Approximation Algorithms for Directed Steiner Tree and Traveling Salesman Problems

by

Seyyed Ramin Mousavi Haji

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing Science

University of Alberta

Abstract

In this thesis, we design approximation algorithms for a variety of problems in Network Design. The first problem we consider is the DIRECTED STEINER TREE (DST) problem where we want to find a cheapest way of connecting a subset of nodes (terminal nodes) from a root node in a directed network. We give the first logarithmic approximation algorithm for DST on planar graphs.

Another restriction of DST we consider is to quasi-bipartite instances on planar graphs. In these instances the underlying graph is planar and no two non-terminal nodes are adjacent. Here we get the first constant factor approximation. We further extend this result to a more general family of graphs called minor-free graphs. Our approach is based on a non-standard primal-dual framework and also bounds the integrality gap of the classical linear programming (LP) relaxation for DST.

The third problem we study is a variant of the TRAVELING SALESMAN problem (TSP), one the most famous problem in combinatorial optimization. In TSP, we are given a set of cities, the task is to find a minimum cost tour (closed walk) visiting all the cities. We initiate the study of generalization of TSP, where we are interested in tours that respect some given degree bounds, i.e., a feasible tour must not pass through a location more than a given bound. We further generalize this problem to the setting where we only need to visit a subset of locations in the network. It is easy to see unless $P = NP$, we cannot hope for an algorithm that does not violate the degree bounds at all.

On the other hand, the problems we consider are a generalization of TSP, therefore approximating the cost factor better than the approximation factor for TSP is a very challenging problem. In this thesis, we study the trade-offs between the cost of the tour and the degree violation of the nodes in the tour. We develop LP-based rounding algorithms for these TSP variants which in turn bound the integrality gap of a natural LP relaxations for these problems.

Preface

In this thesis, three results based on three different publications are presented. All three results are joint work with my supervisor Zachary Friggstad.

Acknowledgements

First and foremost, I am extremely grateful to my advisor Zachary Friggstad. I am deeply indebted to Zac for teaching me how to do research and his invaluable advice on various matters, be it writing papers, preparing slides, and giving talks. And finally, thank you for being extremely patient with me in all these years.

I would also like to thank two of my wonderful collaborators Mohammad R. Salavatipour and Sayan Bandyapadhyay. Although the results in these collaboration did not appear in this thesis, I learnt a lot from you guys.

I also thank Mohammad R. Salavatipour, Viswanath Nagarajan, Martin Müeller, and Michael Buro for being on my examining committee.

My PhD experience would not be nearly as complete as it was without my fellow swimming club friends. It is a long list of people in the club who made my life in Edmonton more enjoyable even in harsh winters. It was a pleasure to swim with all you amazing teammates.

Finally and most importantly, I thank my parents and my brother for life-time of support and the encouragement to pursue anything of interest. I would not be here today without my brother for making me interested in math and science in the first place.

Contents

List of Figures

Chapter 1

Introduction

Network design is one of the most fundamental area in combinatorial optimization due to its contribution to both theory and practice. From the practical side, network design has applications in VLSI chip design, traffic and telecommunication networks, to name but a few. From the theory side, the study of network design contributed to the development of many algorithmic ideas, e.g. the *iterative rounding* techniques and primal-dual framework in the past three decades.

A typical problem in network design deals with a (directed or undirected) graph $G = (V, E)$ with non-negative edge costs and one seeks a subgraph of G with certain graph properties while minimizing the total cost of edges in the subgraph. Many graph properties are determined by connectivity requirements. For example, a connectivity requirement may be that the output subgraph must contain a path between two given nodes s and t which becomes the SHORTEST PATH problem, or in a directed graph we might be interested in a cheapest spanning subgraph rooted at a vertex r such that every vertex is reachable from r in this subgraph which is known as the ARBORESCENCE problem. The decision versions of both of these problems are in complexity class P, i.e., polynomial time decidable.

One of the classical problems in network design is to find a cheapest way of connecting a designated subset of vertices in a (directed or undirected) network. More formally, in the DIRECTED STEINER TREE problem (DST) we are given a directed graph $G = (V, E)$, a *root node* $r \in V$, non-negative edge

(a) (b)

Figure 1.1: In (a) we are given an instance of DST where the terminal nodes are depicted with squares, and the edge costs are shown next to the edges. In (b), green edges form a feasible solution for the DST instance in (a).

costs $c_e \geq 0$ for all $e \in E$, and a set of *terminal nodes* $X \subseteq V \setminus \{r\}$. The goal is to find a cheapest subgraph (i.e., the total cost of the edges in the subgraph) such that there is a dipath from r to every $x \in X$ in the subgraph. The nodes in $V \setminus (\{r\} \cup X)$ are called Steiner nodes. DST becomes the ARBORESCENCE problem when $X = V$. Throughout we let $n := |V|$ and $k := |X|$. See Figure 1.1 for an example.

One key aspect of DST lies in the fact that it generalizes many other important problems, e.g. UNDIRECTED STEINER TREE, SET COVER, and GROUP STEINER TREE. This generalization comes at the cost of being notoriously hard problem, i.e., it is known that the problem is NP-hard [39]. For this reason, a large effort has been put toward designing *approximation algorithms* for DST.

An *α-approximation algorithm* ($\alpha \geq 1$) for a minimization problem computes a solution of cost at most α times the cost of an optimal solution in polynomial time in the size of the input unless stated otherwise.

Not only is DST NP-hard, but it cannot be approximated within $O(\log^{2-\varepsilon} n)$ for any $\varepsilon > 0$ unless $NP \subseteq \text{ZTIME}\,(n^{\text{polylog}\,(n)})$ [35] (i.e., every problem in NP admits a probabilistic algorithm whose expected running time is $n^{\text{polylog}\,(n)}$, where n is the size of the input, with zero error probability). The best upper bound known is from more than two decades ago [11] who

gave a $O(k^{\varepsilon})$-approximation for any constant $\varepsilon > 0$. The same paper [11] also showed a $O(\log^3 k)$-approximation in $O(n^{\text{polylog}(k)})$ time (a.k.a. quasi-polynomial time). This was recently improved by Grandoni, Laekhanukit and Li [32], who gave a quasi-polynomial time $O(\frac{\log^2 k}{\log \log k})$-approximation factor for DST. They also provided a matching lower bound in that no asymptotically-better approximation is possible even for quasi-polynomial time algorithms under more refined complexity theoretic assumptions.

The undirected version of DST (i.e., UNDIRECTED STEINER TREE) is better understood. In this problem, the graph is undirected and there is no root node. The goal is to find a cheapest subgraph (in this case a tree) that connects all the terminal nodes. A series of papers steadily improved over the simple 2-approximation [40], [59], [62], [72] culminating in a $\ln 4 + \varepsilon$ for any constant $\varepsilon > 0$ [7]. Bern and Plassmann [5] showed that unless $P = NP$ there is no approximation factor better than $\frac{96}{95}$ for UNDIRECTED STEINER TREE.

