interpret LP(SE-Dual), one can think of x_{uv} as capacity of edge (u, v). There are k circulations - f_i for every required vertex i each satisfying the *capacity constraint* and moreover, the supplies (excess flows) for f_i at every vertex v is $\alpha(v)$ (it could be negative) except at the vertex i, where it is $\alpha(i) + 2$. These are the *supply constraints*. The last equality constraint implies the total supplies sum up to zero, as it should be in a circulation.[4]

Before going on to describe the equivalence of LP(BCR) and LP(SE-Dual), we show a direct proof that LP(SE-Dual) is a relaxation of the minimum Steiner tree problem giving an alternate proof of Theorem 3.3.

Alternate proof of Theorem 3.3: Given any Steiner tree T, consider the following solution to LP(SE-Dual): $x_{uv} = 1$ for all $(u, v) \in E(T)$, and 0 for all other edges; $\alpha(v) = deg_T(v) - 2$ for all $v \in V(T)$ and 0 for all other vertices. Note that $\sum_v \alpha(v) = -2$ and $\sum_{(u,v)\in E} c(u, v)x_{uv}$ is the cost of the tree. It remains to define the circulations. For required vertex i, consider the unique out-tree rooted at i formed by directing all edges of T away from i. For every edge (u, v) in T, let $f_i([u, v]) = 1$ if (u, v) is directed from u to v in the out-tree, or $f_i([v, u]) = 1$ if the arc is oriented otherwise. All other arcs carry 0 circulation. Note that for any vertex v in the tree other than i, the total flow coming in is 1, and flow going out is $deg_T(v) - 1$; while for i, the total flow going out is $deg_T(i)$. Thus the circulation satisfies the supply constraints. This shows that LP(SE-Dual) is a relaxation of the minimum Steiner tree problem. □

[4]Note that the constraint of the LP's only imply that the supplies are *at least* $\alpha(v)$ or $\alpha(i) + 2$. Since the sum of supplies adds up to 0 and the sum of the $\alpha(v)$'s equals -2, all these inequalities are actually tight.

We now go on to show the equivalence between LP(SE-Dual) and LP(BCR). In [GM93], Goemans and Myung provide two vertex weighted relaxations which are equivalent to the bidirected cut relaxation. Although our relaxation is different, our proof of equivalence follows on similar lines.

Theorem 3.4 *Given any graph G, $SE(G) = BCR(G)$.*

Proof: We show there exists a feasible solution to LP (BCR) of value Γ if and only if there exists a feasible solution to LP (SE-Dual) of value Γ. Recall that LP(BCR) (with root $r \in R$) was the following:

$$\min\{ \sum_{[u,v]\in\vec{E}} c([u,v])y_{[u,v]} : \quad y(\delta^+(U)) \geq 1, \quad \forall U \in \mathcal{U}; \quad y \geq 0\}$$

LP (BCR) $\geq$ **LP** (SE-Dual)**:** Let $\{y_{[u,v]}\}_{[u,v]\in\vec{E}}$ be a feasible solution of LP (BCR) of cost Γ with root r. The corresponding solution to LP (SE-Dual) is as follows: $x_{uv} := y_{[u,v]} + y_{[v,u]}$ and $\alpha(v)$ is the *supply* at vertex v which is the difference of the outgoing $y_{[v,u]}$'s and the incoming $y_{[u,v]}$'s, except at r where $\alpha(r)$ is the supply-2. That is,

$$\alpha(v) = \begin{cases} \sum_{u:(u,v)\in E}(y_{[v,u]} - y_{[u,v]}) & \text{if } v \neq r \\ \sum_{u:(u,v)\in E}(y_{[v,u]} - y_{[u,v]}) - 2 & \text{if } v = r \end{cases}$$

Note that $\sum_v \alpha(v) = -2$.

Now we describe the circulations. The circulation f_r corresponding to r just mimics the $y_{[u,v]}$'s. That is $f_r([u,v]) := y_{[u,v]}$ and $f_r([v,u]) := y_{[v,u]}$, for all edges (u,v). To get the circulations corresponding to another required vertex j, we use the fact that the *minimum r-j cut* (w.r.t. capacities y) in the digraph with arc set $\vec{E}$ and capacities $y_{[u,v]}$'s is at least 1 since the solution is feasible for LP (BCR). This implies there is a standard flow g_{rj} from r to j in this digraph. The flow f_j is found by *subtracting* $2g_{rj}$ from f_r.

To be precise, for each edge (u,v) we set

$$f_j([u,v]) := \max[0, f_r([u,v]) - 2g_{rj}([u,v])] + \max[0, 2g_{rj}([v,u]) - f_r([v,u])]$$

and

$$f_j([v,u]) := \max[0, f_r([v,u]) - 2g_{rj}([v,u])] + \max[0, 2g_{rj}([u,v]) - f_r([u,v])]$$

It is easy to check that f_j satisfies the constraints of LP(SE-Dual). Since g_{rj} increases the supply of j by 2 and decreases that of i by 2, the supply constraints are guaranteed. The capacity constraints are guaranteed by noting

$$f_j([u,v]) + f_j([v,u]) \leq \max[(f_r([u,v]) + f_r([v,u])), (f_r([u,v]) - f_r([v,u]) + 2g_{rj}([v,u]))]$$

both of which are less than $y_{[u,v]} + y_{[v,u]} = x_{uv}$. Thus the solution is feasible for LP(SE-Dual) and is of value Γ.

LP(SE-Dual) $\geq$ **LP**(BCR): Let $(\{x\}, \{f_i\}, \{\alpha\})$ (respectively over edges, arcs and vertices) be a solution to LP (SE-Dual). Without loss of generality, by adding circulations if necessary, we can assume for all edges (u,v), and i we have $x_{uv} = f_i([u,v]) + f_i([v,u])$. For LP(BCR), let r be the

chosen root. Then the solution is: for all edges (u,v), $y_{[u,v]} := f_r([u,v])$ and $y_{[v,u]} := f_r([v,u])$ To see feasibility for LP (BCR), we must show across any cut S separating r and a required vertex j, we have $\sum_{[u,v]\in\delta^+(S)} f_r([u,v]) \geq 1$. To see this, consider the flow g_{rj}: the difference between f_r and f_j. To be precise:

$$g_{rj}([u,v]) = \max[0, f_r([u,v]) - f_j([u,v])] + \max[0, f_j([v,u]) - f_r([v,u])]$$

Observe that the supply of the flow g_{rj} on every vertex other than r or j is 0 and on r is 2 and j is -2. This is because the supplies of f_r and f_j match on every vertex other than r and j where they differ by 2. Thus it is a standard flow from r to j of value 2, implying across any cut S as above, $\sum_{[u,v]\in\delta^+(S)} g_{rj}([u,v]) \geq 2$. The proof ends by noting for any arc $[u,v]$, $g_{rj}([u,v]) \leq 2f_r([u,v])$, because each term in the above definition of g_{rj}, is less than $f_r([u,v])$. The second term is less since $f_r([u,v]) + f_r([v,u]) = x_{uv} \geq f_j([v,u])$. □

What the above theorem shows is that the LP(SimpEmb) can be interpreted as a geometric dual to the bidirected cut relaxation. In Section 4 we explain how we design algorithms using the primal-dual schema with this new geometric dual.

3.3 A stronger LP relaxation

Note that in LP(SE-Dual), the $\alpha(v)$'s are free variables. This is because the interpretation of $\alpha(v)$ as $deg_T(v) - 2$ implies that for leaves $\alpha(v)$ could be negative. However, in any optimal Steiner tree, no Steiner vertex would be a leaf. Therefore, we could add the constraint $(\alpha(v) \geq 0,\ \forall v \in S)$, and the new relaxation, call it LP(SE-Dual2), would still be a feasible relaxation for the minimum Steiner tree problem.

$$\begin{aligned}
\min\ \{ \sum_{(u,v)\in E} c(u,v)x_{uv} : \qquad & \text{(SE-Dual2)}\\
x_{uv} \geq f_i([u,v]) + f_i([v,u]),\ \forall i \in [k],\ (u,v) \in E;& \\
\sum_{v:(u,v)\in E} (f_i([u,v]) - f_i([v,u])) \geq \alpha(u),\ \forall i \in [k], u \in V \setminus i;& \\
\sum_{v:(i,v)\in E} (f_i([i,v]) - f_i([v,i])) \geq \alpha(i) + 2, \qquad \forall i \in [k];& \\
\sum_v \alpha(v) = -2;& \\
f_i([u,v]), f_i([v,u]), x_{uv} \geq 0, \quad \forall (u,v)\ \in E, i \in [k];& \\
\alpha(v) \geq 0, \quad \forall v \in S\} \qquad & (1)
\end{aligned}$$

The corresponding change in the simplex-embedding LP is that for the Steiner vertices, the equality constraint $\sum_{i\in[k]} z_v(i) = \lambda$ is relaxed to an inequality constraint $\sum_{i\in[k]} z_v(i) \geq \lambda$. Call this new LP, LP(SimpEmb2). In other words, the Steiner vertices are allowed to embed "above" the simplex. It is not too hard to extend Theorem(3.2) to capture this as well.

It is clear that the optimum value of LP(SimpEmb2) is at least that of LP(SimpEmb) and thus it gives as good a lower bound. In fact, the following example in Figure(1) shows that on some

instances the former is strictly better, and thus LP(SE-Dual2) is strictly a better relaxation than LP(SE-Dual) and also by Theorem(3.4), better than LP(BCR).

