A Constant-Factor Approximation for Quasi-Bipartite Directed Steiner Tree on Minor-Free Graphs

Zachary Friggstad ✉
Department of Computing Science, University of Alberta, Canada

Ramin Mousavi ✉
Department of Computing Science, University of Alberta, Canada

Abstract

We give the first constant-factor approximation algorithm for quasi-bipartite instances of Directed Steiner Tree on graphs that exclude fixed minors. In particular, for K_r-minor-free graphs our approximation guarantee is $O(r \cdot \sqrt{\log r})$ and, further, for planar graphs our approximation guarantee is 20.

Our algorithm uses the primal-dual scheme. We employ a more involved method of determining when to buy an edge while raising dual variables since, as we show, the natural primal-dual scheme fails to raise enough dual value to pay for the purchased solution. As a consequence, we also demonstrate integrality gap upper bounds on the standard cut-based linear programming relaxation for the Directed Steiner Tree instances we consider.

2012 ACM Subject Classification Theory of computation → Routing and network design problems

Keywords and phrases Directed Steiner tree, Combinatorial optimization, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2023.13

Category APPROX

Related Version *Full Version*: https://arxiv.org/abs/2111.02572

1 Introduction

In the Directed Steiner Tree (DST) problem, we are given a directed graph $G = (V, E)$ with edge costs $c(e) \geq 0$ for all $e \in E$, a root node $r \in V$, and a collection of terminals $X \subseteq V \setminus \{r\}$. The nodes in $V \setminus (X \cup \{r\})$ are called *Steiner* nodes. The goal is to find a minimum cost subset $F \subseteq E$ such that there is an $r - t$ path using only edges in F for every terminal $t \in X$. Note any feasible solution that is inclusion-wise minimal must be an arborescence rooted at r. Throughout, we let n denote $|V|$.

One key aspect of DST lies in the fact that it generalizes many other important problems, e.g. Set Cover, (non-metric, multilevel) Facility Location, and Group Steiner Tree. Halperin and Krauthgamer [23] showed Group Steiner Tree cannot be approximated within $O(\log^{2-\epsilon} n)$ for any $\epsilon > 0$ unless NP $\subseteq$ ZTIME $(n^{\text{polylog}\,(n)})$ and therefore the same result holds for DST.

Building on a height-reduction technique of Calinescu and Zelikovsky [5, 36], Charikar et al. give the best approximation for DST which is an $O(|X|^\epsilon)$-approximation for any constant $\epsilon > 0$ [8] and also an $O(\log^3 |X|)$-approximation in $O(n^{\text{polylog}(|X|)})$ time (quasi-polynomial time). More recently, Grandoni, Laekhanukit, and Li [21] obtained a quasi-polynomial time $O(\frac{\log^2 |X|}{\log\log |X|})$-approximation factor for Directed Steiner Tree which is the best possible for quasi-polynomial time algorithms, assuming both the Projection Game Conjecture

Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2023).
Editors: Nicole Megow and Adam D. Smith; Article No. 13; pp. 13:1–13:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

and NP $\not\subseteq \bigcap_{0<\delta<1}$ ZTIME(2^{n^δ}). Ghuge and Nagarajan [18] studied a variant of DST called the Directed Tree Orienteering problem and presented an $O(\frac{\log|X|}{\log\log|X|})$-approximation in quasi-polynomial time which yields the same approximation guarantee as in [21] for DST.

Methods based on linear programming have been less successful. Zosin and Khuller [38] showed the integrality gap of a natural flow-based LP relaxation is $\Omega(\sqrt{|X|})$ but n, the number of vertices, in this example is exponential in terms of $|X|$. More recently, Li and Laekhanukit [27] provided an example showing the integrality gap of this LP is at least polynomial in n. On the positive side, [33] shows for ℓ-layered instances of DST that applying $O(\ell)$ rounds of the Lasserre hierarchy to a slight variant of the natural flow-based LP relaxation yields a relaxation with integrality gap $O(\ell \cdot \log|X|)$. This was extended to the LP-based Sherali-Adams and Lovász-Schrijver hierarchies by [14].

We consider the cut-based relaxation (**Primal-LP**) for DST, which is equivalent to the flow-based relaxation considered in [38, 27]; the flow-based relaxation is an extended formulation of (**Primal-LP**). Let $\delta^{in}(S)$ be the set of directed edges entering a set $S \subseteq V$,

$$\begin{aligned} \text{minimize:} \quad & \sum_{e\in E} c(e)\cdot x_e && \textbf{(Primal-LP)} \\ \text{subject to:} \quad & x(\delta^{in}(S)) \geq 1 \quad \forall S\subseteq V\setminus\{r\},\ S\cap X\neq\emptyset && (1) \\ & x \geq 0 \end{aligned}$$

It is useful to note that if $|X| = 1$ (the shortest $s-t$ path problem) or $X \cup \{r\} = V$ (the minimum cost arborescence problem), the extreme points of (**Primal-LP**) are integral, see [29] and [11] respectively.

The undirected variant of Steiner Tree has seen more activity[1]. A series of papers steadily improved over the simple 2-approximation [37, 25, 30, 32] culminating in a $\ln 4 + \epsilon$ for any constant $\epsilon > 0$ [4]. Bern and Plassmann [2] showed that unless P = NP there is no approximation factor better than $\frac{96}{95}$ for Steiner Tree. However, there is a PTAS for Steiner Tree on planar graphs [3] and more generally [1] obtains a PTAS for Steiner Forest on graphs of bounded-genus.

Another well-studied restriction of Steiner Tree is to quasi-bipartite graphs. These are the instances where no two Steiner nodes are connected by an edge (i.e., $V \setminus (X \cup \{r\})$ is an independent set). Quasi-bipartite instances were first studied by Rajagopalan and Vazirani [31] in order to study the bidirected-cut relaxation of the Steiner Tree problem: this is exactly (**Primal-LP**) where we regard both directions of an undirected edge as separate entities. Feldmann et al. [13] studied Steiner Tree on graphs that do not have an edge-induced claw on Steiner vertices, i.e., no Steiner vertex with three Steiner neighbours, and presented a faster ln(4)-approximation than the algorithm of [4]. Currently, the best approximation in quasi-bipartite instances of Steiner Tree is $\frac{73}{60}$-approximation [19].

A natural question is to study the complexity of DST on these restricted instances. Hibi and Fujito [24] presented an $O(\log|X|)$-approximation algorithm for this case. Assuming P $\neq$ NP, this result asymptotically matches the lower bound $(1 - o(1)) \cdot \ln|X|$ for any $\epsilon > 0$; this lower bound comes from the hardness of Set Cover [12, 10] and the fact that the quasi-bipartite DST problem generalizes the Set Cover problem. Friggstad, Könemann, and Shadravan [15] showed that the integrality gap of (**Primal-LP**) is also $O(\log|X|)$ by a primal-dual algorithm and again this matches the lower bound on the integrality gap of this

[1] One usually does not specify the root node in Steiner Tree, the goal is simply to ensure all terminals are connected.

LP up to a constant. Chan et al. [7] studied the k-connected DST problem on quasi-bipartite instances in which the goal is to find a minimum cost subgraph H such that there are k edge-disjoint paths (in H) from r to each terminal in X. They gave an upper bound of $O(\log|X| \cdot \log k)$ on the integrality gap of the standard cut-based LP (put k instead of 1 in the RHS of the constraints in (**Primal-LP**)) by presenting a polynomial time randomized rounding algorithm.

Very recently, [17] show a combinatorial $O(\log|X|)$-approximation algorithm for DST on planar graphs based on shortest path separators in planar graph by Thorup [35]. It is also worth noting that Demaine, Hajiaghayi, and Klein [9] show that if one takes a standard flow-based relaxation for DST in planar graphs and further constraints the flows to be "non-crossing", then the solution can be rounded to a feasible DST solution while losing only a constant factor in the cost. To date, we do not know how to compute a low-cost, non-crossing flow in polynomial time for DST instances on planar graphs.