Studying the complexity of network design problems on restricted metrics such as planar graphs and more generally, graphs that exclude a fixed minor has been a fruitful research direction. Garey and Johnson [27] showed the UNDIRECTED STEINER TREE problem is NP-hard even on planar graphs. Borradaile et al. [6] gave the first *polynomial time approximation scheme* (PTAS) for UNDIRECTED STEINER TREE on planar graphs and more generally [4] obtained a PTAS for STEINER FOREST on graphs of bounded-genus. Intuitively, graphs with bounded-genus can be drawn without edges crossing each other on a sphere with bounded number of handles (e.g., planar graphs can be drawn on sphere with no handles). A PTAS is a $(1+\varepsilon)$-approximation algorithm with running time being polynomial for any constant $\varepsilon > 0$. Very recently, Cohen-Addad [13] presented a *quasi-polynomial time approximation scheme* (QPTAS) for UNDIRECTED STEINER TREE on minor-free graphs (this family for example contains planar graphs and bounded-genus graphs). A QPTAS is a $(1+\varepsilon)$-approximation whose running time is $n^{O(\text{polylog}\, n)}$ for any constant $\varepsilon > 0$. In this sense, we know it is "easier" to approximate UNDIRECTED STEINER TREE on planar graphs than in general graphs. However,

this distinction was not clear in the directed setting. The first two chapters of this thesis address this distinction.

Question 1. *In what settings (e.g., what family of graphs), is DST easier to approximate than in general graphs?*

Another example of network design problems, probably the most famous problem in combinatorial optimization, is the TRAVELING SALESMAN problem (TSP). Given a set of cities in a road network represented by a graph, the task is to find a minimum cost tour (closed walk) visiting all cities. This problem has various practical applications from logistics to genome sequencing. TSP hosts a wide variety of algorithmic techniques including the Christofides-Serdyukov algorithm [12], [65] from 1976 to a more complex improvement by Karlin, Klein, and Oveis Gharan [38] from 2021.

In classic TSP, a minimum cost tour may pass through a location many times. However, in coming up with a travel plan on a road network, there are other factors to consider besides the cost. For example, one might prefer to go on a longer trip to avoid busy intersections or narrow junctions or at least limit the number of times they pass through busy locations. Another reason could be that a disruptive big truck prefers to avoid narrow intersections as much as possible. This motivates us to generalize TSP in the following way. We are given an undirected graph $G = (V, E)$, non-negative edge costs $c_e \geq 0$ for all $e \in E$ (not necessarily forming a metric), and even integer degree bounds b_v for all $v \in V$. The goal is to find a minimum cost tour that passes through v at most $\frac{b_v}{2}$ times for all $v \in V$ (i.e., the number of edges incident to v in the tour is at most b_v). We call this problem BOUNDED-DEGREE TRAVELING SALESMAN problem (BDTSP). See Figure 1.2 for an example.

In real-world applications, TSP problems are often concerned with visiting a subset of nodes of some larger graph. Thus, we consider the BOUNDED-DEGREE SUBSET TRAVELING SALESMAN problem (BDSTSP) in which we are given an undirected graph $G = (V, E)$ with edge costs $c_e \geq 0$ for all $e \in E$, a subset of terminals $X \subseteq V$ to visit ($|X| \geq 2$), and even integer bounds $b_v \geq 0$

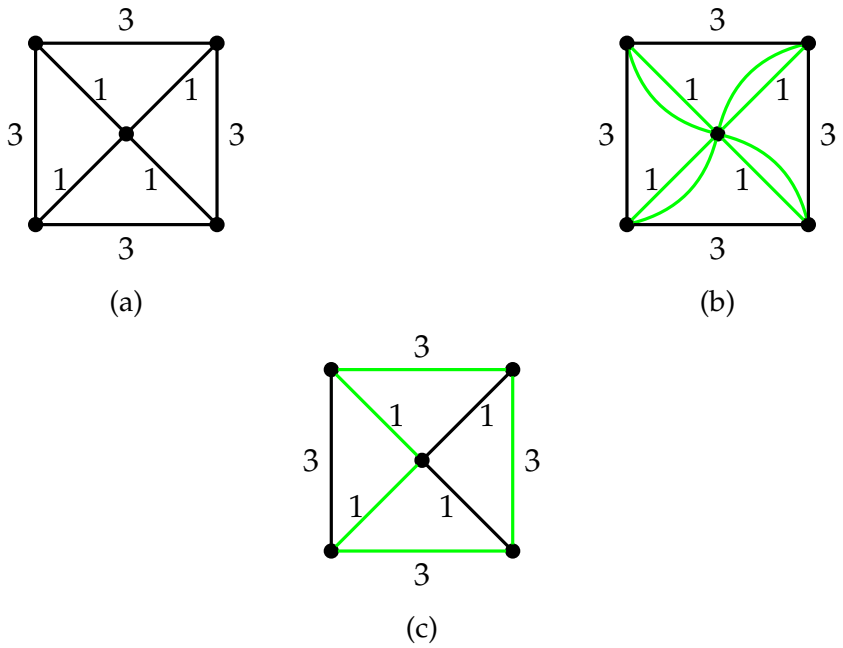


Figure 1.2: In (a) we are given a road network with edge costs shown next to the edges. If we are interested only in a minimum cost tour (like in TSP) then the green edges in (b) is such solution with cost 8. Now suppose we are given degree bounds 2 on all the vertices and our tour must respect these bounds. Then, the solution in (b) is not feasible for this instance of BDTSP. In (c) we show a tour of cost 11 that respects all the degree bounds.

for all nodes $v \in V$. The goal is to find a minimum cost tour $\mathcal{Q}$ spanning all terminals such that $d_{\mathcal{Q}}(v) \leq b_v$ (i.e., the number of edges in the multiset $\mathcal{Q}$ incident to v is at most b_v). Note that this is a generalization of BDTSP (i.e., $X = V$). Without these degree bounds, the problem is equivalent to TSP, e.g. by considering the metric completion of the underlying graph and then restricting it to X.

Finding special subgraphs whose vertices satisfy some degree bounds has been an active research area in computer science and operations research, e.g., BOUNDED-DEGREE SPANNING TREES [29], [66], BOUNDED-DEGREE STEINER NETWORKS [47], [49], [50], [54], BOUNDED-DEGREE ELEMENT-CONNECTIVITY and BOUNDED-DEGREE VERTEX-CONNECTIVITY [26], [43]. However, to the best of our knowledge, BDTSP has not been previously considered. In Chapter 5, we study the following question:

Question 2. *What could one hope to achieve for* BOUNDED-DEGREE SUBSET TRAVELING SALESMAN *in terms of algorithm design with provable guarantees?*

1.1 Outline and contributions of this thesis

In Chapter 2 we introduce some basic notation and briefly summarize well-known facts from combinatorial optimization that we will use in later parts of this thesis.