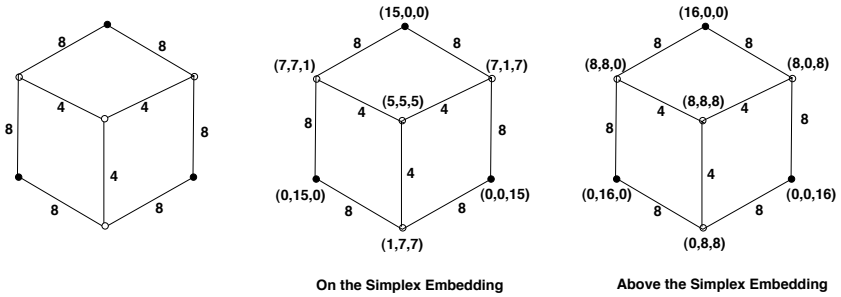

Figure 1: In the figure, black vertices are required and white ones are Steiner. Integrality gap of the bidirected cut relaxation for the graph is known to be 16/15 (due to Goemans, as reported in Chap 23 of [Vaz01]). The middle figure shows an embedding on the 15-simplex attaining a value of 30. The figure to the right shows how we can get a higher value of 32 if we allow Steiner vertices to move above the 16-simplex. Note that the Steiner vertex at the center is not on the 16-simplex.

Nevertheless, the example of Skutella which shows a gap of 8/7 for the bidirected cut relaxation can be shown to also give the same gap for LP(SE-Dual2). We provide Skutella's example and integrality gap of 8/7 for the stronger LP in Appendix A. Henceforth, in the remainder of the paper we will investigate LP(SimpEmb) and thus all our results will imply results for the bidirected cut relaxation.

3.4 Primal-Dual schema with the geometric dual

All the algorithms we give for the minimum Steiner tree problem fall in the following primal-dual schema: we construct a Steiner tree T and a feasible embedding z of the vertices of the graph onto a λ-simplex; moreover we prove $c(T) \leq \rho \cdot \gamma(z)$, and thus get a ρ-approximation for the Steiner tree problem as well as an upper bound on the integrality gap of LP(SE-Dual) and LP(BCR).

In the remainder of the paper we restrict ourselves to the special class of graphs called *quasi-bipartite graphs*. Such graphs do not have Steiner-Steiner edges. The best known upper bound on the integrality gap of LP(BCR) on such graphs was 3/2[RV99].

For such graphs, we first describe the dual growing procedure, EMBED, which gives us a feasible embedding of vertices. EMBED has the property that on quasi-bipartite graphs, if the MST on the required vertices is the optimal Steiner tree, then it returns an embedding z such that $\gamma(z)$ equals the cost of the MST on terminals. Otherwise, it returns a Steiner vertex v such that $MST(R \cup v) < MST(R)$.

The EMBED algorithm stops after returning the first Steiner vertex. In fact, one can extend EMBED to obtain more Steiner vertices, however a natural extension to do so doesn't give good enough dual – that is, there is an example where the natural dual growing procedure stops with a dual which is around 1/2 the cost of the optimal tree. For details, refer Section 5. In Section 5, we show how to modify EMBED and give a primal-dual algorithm achieving the upper bound of 3/2 for quasi-bipartite graphs. The algorithm can be made to run in (almost) linear time. We note that Rizzi's [Riz03] algorithm runs in time $O(n^2 T(n,m) \log_2 c_{max})$ where $T(n,m)$ is the time taken to compute an MST in a n-vertex, m-edge graph, while c_{max} is the maximum cost of an edge.

In Sections 6 and 7, we use EMBED *in rounds* (with a processing step before each round) to get $\sqrt{2}$ and $\frac{4}{3}$ factor approximations respectively for quasi-bipartite graphs. The $\sqrt{2}$ algorithm runs in strongly polynomial time, while the 4/3 algorithm takes the same time as Rizzi's algorithm.

4 The EMBED algorithm

In this section, we describe the dual growing procedure EMBED which given a quasi-bipartite graph G and a cost function c does the following.

Case 1: If $MST(R)$ is the optimal Steiner tree, then it returns a feasible embedding z such that $\gamma(z) = MST(R)$. Note, in this case, $MST(R) = OPT = BCR$, since $MST(R) \geq OPT \geq BCR \geq \gamma(z)$.

Case 2: Or, returns a Steiner vertex v whose addition *strictly* helps the MST on R, that is, $MST(R \cup v) < MST(R)$. We say that EMBED *crystallizes*[5] v.

The following theorem is immediate.

Theorem 4.1 *Given a quasi-bipartite instance G, if the addition of no Steiner vertex reduces the cost of $MST(R)$, then $MST(R) = BCR(G)$. In particular the integrality gap for the instance is* 1.

Note that the theorem implies the following important property about the bidirected cut relaxation for quasi-bipartite graphs: If the minimum spanning tree is optimal, then the relaxation is exact. We are now ready to describe EMBED.

The following is a continuous description of the algorithm, which can be easily discretized. The algorithm has a notion of time. Time starts at $t = 0$ and increases at unit rate. At any time t, all required vertices are on the t-simplex, all Steiner vertices are below the t-simplex (sum of coordinates is less than t). The algorithm maintains a set of terminal-terminal edges T, which form a forest at any time t. Let K denote a connected component of required vertices formed with the edges of T and $\mathcal{K}$ denote the set of all connected components. At time $t = 0$, the algorithm starts with $T = \varnothing$ and the components are singleton required vertices. All vertices start at the origin at $t = 0$. Before describing how the embedding at time t is formed, we make a few definitions.

Definition 1 *At any point of time, $d(u, v)$ be the half-ℓ_1 distance between z_u and z_v, that is, $d(u, v) := \frac{1}{2}\sum_i |z_u(i) - z_v(i)|$. Let $d^+(\{u, v\}) := \sum_{i \in X_u}(z_u(i) - z_v(i))$, where X_u are the coordinates in which u dominates v. That is, $X_u := \{i : z_u(i) \geq z_v(i)\}$. Note that d^+ takes an ordered pair of vertices.*

Claim 4.2 *If u and v are both on a λ-simplex (for some $\lambda > 0$), then $d^+(\{u, v\}) = d^+(\{v, u\}) = d(u, v)$.*

Proof: Note by definition, $d(u, v) = \frac{1}{2}(d^+(\{u, v\}) + d^+(\{v, u\}))$. Also, if u, v are both on the λ-simplex we get $d^+(\{u, v\}) = \sum_{i \in X_u} z_u(i) - \sum_{i \in X_u} z_v(i) = \lambda - \sum_{i \in X_v} z_u(i) - (\lambda - \sum_{i \in X_v} z_v(i)) = \sum_{i \in X_v} z_v(i) - \sum_{i \in X_v} z_u(i) = d^+(\{v, u\})$. □

[5]The notation of crystallizing is from [RV99]; they also define a process called crystallization to find 'useful' Steiner vertices.

Definition 2 *Given a vertex v and a component K, let $d(v,K) := \min_{u\in K} d(u,v)$. Similarly, let $d^+(\{v,K\}) := \min_{u\in K} d^+(\{v,u\})$ and $d^+(\{K,v\}) := \min_{u\in K} d^+(\{u,v\})$. For any two components $K, L \in \mathcal{K}$, let $c(K,L) := \min_{u\in K, v\in L} c(u,v)$. An edge (u,v) is* tight *if $d(u,v) = c(u,v)$.*

Definition 3 *A Steiner vertex v* links *to a component K if there exists $i \in K$ so that $d^+(\{i,v\}) = c(i,v)$. The edge (i,v) is called a* link *of v to K.*

We first give the intuition of the algorithm. At time $t = 0$ all vertices are embedded to the origin of R^k_+. Note that we are in k-dimensions and thus we correspond every coordinate to a required vertex. As time moves on, required vertices increase their respective coordinates till some edge goes tight in which case the vertices merge to form components and their coordinate increases are "coupled". Steiner vertices v stay at the origin till some required vertex i goes "far enough" so that $d^+(\{i,v\} = c(i,v)$. We use the $d^+()$ distance rather than the $d()$ since the Steiner vertices are not on the simplex, but below it. Recall that when on the simplex (Claim 4.2), the two distances match. The algorithm goes on till either all the required vertices merge into a single component (Case 1), or a Steiner vertex hits the simplex (Case 2). We now give the details of the algorithm.

Algorithm 1 EMBED

Required vertices: For each component K and required vertex $i \in K$, the algorithm increases the jth coordinate of i at rate $1/|K|$, for each $j \in K$. Clearly, this will keep required vertex i on the t-simplex. When an edge (i,j) goes tight, the algorithm merges the components containing i and j and adds (i,j) to T. It is instructive to note that when restricted to only required vertices, this actually mimics Kruskal's MST algorithm.

Steiner vertices: For each Steiner vertex v, the algorithm increases its coordinates depending on the components it has linked to. For each component K that v is linked to, the coordinates $z_v(i)$ for all $i \in K$ increase at rate $1/|K|$. Thus, henceforth $d^+(\{i,v\})$ remains the same for all terminals $i \in K$. Note that at $t = 0$, the Steiner vertex is linked to no component and remains at the origin till $t = c(i,v)$, where i is the closest terminal to v. Also note that if a Steiner vertex v is linked to *exactly one* component K, then rate of growth of $z_v(i)$ equals that of $z_i(i)$, for all $i \in K$.

The algorithm terminates if the number of components becomes 1 (Case 1) or a Steiner vertex v hits the simplex, that is $\sum_{j=1}^k z_v(j) = t$, and is linked to more than one component (Case 2). In Case 1, the algorithm runs the following projection step.

Projection Step: If Case 1 happens at time $t = \lambda$, for every Steiner vertex v and coordinate j, $z_v(j) := z_v(j)\frac{\lambda}{\sum_i z_v(i)}$. Note that each Steiner vertex is projected on to the λ-simplex. The coordinates of the required vertices are kept the same.

The embedding procedure is described in Algorithm 1. The example in Figure 4 illustrates the algorithm on a graph with three required vertices.

We now show Algorithm 1 satisfies the conditions mentioned at the beginning of the section. In Case 1, the algorithm returns a tree T and embedding z. The way the required vertices are moved, it is clear that terminals stay on the t-simplex and no terminal-terminal edge is over tight.