It remains an open question whether DST on planar graphs admits a constant factor approximation or even a PTAS, or not. We make some progress on this question. We show quasi-bipartite DST on planar graphs and more generally graphs excluding a fixed minor admit a constant factor approximation. In contrast to the approach in [17], our algorithm is LP-based and bounds the integrality gap of the natural cut-based LP. Our algorithm also works in the more general setting of minor-free graphs, whereas the DST approximation in [17] is specific only to planar graphs.

1.1 Primal-Dual Approximations for Steiner Tree Problems

Consider the Node-Weighted Steiner Tree (NWST) problem which is similar to undirected Steiner Tree except the weight function is on the Steiner vertices instead of edges and can also be viewed as a special case of DST. Guha et al. [22] presented a primal-dual algorithm with guarantee of $O(\ln n)$ which is asymptotically tight since NWST also generalizes set cover. Könemann, Sadeghian, and Sanità [26] give an $O(\log n)$-approximation using the primal-dual framework for a generalization of NWST called Node-Weighted Prize Collecting Steiner Tree[2].

Demaine, Hajiaghayi, and Klein [9] considered a generalization of NWST called Node-Weighted Steiner Forest (NWSF) on planar graphs and using the generic primal-dual framework of Goemans and Williamson [20] they showed a 6-approximation and further they extended their result to minor-free graphs. Later Moldenhauer [28] simplified their analysis and showed an approximation guarantee of 3 for NWSF on planar graphs.

An interesting, non-standard use of the primal-dual scheme is in the work of Chakrabarty, Devanur, and Vazirani [6] for undirected, quasi-bipartite instances of Steiner Tree. They introduce a new "simplex-embedding" LP relaxation and their primal-dual scheme raises dual variables with different rates. It is worth noting that although they also obtain upper bound for the integrality gap of the so-called *bidirected cut relaxation* (BCR) of quasi-bipartite instances of Steiner Tree, the algorithm and the simplex-embedding LP relaxation itself are valid only in the undirected setting.

[2] A key aspect of their algorithm is that it is also *Lagrangian multiplier preserving.*

1.2 Our contributions

We present a primal-dual algorithm for quasi-bipartite DST on minor-free graphs.

Generally, it is difficult to effectively utilize primal-dual algorithms in directed network design problems. This is true in our setting as well: we begin by showing a standard primal-dual algorithm (similar to the primal-dual algorithm for the minimum-cost arborescence problem) does not grow sufficiently-large dual to pay for the set of edges it purchases within any constant factor.

We overcome this difficulty by highlighting different roles for edges in connecting the terminals to the root. For some edges, we maintain two slacks: while raising dual variables these two slacks for an edge may be filled at different rates (depending on the edge's role for the various dual variables being raised) and we purchase the edge when one of its slacks is exhausted. Furthermore, unlike the analysis of standard primal-dual algorithms where the charging scheme is usually more local (i.e., charging the cost of purchased edges to the dual variables that are "close by"), we need to employ a more global charging scheme. Our approach also provides an $O(1)$ upper bound on the integrality gap of the natural cut-based relaxation (**Primal-LP**) for graphs that exclude a fixed minor.

We summarize our results here.

▶ **Theorem 1.** *There is an $O(r \cdot \sqrt{\log r})$-approximation algorithm for DIRECTED STEINER TREE on quasi-bipartite, K_r-minor free graphs. Moreover, the algorithm gives an upper bound of $O(r \cdot \sqrt{\log r})$ on the integrality gap of* (**Primal-LP**) *for DST instances on such graphs.*

▶ Remark 2. The running time of our algorithm is $O(|V|^c)$ where c is a fixed constant that is independent of r. Also, we only require that every (simple) minor of the graph has bounded average degree to establish our approximation guarantee. In particular, if every minor of the input (quasi-bipartite) graph has degree at most d, then the approximation factor will be $O(d)$.

▶ **Theorem 3.** *There is a 20-approximation algorithm for DIRECTED STEINER TREE on quasi-bipartite, planar graphs. Moreover, the algorithm gives an upper bound of 20 on the integrality gap of* (**Primal-LP**) *for DIRECTED STEINER TREE instances on such graphs.*

We also verify that STEINER TREE (and, thus, DIRECTED STEINER TREE) remains NP-hard even when restricted to quasi-bipartite, planar instances. Similar results are known, but we prove this one explicitly since we were not able to find this precise hardness statement in any previous work.

▶ **Theorem 4.** *STEINER TREE instances on bipartite planar graphs where the terminals are on one side and the Steiner nodes are on the other side is* NP*-hard.*

The above hardness result shows DST instances on quasi-bipartite, planar graphs is NP-hard as well.

1.3 Organization of the paper

In Section 2, we state some definition and notation where we use throughout the paper. In Section 3 we present an example that shows the most natural primal-dual algorithm fails to prove our approximation results, this helps the reader understand the key difficulty we need to overcome to make a primal-dual algorithm work and motivates our more refined approach. In Section 4 we present our primal-dual algorithm and in Section 5 we present the analysis.

The analysis contains three main subsections where in each section we present a charging scheme. The first two charging schemes are straightforward but the last one requires some novelty. Finally, we put all these charging schemes together in Subsection 5.4 and prove Theorems 1 & 3. The proof of the hardness result (Theorem 4) is deferred to the full version of the paper [16].

2 Preliminaries

In this paper, graphs are simple directed graphs unless stated otherwise. By simple we mean there are no parallel edges[3]. Note that we can simply keep the cheapest edge in a group of parallel edges if the input graph is not simple; the optimal value for DST problem does not change.

Throughout this paper, we fix a directed graph $G = (V, E)$, a cost function $c : E \to \mathbb{R}_{\geq 0}$, a root r, a set of terminals $X \subseteq V \setminus \{r\}$, and no edge between any two Steiner nodes, as the input to the DST problem. We denote the optimal value for DST instance by OPT.

Given a subgraph G' of G we define $\delta^{in}_{G'}(S) = \{e = (u, v) \in E(G') : \ u \in V \setminus S, \ v \in S\}$ (i.e., the set of edges in G' entering S) we might drop the subscript if the underlying subgraph is G itself. For an edge $e = (u, v)$, we call u the **tail** and v the **head** of e. By a **dipath** we mean a directed path in the graph. By **SCCs** of $F \subseteq E$ we mean the strongly connected components of (V, F) that contains either the root node or at least one terminal node. So for example, if a Steiner node is a singleton strongly connected component of (V, F) then we do not refer to it as an SCC of F. Due to the quasi-bipartite property, these are the only possible strongly connected components in the traditional sense of (V, F) that we will not call SCCs. Observe F is a feasible DST solution if and only if each SCC is reachable from r.

An *arborescence* $T = (V, E)$ rooted at $r \in V$ is a directed tree oriented away from the root such that every vertex in V is reachable from r. By *height* of a vertex u in T we mean the number of edges between r (the root) and u in the dipath from r to u in T. We let T_u denotes the subtree of T rooted at u.

Our discussions, algorithm, and the analysis rely on the concept of *active sets*, so we define them here.

▶ **Definition 5** (Violated set). *Given a DST instance and a subset $F \subseteq E$, we say $S \subseteq V \setminus \{r\}$ where $S \cap X \neq \emptyset$ is a* violated *set with respect to F if $\delta^{in}_F(S) = \emptyset$.*

▶ **Definition 6** (Active set). *Given a DST instance and a subset $F \subseteq E$, we call a minimal violated set (no proper subset of it, is violated) an* active *set (or active moat) with respect to F.*

We use the following definition throughout our analysis and (implicitly) in the algorithm.

▶ **Definition 7** (F-path). *We say a dipath P is a F-path if all the edges of P belong to $F \subseteq E$. We say there is a F-path from a subset of vertices to another if there is a F-path from a vertex of the first set to a vertex of the second set.*

In quasi-biparitite graphs, active moat have a rather "simple" structure, our algorithm will leverage the following properties.