Chapter 3. In this chapter, we study the first question posed above. More precisely, we present an $O(\log k)$-approximation for DST on planar graphs. Given the hardness result for DST on general graphs (i.e., DST cannot be approximated within $O(\log^{2-\varepsilon} n)$ for any $\varepsilon > 0$ unless $NP \subseteq \text{ZTIME}\,(n^{\text{polylog}\,(n)})$ [35]), this shows DST is easier to approximate on planar graphs than in general graphs. We use Thorup's separator result [68] for planar graphs. This separator paired with a simple aggressive guessing of the optimal value leads to an easy $O(\log k)$-approximation in quasi-polynomial time. Then, we show that one

could be smarter in the guessing part and reduce the running time to polynomial. These results appeared in [25].

Our results here trivially extend to the *node-weighted* instances of DST where the Steiner nodes have non-negative cost. This is an extension of DST because we can subdivide an edge with a Steiner node of the same cost. Thus, in general graphs, node-weighted instances of DST reduces to the edge-weighted instances. However, this reduction does not preserve planarity. Here we show our approach also yields an $O(\log k)$-approximation for node-weighted instances of DST.

Another generalization that we consider is *multi-rooted* instances where instead of one root, we are given multiple roots and a feasible solution is a subgraph such that every terminal node is reachable from a root node. In general graphs, there is an easy reduction from multi-rooted instances to the single rooted instance: add an auxiliary node r^* and orient an edge of cost zero from r^* to all the original roots. However, like before this reduction does not preserve planarity, see Figure 1.3. For this generalization, we give two approximation algorithms: first one is a polynomial time approximation algorithm whose approximation factor depends on the number of roots, more specifically, the approximation factor is $O(R + \log k)$, where R is the number of roots. The second algorithm recovers the same approximation factor as in the single rooted case (i.e., $O(\log k)$); however, the running time is exponential in the number of roots and $\log k$. When $R = \operatorname{polylog} k$, the second algorithm recovers the same approximation factor as in the single rooted case in quasi-polynomial time.

Chapter 4. In this chapter we continue our study of Question 1. Another well-studied setting in UNDIRECTED STEINER TREE and DST is quasi-bipartite instances. These are instances where there is no edge between any two Steiner nodes (i.e., $V \setminus (X \cup \{r\})$ is an independent set in the input graph).

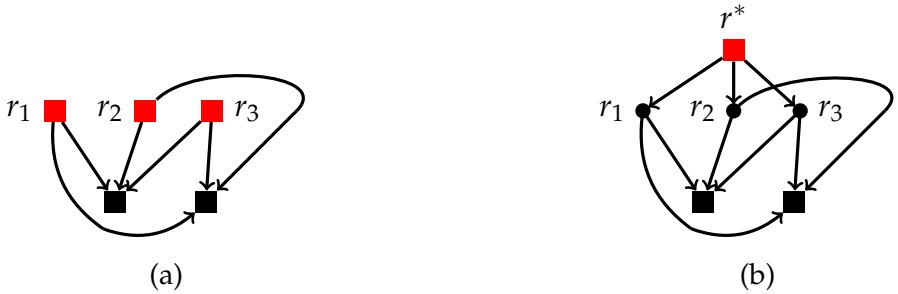


Figure 1.3: In (a) we are given a planar MR-DST instance with three roots shown in red. In (b) we show the resulting single rooted DST instance obtained from the reduction discussed before. Note that this graph is isomorphic to $K_{3,3}$ (i.e., complete bipartite graph with three nodes on each side) which is not planar.

Quasi-bipartite instances were first studied in UNDIRECTED STEINER TREE setting by Rajagopalan and Vazirani [60]. Feldmann et al. [19] studied UNDIRECTED 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 [7]. Currently, the best approximation in quasi-bipartite instances of UNDIRECTED STEINER TREE is $\frac{73}{60}$-approximation [30].

Hibi and Fujito [36] presented an $O(\log |X|)$-approximation algorithm for quasi-bipartite instances of DST. Assuming $P \neq NP$, this result asymptotically matches the lower bound $(1 - o(1)) \cdot \ln |X|$ for any $\varepsilon > 0$; this lower bound comes from the hardness of SET COVER [15], [18] and the fact that quasi-bipartite DST generalizes the SET COVER problem.

Our contribution here is an $O(1)$-approximation for quasi-bipartite DST on graphs excluding a fixed minor (e.g. planar graphs). We also show this problem is NP-hard. The results in this chapter are published in [23].

Our technique is based on a well-known framework in algorithm design called primal-dual algorithms. However, we show the generic

primal-dual framework used in past work is not enough. 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 for graphs that exclude a fixed minor.

Chapter 5. In this chapter, we address Question 2. We need the following notation to discuss our results. An $(\alpha, +d)$-approximation algorithm for BDSTSP, outputs a tour whose cost is at most α times the value of an optimal tour and violates the degree bound by an additive factor of d (i.e. the degree of the tour at v will be at most $b_v + d$). All of our results are based on LP rounding and therefore bound the integrality of the respected LPs.

We begin by noting that the integrality gap of a natural LP for BDTSP is $(\frac{4}{3}, +2)$, i.e., any tour has cost at least $4/3$ times the LP optimum and any tour violates the degree bound of at least one vertex by at least $+2$. Furthermore, we cannot hope for any approximation algorithm that outputs a tour of cost at most α times the optimal cost (for any α) with no violation in the degree bounds as otherwise it decides the HAMILTONIAN CYCLE problem which is known to be NP-complete. Motivated by these observations, we investigate the trade-offs between the cost of the tour and degree violation of vertices in the tour.

The general framework achieving our approximation algorithms are similar to Wolsey's analysis [70] of the Christofides-Serdyukov al-

gorithm for TSP. In particular, almost the same approach leads to a $(\frac{3}{2}, +4)$-approximation for BDTSP. For BDSTSP, Wolsey's technique does not work readily. However, we show a simple $(\frac{5}{3}, +4)$-approximation which is reminiscent of the work on the TRAVELING SALESMAN PATH problem [3]. In all the above algorithms, we use a result of Lau and Singh [49] on BOUNDED-DEGREE STEINER NETWORKS as a black-box.