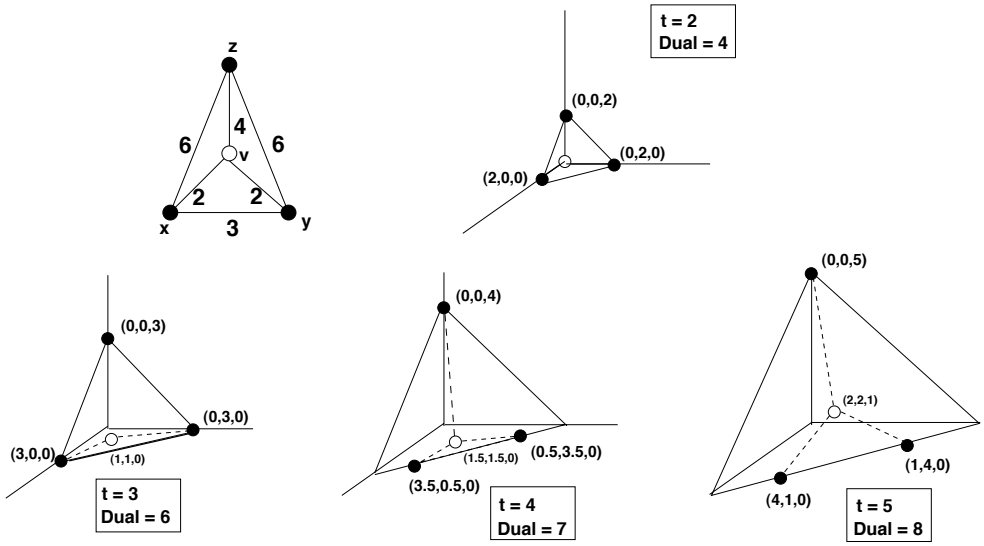

Figure 2: Snapshots of the running of EMBED on the graph above at times $t = 2, 3, 4, 5$. At time $t = 2$, the Steiner vertex v links to the required vertices x and y, and increases its x and y coordinates at rate 1. At time $t = 3$, x, y merge. The edge (x, y) goes into $Remove(v)$. At time $t = 4$, v links to z, and moves in the zth coordinate as well. At $t = 5$, it hits the 5-simplex, terminating the algorithm. The tree shown with dotted lines pays exactly for the dual and is cheaper than the MST.

Steiner vertices always stay *below* the t-simplex – if they hit the t-simplex since we are in Case 1 there is exactly one component to which it is linked and thus due to the movement of the terminals, the Steiner remains on a t-simplex throughout. To see that terminal-Steiner edges are not over tight, note that before the projection step, we have for every terminal i and every Steiner vertex v, $d^+(\{i, v\}) \leq c(i, v)$ (otherwise v links to the component containing i). After projection step, the coordinates of the Steiner vertex only increase, which means $d^+(\{i, v\})$ only decreases. Moreover, since v is on the simplex, we have (by Definition 1) $d^+(\{i, v\}) = d(i, v)$ and thus $d(i, v) \leq c(i, v)$.

We now need to show that tree T has cost $\gamma(z)$. In fact we prove something stronger. Given any connected component K, denote the restriction of T to K as $T[K]$.

Lemma 4.3 *At any instant of time t, for any connected component K,* $c(T[K]) = \sum_{i \in K} z_i(i) - t$.

Proof: At time $t = 0$, the lemma holds vacuously. Since the quantity $\sum_{i \in K} z_i(i)$ increases at the same rate as time, we need to prove the lemma only in the time instants when two components merge. Suppose K, K' merge at time instant t due to edge (i, j) which comes in the tree, with $i \in K, j \in K'$. Note $d(i, j) = c(i, j) = t$. So for the new connected component $K \cup K'$, $\sum_{i \in K \cup K'} z_i(i) - t = \sum_{i \in K} z_i(i) - t + \sum_{i \in K'} z_i(i) - t + t = c(T[K]) + c(T[K']) + c(i, j) = c(T[K \cup K'])$. □

Thus, we have established correctness if we are in Case 1.

In Case 2, when v hits the simplex (suppose at time $t = \lambda$), the algorithm returns v as the

Steiner vertex helping the minimum spanning tree. Since v hits the simplex there must be at least two components to which it is linked to. Suppose they are $\hat{K}_1, \ldots, \hat{K}_{\hat{r}}$ with $\hat{r} \geq 2$. Let $\hat{P} = \bigcup_{\ell=1}^{\hat{r}} \hat{K}_\ell$ be the vertices of K_ℓ's. We now show $MST(P \cup v) < MST(\hat{P})$ which will imply $MST(R \cup v) < MST(R)$ by extending the MST to all required vertices.

With each Steiner vertex v, we associate a subset of edges $Remove(v)$ of T. Suppose v is linked to K and K' and these merge at time t, due to edge (i, j), $i \in K$ and $j \in K'$. At this point, (i, j) is added to the set $Remove(v)$. Thus, a Steiner vertex may have more than one link into the same component, but for each extra link, there is an edge in $Remove(v)$. At any instant of time, let v be linked to $K_1, \ldots, K_r$, let $P := \bigcup_{\ell=1}^{r} K_\ell$ and let T_v be the tree formed by adding all the links incident at v to $\bigcup_{\ell=1}^{r} T[K_\ell]$ and deleting $Remove(v)$. The proof of the following lemma is very similar to that of Lemma 4.3.

Lemma 4.4 *At any instant of time, $c(T_v) = \sum_{i \in P} z_i(i) - \sum_{i \in P} z_v(i)$.*

Proof: At time $t = 0$, the lemma holds vacuously. Since the quantity $\sum_{i \in P} z_v(i)$ increases precisely at the rate $\sum_{i \in P} z_i(i)$, we need only check the lemma when v links to a *new* component K. Suppose this happens at time t. This means, there is a terminal $j \in K$ with $c(v, j) = t$. Note that $\sum_{i \in K} z_i(i) = t$, by the way required vertices move. Thus the increase in both left-hand side and right-hand side is t, and thus equality holds. □

Hence at time $t = \lambda$, when v hits the simplex, $\sum_{i \in \hat{P}} z_v(i) = \lambda$, and so $c(T_v) = \sum_{i \in \hat{P}} z_i(i) - \lambda$. From the proof of Lemma 4.3, we have that the RHS is at most $MST(\hat{P})$ with equality iff all the vertices of $\hat{P}$ are in a single component. Since $\hat{r} \geq 2$, we see that v improves the MST. Thus, we have proved the following theorem.

Theorem 4.5 *Given a quasi-bipartite graph G, the algorithm* Embed *either returns a terminal spanning tree T and feasible embedding z with $c(T) = MST(R) = \gamma(z)$; or returns a Steiner vertex v with $MST(R \cup v) < MST(R)$.*

Remark: Note that the above algorithm and analysis do not use the fact the cost satisfies triangle inequality. We would need this for our algorithms in Sections 6 and 7 to work.

5 A $\frac{3}{2}$-factor approximation algorithm

The dual growing procedure Embed suggests the following primal-dual algorithm: At each step, maintain the connected components of T; when a Steiner vertex v hits the simplex, *merge* all the components v was linked to by adding v and the various links connected to it to T and continue the dual growing procedure.

However, as the example in Figure(5) suggests, such an algorithm cannot give anything better than a factor 2.

The above example suggests a shortcoming of the Embed procedure: it grows the coordinates of all the Steiner vertices, although any optimum Steiner tree contains at most one Steiner vertex of degree larger than 2. To be precise, the k vertices at the bottom increases the coordinates of all the ℓ Steiner vertices at a very high rate (at rate k), although only one of them is useful for connecting the k terminals.

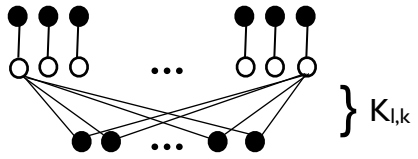

Figure 3: In the above graph, the black vertices are terminals and the white vertices are Steiner. There are ℓ Steiner vertices connected to k terminals via a complete bipartite graph $K_{\ell,k}$. Think of $\ell \gg k \gg 1$. There are ℓ other terminals forming a perfect matching with the l Steiner vertices. Each edge is of length 1. Note that the optimum Steiner tree costs at least 2ℓ. In the procedure EMBED, all the Steiner vertices hit the t-simplex simultaneously at time $t = 1 + \frac{1}{k}$. Subsequently, no more dual can be grown and thus the total dual obtained is $(k + \ell - 1)(1 + 1/k) \sim k + \ell$.

The main idea behind the 3/2-factor algorithm is that it uses EMBED to recognize *useful* (degree ≥ 3) Steiner vertices *early* (not wait till they hit the simplex) and on recognizing such a vertex, merge the components linked to the vertex so that these components do not raise the coordinates of other Steiner vertices.

In short, the algorithm runs EMBED until some Steiner vertex is linked to three distinct connected components. If that is the case, it merges all the components, and adds the Steiner vertex and the three links to the tree. The idea is that this Steiner vertex should be helpful. With this modification, we see that if some Steiner vertex hits the simplex, it must do so being connected to at most *two* components. In this case, the algorithm merges these components, adds the Steiner vertex and the (possibly two) links to the tree, and continues. The algorithm terminates when there is a single component.

The algorithm maintains a set of edges and vertices, T, which is initialized to $E(T) = \varnothing$ and $V(T) = R$. At every time instant t, the algorithm maintains a family of connected components $\mathcal{K}$ of T; and an embedding $z : V \to \mathbb{R}^k_+$ of the vertices. We emphasize that components in $\mathcal{K}$ contain Steiner vertices unlike those in EMBED. At time $t = 0$, all the vertices of the graph are embedded at the origin. Time increases at rate 1. We state the algorithm formally in Algorithm 2 SMART-EMBED. We use the definitions 1,2 and 3 from Section 4.

It is clear that the algorithm returns a Steiner tree T in the end. Moreover, the feasibility of the embedding returned by EMBED also implies the feasibility of the embedding z returned by SMART-EMBED. We finish the section with the following theorem which shows that the tree is within 3/2 of the optimal.