▶ **Lemma 8.** *Consider a subset of edges F and let A be an active set with respect to F. Then, A consists of exactly one SCC C_A of F, and any remaining in $A \setminus C_A$ are Steiner nodes. Furthermore, for every Steiner node in $A \setminus C_A$ there are edges in F that are oriented from the Steiner node to C_A.*

[3] Two edges are parallel if their endpoints are the same and have the same orientation.

Proof. By definition of violated sets, A does not contain r. If A contains only one terminal, then the first statement holds trivially. So consider two terminals t and t' in A. We show there is a F-path from t to t' and vice versa. Suppose not and wlog assume there is no F-path from t' to t. Let $B := \{v \in A : \exists F - path\ from\ v\ to\ t\}$. Note that B is a violated set and $B \subsetneq A \setminus \{t'\}$ which violates the fact that A is a minimal violated set. Therefore, exactly one SCC of F is in A.

Next we prove the second statement. Let s be a Steiner node (if exists) in $A \setminus C_A$. If there is no edge in F oriented from s to C_A, then $A \setminus \{s\}$ is a violated set, because the graph is quasi-bipartite and the fact that A is a violated set itself, contradicting the fact that A is a minimal violated set. ◀

Note that the above lemma limits the interaction between two active moats. More precisely, two active moats can only share Steiner nodes that lie outside of the SCCs in the moats.

▶ **Definition 9** (The SCC part of active moats). *Given a set of edges F and an active set A (with respect to F), we denote by C_A the SCC (with respect to F) inside A.*

We use C_A rather than C_A^F because the set F will always be clear from the context.

Finally we recall bounds on the size of K_r-minor free graphs that we use at the end of our analysis.

▶ **Theorem 10** (Thomason 2001 [34]). *Let $G = (V, E)$ be a K_r-minor free graph with no parallel edges. Then, $|E| \leq O(r \cdot \sqrt{\log r})|V|$ and this bound is asymptotically tight. The constant in the O-notation in the above theorem is at most 3 for large enough r.*

Bipartite planar graphs are K_5-minor free, but we know of explicit bounds sizes. The following is the consequence of Euler's formula that will be useful in our tighter analysis for quasi-bipartite, planar graphs.

▶ **Lemma 11.** *Let $G = (V, E)$ be a bipartite planar graph with no parallel edges. Then, $|E| \leq 2 \cdot |V|$.*

3 Standard primal-dual algorithm and a bad example

Given a DST instance with $G = (V, E)$, $r \in V$ as the root, and $X \subseteq V - \{r\}$ as the terminal set, we define $\mathcal{S} := \{S \subsetneq V : r \notin S,\ and\ S \cap X \neq \emptyset\}$. We consider the dual of (**Primal-LP**).

$$\text{maximize:} \quad \sum_{S \in \mathcal{S}} y_S \qquad \textbf{(Dual-LP)}$$

$$\text{subject to:} \quad \sum_{\substack{S \in \mathcal{S}: \\ e \in \delta^{in}(S)}} y_S \leq c(e) \quad \forall e \in E \qquad (2)$$

$$y \geq 0$$

As we discussed in the introduction, a standard primal-dual algorithm solves arborescence problem on any directed graph [11]. Naturally, our starting point was to investigate this primal-dual algorithm for DST instances. We briefly explain this algorithm here. At the beginning we let $F := \emptyset$. Uniformly increase the dual constraints corresponding to active moats and if a dual constraint goes tight, we add the corresponding edge to F. Update the active sets based on F (see Definition 6) and repeat this procedure. At the end, we do a reverse delete, i.e., we go over the edges in F in the reverse order they have been added to F and remove it if the feasibility is preserved. Unfortunately, for DST instances in quasi-bipartite planar graphs, there is a bad example (see Figure 1), that shows the total growth of the dual variables is $2 + k \cdot \epsilon$ while the optimal solution costs $2 + k + k \cdot \epsilon$ for arbitrarily large k. So the dual objective is not enough to pay for the cost of the edges in F (i.e., we have to multiply the dual objective by $O(k)$ to be able to pay for the edges in F).

What is the issue and how can we fix it? One way to get an $O(1)$-approximation is to ensure at each iteration the number of edges (in the final solution) whose dual constraints are losing slack at this iteration is proportioned to the number of active moats. In the bad example (Figure 1), when the bottom moat is paying toward the downward blue edges, there are only two active moats but there are k downward blue edges that are currently being paid for by the growing dual variables.

To avoid this issue, we consider the following idea: once the bottom active moat grew enough so that the dual constraints corresponding to all the downward blue edges are tight we purchase an arbitrary one of them, say (r, z_k) for our discussion here. Once the top active moat reaches z_1 instead of skipping the payment for this edge (since the dual constraint for (w_2, z_1) is tight), we let the active moat pay towards this edge again by ignoring previous payments to the edge, and then we purchase it once it goes tight. Note that now we violated the dual constraint for (w_2, z_1) by a multiplicative factor of 2. Do the same for all the other downward blue edges (except (r, z_k) that was purchased by the bottom moat). Now it is easy to see that we grew enough dual objective to approximately pay for the edges that we purchased. We make this notion precise by defining different roles for downward blue edges in the next section. In general, each edge can serve up to two roles and has two "buckets" in which it receives payment: each moat pays towards the appropriate bucket depending on the role that edge serves for that moat. An edge is only purchased if one of its buckets is filled and some tiebreaking criteria we mention below is satisfied.

4 Our primal-dual algorithm

As we discussed in the last section, we let the algorithm violate the dual constraint corresponding to an edge by a factor of 2 and hence we work with the following modified Dual-LP:

$$\text{maximize:} \quad \sum_{S \in \mathcal{S}} y_S \qquad \textbf{(Dual-LP-Modified)}$$

$$\text{subject to:} \quad \sum_{\substack{S \in \mathcal{S}: \\ e \in \delta^{in}(S)}} y_S \le 2 \cdot c(e) \quad \forall e \in E \qquad (3)$$

$$y \ge 0$$

Note that the optimal value of (**Dual-LP-Modified**) is at most twice the optimal value of (**Dual-LP**) because consider a feasible solution y for the former LP then $\frac{y}{2}$ is feasible for the latter LP.

Let us define the different buckets for each edge that are required for our algorithm.

Antenna, expansion and killer buckets. We say edge $e = (u, v)$ is an *antenna* edge if $u \notin X \cup \{r\}$ and $v \in X$, in other words, if the tail of e is a Steiner node and the head of e is a terminal. For every antenna edge we associate an antenna bucket with size $c(e)$. For every non-antenna edge e, we associate two buckets, namely *expansion* and *killer* buckets, each of size $c(e)$. The semantics of these labels will be introduced below.