In order to improve the cost factor, we first augment the natural LP for BDSTSP with non-trivial constraints asserting the degree of Steiner cuts should be at least the degree of any Steiner node in the cut. Then, we modify the iterative rounding algorithm of [49] using splitting off techniques by Mader [55] to obtain a more "structured" Steiner tree. Namely, some Steiner nodes are designated *dangerous* because they have low fractional degree in our LP solution: our modification ensures dangerous nodes will have even degree in the resulting tree. Finally, we show how this Steiner tree helps us to obtained a better bounded-degree Y-join to fix the degree parity of odd-degree vertices. Overall we get $(\frac{13}{8}, +6)$ and $(\frac{3}{2}, +8)$ approximation factors for BDSTSP. The results in this chapter appeared in [24].

Chapter 2

Preliminaries

The purpose of this chapter is to introduce some notation and classic results in combinatorial optimization used throughout this thesis. Apart from the first section that discusses the notation, the rest of this chapter may be skipped depending on the familiarity of the reader with the subjects.

2.1 Notation and terminology

We define the following three classes of algorithms based on their running time as a function of the input size $n := |I|$:

- Polynomial time: the running time is $O(n^c)$ for some constant $c > 0$.
- Quasi-polynomial time: the running time is $n^{O(\log^c n)}$ for some constant $c > 0$.
- Exponential time: the running time is $O(2^{n^c})$ for some constant $c \geq 1$.

An algorithm for a minimization (resp. maximization) problem is said to be an ***α-approximation algorithm*** for $\alpha \geq 1$ (resp. $\alpha \leq 1$) if for every instance I of the problem, it outputs a solution whose value is at most $\alpha \cdot \text{opt}_I$ (resp. at least $\alpha \cdot \text{opt}_I$) where opt_I is the optimum value of instance I, and runs in polynomial time in $|I|$ unless stated otherwise. We sometimes drop the subscript if the instance is clear from the context. The running time of an approximation algorithm is assumed to be polynomial in the size of the input unless stated otherwise. For the rest of the discussion here we only consider

minimization problems; however, the definition for maximization versions are similar. A ***polynomial time approximation scheme*** (PTAS) for a minimization problem is a $(1+\varepsilon)$-approximation algorithm whose running time is polynomial for any constant $\varepsilon > 0$. A ***quasi-polynomial time approximation scheme*** (QPTAS) for a minimization problem, is a $(1+\varepsilon)$-approximation algorithm whose running time is quasi-polynomial for any constant $\varepsilon > 0$.

We denote a *graph* with a tuple $G = (V, E)$, where V is the vertex set and E is the edge multi-set. An *edge* between u and v is denoted by $e = (u, v)$ where in directed graphs (digraphs for short) this means the edge is oriented from u to v. For brevity, in undirected graphs we denote an edge with endpoints u and v by uv. We will make it clear when we are discussing a directed or undirected graph.

Graphs discussed in this thesis are multi-graphs, i.e., there could be multiple edges between two endpoints (parallel edges) but no self-edges (loops), unless stated otherwise. For a (sub)graph H, we denote by $V(H)$ the vertex set of H and $E(H)$ the edge set of H. Given a subset $S \subseteq V$, denote by $E[S]$ the set of edges in E with both endpoints in S.

Let $G = (V, E)$ be a directed graph. We say G is ***strongly connected*** if for any pair of vertices $u, v \in V$ there are directed path (dipaths for short) from u to v and vice-versa in G. We say a set $S \subseteq V$ is a ***strongly connected component*** (SCC) of G if $(S, E[S])$ is strongly connected graph and S is inclusion-wise maximal (i.e., there is no set $V \supseteq T \supset S$ such that $(T, E[T])$ is strongly connected). We can view $F \subseteq E$ as a subgraph of G where the vertex set is $V(F)$, i.e., all the endpoints of edges in F, and the edge set is F. So $F \subseteq E$ is an SCC if $(V(F), F)$ is a strongly connected graph.

Fix a graph $G = (V, E)$. By ***contraction*** of an edge $e = (u, v)$, denoted by G/e, we mean identification of u and v, i.e., replacing u and v by a new vertex w adjacent to all the former neighbours (adjacent vertices) of u and v. We delete all the loops but keep all the parallel edges created in this process. Suppose (u, z) is an edge, then we say the ***corresponding edge*** of (w, z) in the original graph is (u, z). Let H be a subgraph of G. We denote by G/H the

resulting graph obtained by contracting all the edges in H and denote $G \setminus H$ the resulting graph by removing H from G, i.e., removing all the vertices in $V(H)$ and edges incident to those vertices.

A graph H is a ***minor*** of a graph G if it can be obtained by repeatedly deleting vertices, deleting edges, and contracting edges of G. We say G has an H-minor if there is a minor of G that is isomorphic to H and we say G is H-minor free otherwise. A family of graphs is H-minor free if each graph in the family is H-minor free. For example, it is known that planar graphs is a family of graphs that is K_5-minor free and $K_{3,3}$-minor free, where K_5 is a complete graph on 5 vertices and $K_{3,3}$ is a complete bipartite graph with 3 vertices on each side [46], [69].

Often the graphs we consider come with edge costs $c_e \geq 0$ for all $e \in E$. For a subset of edges $F \subseteq E$, the ***cost*** of F is defined as $\text{cost}_c(F) := \sum_{e \in F} c_e$. Similarly, for a vector $x \in \mathbb{R}^E$ (i.e., an $|E|$-dimensional vector with real value coordinates and the each coordinate corresponds to an edge in E) we define $x(F) := \sum_{e \in F} x_e$ and $\text{cost}_c(x) := \sum_{e \in E} c_e \cdot x_e$ for any $F \subseteq E$. If the edge cost c is clear from the context, we may drop the subscripts in $\text{cost}_c(.)$. To keep the notation light, for a subgraph H we may use $\text{cost}(H)$ or $x(H)$ instead of $\text{cost}(E(H))$ and $x(E(H))$, respectively.

The ***degree*** of a vertex v in an undirected (sub)graph H is the number of edges in H (with multiplicity) whose one endpoint is v and denoted by $d_H(v)$. Given subset of edges F, we define $odd(F) := \{v \in V : \ d_H(v) \text{ is odd}\}$. An undirected graph $G = (V, E)$ is ***Eulerian*** if $d_G(v)$ is even for all $v \in V$.

For a $U \subseteq V$ we define

$$\delta_G(U) := \{e \in E : \text{ exactly one endpoint of e is in U}\}$$

to be the set of edges in the cut corresponding to U. We say U is a s,t-cut if $|U \cap \{s,t\}| = 1$.

If G is directed, then for $U \subseteq V$ we define

$$\delta^{in}_G(U) := \{(v,u) \in E : \ u \in U, \ v \notin U\}$$

to be the set of incoming edges of U. Similarly, the set of outgoing edges of U is defines as $\delta^{out}_G(U) := \delta^{in}(V \setminus U)$. We define $\delta^{in}_G(v) := \delta^{in}_G(\{v\})$ and write $|\delta^{in}_G(v)|$ as the ***indegree*** of v. We omit the subscript above if the underlying graph is clear from the context.