Theorem 5.1 *The algorithm* SMART-EMBED *returns a tree T and a feasible embedding z, with $c(T) \leq \frac{3}{2}\gamma(z)$.*

Proof:
We show that at any time t, for any connected component K we have $c(T[K]) \leq \frac{3}{2}(\sum_{i \in K \cap R} z_i(i) - t)$. Henceforth, we abuse notation and denote $c(T[K])$ by $c(K)$. The proof follows by induction on the size of K. Moreover it is enough to check that the inequality is preserved after every merge step as between merge steps both sides of the inequality remain unchanged.

Algorithm 2 SMART-EMBED

Component Vertices: For each component $K \in \mathcal{K}$, and for each terminal $i \in K \cap R$, the algorithm increases the ith coordinate of every vertex $v \in K$ (required or Steiner) at rate $1/|K \cap R|$.

Other Vertices: For each Steiner vertex v not in any component, the algorithm increases its coordinates depending on the components it has linked to. For each component K that v is linked to, the coordinates of v corresponding to K increase at rate $1/|K \cap R|$.

Merge Step: A Merge Step happens when one of the three events take place:

1. For some two components $K, L \in \mathcal{K}$, we have $c(K, L) = t$. Recall that $c(K, L) := \min_{u \in K, v \in L} c(u, v)$ and now u or v (but not both since graph is quasi-bipartite) could be Steiner vertices. Since we project (as we see below) Steiner vertices of a component to the t-simplex, the distance between these components is also t.

   In this case, we merge K and L into one component. All Steiner vertices linked to either K or L now link to the new component. Moreover, if a Steiner vertex is linked to both K and L, the costlier of its links to K and L is removed. This maintains that a Steiner vertex has only one link to a component. Let (u, v) be the edge which achieves $c(K, L)$. Add (u, v) to T.

2. Some Steiner vertex v not in any component hits the t-simplex, that is, $\sum_{i \in [k]} z_v(i) = t$. In this case, merge v and all the components it links to into one single component. For each component K, v links to, add the cheapest links of v to K to T. Add v to T as well.

3. Some Steiner vertex v not in any component links to three components K_i, K_j, K_l. In this case, merge v and K_i, K_j, K_l into one component. Once more all Steiner vertices linked to either K_i,K_j or K_l now link to the new component. Steiner vertices which link to two out of the three, discard their costlier link, as in Step 1. Add v and its links to K_i, K_j, K_l to T. Also project v onto the t-simplex as in EMBED.

Projection Step: Once the number of components is 1, we do a projection step as in the EMBED procedure.

Suppose at time t, a merging of type 1 occurs: components K and L merge into one component, and the edge (u, v) is added to T. Therefore, we have $c(K) \leq \frac{3}{2}(\sum_{i \in K \cap R} z_i(i) - t)$ and $c(L) \leq \frac{3}{2}(\sum_{i \in L \cap R} z_i(i) - t)$. The cost of the new component $(K \cup L)$ increases by the cost of the edge (u, v). Note that by definition, $c(u, v) = t$. That is,

$$\begin{aligned} c(K \cup L) = c(K) + c(L) + t &\leq \frac{3}{2}\Big(\sum_{i \in K \cap R} z_i(i) - t\Big) + \frac{3}{2}\Big(\sum_{i \in L \cap R} z_i(i) - t\Big) + t \\ &\leq \frac{3}{2}\Big(\sum_{i \in (K \cup L) \cap R} z_i(i) - t\Big) \end{aligned} \tag{2}$$

Suppose at time t, a merging of type 2 occurs: the Steiner vertex v hits the simplex. First observe that there are exactly 2 components v is linked to. The algorithm maintains that a Steiner vertex is linked to *at most* 2 components. Moreover, if there is only 1 component v is linked to,

the rate of growth of sum of its coordinates is the same as that for the terminals and the Steiner vertex won't hit the simplex.

Call the two components K and L. Let the links to K and L be (v,i) and (v,j) respectively. Without loss of generality let (v,i) be the shorter one with $c(v,i) =: a \leq c(v,j) =: b \leq t$. Note that the increase in the left hand side is $a+b$. That is, $c(K \cup L \cup v) = c(K) + c(L) + (a+b)$. Now note that whenever a link from v is removed (this happens only when two components linked to the same Steiner vertex merge), we always remove the costlier link from v. Thus, the smaller link (v,i) is in fact the *first* link of v. That is, before time a, the coordinates of v were all 0. By the running of the algorithm, a Steiner vertex is linked to at most two components at all times. Thus, the rate of increase of sum of coordinates is at most 2. Therefore at time t when the vertex hits the simplex, the sum of coordinates (which is t since it hits the simplex) is at most $2(t-a)$. This gives a bound on t: $2(t-a) \geq t$ implying $t \geq 2a$. Furthermore, note that $t \geq b$ since the terminal j links to a vertex v at time $c(j,v) = b$ which must be before t. Thus $a + b \leq 3t/2$. This gives us

$$c(K \cup L \cup v) \leq c(K) + c(L) + \frac{3}{2}t \leq \frac{3}{2}(\sum_{i \in (K \cup L) \cap R} z_i(i) - t)$$

where the last inequality follows as in Inequality 2.

Suppose a merging of type 3 happens at time t: a Steiner vertex links to components K, L, M via links $(v,i), (v,j)$ and (v,l) respectively. Each of these links have cost less than t. Thus the cost of the new component

$$c(K \cup L \cup M \cup v) \leq c(K) + c(L) + c(M) + 3t \leq \frac{3}{2}(\sum_{i \in (K \cup L \cup M)} z_i(i) - t)$$

where the last inequality follows from induction on the components K, L, M. This completes the proof. □

The above algorithm can be implemented in time $O((|E| + |V|)\log(|V|))$ thus giving a nearly linear time 3/2-factor algorithm for quasi-bipartite graphs. However, the implementation details are a little tedious and not quite the main focus of this paper, and so we defer the proof of the next theorem to Appendix B.

Theorem 5.2 *Given a weighted quasi-bipartite graph $G(V = R \cup S, E; c)$, there exists an algorithm which runs in time $O((|E|+|V|)\log(|V|))$ and returns a tree T with cost within $3/2$ times the optimal Steiner tree.*

6 A $\sqrt{2}$ Factor Approximation Algorithm

In this section and the next, we give algorithms for the Steiner tree problem using EMBED as a black-box unlike the 3/2-factor algorithm. We will require nothing more than the statement of Theorem 4.5. Both algorithms would use EMBED in rounds to obtain *useful* Steiner vertices, although each round will be preceded by a pre-processing step. In this section we look at a $\sqrt{2}$-factor algorithm.

Notation 1 *$MST(U; c)$ denotes the minimum cost spanning tree on vertices U given the costs c. Based on the context, it also denotes the cost of this tree.*

We start by giving a high level idea of our algorithm. The algorithm will finally return a cost c_2 and a subset of Steiner vertices $X \subseteq S$ such that

1. The optimal Steiner tree w.r.t. c_2 is the MST on the terminals. Equivalently, by Theorem 4.5, EMBED when run on G, c_2 terminates with a feasible embedding z with $\gamma(z) = MST(R; c_2)$.
2. $MST(X \cup R; c) \leq \sqrt{2} \cdot MST(R; c_2)$

The costs c_2 will be only smaller than c; therefore, z is also feasible for c. Hence, the two conditions imply that we get a factor $\sqrt{2}$ approximation.

Initially, $X = \emptyset$ and we obtain c_2 by reducing the costs of the required-required edges by a factor of $\sqrt{2}$ and leaving the costs of required-Steiner edges unchanged. We denote the reduced cost at this point as c_1 which we use later. Clearly Condition 2 is satisfied now, and will remain an *invariant* of the algorithm.

Suppose that condition 1 is not satisfied, that is, when EMBED is run on G, c_2, a Steiner vertex $v \in S$ hits the simplex. At this point, the algorithm adds v to X, and modifies c_2 by reducing the costs of certain required-required edges further, as detailed below. This has the effect that if EMBED is run with these new costs, v does not hit the simplex. Moreover, the Condition 2 above is maintained. Hence in each iteration, a new Steiner vertex is added to X, implying termination in at most $|S|$ rounds.

We now give the intuition behind modifying the costs so that the invariant is maintained. The first step of scaling all the required-required edges acts as a "global filter" which filters out Steiner vertices that only help a little. If a Steiner vertex v now hits the simplex, then adding it to X reduces the cost of $MST(R \cup X; c)$ so much that decreasing the cost of required-required edges "local" to it to $\frac{1}{2}$ of the original costs still maintains the invariant Condition 2. This requires an involved argument (Lemma 6.2) that amortizes the improvements due to all the vertices previously added to X. Moreover, since all the local required-required edges have been decreased to half their original cost, this has the additional desired effect that running EMBED on the new costs doesn't let v hit the simplex – that is v itself is filtered out.

Now the formal description of the algorithm follows.

Definition 4 *Applying the* global filter *with parameter $\rho > 1$ gives a cost c_1 defined as $c_1(i,j) = \frac{c(i,j)}{\rho}$ for all $i, j \in R$, and $c_1(i,v) = c(i,v)$ for all $i \in R$ and $v \in S$.*

Definition 5 *Applying a* local *filter w.r.t $X \subseteq S$ gives a cost c_2. Let $T = MST(R \cup X; c_1)$, and for each $u \in X$, $Clos(u)$ denote the closest required vertex to u. Let $\Gamma_T(u)$ be the neighbors of u in T_1. The cost c_2 is defined as*

$$c_2(i,j) = \begin{cases} \frac{1}{2}c(i,j) & \text{if there exists } u \in X \text{ such that } i = Clos(u) \text{ and } j \in \Gamma_T(u) \\ c_1(i,j) & \text{otherwise} \end{cases}$$

Algorithm 3 PRIMAL-DUAL

1. Apply global filter with parameter $\rho = \sqrt{2}$ to get c_1.
   Initialize $X \leftarrow \varnothing$; $c_2 \leftarrow c_1$.