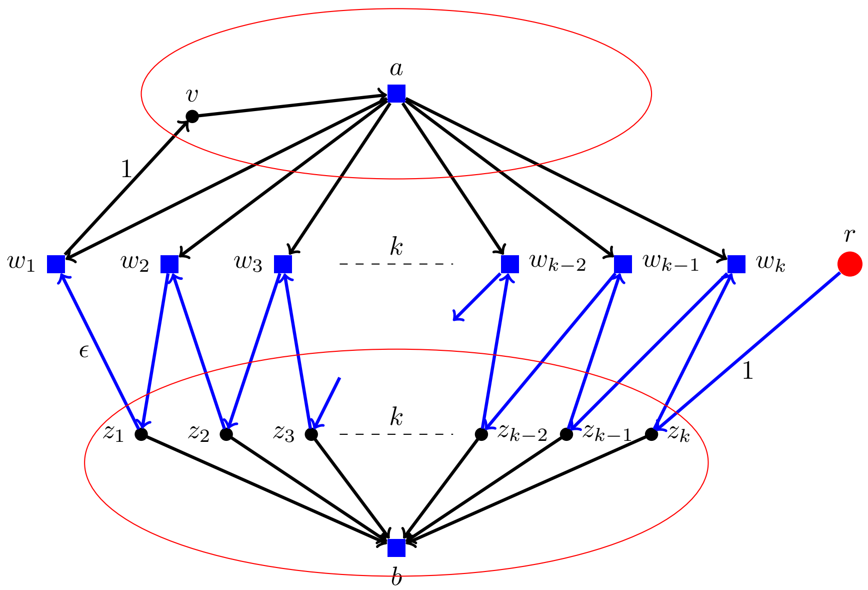

Figure 1 This is an example to show why a standard primal-dual algorithm fails. The square vertices are terminals. The downward blue edges (i.e., (w_i, z_{i-1})'s for $2 \le i \le k$) have cost 1, the upward blue edges (i.e., (z_i, w_i)'s for $1 \le i \le k$) have cost ϵ. The cost of the black edges are 0 except (w_1, v) who has cost 1. Note any feasible solution contains all the blue edges and the cost of an optimal solution is $k + k \cdot \epsilon + 1$. However, it is easy to see the total dual variables that are grown using a standard primal-dual algorithm is $2 + k \cdot \epsilon$.

Now we, informally, describe our algorithm, see Algorithm 1 for the detailed description. Recall the definition of active moats (Definition 6).

Growth phase. At the beginning of the algorithm we set $F := \emptyset$ and every singleton terminal is an active moat. As long as there is an active moat with respect to F do the following: uniformly increase the dual variables corresponding to the active moats. Let $e \notin F$ be an antenna edge with its head in an active moat, then the active moat pays towards the antenna bucket of e. Now suppose $e = (u, v) \notin F$ is a non-antenna edge, so $u \in X \cup \{r\}$. For every active moat A that contains v, if C_A (see Definition 9) is a subset of an active set A' with respect to $F \cup \{e\}$, then A pays toward the expansion bucket of e and otherwise A pays towards the killer bucket of e.

Uniformly increase the dual variables corresponding to active moats until a bucket for an edge e becomes full (antenna bucket in case e is an antenna edge, and expansion or killer bucket if e is a non-antenna edge), add e to F. Update the set of active moats $\mathcal{A}$ according to set F.

Pruning. Finally, we do the standard reverse delete meaning we go over the edges in F in the reverse order they have been added and if the resulting subgraph after removing an edge is still feasible for the DST instance, remove the edge and continue.

The following formalizes the different roles of a non-antenna edge that we discussed above.

▶ **Definition 12** (Relation between non-antenna edges and active moats). *Given a subset of edges $F \subseteq E$, let $\mathcal{A}$ be the set of all active moats with respect to F. Consider a non-antenna edge $e = (u, v)$ (so $u \in X \cup \{r\}$). Suppose $v \in A$ where $A \in \mathcal{A}$. Then,*

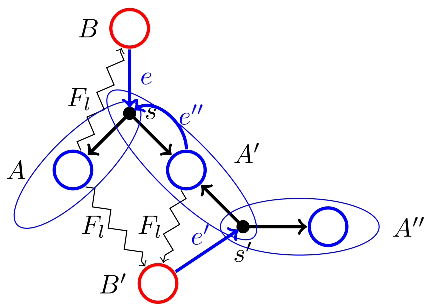

Figure 2 Above is a part of a graph at the beginning of iteration l in the algorithm. F_l denotes the set F at this iteration. The circles are SCCs in (V, F_l). Blue circles are inside some active moats shown with ellipses. The black dots s and s' are Steiner nodes. The black edges and the zigzag paths are in F_l. The edges e, e', and e'' have not been purchased yet (i.e., $e, e', e'' \notin F_l$). Since C_A is a subset of an active moat namely $A \cup B \cup \{s\}$ with respect to $F_l \cup \{e\}$, e is an expansion edge with respect to A. However, e is a killer edge with respect to A' and e'' is a killer edge with respect to A. Finally, e' is a killer edge with respect to A' (and A'') because there is a $F_l \cup \{e'\}$-path from C_A to $C_{A'}$ (and $C_{A''}$), therefore $C_{A'}$ (and $C_{A''}$) cannot be inside an active moat with respect to $F_l \cup \{e'\}$.

- *we say e is an expansion edge with respect to A under F if there is a subset of vertices A' that is active with respect to $F \cup \{e\}$ such that $C_A \subsetneq A'$,*
- *otherwise we say e is a killer edge with respect to A.*

For example, all exiting edges from r that are not in F is a killer edge with respect to any active moat (under F) it enters. See Figure 2 for an illustration of the above definition.

Intuition behind this definition. A non-antenna edge $e = (u, v)$ is a killer edge with respect to an active moat A, if and only if, there is a dipath in $F \cup \{e\}$ from r or $C_{A'}$ to C_A where $A' \neq A$ is an active moat with respect to F. Note that adding e to F will make the dual variable corresponding to A stop growing and that is why we call e a killer edge with respect to A. For example, in Figure 2, both e and e' are killer edges with respect to A'. On the other hand, if $e = (u, v)$ is an expansion edge with respect to A, then C_A will be a part of a "bigger" active moat with respect to $F \cup \{e\}$ and hence the name expansion edge for e. For example, in Figure 2, e is an expansion edge with respect to A because in $F \cup \{e\}$, $A \cup B \cup \{s\}$ is an active moat whose SCC contains C_A.

The complete description of the algorithm is given in Appendix A. Note that the purchased edge e_l at iteration l enters some active moat at iteration l.

After the algorithm finishes, then we label non-antenna edges by expansion/killer as determined by the following rule:

▶ **Definition 13** (Killer and expansion edges)**.** *Consider iteration l of the algorithm where we added a non-antenna edge e_l to F. We label e_l as expansion (killer) if the expansion (killer) bucket of e becomes full at iteration l, break ties arbitrarily.*

Following remark helps to understand the above definition better.

▶ Remark 14. It is possible that one bucket becomes full for an edge yet we do not purchase the edge with that bucket label (killer or expansion) due to tiebreaking when multiple buckets become full. For example, this would happen in our bad example for the downward blue edges: their killer buckets are full yet all but one are purchased as expansion edges.

Let us explain the growth phase of Algorithm 1 on the bad example in Figure 1. Since the early iterations of the algorithm on this example are straightforward, we start our explanation from the iteration where the active moats are $A = \{b, z_1, z_2, ..., z_k\}$ and $A' = \{a, v\}$.

Every (w_i, z_{i-1}) for $2 \leq i \leq k$ is a killer edge with respect to A so A pays toward the killer buckets of these edges. At the same iteration, (w_1, v) is an expansion edge with respect to A' so A' pays toward the expansion bucket of this edge. Now the respected buckets for all mentioned edges are full. Arbitrarily, we pick one of these edges, let us say (w_k, z_{k-1}), and add it to F. Then, A stops growing. In the next iteration, we only have one active moat A'. Since (w_1, v) is still expansion edge with respect to A' and its (expansion) bucket is full, in this iteration we add (w_1, v) to F and after updating the active moats, again we only have one active moat $\{a, v, w_1\}$ which by abuse of notation we denote it by A'. Next iteration we buy the antenna edge (z_1, w_1) and the active moat now is $A' = \{a, v, w_1, z_1\}$. In the next iteration, the crucial observation is that the killer bucket of (w_2, z_1) is full (recall the A payed toward the killer bucket of (w_2, z_1)); however, (w_2, z_1) is an expansion edge with respect to A' so A' will pay towards its expansion bucket and then purchases it. Similarly, the algorithm buys (w_i, z_{i-1})'s except (w_k, z_{k-1}) because this edge is in F already (recall we bought this edge with A). Finally, (r, z_k) is a killer edge with respect to the active moat in the last iteration and we purchase it.

5 The analysis

Because of the space constraints, we defer most of the proofs to the full version of the paper [16] and defer to the full version.