Let $F \subseteq E$, then the ***characteristic vector*** of F, $\chi(F)$ is an $|E|$-dimensional vector whose coordinates correspond to E and the e-th coordinate is 1 if $e \in F$ and zero otherwise. Let F be a multi-subset of E (so there could be multiple occurrences of an element in E), then the ***incident vector*** of F is an $|E|$-dimensional vector where its e-th coordinate is equal the number of occurrences of e in F.

An ***arborescence*** in a directed graph $G = (V, E)$ rooted at $r \in V$ is a spanning subgraph with $|V| - 1$ edges and indegree of every vertex in $V \setminus \{r\}$ is exactly one and indegree of r is zero.

2.2 Linear programming

In ***linear programming*** (LP), we are given a matrix $A \in \mathbb{R}^{m \times n}$, vectors $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$. The goal is to find $x \in \mathbb{R}^n_{\geq 0}$ such that $Ax \geq b$ and $c^T x$ is minimized. We formulate this problem as follows:

$$\begin{aligned} \text{minimize:} \quad & c^T x && \text{(Primal-LP)} \\ \text{subject to:} \quad & Ax \geq b && (2.1) \\ & x \geq 0 \end{aligned}$$

The polyhedron $P := \{x \in \mathbb{R}^n : Ax \geq b,\ x \geq 0\}$ is the set of feasible solutions for (Primal-LP). If A and b are rational, then P is called *rational* polyhedron. A bounded polyhedron is called *polytope*. We say $x \in P$ is an ***extreme point*** of P if there does not exist a non-zero vector $y \in \mathbb{R}^n$ such that $x + y, x - y \in P$. A polyhedron P is ***integral*** if every extreme point of P is integral.

There are different forms in which an LP can be represented. However, all these forms are equivalent to the form we consider above.

Khachiyan [42] proved linear programs can be solved in polynomial time in the size of the input for rational A, b, and c. Here "solve" means either the algorithm finds an optimum solution, or declares it is infinite, or outputs the feasible region is empty. This result requires the LP to be given explicitly. Khachiyan's algorithm is often referred to as ***ellipsoid method***. Later Grötschel, Lovász, and Schrijver [33] showed that for the ellipsoid method to work it is "sufficient"[1] if the feasible region, P, is given only implicitly via a ***separation oracle***. A separation oracle for a convex set P, receives a vector $y \in \mathbb{Q}^n$ and either decides $y \in P$ or finds a vector $a \in \mathbb{Q}^n$ such that $a^T x < a^T y$ for all $x \in P$.

Often in applications of linear programs in algorithm design, we need to compute an optimal solution that is an extreme point of the LP. Grötschel, Lovász, and Schrijver [33] showed this is doable in polynomial time given a separation oracle. Again this result uses some mild assumptions that hold for all the problems we consider in this thesis.

The following is an important property of extreme points of a polyhedron.

Lemma 1 (Rank Lemma [48]). *Let $P = \{x \in \mathbb{R}^n : Ax \geq b,\ x \geq 0\}$ be a polyhedron and let x be an extreme point of P such that $x_i > 0$ for each $1 \leq i \leq n$. Then, any maximal number of linearly independent rows a_i's of A with corresponding tight constraints (i.e., $a_i^T x = b_i$) equals the number of variables (i.e., n).*

LP duality

The ***dual*** of (Primal-LP) is

$$\begin{aligned} \text{maximize:} \quad & b^T y && \text{(Dual-LP)} \\ \text{subject to:} \quad & A^T y \leq c \\ & y \geq 0 \end{aligned}$$

[1] With some reasonable assumptions such as the feasible region is a rational polytope, we have a bound on the bit complexity of the vertices of the polytope, and knowing a point in the interior of the polytope. The LP relaxation for combinatorial problems usually satisfy these conditions.

We have the following relations between (Primal-LP) and (Dual-LP) that can be found in any combinatorial optimization textbook, e.g. [45].

Theorem 2. *Let $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and $c \in \mathbb{R}^n$. Then, we have*

- *(Weak duality) For any $x \in \mathbb{R}^n$ feasible for* (Primal-LP) *and any $y \in \mathbb{R}^m$ feasible for* (Dual-LP) *we have $b^T y \leq c^T x$.*
- *(Strong duality) If* (Primal-LP) *is feasible and bounded so is* (Dual-LP) *and both LPs have the same optimal value.*

LP relaxations

Given an optimization problem whose feasible solutions can be represented by incident vectors (e.g. the incident vector could indicate whether an edge is in the solution or not by having 0 or 1 in the corresponding entry), we say an LP is a ***relaxation*** for this problem if the incident vector of feasible solutions of I (an instance of the problem) are feasible solutions of the LP with the same cost. For example, consider the ARBORESCENCE problem, where we are given a directed graph $G = (V, E)$, a root node $r \in V$, and edge costs $c_e \geq 0$ for all $e \in E$. The goal is to find an arborescence with minimum cost. The following is an LP relaxation for this problem.

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

Note that the optimum value of a relaxation for an instance I (of a minimization problem) is at most opt_I, the optimal value of the instance I. We say an LP is ***integral*** if it has an integral optimal solution (i.e., all the coordinates of the vector is integers). For example, the result discussed in the next section shows (**Arb LP**) is integral.

Integrality gap

Consider a minimization problem with an LP relaxation. Let I be an instance of this problem and let LP_I be the optimum value of the LP relaxation for instance I. The ***integrality gap*** of this LP is define as

$$\sup_I \frac{\text{opt}_I}{LP_I}.$$

We note that sometimes the integrality gap depends on the size, n, of the input, e.g. $\log n$, $\sqrt{n}$. So the integrality gap could be a function of the input size.

The importance of this notion lies in the fact that it gives a conditional lower bound for the approximability of the underlying problem using the LP relaxation. More precisely, if the integrality gap of an LP relaxation for a problem is β, then we cannot get better than β-approximation for the problem by rounding a solution of this relaxation to an integral one.

2.3 Arborescences and primal-dual algorithms

As we discussed in the introduction, one of the tools we use in this thesis is the *primal-dual* framework. In this approach one simultaneously builds an approximate solution for the problem and a feasible (fractional) solution for the dual of an LP relaxation of the problem. We demonstrate this approach with proof sketches of the analysis in more details on the ARBORESCENCE problem (defined in the previous section). We demonstrate Edmonds' algorithm [16][2] for ARBORESCENCE here because this algorithm captures the generic primal-dual framework that is used in many combinatorial optimization problems, see [31] for many such examples. Furthermore, this algorithm is a starting point in our algorithm for quasi-bipartite DST on planar graphs, see Chapter 4.