2. **Repeat** till EMBED returns z
   Run EMBED on G, c_2.
   If EMBED returns v then
   $X = X \cup v$; Apply local filter w.r.t X to get c_2.

3. Return $T_1 = MST(R \cup X; c), z$.

Claim 6.1 *At each round a new Steiner vertex enters X.*

Proof: Let c_2 be obtained by applying the local filter w.r.t. X. Let $T_1 = MST(R \cup X; c_1)$. We claim that EMBED run on G, c_2 will not crystallize any vertex $v \in X$. This will complete the proof.

Suppose EMBED on $G; c_2$ does crystallize $v \in X$. Then by Theorem 4.5 $MST(R \cup v; c_2) < MST(R; c_2) =: T_2$. That is, there exist a set of edges A, each of the form (v, j) where j is a terminal; and a set of terminal-terminal edges in $E(T_2)$ so that $c_2(A) = c_1(A) < c_2(B)$; and $T_2 \setminus B \cup A$ is a tree spanning $R \cup v$.

Call an edge *diminished* if $c_2(e) = \frac{1}{2}c(e)$. Now, among the edges in B, some are diminished and some are in $E(T_1)$; and each diminished edge e corresponds to two edges e_1 and e_2 in $E(T_1)$ one of which, say e_1, has $c_1(e_1) \geq c_2(e)$. Thus from B, we get a subset of edges $B' \subseteq E(T_1)$ with $c_1(B') \geq c_2(B)$. Moreover, note that $T_1 \setminus B' \cup A$ is a valid spanning tree of $R \cup X \cup v$. This implies $MST(R \cup X \cup v; c_1) < MST(R \cup X; c_1)$, which is a contradiction. □

Lemma 6.2 *(Invariant Condition 2) Let X be the set of Steiner vertices at any stage of the algorithm. Then, $MST(R \cup X; c) \leq \sqrt{2} \cdot MST(R; c_2)$.*

Proof: Let $T_1 := MST(R \cup X; c_1)$. Clearly, $MST(R \cup X; c) \leq c(T_1)$. Let $T_1 = E_0 \cup E_1$, where E_0 denotes the required-Steiner edges and E_1 denotes the required-required edges of T_1. We bound the costs of these two sets separately. Let E_2 be the set of edges modified by the local filter, that is, e such that $c_2(e) = \frac{1}{2}c(e)$. Recall such edges are called *diminished*. Define T_2 to be $E_2 \cup E_1$. It can be shown that T_2 is an $MST(R, c_2)$.

Claim 6.3 *T_2 is an MST of R with respect to cost c_2.*

Proof: By definition of the local filter, it is clear that $E_2 \cup E_1$ is a terminal spanning tree. If it is not a minimum one w.r.t cost c_2, there exists an edge $e \in E(T_2)$ and an edge $f \notin E(T_2)$ such that $T_2 - e + f$ is spanning and $c_2(f) < c_2(e)$. Note that $c_2(f) = c_1(f)$, since $f \notin E_2$. Also, since $f \notin E_1 \cup E_0$, $T_1 + f$ has a cycle, and in fact it is the precisely the cycle formed in $T_2 + f$ with the edges in E_2 replaced by length 2 paths containing a Steiner vertex. Moreover, by definition of the local filter, there will be an edge e' in the cycle in $T_1 + f$ such that $c_1(e') \geq c_2(e)$, and thus $T_1 + f - e'$ will be a cheaper spanning tree of $R \cup X$ w.r.t. costs c_1, contradicting the choice of T_1. □

We have $c(T_1) = c(E_0) + c(E_1)$, $c_2(T_2) = c_2(E_2) + c_2(E_1)$. The proof follows from the following.

- $c(E_0) \leq \sqrt{2} c_2(E_2)$.
  This is essentially a consequence of the observation that $c_1(T_1) \leq MST(R; c_1)$. Since $T_2 = E_1 \cup E_2$ is also a spanning tree of R, we get $c_1(T_1) \leq c_1(T_2)$. Expanding the costs, we get

  $$c_1(E_0) + c_1(E_1) \leq c_1(E_2) + c_1(E_1).$$

  Since E_0 are required vertex-Steiner edges, $c_1(E_0) = c(E_0)$. $c_1(E_2) = c(E_2)/\sqrt{2} = \sqrt{2} c_2(E_2)$ by definition, giving us $c(E_0) \leq \sqrt{2} c_2(E_2)$.

- $c(E_1) \leq \sqrt{2} c_2(E_1)$.
  Since E_1 costs are not modified by the local filter, $c_2(E_1) = c_1(E_1)$ and in fact the relation holds with equality.

□

Theorem 6.4 *The algorithm* Primal-Dual *terminates in at most $|S|$ rounds, returning a Steiner tree T_1 and a feasible embedding z of G, c such that $c(T_1) \leq \sqrt{2} \cdot \gamma(z) \leq \sqrt{2} \cdot OPT$.*

Proof: By Claim 6.1, the algorithm terminates in $|S|$ rounds. Moreover, we have $\gamma(z) = MST(R; c_2)$. Using Lemma 6.2, we get $MST(R \cup X; c) \leq \sqrt{2} \cdot MST(R; c_2) = \sqrt{2} \cdot \gamma(z) \leq \sqrt{2} \cdot OPT$. □

In fact, the above algorithm has a faster implementation. Although the algorithm constructs the set X in a certain order, it turns out that the order does not matter. Hence it is enough to simply apply the global filter and go through the Steiner vertices (in any order) once, picking the ones that help. We describe this in Algorithm 4 Reduced One-Pass Heuristic .

Algorithm 4 Reduced One-Pass Heuristic

1. Apply global filter with parameter $\rho = \sqrt{2}$ to get c_1.
   Initialize $X \leftarrow \varnothing$;
2. For all $v \in S$,
   If $MST(R \cup X \cup v; c_1) < MST(R \cup X; c_1)$, then
   $X = X \cup v$;
3. Return $T_1 = MST(R \cup X; c_1)$.

Theorem 6.5 *There exists a feasible embedding z of G, c such that for T_1 returned by Algorithm* Reduced One-Pass Heuristic*, $c(T_1) \leq \sqrt{2} \cdot \gamma(z)$.*

The proof of Theorem (6.5) is quite similar to Theorem (6.4) and is deferred to Appendix C for completeness. Note that the above algorithm makes at most $|S|$ minimum spanning tree computations and is hence is very efficient. In particular, it runs in strongly polynomial time.

7 A $\frac{4}{3}$ Factor Approximation Algorithm

The primal-dual $\frac{4}{3}$ approximation algorithm is along the lines of the one in the previous section, with the major difference being that it drops Steiner vertices from X when beneficial. The other differences are that it applies the global filter with $\rho = 4/3$, and the definition of a local filter is somewhat different. And like the earlier algorithm, the order of vertices picked/dropped does not matter. As a result it can be implemented as a local search algorithm with an extra global filtering step, which is what we present here.

Algorithm 5 REDUCED-LOCAL-SEARCH

1. Apply global filter with parameter $\rho = 4/3$ to get c_1.
   Initialize $X \leftarrow \varnothing$, $T_1 = MST(R; c_1)$;
2. Repeat
   If $\exists v$ such that $MST(R \cup X \cup v; c_1) < c_1(T_1)$, $X = X \cup v$.
   If $\exists v$ such that $MST(R \cup X \setminus v; c_1) < c_1(T_1)$, $X = X \setminus v$.
   $T_1 = MST(R \cup X; c_1)$.
   Until No such v exists.
3. Return T_1.

The plain local search algorithm (without the global filtering step) was studied [RV99] who showed that this algorithm gives a 3/2 factor approximation for quasi-bipartite graphs. This factor is tight. So the simple modification of applying a global filter provably improves the performance of this algorithm. It was shown in [Riz03] that this algorithm can be implemented efficiently in time $O(|V||S|T(|V|, |E|) \log_2(c_{max}))$ where $T(n, m)$ is the time taken to compute the minimum spanning tree in an n-vertex m-edge graph, and c_{max} is the maximum cost of an edge.

We show that T_1 returned by the algorithm is within 4/3 of the optimal by exhibiting an embedding z of value greater than 3/4 times the cost of T_1. As in Section (6), the analysis proceeds by defining cost c_2 and constructing tree T_2. The factor 4/3 comes from the parameter ρ used in the global filter and the following property of T_1.

Lemma 7.1 *The degree of every Steiner vertex in T_1 is at least 4.*

Proof: It is easy to see that T_1 doesn't have vertices of degree 1 or 2. Suppose there existed a Steiner vertex $v \in T_1$ with $deg(v) = 3$. Let a, b, c be the required vertices connected to v and assume $c_1(va) \leq c_1(vb) \leq c_1(vc)$ without loss of generality. Now by triangle inequality property of c, we know $c(va) + c(vb) \geq c(ab)$. Since $c(va) = c_1(va)$ and $c(vb) = c_1(vb)$, we get $\frac{3}{4}(c_1(va) + c_1(vb)) \geq \frac{3}{4}c(ab) = c_1(ab)$. Similarly $\frac{3}{4}(c_1(va) + c_1(vc)) \geq c_1(ac)$. Thus $c_1(ab) + c_1(ac) \leq \frac{3}{4}(2c_1(va) + c_1(vb) + c_1(vc)) \leq c_1(va) + c_1(vb) + c_1(vc)$. Thus $MST(R \cup X)$ would choose[6] (ab) and (ac), rather than choosing $(va), (vb), (vc)$. □

[6]It would seem that (ab) and (ac) could be equal to $(va) + (vb) + (vc)$ and improve the MST – in this case we break ties by excluding Steiner vertices if possible

Theorem 7.2 *For the tree T_1 returned by* REDUCED-LOCAL-SEARCH*, there exists a feasible embedding z such that $c(T_1) \leq \frac{4}{3} \cdot \gamma(z)$.*

Proof:

As in the proof of Lemma 6.2, denote the edges of T_1 as $E_1 \cup E_0$. Define c_2 as: For every Steiner vertex $v \in T_1$ and for every $j \neq Clos(v)$ connected to v in T_1, let $c_2(Clos(v), j) = c_1(vj)$. Note that $c_1(vj) \leq c_1(Clos(v), j)$, for otherwise T_1 would have picked $(Clos(v), j)$ instead of (vj). Call these required vertex-required vertex edges *diminished*. For every other edge, $c_2(e) := c_1(e)$. Let E_2 be the set of diminished edges and let $T_2 := E_1 \cup E_2$, be a required vertex spanning tree. By the conditions of the algorithm, since T_1 is an MST of $R \cup X$ with costs c_1 and no Steiner vertices help X, T_2 is an MST of R with costs c_2 and no Steiner vertex helps T_2. Thus, by Theorem (4.1) running EMBED on G, c_2 returns a feasible embedding z of value $c_2(T_2)$. We now bound the cost of T_1.