The general framework for analyzing primal-dual algorithms is to use the dual constraints to relate the cost of purchased edges and the dual variables. However, here we do not use the dual constraints and rather we use the buckets we created for each edge. Recall $\overline{F}$ is the solution output by Algorithm 1. We define $\overline{F}_{\text{Killer}}$ to be the set of edges in $\overline{F}$ that was purchased as killer edge (recall definition 13). Similarly define $\overline{F}_{\text{Exp}}$ and $\overline{F}_{\text{Ant}}$. For each iteration l, we denote by F_l the set F at this iteration, $\mathcal{A}_l$ denotes the set of active moats with respect to F_l, and ϵ_l is the amount we increased the dual variables (corresponding to active moats) with at iteration l. Finally, Let y^* be the dual solution for (**Dual-LP-Modified**) constructed in the course of the algorithm. We use the following notation throughout the analysis.

▶ **Definition 15.** *Fix an iteration l. For any $A \in \mathcal{A}_l$, let*

$$\Delta^l_{\text{Killer}}(A) := \{e \in \overline{F}_{\text{Killer}} :\ e \text{ is killer with respect to } A \text{ under } F_l\},$$

in other words, $\Delta^l_{\text{Killer}}(A)$ is the set of all killer edges in $\overline{F}$ such that they are killer edge with respect to A at iteration l. Similarly define $\Delta^l_{\text{Exp}}(A)$.

Let $\Delta^l_{\text{Ant}}(A) := \{e \in \overline{F}_{\text{Ant}} :\ e \in \delta^{in}(A)\}$. Finally, we define

$$\Delta^l(A) := \Delta^l_{\text{Killer}}(A) \cup \Delta^l_{\text{Exp}}(A) \cup \Delta^l_{\text{Ant}}(A).$$

Note $\Delta^l_{\text{Killer}}(A)$, $\Delta^l_{\text{Exp}}(A)$, and $\Delta^l_{\text{Ant}}(A)$ are pairwise disjoint for any $A \in \mathcal{A}_l$.

Suppose we want to show that the performance guarantee of Algorithm 1 is $2 \cdot \alpha$ for some $\alpha \geq 1$, it suffices to show the following: for any iteration l we have

$$\sum_{S \in \mathcal{A}_l} |\Delta^l(S)| \leq \alpha \cdot |\mathcal{A}_l|. \tag{4}$$

Once we have (4), then the $(2 \cdot \alpha)$-approximation follows easily:

$$\sum_{e \in \overline{F}} c(e) = \sum_{e \in \overline{F}_{\text{Killer}}} \sum_{l} \sum_{\substack{S \in \mathcal{A}_l: \\ e \in \Delta^l_{\text{Killer}}(S)}} \epsilon_l + \sum_{e \in \overline{F}_{\text{Exp}}} \sum_{l} \sum_{\substack{S \in \mathcal{A}_l: \\ e \in \Delta^l_{\text{Exp}}(S)}} \epsilon_l + \sum_{e \in \overline{F}_{\text{Ant}}} \sum_{l} \sum_{\substack{S \in \mathcal{A}_l: \\ e \in \Delta^l_{\text{Ant}}(S)}} \epsilon_l \tag{5}$$

$$= \sum_{l} \epsilon_l \cdot \sum_{S \in \mathcal{A}_l} |\Delta^l(S)| \tag{6}$$

$$\leq \alpha \cdot \sum_{l} |\mathcal{A}_l| \epsilon_l \tag{7}$$

$$= \alpha \cdot \sum_{S \subseteq V \setminus \{r\}} y^*_S \tag{8}$$

$$\leq \alpha \cdot \big(2 \cdot \text{OPT}(\textbf{Dual} - \textbf{LP})\big) \tag{9}$$

$$= 2 \cdot \alpha \cdot \text{OPT}(\textbf{Primal} - \textbf{LP}) \tag{10}$$

$$\leq 2 \cdot \alpha \cdot \text{OPT}, \tag{11}$$

where the first equality follows from the algorithm, the second equality is just an algebraic manipulation, (7) follows from (4). Equality (8) follows from the fact we uniformly increased the dual variables corresponding to active moats by ϵ_l at iteration l, (9) follows from feasibility of $\frac{y^*}{2}$ for (**Dual-LP**), and (10) follows from strong duality theorem for linear programming.

It remains to show (4) holds. Consider iteration l. Using the bound on the total degree of nodes in G (using minor-free properties) to show (4), it suffices to bound the number of edges in $\bar{F}_{\text{Ant}} \cup \bar{F}_{\text{Killer}} \cup \bar{F}_{\text{Exp}}$ that are being paid by some active moat at iteration l, by $O(|\mathcal{A}_l|)$. We provide charging schemes for each type of edges, separately. Since G is quasi-bipartite, it is easy to show that for each active moat $A \in \mathcal{A}_l$, there is at most one antenna edge in $\bar{F}$ that enters A, this is proved in Section 5.1. The charging scheme for killer edges is also simple as one can charge a killer edge to an active moat that it kills; this will be formalized in Section 5.2. However, the charging scheme for expansion edges requires more care and novelty. The difficulty comes from the case that an expansion edge is not pruned because it would disconnect some terminals that are not part of any active moat that e is entering this iteration.

Our charging scheme for expansion edges is more global. In a two-stage process, we construct an auxiliary tree that encodes some information about which nodes can be reached from SCCs using edges in F_l (which is the information we used in the definition of expansion edge). Then using a token argument, we leverage properties of our construction to show the number of expansion edges is at most twice the number of active moats in any iteration. These details are presented in 5.3. Finally, in Section 5.4 we put all the bounds we obtained together and derive our approximation factors.

5.1 Counting the number of antenna edges in an iteration

Fix an iteration l. Recall F_l denotes the set F at iteration l, and $\mathcal{A}_l$ denotes the set of active moats with respect to F_l. It is easy to bound the number of antenna edges in $\overline{F}$ against $|\mathcal{A}_l|$. We do this in the next lemma.

▶ **Lemma 16.** *At the beginning of each iteration l, we have* $\sum_{A \in \mathcal{A}_l} |\Delta^l_{\text{Ant}}(A)| \leq |\mathcal{A}_l|$.

5.2 Counting the number of killer edges in an iteration

We introduce a notion called *alive* terminal which helps us to bound the number of killer edges at a fixed iteration against the number of active moats in that iteration. Also this notion explains the name killer edge. Throughout the algorithm, we show every active moat contains exactly one alive terminal and every alive terminal is in an active moat.

We consider how terminals can be "killed" in the algorithm by associating active moats with terminals that have not yet been part of a moat that was killed. At the beginning of the algorithm, we mark every terminal alive, note that every singleton terminal set is initially an active moat as well. Let $e_l = (u, v)$ be the edge that was added to F_l at iteration l. If $e_l = (u, v)$ is a non-antenna edge, then for every active set A such that e_l is a killer edge with respect to A under F_l, mark the alive terminal in A as *dead*[4]. If $e_l = (u, v)$ is an antenna edge, then for every active moat A such that $e_l \in \delta^{in}(A)$ and C_A is **not** in any active moat with respect to $F_l \cup \{e_l\}$, then mark the alive terminal in A as dead[5].

The important observation here is that by definition, if e_l is a killer edge, then there must be an active set that satisfies the above condition, hence there is at least one alive terminal that will be marked dead because of e_l. In the case that e_l is bought as killer edge, arbitrarily pick an alive terminal t_{e_l} that dies because of e_l and assign e_l to t_{e_l}. Note that t_{e_l} was alive until e_l was added to F_l.

▶ **Definition 17.** *Fix an iteration l. We define*

$$\overline{F}^l_{\text{Killer}} := \bigcup_{A \in \mathcal{A}_l} \Delta^l_{\text{Killer}}(A),$$

in other words, $\overline{F}^l_{\text{Killer}}$ is the set of all killer edges in $\overline{F}$ such that some active moat(s) is paying toward their killer bucket at iteration l.