We consider (**Arb LP**) as a relaxation for the ARBORESCENCE problem. The dual of this relaxation is define as follows.

[2]With slight modification where in [16] the active dual variables are raised one at the time but the way we present it here is all the active dual variables are raised at the same time. The latter approach works as well. The reason we present it like this is because it is more aligned with the generic primal-dual framework.

$$\begin{aligned} \text{minimize:} \quad & \sum_{\emptyset \neq S \subseteq V \setminus \{r\}} y_S && \textbf{(Arb-dual LP)} \\ \text{subject to:} \quad & \sum_{S: e \in \delta^{in}(S)} y_S \leq \quad c_e \quad \forall e \in E && (2.2) \\ & y \geq \quad 0 \end{aligned}$$

Note that in any feasible solution for ARBORESCENCE, for every subset of vertices S that separates a terminal node from the root node, there must be at least one edge entering S. At each iteration of the primal-dual algorithm, we have a "partial" solution for the instance meaning there are some sets that violate the mentioned property and we would like to extend this partial solution to a feasible solution. This motivates the following definition: given a subset of edges $F \subseteq E$, we say a set $\emptyset \neq S \subseteq V \setminus \{r\}$ is ***violated*** if $\delta^{in}_F(S) = \emptyset$. We call a minimal (inclusion-wise) violated set, an ***active*** set (or sometimes also called active moat). For example, when $F = \emptyset$ then every singleton $\{v\}$ is an active set for all $v \in V \setminus \{r\}$.

Algorithm 1 Primal-dual algorithm for ARBORESCENCE

Input: Directed graph $G = (V, E)$, a root node $r \in V$, and edge cost $c_e \geq 0$ for all $e \in E$.
Output: An Arborescence rooted at r.

$F \leftarrow \emptyset$.
$\ell \leftarrow 0$
Growing phase:
$\mathcal{A}$, initially, is the set to all singletons $\{v\}$'s for $v \in V \setminus \{r\}$.
while $\mathcal{A} \neq \emptyset$ **do**
  $\ell \leftarrow \ell + 1$
  Increase the dual variables corresponding to all the active moats until for some edge e_ℓ we have $\sum_{S: e_\ell \in \delta^{in}(S)} y_S = c_{e_\ell}$. Add e_ℓ to F and update $\mathcal{A}$ based on the updated set F.
Deletion phase:
$\overline{F} \leftarrow F$.
for i from ℓ to 0 **do**
  if $\overline{F} \setminus \{e_i\}$ is an arborescence **then**
    $\overline{F} \leftarrow \overline{F} \setminus \{e_i\}$.
return $\overline{F}$.

Algorithm 1 falls into the following framework. Increase all the dual variables corresponding to active moats uniformly until a dual constraint becomes tight. Then, add the "object" (here objects are edges), corresponding to the tight constraint, to the solution. And at the end, we delete "redundent" edges in the reverse order they have been purchased. Many existing primal-dual algorithms in the literature use this framework and because of this we refer to Algorithm 1, the *standard primal-dual* algorithm.

Example 3. *Let us show how Algorithm 1 works on an example. In Figure 2.1(a) we are given an instance of the* ARBORESCENCE *problem. Active moats are shown in blue. The moats that become inactive are depicted in red, and green edges are the ones the algorithm adds them to F. Letters in parenthesis refer to the subfigures in Figure 2.1. In (b), all the singleton vertices are active and we raise their corresponding dual variables to* 1*. Then, the active moats* $\{u\}, \{v\}, \{w\}$ *become inactive because there are edges of cost* 1 *entering them. The algorithm adds these edges to F, this is shown in (c). Next, the only active moat,* $\{z\}$*, raises its dual variable from* 1 *to* 2 *and the edge* (w, z) *will be added to F. Now,* $\{z\}$ *becomes inactive but we have a new SCC* $\{v, w, z\}$ *that is violated. This process continues until there is no active moat, see figures (d)-(f). Finally, edges* (z, v) *and* (v, u) *get deleted in the reverse delete process. The final arborescence output by the algorithm is shown in (g).*

One can show that Algorithm 1 outputs an optimal arborescence. The high-level idea behind the analysis is as follows. Fix an iteration ℓ. One can view the growth phase of the algorithm as active moats raising dollars until we have for some edge (u, v), the sum of all the dollars in each moat that contains v is equal to the cost of (u, v). Then, we can purchase (u, v). The following are the steps one could take to show Algorithm 1 outputs an optimal arborescence. Fix an iteration ℓ.

(i) One can show every active moat at iteration ℓ is an SCC (i.e., strongly connected component).

(ii) The number of edges in the final solution (after deletion phase) entering an active moat A at iteration ℓ is exactly 1. This is guaranteed by the