We have $c(T_1) = c(E_1) + c(E_0) = c(E_1) + c_1(E_0)$. Note that $c_2(T_2) = c_1(E_1) + c_2(E_2)$ since E_1 is not diminished. As in the proof of Theorem (6.4), we argue term by term. By definition we have $c(E_1) = \frac{3}{4} c_1(E_1)$.

Every Steiner vertex $v \in T_1$ contributes $deg(v) - 1$ edges to E_2 and $deg(v)$ edges in E_0, where $deg(v)$ is the degree of v in T_1. By definition the $deg(v) - 1$ edges have cost exactly the cost of the largest $deg(v) - 1$ edges of the $deg(v)$ edges it contributes to E_0. By lemma (7.1), $deg(v) \geq 4$ and thus we get $c_1(E_0) \leq \frac{4}{3} c_2(E_2)$. Adding, we get $c(T_1) \leq \frac{4}{3} c_2(T_2) = \frac{4}{3} \gamma(z)$. □

References

[AC04] A. Agarwal and M. Charikar. On the advantage of network coding for improving network throughput. In *Proceedings of the IEEE Information Theory Workshop*, 2004.

[CKR98] G. Calinescu and H. J. Karloff and Y. Rabani. An improved algorithm for the MULTIWAY CUT. *Proceedings of Symposium on Theory of Computation (STOC)*, 1998.

[CGK05] C. Chekuri and A. Gupta and A. Kumar. On a bidirected relaxation for the MULTIWAY CUT problem. *Discrete Applied Math.*, 150(1-3):67–79, 2005.

[CC02] M. Chlebik and J. Chlebikova. Approximation hardness of the steiner tree problem on graphs. *Proceedings of Scandinavian Workshop on Algorithm Theory*, 2002.

[CLRS01] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms* Second Edition. MIT Press and McGraw-Hill, 2001.

[CR94a] S. Chopra and M. R. Rao. The Steiner tree problem I: Formulations, compositions and extension of facets. *Math. Programming*, 64:209–229, 1994.

[CR94b] S. Chopra and M. R. Rao. The Steiner tree problem II: Properties and classes of facets. *Math. Programming*, 64:231–246, 1994.

[CRS96] R. Courant, H. Robbins, and I. Stewart. *What Is Mathematics?: An Elementary Approach to Ideas and Methods.* Oxford Papebacks, 1996.

[Edm67] J. Edmonds. Optimum branchings. *Journal of Research of the National Bureau of Standards. Section B*, 71:233–240, 1967.

[GM93] M. Goemans and Y. Myung. A catalog of Steiner tree formulations. *NETWORKS*, 23:19–23, 1993.

[Goe96] M. Goemans. Personal communication with third author. 1996.

[GW95] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, pages 1115–1145, 1995.

[HRW92] F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner Tree Problem*, volume 53 of *Annals of Discrete Mathematics*. North-Holland, Amsterdam, Netherlands, 1992.

[IT94] A. O. Ivanov and A. A. Tuzhilin. The Steiner problem and its generalizations. CRC Press, BocaRaton, Ann Arbor, London, Tokyo, 1994.

[JV02] K. Jain and V. V. Vazirani. Equitable cost allocations via primal-dual-type algorithms. In *Proceedings of 33rd ACM Symposium on Theory of Computing*, 2002.

[KKS+04] D. Karger and P. Klein and C. Stein and M. Thorup and N. Young. Rounding Algorithms for a Geometric Embedding of Minimum Multiway Cut. *Math. of Operations Research*, 29(3): 436–461, 2004.

[KPT07] J. Könemann, D. Pritchard, and K. Tan. A partition based relaxation for Steiner trees. *Manuscript*, 2007.

[Riz03] R. Rizzi. On Rajagopalan and Vazirani's 3/2-approximation bound for the iterated 1-Steiner heuristic. *Information Processing Letters*, 86:335–338, 2003.

[RV99] S. Rajagopalan and V. Vazirani. On the bidirected cut relaxation for the metric Steiner tree problem. In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, 1999.

[RZ05] G. Robins and A. Zelikovsky. Tighter bounds for graph Steiner tree approximation. *SIAM Journal of Discrete Mathematics*, 19:122–134, 2005.

[Vaz01] V. V. Vazirani *Approximation Algorithms.* Second Edition, Springer, 2001.

[Won84] R. T. Wong. A dual ascent approach for Steiner trees on a directed graph. *Mathematical Programming*, 28:271–287, 1984.

A An 8/7 Integrality Gap Example for LP (SE-Dual2)

In this section we show that the LP relaxation LP SE-Dual2 described in Section 3 has integrality gap lower bounded by 8/7. We first recall the LP relaxation.

$$\begin{aligned}
\min \ \{ \sum_{(u,v)\in E} c(u,v)x_{uv} : \qquad & \text{(SE-Dual2)}\\
x_{uv} \geq f_i([u,v]) + f_i([v,u]),\ \forall i \in [k],\ (u,v) \in E;& \\
\sum_{v:(u,v)\in E} (f_i([u,v]) - f_i([v,u])) \geq \alpha(u),\ \forall i \in [k], u \in V \setminus i;& \\
\sum_{v:(i,v)\in E} (f_i([i,v]) - f_i([v,i])) \geq \alpha(i) + 2, \qquad \forall i \in [k];& \\
\sum_{v} \alpha(v) = -2;& \\
f_i([u,v]), f_i([v,u]), x_{uv} \geq 0, \quad \forall (u,v) \in E, i \in [k];& \\
\alpha(v) \geq 0, \quad \forall v \in S\} \qquad & (3)
\end{aligned}$$

The example showing a gap of 8/7 is due to Martin Skutella which was reported by Könemann et.al. [KPT07]. We produce the example below from their paper.

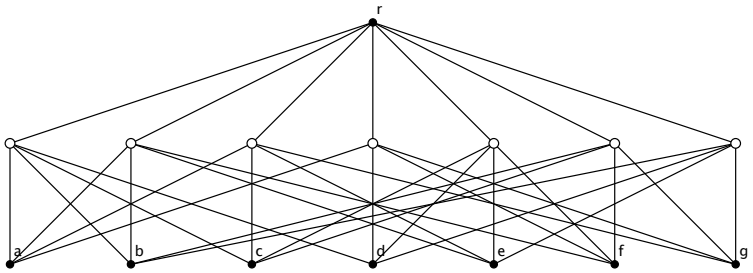

Figure 4: The nodes in white are Steiner nodes and nodes in black are required. Each edge has unit cost. The optimum Steiner tree is of cost 10.

It can be seen (by inspection, or there exists a proof in [KPT07]) that the optimum Steiner tree costs 10. We now demonstrate a solution to LP SE-Dual2 of cost 35/4 which will show that the gap of the LP is at least 8/7. In fact the cost 35/4 is optimal for the LP since the bidirected cut value is exactly 35/4 for this example, and as proved in Section 3, LP SE-Dual2 can only be larger.

The solution gives $x_e = 1/4$ for every edge. Note that the value is then 35/4. The α values are as follows: $\alpha(r) = -1/4$. $\alpha(i) = -1$ for every other required vertex. $\alpha(u) = 3/4$ for every Steiner vertex u. Note that $\sum_{v\in V} \alpha(v) = (-7 + 21/4 - 1/4) = -2$. It remains to describe the circulations. We will describe the circulation f_r for the root r and f_a for the terminal a. All the other circulations are symmetric to f_a.

f_r sends a flow of $1/4$ from r to every Steiner vertex and a flow of $1/4$ from every Steiner vertex to terminal which is not r. Thus, the supply on r is $7/4 = 2 + (-1/4)$. The supply on a Steiner vertex is $1 - 1/4 = 3/4$ and that on a non-root terminal is -1.

To describe f_a, let S be the set of Steiner vertices incident on a. f_a is the same as f_r except on the edges of the form (v, a) and (r, v) for $v \in S$. In f_r, $f_r(v, a) = f_r(a, v) = 1/4$ for all $v \in S$. In f_a, these are reversed. That is, $f_a(a, v) = f_a(v, r) = 1/4$ for all $v \in S$. All the other flows are the same. Now, the total supply on r is $(-1 + 3/4) = -1/4$, the total supply on Steiners remain the same, and the total supply on a is $1 = (-1 + 2)$. Thus, we get a feasible solution to LP SE-Dual2 of value $35/4$.

B An Implementation of Smart Embed in time $O((|E|+|V|)\log|V|)$

Theorem B.1 *(Theorem 5.2) Given a weighted quasi-bipartite graph $G(V = R \cup S, E; c)$, there exists an algorithm (Algorithm 6 below) which runs in time $O((|E| + |V|)\log(|V|))$ and returns a tree T with cost within $3/2$ times the optimal Steiner tree.*

We now describe an implementation of the Algorithm Smart-Embed which runs in almost linear time ($O((|E| + |V|)\log|V|$). Note that Smart-Embed has a notion of time, and this implementation mimics the same by only maintaining the times at which the merge operations of Smart-Embed take place.