Now we can state the main lemma of this section.

▶ **Lemma 18.** *At the beginning of each iteration l, we have $|\overline{F}^l_{\text{Killer}}| \leq |\mathcal{A}_l|$.*

Note that the above lemma does not readily bound $\sum_{A \in \mathcal{A}_l} |\Delta^l_{\text{Killer}}(A)|$ against $|\mathcal{A}_l|$ which is required to prove inequality (4). We need the properties of minor-free graphs to do so. In the next section we prove a similar result for expansion edges and then using the properties of the underlying graph, we demonstrate our approximation guarantee.

5.3 Counting the number of expansion edges in an iteration

The high level idea to bound the number of expansion edges is to look at the graph $\overline{F} \cup F_l$ and contract all SCCs[6] of (V, F_l). Then, we construct an auxiliary tree that highlights the role of expansion edges to the connectivity of active moats. Finally, using this tree we provide our charging scheme and show the number of edges in $\overline{F}_{\text{Exp}}$ that are being paid by some active moats at iteration l is at most twice the number of active moats.

[4] It is possible, e_l is bought as an expansion edge but kills some alive terminals. For example, in Figure 2 suppose e is being added to F_l at iteration l as an expansion edge (note that A pays toward the expansion bucket of e). Then, we mark the alive terminal in A' as dead because e is a killer edge with respect to A' under F_l.

[5] For example, suppose the antenna edge $e_l = (u, v) \in \delta^{in}(A)$ is being added to F_l and u is in $C_{A'}$ for some active moat A'. Then, after adding e_l to F_l, we mark the alive terminal in A as dead.

[6] Recall that we do NOT call a Steiner node that is a singleton strongly connected component of (V, F_l) an SCC. So every SCC in (V, F_l) is either $\{r\}$ or contains at least one terminal node.

We fix an iteration l for this section. First let us recall some notation and definition that we use extensively in this section. (i) $\overline{F}$ is the output solution of the algorithm. (ii) $F_l \subseteq E$ is the set of purchased edges in the growing phase up to the beginning of iteration l (i.e., set F in the algorithm at iteration l). (iii) $\mathcal{A}_l$ is the set of active moats with respect to F_l (see Definition 6). Recall each $A \in \mathcal{A}_l$ is consists of an SCC (with respect to edges in F_l) and bunch of Steiner nodes. Denote by C_A the SCC part of A.

We define an analogous of Definition 17 for expansion edges.

▶ **Definition 19.** *Fix an iteration l. Then, we define*

$$\overline{F}^l_{\text{Exp}} := \bigcup_{A \in \mathcal{A}_l} \Delta^l_{\text{Exp}}(A),$$

in other words, $\overline{F}^l_{\text{Exp}}$ is the set of all expansion edges in $\overline{F} \setminus F_l$ such that some active moat(s) is paying toward their expansion bucket at iteration l.

This section is devoted to prove the following inequality.

▶ **Lemma 20.** *At the beginning of each iteration l of the algorithm, we have $|\overline{F}^l_{\text{Exp}}| \leq 2 \cdot |\mathcal{A}_l|$.*

Sketch of the proof. We start by giving a sketch of the proof of Lemma 20. Consider the subgraph $F_l \cup \overline{F}$ of G. Contract every SCC of (V, F_l) and denote the resulting subgraph by H (keeping all copies of parallel edges that may result). For every non-root, non-Steiner node $v \in V(H)$, we call v active if it is a contraction of an SCC that is a subset of an active moat in $\mathcal{A}_l$, otherwise we call v inactive. Note that r is a singleton SCC in (V, F_l) and therefore $r \in V(H)$. We call an edge in $E(H)$ an expansion edge, if its corresponding edge is in $\overline{F}^l_{\text{Exp}}$. Note that every non root vertex in $V(H)$ is either labeled active/inactive, or it is a Steiner node. Lemma 20 follows if we show the number of expansion edges in H is at most twice the number of active vertices in H. As we stated at the beginning of this section, we use an arborescence that highlights the role of expansion edges to the connectivity of active vertices in H. A bit more formally, we show if every expansion edge is "good" with respect to the arborescence, which is formalized below, then every expansion edge is "close" to an active vertex in H and we use this in our charging scheme.

Given an arborescence T, define $\text{Elevel}_T(v)$ to be the expansion level of v with respect to T, i.e., the number of expansion edges on the dipath from r to v in T.

▶ **Definition 21.** *Given an arborescence T and an expansion edge $e = (u, v)$, we say e is a* good *expansion edge with respect to T if one of the following cases happens:*

- *Type 1: If u has an active ancestor w such that* $\text{Elevel}_T(w) = \text{Elevel}_T(u)$.
- *Type 2: If e is not of type 1 and the subtree rooted at u has an active vertex w such that* $\text{Elevel}_T(w) \leq \text{Elevel}_T(u) + 1$.

Every expansion edge that is not of type 1 or type 2, is called a bad *expansion edge with respect to T.*

A starting point for an arborescence that every expansions edge is good, is a shortest path arborescence rooted at r in H where each expansion edge has cost 1 and the rest of the edges have cost 0. However, as Figure 3 shows, there could be some *bad* expansion edges in this arborescence. For example, e is a *bad* expansion edge with respect to the arborescence in Figure 3 (b). Since B_2, the tail of e, is an inactive vertex, there must be an active vertex, namely A_3, that has a dipath from A_3 to B_2 in F_ℓ. Then, we can "cut" the subtree rooted at B_2 and "paste" it under A_3 as shown in Figure 3(c). It is easy to verify that now every expansion edge is good with respect to the arborescence in Figure 3(c). We formalize this

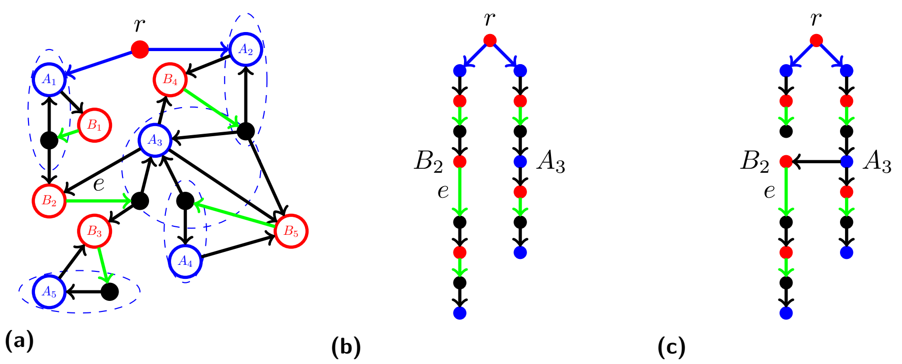

Figure 3 (a) shows subgraph $F_l \cup \bar{F}$ of G, in particular, the SCCs of (V, F_l) are shown with circles but the nodes inside SCCs are not shown for simplicity. The blue SCCs are inside some active moats shown with dashed ellipses. Contracting all the SCCs result in the graph H discussed before. Black edges are in F_l, blue edges are in $\bar{F} \setminus F_l$, and green edges are in $\bar{F}^l_{\text{Exp}}$. In (b), we have a shortest path arborescence rooted at r in H where the cost of edges is one if it is green and zero otherwise. Recall the definition of H at the beginning of this sketched proof. Note that e is a bad expansion edge with respect to this arborescence. In (c), we show how to construct an arborescence using cut-and-paste procedure so that every expansion edge is a good expansion edge in the resulting arborescence.

"cut and paste" procedures in Algorithm 2 in the full version of the paper and prove the output of the algorithm is an arborescence with the property that every expansion edge is good. Given an arborescence that every expansion edge is good, we show there is a rather natural charging scheme that proves Lemma 20.