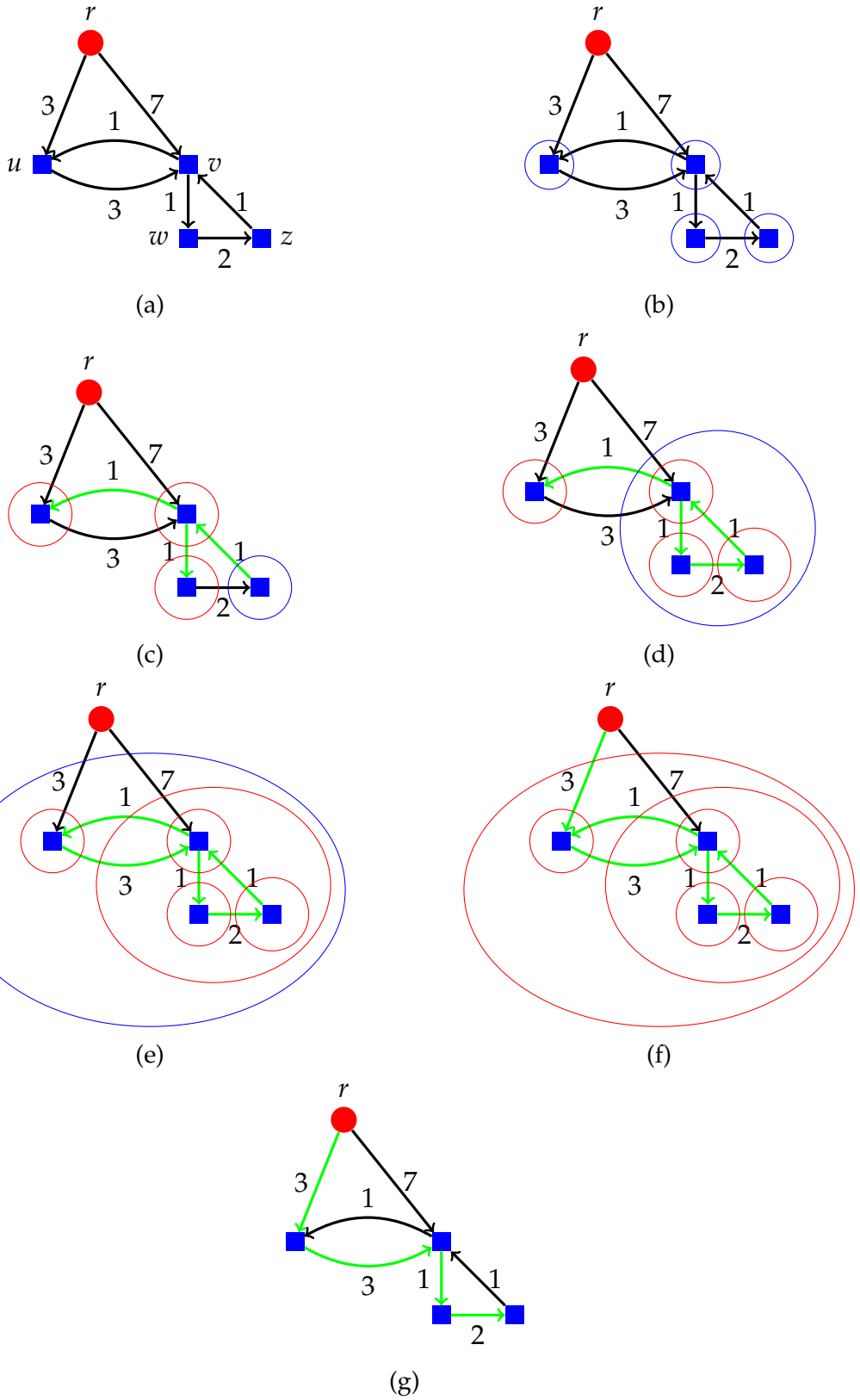


Figure 2.1: Primal-dual algorithm for ARBORESCENCE.

deletion phase and property (i).

(iii) Property (ii) ensures every dollar raised by active moat A at iteration ℓ goes toward the cost of the edge in $\overline{F}$ entering A in this iteration. Since there is only one edge entering A, there is no double charging.

(iv) Property (iii) ensures the dual raised in total is enough to pay for every edge that is bought and we can write

$$\sum_{e \in \overline{F}} c_e = \sum_{S \subseteq V} y_S \leq \text{opt},$$

where the last inequality follows from the weak duality (see Theorem 2).

2.4 Laminar families

Let A and B be subsets of a common set. We say A and B are ***crossing*** if all sets $A \cap B$, $A \setminus B$, $B \setminus A$ are non-empty, and call them ***non-crossing*** otherwise.

Definition 4 (Laminar family)**.** *Let V be a finite set and let $\mathcal{L}$ be a family of subsets of V. We say $\mathcal{L}$ is* laminar *if any pair of sets in $\mathcal{L}$ are non-crossing.*

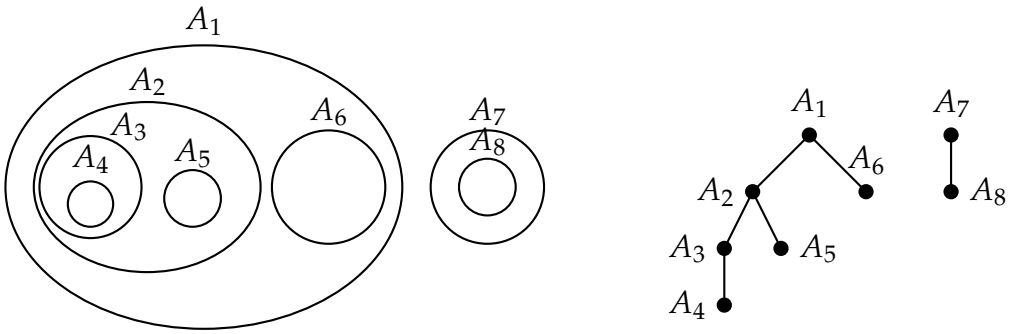
A laminar family $\mathcal{L}$ defines naturally a forest L (see Figure 2.2): each node of L corresponds to a set in $\mathcal{L}$, and there is an edge from set A to set B if A is the smallest set containing B. A is called the *parent* of B and B is called the *child* of A. A node with no parent is called *root* node.

The following is a simple known fact about any laminar family that will be useful and its proof can be found, for example, in Chapter 4 of [48].

Lemma 5. *Let V be a finite set and $\mathcal{L}$ be a laminar family of non-empty subsets of V. Then, $|\mathcal{L}| \leq 2 \cdot |V| - 1$.*

2.5 Steiner trees and iterative rounding

In the UNDIRECTED STEINER TREE problem (or just STEINER TREE for short), we are given an undirected graph $G = (V, E)$ with edge costs $c_e \geq 0$ for



(a) Laminar family $\mathcal{L}$ (b) Corresponding forest L

Figure 2.2

all $e \in E$ and a collection of terminals $X \subseteq V$. The nodes in $V \setminus X$ are called *Steiner* nodes. The goal is to find a minimum cost tree $F \subseteq E$ spanning X. Note that in the case of $X = V$ this problem becomes MINIMUM COST SPANNING TREE, and if $X = \{s, t\}$ then it turns to the *st*-SHORTEST PATH problem which in both cases can be solved efficiently. However, in general, the STEINER TREE problem cannot be approximated better than $\frac{96}{95}$ unless $P = NP$ [5]. A series of papers steadily improved over the simple 2-approximation [40], [59], [62], [72] culminating in a $(\ln 4 + \varepsilon)$-approximation for any constant $\varepsilon > 0$ [7]. There is a natural LP relaxation for this problem:

$$\begin{aligned} \text{minimize:} \quad & \sum_{e \in E} c_e \cdot x_e & \textbf{(ST LP)} \\ \text{subject to:} \quad & x(\delta(S)) \geq 1 \quad \forall S \neq X,\ S \cap X \neq \emptyset \\ & x \geq 0 \end{aligned}$$

In a seminal paper [37], Jain showed the integrality gap of the above LP is at most 2. Actually he showed the integrality gap of 2 for a more general optimization problems. To prove this he developed an *iterative rounding* scheme where he showed in an extreme point solution of (**ST LP**) there is a variable $x_e \geq \frac{1}{2}$. This suggests the following recursive algorithm. Start with $T = \emptyset$. Find an extreme point x of (**ST LP**). Pick an edge e with $x_e \geq \frac{1}{2}$ and add it to T. Delete e from the graph and update the LP by removing all the constraints for all cuts that contains exactly one endpoint of e and repeat.

One can generalize STEINER TREE by augmenting it with degree bounds.