For every edge $(u, v) \in E$, we maintain a variable `t(u,v)` $:= c(u, v)$ the cost of the edge. Thus `t(u,v)`'s are *static* variables. The idea is that the edge (u, v) comes to play in Smart-Embed precisely at time $t =$ `t(u,v)`. For instance, if the vertices u and v are required, at time $t =$ `t(u,v)` the components containing them merge. Moreover a Steiner vertex v links to a component at time `t(u,v)` for some terminal u in the component.

We also maintain a time-variable `t(v)` for every Steiner vertex v which is not in any component. `t(v)` stands for the time at which the Steiner vertex v will hit the simplex if no other merge steps occur in between. Initially `t(v)` is set to ∞ (which for implementation issues can be assumed to be an integer larger than sum of all edge weights).

We maintain a list L of `t(u,v)`'s (for all edges (u, v)) and `t(v)`'s (for Steiner vertices v). These are precisely the set of times at which the merge steps of Smart-Embed take place. We remark that the list is not static as `t(v)`'s can change as the algorithm progresses. However, we always maintain L sorted in non-decreasing order of times. The implementation reads the list L in the sorted order and depending on the entry read, and whenever applicable, performs one of the three merge steps of Smart-Embed. We show that these merges can all be made to run with an amortized $O(\log|V|)$ time per entry in L, and since there are at most $O(|E| + |V|)$ entries in L, the implementation runs in almost linear time of $O((|E| + |V|)\log|V|)$.

Data Structures:
We now describe the data structure used for the implementation. The data structures are similar to disjoint set data structures implemented via linked lists (see for example, [CLRS01]). These are used in an implementation of Kruskal's minimum spanning tree algorithm.

For each vertex v of the graph, we maintain 4 pointers: *head* - which points to the connected component containing v, *next* - which points to the next vertex in v's connected component, *copy1* and *copy2* - these point to two copies of the same vertex, as we see these will be used by only Steiner vertices and for terminals the two pointers will always be `null`.

The components $K \in \mathcal{K}$ are stored as linked lists of vertices. For every component, one of the vertices will be an identifier vertex. Each vertex in the component will point to this identifier vertex via their *head* pointers. Thus, given a vertex v, the component which contains it can be found in $O(1)$ time. Initially, we start with $|V|$ linked lists, one for each vertex. However, there are only $|R|$ (number of terminals) components corresponding to the terminals; the *head* pointers of the Steiner vertices are initialized to `null`. For each component, we also maintain the size of the component. We also maintain a pointer *tail* which points to the last vertex in the list.

For each Steiner vertex $v \in S$, we store two copies of it, call them v_A and v_B. These correspond to the two possible components a Steiner vertex could link to. Each of these copies are linked with the original vertex via their *copy* pointers. Whenever a Steiner vertex v is picked in a component, these copies are deleted and the Steiner vertex is treated like a terminal henceforth.

For every component K, we maintain another linked-list, $\textsc{Steiner}(K)$, which contains the list of all Steiner vertices K is linked to. The list contains only copies of the Steiner vertices and not the original Steiner vertex. $\textsc{Steiner}(K)$ is stored similarly as K is stored: a linked list with all Steiner vertices pointing to an identifier vertex for $\textsc{Steiner}(K)$. The lists K and $\textsc{Steiner}(K)$ are doubly linked to each other for all K, thus given a Steiner vertex v one can find the components to which it is linked to in $O(1)$ operations. In fact, we will call this function $\textsc{Link}(v)$ which returns a set (which we also call $\textsc{Link}(v)$, abusing notation) of (identifiers of) connected components linked to v. $\textsc{Link}(v)$ is of at most size two, for all Steiner vertices v. For any Steiner vertex v, we also maintain a set of edges $\textsc{link-edges}(v)$ which contains the *unique* links of v to the various components in $\textsc{Link}(v)$.

To calculate `t(v)`, we maintain a bunch of variables for every Steiner vertex v. We store a variable $\texttt{z}(v)$ for every Steiner vertex which is supposed to contain the sum-of-coordinates of v as in Smart-Embed. $\texttt{z}(v)$ is initialized to 0 for every Steiner vertex. We maintain a variable $\texttt{last}(v)$, which is the last time when a modification had to be made to $\texttt{z}(v)$. These will correspond to instances when v links to a component, or when two components linked to v merge. $\texttt{last}(v)$ is also initialized to 0. We maintain $\texttt{rate}(v)$, which is the rate at which $\texttt{z}(v)$ increases since $\texttt{last}(v)$. $\texttt{rate}(v)$ will always equal to $|\textsc{Link}(v)|$. Using $\texttt{last}(v), \texttt{rate}(v)$ and $\texttt{z}(v)$, at any time `t(u,v)`, one can easily figure out `t(v)`: if $\texttt{rate}(v) = 1$, $\texttt{t(v)} = \infty$, otherwise $\texttt{t(v)} = \texttt{t(u,v)} - \texttt{z}(v)$.

We now describe how Smart-Embed can be implemented using these data structures. Initially, the set of components $\mathcal{K}$ is initialized to be all singleton terminals. That is, all terminals have their *head* pointer pointing to themselves. All `t(v)`'s are set to ∞. The list L is read in non-decreasing order. If the entry read is `t(u,v)` for some edge (u, v) and u and v are not in the same component, then the following is done. If u and v are both terminals, the components corresponding to them are merged and (u, v) is added to the tree. The set of Steiner vertices $\textsc{Steiner}(K(u))$ and $\textsc{Steiner}(K(v))$ are also merged and for Steiner vertices w linked to both $K(u)$ and $K(v)$, $\texttt{z}(w), \texttt{last}(w), \texttt{rate}(w)$ and `t(w)` are modified. If u is a terminal and v is a vertex not linked to $K(u)$, then either v links to $K(u)$ if $|\textsc{Link}(v)| \leq 1$, or a merge step corresponding to merge step 3 of Smart-Embed takes place. If the entry of L read corresponds to `t(v)` for some Steiner vertex v, then it means v has hit the simplex in Smart-Embed. The components in $\textsc{Link}(v)$ are merged and the link-edges

LINK-EDGES(v) are added to the tree. The algorithm terminates when there is a single component formed.

What remains to be described is how to perform the merge steps in $O(\log |V|)$ amortized time per entry of L. It is known that if the sets to be merged are always disjoint, then this can be done using the disjoint-set data structures. In our case, any two components are disjoint. However, for two components K_1 and K_2, the list of linked Steiner vertices STEINER(K_1) and STEINER(K_2) might not be disjoint and the above argument for disjoint data structures fail. Nevertheless, we can show that the total time taken by the merge steps can be bounded by $O(|E| + |V| \log |V|)$. The main idea is this: if the two sets STEINER(K_1) and STEINER(K_2) have a "big" enough union, then the argument for disjoint set data structures can be made to go through. If the union is not "big", then the intersection must be "big". Note that a vertex v in the intersection of STEINER(K_1) and STEINER(K_2) is a vertex linked to both K_1 and K_2. When two such components merge, the costlier of the two link-edges in LINK-EDGES(v) is discarded and this never plays a role in the algorithm subsequently. Thus, all the operations (which are only $O(1)$) made on v in this merge operation can now be charged to the costlier edge in LINK-EDGES(v). The algorithm details are in Algorithm 6: Implementation of SMART-EMBED.

Algorithm 6 Implementation of SMART EMBED

1. **Initialization:**

   - For every vertex $u \in V$, create linked list containing u with pointers *head*, *next*, *copy1*, *copy2*. For terminals $i \in R$, *head* points to itself, *next* is initialized to null, and the two copy pointers are always set to null. We also initialize the *tail* pointers for each to point to itself.
   - For Steiner vertices $v \in S$, *head* and *next* both point to null. For each v, create two linked lists v_A, v_B which are the two copies of v. The two copy pointers of v point to v_A and v_B respectively. All pointers of the copy nodes are set to null.
   - For every Steiner vertex $v \in S$, create linked list LINK(v) containing two pointers initially set to null. Also create a list LINK-EDGES(v) initially set to null. Initialize variables $\texttt{z}(v)$ to 0, $\texttt{last}(v)$ to 0, $\texttt{rate}(v)$ to 0 and `t(v)` to ∞.
   - Initialize the tree T to be a linked list of edges initialized to an empty list. L be the list of `t(u,v)`'s and `t(v)`'s. Along with the value of the $t()$, we also store the edge or the Steiner vertex responsible for it. L can be stored as a heap so that insertion in sorted can be done in $O(\log |L|)$ time and the list can be read out in a non-decreasing order with $O(1)$ time per read.

2. Read the entries of L in non-decreasing order. If the entry read corresponds to a Steiner vertex (that is `t(v)` for some Steiner vertex v), then merge the components v is linked to along with the Steiner vertex v: For $K_1, K_2 \in$ LINK(v), **Merge**(K_1, K_2, v). Delete the copies v_A, v_B, delete `t(v)` from L, and add the links of v to the tree, that is, append LINK-EDGES(v) to T.

3. If the next term in the list L is the edge (u, v), *then*

   (a) If u and v are in the same component, delete `t(u,v)` from L and proceed to the next entry.

   (b) If $u \in R, v \in R$, merge the two components containing them and add the edge to the tree: **Merge**$(K(u), K(v))$ and append (u, v) to T.