Charging scheme. At the beginning we label every token **unused**. We process all the vertices with height l. For each expansion edge whose tail has height l we assign an unused token to it and change the label of the assigned token to **used**. Then we move to height $l-1$ and repeat the process. Fix height l. We do the following for every vertex u with this height: if there is no expansion edge whose tail is u then mark u as processed. Otherwise let $(u, v_1), ..., (u, v_k)$ be all the expansion edges whose tail is u. Note that by definition of type 1 and 2, either (i) all (u, v_i)'s are type 1 or (ii) all are type 2. Base on these two cases we do the following:

(i) Let $(u, v_1), ..., (u, v_k)$ be the expansion edges of type 1. For each $1 \le i \le k$ there is at least one unused token in $T^*_{v_i}$. Pick one such unused token and assign it to (u, v_i) and change its label to used. Mark u as processed.

(ii) Let $(u, v_1), ..., (u, v_k)$ be the expansion edges of type 2. For each $1 \le i \le k$ there is at least one unused token in $T^*_{v_i}$. Pick one such unused token and assign it to (u, v_i) and change its label to used. Furthermore, after this there is at least one more unused token in T^*_u. Mark u as processed.

The proof of correctness of this charging scheme is based on induction on the height l. ◄

5.4 Putting everything together

Fix an iteration l. We use Lemmas 18 & 20 and the properties of graph G to bound $\sum_{A \in \mathcal{A}_l} |\Delta^l_{\text{Killer}}(A) \cup \Delta^l_{\text{Exp}}(A)|$. Consider an active moat A and its SCC C_A. We show there is at most one killer/expansion edge that enters C_A. So the remaining killer/expansion edges must enter some Steiner node in $A \setminus C_A$. We use this fact later.

▷ Claim 22. Fix an iteration l and an active moat $A \in \mathcal{A}_l$. There is at most one edge in $\Delta^l_{\text{Killer}}(A) \cup \Delta^l_{\text{Exp}}(A)$ whose head is in C_A.

Consider the graph $F_l \cup \overline{F}$. Remove all vertices that are not in an active moat at this iteration. For each active moat A, remove all Steiner nodes in $A \setminus C_A$ that are not the head of any edge in $\overline{F}^l_{\text{Killer}} \cup \overline{F}^l_{\text{Exp}}$. Then, for each $A \in \mathcal{A}_l$ contract C_A to a single vertex and call the contracted vertex by C_A. Finally, if there are parallel edges, arbitrarily keep one of them and remove the rest[7]. Call the resulting graph G'.

Now we relate the sum we are interested in to bound with the sum of the indegree of vertices in G'.

▷ Claim 23. For each active moat $A \in \mathcal{A}_l$, we have

$$|\Delta^l_{\text{Killer}}(A) \cup \Delta^l_{\text{Exp}}(A)| \le |\delta^{in}_{G'}(C_A)| + 1. \tag{12}$$

Next, using Lemmas 18 & 20 we bound the number of vertices in G'.

▷ Claim 24. Fix an iteration l. Then, $|V(G')| \le 4 \cdot |\mathcal{A}_l|$.

Finally, we prove Theorems 1 & 3.

Proof of Theorem 1. Since G is K_r-minor free so does G'. So we can write

$$\begin{aligned}
\sum_{A\in\mathcal{A}_l} \left|\Delta^l_{\text{Killer}}(A) \cup \Delta^l_{\text{Exp}}(A)\right| &\le \sum_{A\in\mathcal{A}_l} \left(|\delta^{in}_{G'}(C_A)| + 1\right) \\
&= |E(G')| + |\mathcal{A}_l| \\
&\le O(r \cdot \sqrt{\log r}) \cdot 4 \cdot |\mathcal{A}_l| + |\mathcal{A}_l| \\
&= O(r \cdot \sqrt{\log r})|\mathcal{A}_l|,
\end{aligned} \tag{13}$$

where the inequality follows from Claim 23 and the second inequality follows from Claim 24 together with Theorem 10.

Next we show (4) holds for $\alpha = O(r \cdot \sqrt{\log r})$.

$$\begin{aligned}
\sum_{A\in\mathcal{A}_l} |\Delta^l(A)| &= \sum_{A\in\mathcal{A}_l} |\Delta^l_{\text{Killer}}(A) \cup \Delta^l_{\text{Exp}}(A)| + \sum_{A\in\mathcal{A}_l} |\Delta^l_{\text{Ant}}(A)| \\
&\le O(r \cdot \sqrt{\log r})|\mathcal{A}_l| + |\mathcal{A}_l| \\
&= O(r \cdot \sqrt{\log r})|\mathcal{A}_l|,
\end{aligned}$$

where inequality follows from inequality (13) and Lemma 16.

As we discussed at the beginning of Section 5 that if (4) holds for α then we have a $(2 \cdot \alpha)$-approximation algorithm. Hence, Algorithm 1 is an $O(r \cdot \sqrt{\log r})$-approximation for DST on quasi-bipartite, K_r-minor free graphs. ◀

Proof of Theorem 3. The proof is exactly the same as proof of Theorem 1 except instead of $O(r \cdot \sqrt{\log r})$ in (13) we have 2 because G' is a bipartite planar graph, see Lemma 11. Now we can write $\sum_{A\in\mathcal{A}_l} \left|\Delta^l_{\text{Killer}}(A) \cup \Delta^l_{\text{Exp}}(A)\right| \le 9 \cdot |\mathcal{A}_l|$ and $\sum_{A\in\mathcal{A}_l} |\Delta^l(A)| \le 10 \cdot |\mathcal{A}_l|$. Therefore, (4) holds for $\alpha = 10$ and hence we have a 20-approximation algorithm, as desired. ◀

[7] Note that all the parallel edges are antenna edges and so removing them does not affect the quantity $\sum_{A\in\mathcal{A}_l} |\Delta^l_{\text{Killer}}(A) \cup \Delta^l_{\text{Exp}}(A)|$ we are trying to bound.

References

1 MohammadHossein Bateni, MohammadTaghi Hajiaghayi, and Dániel Marx. Approximation schemes for steiner forest on planar graphs and graphs of bounded treewidth. *Journal of the ACM (JACM)*, 58(5):1–37, 2011.

2 Marshall Bern and Paul Plassmann. The steiner problem with edge lengths 1 and 2. *Information Processing Letters*, 32(4):171–176, 1989.

3 Glencora Borradaile, Philip Klein, and Claire Mathieu. An o (n log n) approximation scheme for steiner tree in planar graphs. *ACM Transactions on Algorithms (TALG)*, 5(3):1–31, 2009.

4 Jarosław Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. Steiner tree approximation via iterative randomized rounding. *Journal of the ACM (JACM)*, 60(1):1–33, 2013.

5 Gruia Calinescu and Alexander Zelikovsky. The polymatroid steiner problems. *J. Combonatorial Optimization*, 33(3):281–294, 2005.

6 Deeparnab Chakrabarty, Nikhil R Devanur, and Vijay V Vazirani. New geometry-inspired relaxations and algorithms for the metric steiner tree problem. *Mathematical programming*, 130(1):1–32, 2011.

7 Chun-Hsiang Chan, Bundit Laekhanukit, Hao-Ting Wei, and Yuhao Zhang. Polylogarithmic approximation algorithm for k-connected directed steiner tree on quasi-bipartite graphs. *arXiv preprint*, 2019. `arXiv:1911.09150`.

8 Moses Charikar, Chandra Chekuri, To-Yat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, and Ming Li. Approximation algorithms for directed steiner problems. *Journal of Algorithms*, 33(1):73–91, 1999.

9 Erik D Demaine, MohammadTaghi Hajiaghayi, and Philip N Klein. Node-weighted steiner tree and group steiner tree in planar graphs. *ACM Transactions on Algorithms (TALG)*, 10(3):1–20, 2014.

10 Irit Dinur and David Steurer. Analytical approach to parallel repetition. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 624–633, 2014.

11 Jack Edmonds. Optimum branchings. *Journal of Research of the national Bureau of Standards B*, 71(4):233–240, 1967.

12 Uriel Feige. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.

13 Andreas Emil Feldmann, Jochen Könemann, Neil Olver, and Laura Sanità. On the equivalence of the bidirected and hypergraphic relaxations for steiner tree. *Mathematical programming*, 160(1):379–406, 2016.

14 Zachary Friggstad, Jochen Könemann, Young Kun-Ko, Anand Louis, Mohammad Shadravan, and Madhur Tulsiani. Linear programming hierarchies suffice for directed steiner tree. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 285–296. Springer, 2014.

15 Zachary Friggstad, Jochen Könemann, and Mohammad Shadravan. A Logarithmic Integrality Gap Bound for Directed Steiner Tree in Quasi-bipartite Graphs . In Rasmus Pagh, editor, *15th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2016)*, volume 53 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 3:1–3:11, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

16 Zachary Friggstad and Ramin Mousavi. A constant-factor approximation for quasi-bipartite directed steiner tree on minor-free graphs. *arXiv preprint*, 2021. `arXiv:2111.02572`.

17 Zachary Friggstad and Ramin Mousavi. An O(log k)-Approximation for Directed Steiner Tree in Planar Graphs. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*, volume 261 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 63:1–63:14, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

18 Rohan Ghuge and Viswanath Nagarajan. Quasi-polynomial algorithms for submodular tree orienteering and other directed network design problems. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1039–1048. SIAM, 2020.

19 Michel X Goemans, Neil Olver, Thomas Rothvoß, and Rico Zenklusen. Matroids and integrality gaps for hypergraphic steiner tree relaxations. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 1161–1176, 2012.

20 Michel X Goemans and David P Williamson. The primal-dual method for approximation algorithms and its application to network design problems. *Approximation algorithms for NP-hard problems*, pages 144–191, 1997.

21 Fabrizio Grandoni, Bundit Laekhanukit, and Shi Li. O (log2 k/log log k)-approximation algorithm for directed steiner tree: a tight quasi-polynomial-time algorithm. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 253–264, 2019.

22 Sudipto Guha, Anna Moss, Joseph Naor, and Baruch Schieber. Efficient recovery from power outage. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 574–582, 1999.

23 Eran Halperin and Robert Krauthgamer. Polylogarithmic inapproximability. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 585–594, 2003.

24 Tomoya Hibi and Toshihiro Fujito. Multi-rooted greedy approximation of directed steiner trees with applications. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 215–224. Springer, 2012.

25 Marek Karpinski and Alexander Zelikovsky. New approximation algorithms for the steiner tree problems. *Journal of Combinatorial Optimization*, 1(1):47–65, 1997.

26 Jochen Könemann, Sina Sadeghian, and Laura Sanita. An lmp o (log n)-approximation algorithm for node weighted prize collecting steiner tree. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 568–577. IEEE, 2013.

27 Shi Li and Bundit Laekhanukit. Polynomial integrality gap of flow lp for directed steiner tree. *arXiv preprint*, 2021. `arXiv:2110.13350`.

28 Carsten Moldenhauer. Primal-dual approximation algorithms for node-weighted steiner forest on planar graphs. *Information and Computation*, 222:293–306, 2013.

29 Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity.* Courier Corporation, 1998.

30 Hans Jürgen Prömel and Angelika Steger. A new approximation algorithm for the steiner tree problem with performance ratio 5/3. *Journal of Algorithms*, 36(1):89–101, 2000.

31 Sridhar Rajagopalan and Vijay V Vazirani. On the bidirected cut relaxation for the metric steiner tree problem. In *SODA*, volume 99, pages 742–751, 1999.

32 Gabriel Robins and Alexander Zelikovsky. Tighter bounds for graph steiner tree approximation. *SIAM Journal on Discrete Mathematics*, 19(1):122–134, 2005.

33 Thomas Rothvoß. Directed steiner tree and the lasserre hierarchy. *arXiv preprint*, 2011. `arXiv:1111.5473`.

34 Andrew Thomason. The extremal function for complete minors. *Journal of Combinatorial Theory, Series B*, 81(2):318–338, 2001.

35 Mikkel Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *Journal of the ACM (JACM)*, 51(6):993–1024, 2004.

36 Alexander Zelikovsky. A series of approximation algorithms for the acyclic directed steiner tree problem. *Algorithmica*, 18(1):99–110, 1997.

37 Alexander Z Zelikovsky. An 11/6-approximation algorithm for the network steiner problem. *Algorithmica*, 9(5):463–470, 1993.

38 Leonid Zosin and Samir Khuller. On directed steiner trees. In *SODA*, volume 2, pages 59–63. Citeseer, 2002.

A Full description of the primal-dual algorithm

Algorithm 1 Primal-Dual Algorithm for DST on Quasi-Bipartite Graphs.

Input: Directed quasi-bipartite graph $G = (V, E)$ with edge costs $c(e) \geq 0$ for $e \in E$, a set of terminal $X \subseteq V \setminus \emptyset$, and a root vertex r.
Output: An arborescence $\overline{F}$ rooted at r such that each terminal is reachable from r in $\overline{F}$.

$\mathcal{A} \leftarrow \{\{v\} : v \in X\}$. {The active moats each iteration, initially all singleton terminal set.}
$y^* \leftarrow 0$. {The dual solution}
$F \leftarrow \emptyset$. {The edges purchased}
$l \leftarrow 0$. {The iteration counter}
$b_e^{\text{Ant}} \leftarrow 0$, $b_e^{\text{Exp}} \leftarrow 0$ and $b_e^{\text{Killer}} \leftarrow 0$. {The buckets}
Growing phase:
while until $\mathcal{A} \neq \emptyset$ **do**
  Find the maximum value $\epsilon \geq 0$ such that the following holds:
    (a) for every antenna edge e we have $b_e^{\text{Ant}} + \sum_{\substack{A \in \mathcal{A}: \\ e \in \delta^{in}(A)}} \epsilon \leq c(e)$.
    (b) for every non-antenna edge e we have $b_e^{\text{Exp}} + \sum_{\substack{A \in \mathcal{A}: \\ e \textit{ is expansion} \\ \textit{with resp. to } A}} \epsilon \leq c(e)$.
    (c) for every non-antenna edge e we have $b_e^{\text{Killer}} + \sum_{\substack{A \in \mathcal{A}: \\ e \textit{ is killer with} \\ \textit{resp. to } A}} \epsilon \leq c(e)$.
  Increase the dual variables y^* corresponding to each active moat by ϵ.
  for every antenna edge e **do**
    $b_e^{\text{Ant}} \leftarrow b_e^{\text{Ant}} + \sum_{\substack{A \in \mathcal{A}: \\ e \in \delta^{in}(A)}} \epsilon.$
  end for
  for every non-antenna edge e **do**
    $b_e^{\text{Exp}} \leftarrow b_e^{\text{Exp}} + \sum_{\substack{A \in \mathcal{A}: \\ e \textit{ is expansion} \\ \textit{with resp. to } A}} \epsilon.$
    $b_e^{\text{Killer}} \leftarrow b_e^{\text{Killer}} + \sum_{\substack{A \in \mathcal{A}: \\ e \textit{ is killer with} \\ \textit{resp. to } A}} \epsilon.$
  end for
  pick any single edge $e_l \in \cup_{A \in \mathcal{A}} \delta^{in}(A)$ with one of (a)-(c) being tight (break ties arbitrarily).
  $F \leftarrow F \cup \{e_l\}$.
  update $\mathcal{A}$ based on the minimal violated sets with respect to F.
  $l \leftarrow l + 1$.
end while
Deletion phase:
$\overline{F} \leftarrow F$.
for i from l to 0 **do**
  if $\overline{F} \setminus \{e_i\}$ is a feasible solution for the DST instance **then**
    $\overline{F} \leftarrow \overline{F} \setminus \{e_i\}$.
  end if
end for
return $\overline{F}$