   (c) If $u \in R, v \in S$ (or vice-versa) and $K(u) \notin$ LINK(v), that is, v is not already linked to the component containing u

      i. If $K(v)$ is not null, merge the components containing u and v and add the edge (u, v) to the tree: **Merge**$(K(u), K(v))$ and append (u, v) to T.

      ii. If $|\text{LINK}(v)| = 2$, that is, v is already linked to 2 other components, merge those with the component containing u and add v to the new component, **Merge**$(K(u), \text{LINK}(v), v)$. Add the edge (u, v) and edges in LINK-EDGES(v) to the tree: append (u, v) and LINK-EDGES(v) to T. Also delete the copies of v and delete `t(v)` from L, as in Step 2.

      iii. If $|\text{LINK}(v)| \leq 1$, then do the following: append the edge (u, v) to LINK-EDGES(v); append a copy of v (choose arbitrarily if both copies have their heads pointing to null) to STEINER($K(u)$); append $K(u)$ to LINK(v); recalculate $\texttt{z}(v) = \texttt{z}(v) + \texttt{rate}(v) \cdot (\texttt{t(u,v)} - \texttt{last}(v))$, $\texttt{last}(v) = \texttt{t(u,v)}$ and $\texttt{rate}(v) = \texttt{rate}(v) + 1$. Recalculate `t(v)` to ∞ if (new) $\texttt{rate}(v) = 1$, or $\texttt{t(v)} = (\texttt{t(u,v)} - \texttt{z}(v))$ and place it in the correct order in L.

What remains to be described in the implementation is the **Merge**(K_1, K_2) step. When two components merge, a number of things needs to be taken care of: firstly, the two linked lists corresponding to the components need to be merged. Furthermore, the linked lists STEINER(K_1) and STEINER(K_2) need to be merged *and* care needs to be taken that the duplicate copies of the same Steiner vertex should be removed. Moreover, for precisely these Steiner vertices which were linked to both K_1 and K_2, the variables $\texttt{z}(v), \texttt{last}(v), \texttt{rate}(v)$ and `t(v)` need to be recalculated.

The merge step is very similar to the weighted-union step used in disjoint-set data structures. Since our sets (STEINER(K_1) and STEINER(K_2)) might not be disjoint, we need a little more care. However, recognizing the duplicates only takes constant times more time since the recognition requires going through the vertices of one component, and the weighted union step does precisely that.

We now describe merge step in Algorithm 7.

Algorithm 7 Merge(K_1, K_2)

---

1. Choose the smaller linked list (which can be determined the size variable) of K_1 and K_2, say it is K_1. Append K_1 to K_2 using the *tail* pointer of K_2. Update the *tail* pointer of K_2 to point to the last element of K_1. For every vertex in K_1, make the *head* pointer point to the identifier of K_2

2. Choose the smaller linked list among STEINER(K_1) and STEINER(K_2), say STEINER(K_2) and append STEINER(K_2) to STEINER(K_1) as in the merge of K_1 and K_2. Also make the identifier of K_2 (which now starts a list of the merged K_1 and K_2) point to the identifier of STEINER(K_1) (which now starts a list of the merged STEINER(K_1) and STEINER(K_2)), and vice-versa. This takes care that the LINK(v) operation is consistent with the merges.

   For every vertex v (or rather copy of v) in STEINER(K_2) do the following: check if a copy of it exists in STEINER(K_1). This can be done in $O(1)$ pointer chasing: for a vertex v_A in STEINER(K_2), go to the original Steiner vertex v and figure out the component containing the other copy. If a copy does not exist, move the *head* pointer of v (the copy) to the identifier of STEINER(K_1).

   If a copy exists, and the original Steiner vertex be v, then this means that v is linked to both K_1 and K_2. Remove the copy of v (call it v_A) from STEINER(K_2) and make the *head* pointer of v_A point to `null`. Also remove the costlier edge from LINK-EDGES(v). Furthermore, for v recalculate the variables $\texttt{z}(v), \texttt{last}(v), \texttt{rate}(v)$, `t(v)` as in last step of the implementation above. The only difference being $\texttt{rate}(v)$ decreases by 1 (note that |LINK(v)| also decreases by 1 too).

Proof of Theorem 5.2: It is easy to see that the implementation mimics Algorithm 2 and thus from Theorem 5 the tree T returned is within 3/2 times the optimal tree. We need to argue about the running time.

Note that the sorting of L (storing it as a heap) takes $O((|E| + |V|) \log(|V|))$ time. Suppose that for every entry of L, all the operations of Algorithm 6 took $O(\log |V|)$ time. Then since the size of L is at most $(|E| + |V|)$ we would be done. In fact, it is easy to check that all the steps take $O(\log |V|)$ time except the merge step. We now show that the total time taken in merge steps across the run of the algorithm is $O(|E| + |V| \log |V|)$ and we will be done.

When two components K_1, K_2 are merged, one individual step of merge might take $O(\min(|K_1|, |K_2|) + \min(|\textsc{Steiner}(K_1)|, |\textsc{Steiner}(K_2)|)$ time which is linear. However, since the smaller list is always appended to the larger list, an amortized analysis is possible. Indeed this the most naive implementation of the "union" step in the disjoint step data structures.

Consider the appending of K_1 and K_2. Note that every vertex which changes its *head* pointer ends up in a component at least twice the size of the original component it was in. This is because it was in the smaller component to begin with and the components are disjoint. Thus, since the maximum size of a component in $|V|$, each vertex changes its head pointer at most $O(\log |V|)$ times implying the total number of *head* pointer changes for vertices in components throughout the run of the algorithm is at most $O(|V| \log |V|)$.

However the same accounting does not hold for the appending of $\textsc{Steiner}(K_1)$ and $\textsc{Steiner}(K_2)$. This is because the two sets need not be disjoint and thus the doubling argument does not hold. Nevertheless, we can argue the total number of operations when we append two Steiner lists is also bounded by $O(|E| + |V| \log |V|)$. This is because if the union of the two lists $\textsc{Steiner}(K_1)$ and $\textsc{Steiner}(K_2)$ is not too large (say it is smaller than 1.5 times the size of the smaller list), then the *intersection* will have to be large (larger than 0.5 times the size of the smaller list). Moreover, for every Steiner vertex v in the intersection, we remove one edge from $\textsc{link-edges}(v)$ and this edge never appears again in the algorithm.

Thus, whenever a Steiner vertex changes its *head* pointer, if it moves into a list larger than 1.5 times the list it was in, we charge the movement of the pointer to the vertex. Thus a single Steiner vertex can be charged at most $O(\log |V|)$ times. If the Steiner vertex which changes its *head* pointer doesn't move into a list larger than 1.5 times the list it was in, then we charge the movement of *head* pointers of *all* the Steiner vertices in the smaller list equally to the Steiner vertices in the intersection. Thus by the above observation about intersections being large, every Steiner vertex in the intersection is charged 2. Moreover, vertices in the intersection perform $O(1)$ operations re-evaluating the various variables ($\mathtt{z}(v), \mathtt{last}(v)$,etc), thus the extra 2 can be swept in the $O(1)$. Now, for every Steiner vertex in the intersection, we move the $O(1)$ charge on it, on to the *unique* link-edge incident to it that is removed. The crucial observation is that this link-edge is never brought back on to the algorithm, and thus the charge on it remains $O(1)$.

Therefore, adding up, the total number of operations for merge operations can be charged to vertices and edges, each vertex having charge $O(\log |V|)$ and each edge having charge $O(1)$. The theorem follows. □

C Proof of Theorem 6.5

Theorem C.1 (Theorem 6.5) *There exists a feasible embedding z of G, c such that for T_1 returned by Algorithm* Reduced One-Pass Heuristic, $c(T_1) \leq \sqrt{2} \cdot \gamma(z)$.

Proof: Let T_1 be the tree returned by Algorithm Reduced One-Pass Heuristic with X as the set of Steiner vertices in T_1. Firstly observe that for any vertex $v \in S \setminus X$, $MST(R \cup X \cup v; c_1) \geq MST(R \cup X; c_1)$. This is because, a Steiner vertex which does not help the MST on $R \cup X$ at an earlier iteration *cannot* help the MST at a later iteration; the heaviest edge in the unique path between any two terminals in T_1 keeps on decreasing. Note that we use the fact here that the graph is quasi-bipartite and a Steiner vertex can connect only to terminals.

As in the proof of Lemma 6.2, let $T_1 = E_0 \cup E_1$. Construct the costs c_2 by applying the local filter w.r.t X. Let E_2 be the set of diminished edges and let $T_2 = E_1 \cup E_2$. Once again, one can apply the same proof of Claim 6.3 to show that $T_2 = MST(R, c_2)$. Moreover,

Claim C.2 *For every Steiner vertex $v \in S$, $MST(R \cup v; c_2) \geq MST(R; c_2) = c_2(T_2)$.*

Proof: Suppose for some Steiner vertex v, $MST(R \cup v; c_2) < MST(R; c_2)$. Thus there exist a set of edges A, each of the form (v, j) where j is a terminal; and a set of terminal-terminal edges in $E(T_2)$ so that $c_2(A) = c_1(A) < c_2(B)$; and $T_2 \setminus B \cup A$ is a tree spanning $R \cup v$. Now, among the edges in B, some are diminished and some are in $E(T_1)$; and each diminished edge e corresponds to two edges e_1 and e_2 in $E(T_1)$ one of which, say e_1, has $c_1(e_1) \geq c_2(e)$. Thus from B, we get a subset of edges $B' \subseteq E(T_1)$ with $c_1(B') \geq c_2(B)$. Moreover, note that $T_1 \setminus B' \cup A$ is a valid spanning tree of $R \cup X \cup v$. This implies $MST(R \cup X \cup v; c_1) < MST(R \cup X; c_1)$, which is a contradiction. □

Thus, in the quasi-bipartite graph G with costs c_2, T_2 is an MST spanning the terminals and the addition of no Steiner vertex improves it. So, by Theorem 4.1, we know that running Embed on $(G; c_2)$ will return a feasible dual z with $c_2(T_2) = \gamma(z)$. Since c_2 is only reduced, this embedding is a feasible embedding of (G, c) as well. The proof ends by noting that $c(T_1) \leq \sqrt{2}c_2(T_2)$ which is exactly as in the proof of Lemma 6.2. □