In the BOUNDED-DEGREE STEINER TREE problem (BD-Steiner-TP), in addition to the input of STEINER TREE, we are given degree bounds $b_v \geq 0$ for all $v \in W \subseteq V$. The goal is to find a minimum cost tree T that spans the terminals and $d_T(v) \leq b_v$ for all $v \in W$. A natural relaxation for this problem is given in (**BD-ST LP**). Lau and Singh [49] extended Jain's iterative rounding technique to this problem and showed the following.

Theorem 6 (Theorem 1.1 in [49])**.** *There exists a polynomial time algorithm for BD-Steiner-TP which given an extreme point x^* of* (**BD-ST LP**)*, returns a Steiner tree T of cost at most $2 \cdot \text{cost}(x^*)$ and $d_T(v) \leq b_v + 3$ for all $v \in W$.*

The following is an immediate consequence of the above theorem.

Corollary 7. *Let $\overline{x}$ be any feasible solution to* (**BD-ST LP**)*. Then, in polynomial time, one could get a Steiner tree T of cost at most $2 \cdot \text{cost}(\overline{x})$ and $d_T(v) \leq b_v + 3$ for all $v \in W$.*

Proof. Let x^* be an optimal extreme point solution for (**BD-ST LP**) and then apply Theorem 6 and noticing that $\text{cost}(\overline{x}) \leq \text{cost}(x^*)$. As stated in the linear programming section, there is a polynomial time algorithm that computes an optimal extreme point solution given a separation oracle. One can design a polynomial time separation oracle for (**BD-ST LP**), see the discussion below. So overall the running time of the algorithm is polynomial. □

$$\begin{aligned} \text{minimize:} \quad & \sum_{e \in E} c_e \cdot x_e & & & & \textbf{(BD-ST LP)} \\ \text{subject to:} \quad & x(\delta(v)) \leq & b_v & \quad \forall v \in W \\ & x(\delta(S)) \geq & 1 & \quad \forall S \neq X,\ S \cap X \neq \emptyset \\ & x \geq & 0 \end{aligned}$$

Although (**BD-ST LP**) has exponentially many constraints, there is an easy separation oracle for these constraints. Therefore, it can be solved via ellipsoid method with running time polynomial in the number of variables.

The separation oracle is as follows: given a vector $x \in \mathbb{R}^{|E|}_{\geq 0}$, find a set S^* in G with minimum $x(\delta(S^*))$, this is the MINIMUM-CUT problem which can be solved in polynomial time, for example, see Chapter 8 in [45]. If the minimum cut is less than one, then the incident vector corresponding to $\delta(S^*)$ forms a separating hyperplane otherwise every cut has value at least 1, hence x is a feasible solution for (**BD-ST LP**).

2.6 Y-joins

Let $G = (V, E)$ be an undirected graph and let $F \subseteq E$ be a multi-set of edges. Recall $odd(F)$ is the set of vertices in V whose degree with respect to F is odd.

Definition 8 (Y-join)**.** *Let $G = (V, E)$ be an undirected graph and $Y \subseteq V$ with even cardinality. Then, $J \subseteq E$ is a Y-join if $odd(J) = Y$*[3]*.*

(a) (b)

Figure 2.3: An example of Y-join. The set Y is indicated by large (blue) vertices. A Y-join is indicated by red dashed edges in (b).

An undirected graph G contains a Y-join if and only if every connected component of G contains an even number of elements of Y (for a proof of this fact, see T-join section in [45]). Given edge costs on G, the Y-JOIN problem is to find a Y-join of minimum cost. Edmonds and Johnson [17] showed that the feasible region of the following LP relaxation of Y-JOIN is integral.

[3]In the combinatorial optimization literature, this concept is referred to as T-join. However, since we use T for trees throughout this thesis, we adopt Y-join notation instead.

$$\begin{aligned} \text{minimize:} \quad & \sum_{e \in E} c_e \cdot x_e && && \textbf{(Y-join LP)} \\ \text{subject to:} \quad & x(\delta(S)) \geq && 1 \quad \forall S \subseteq V \text{ s.t. } |S \cap Y| \text{ is odd} && (2.3) \\ & x \geq && 0 \end{aligned}$$

For subsets S where $|S \cap Y|$ is odd, we say S is a Y-odd set and the constraints corresponding to these subsets are called *odd-cut* constraints, i.e., the constraints in (2.3). There is a separation oracle for this LP and so it can be solved in polynomial time, for example, see Section 12.3 in [45].

One can show that in an extreme point x of (**Y-join LP**), there is always an edge $e = (u, v)$ whose x-value is either 0 or 1 (the proof is similar to the same fact about matching polytope given in [48]). This suggests the following simple iterative algorithm. Start with $J = \emptyset$. If $x_e = 0$, then remove e and remove all the constraints corresponding to set S that contains e in the cut. Then, recursively solve the Y-join problem in the residual graph and the updated LP. And if $x_e = 1$, then add e to J, delete e from the graph and recursively solve the $(Y \Delta \{u, v\})$-join problem in the residual graph. This is given as an exercise in Chapter 9 of [48], where $A \Delta B := (A \setminus B) \cup (B \setminus A)$ is the symmetric difference of A and B.

Just like STEINER TREE, we can generalize Y-JOIN to include degree bounds. A natural LP relaxation for this generalization is to take (**Y-join LP**) and add degree constraints to it. A less known fact in combinatorial optimization is that this LP is integral under the condition that degree bounds for vertices in Y are odd and even otherwise. We talk more about this in Chapter 5 where we use this fact.

2.7 The splitting-off procedure

Fix an undirected multi-graph $G = (V, E)$. We say edge e is a ***cut-edge*** if there is a set S such that $\delta(S) = \{e\}$. We denote the minimum cardinality of a u, v-cut by $\lambda_G(u, v)$. We say a pair of edges (su, sv) is a ***splittable pair*** if in

$G' = \big(V, (E \setminus \{su, sv\}) \cup \{uv\}\big)$, we have $\lambda_G(u,v) = \lambda_{G'}(u,v)$ for all $u, v \in V \setminus \{s\}$. In other words, removing su, sv and adding an edge between u and v preserves the minimum u,v-cut value for all $u, v \in V \setminus \{s\}$. This process is called ***splitting-off*** pair (su, sv). The following is a classical splitting-off result by Mader.

Theorem 9 (Mader [20], [55])**.** *Let $G = (V, E)$ be a connected graph and let $s \in V$ be an even degree vertex. Assume there is no cut-edge incident to s. Then, there is a splittable pair (su, sv) for some u, v (possibly $u = v$) adjacent to s.*

When we remove all the splittable pairs so that there is no incident edge to s, we say we *completely split-off* s. Since after applying Mader's theorem, the conditions of the theorem still holds for s, one can repeatedly apply Mader's theorem to completely split-off s while preserving the local connectivities between any pair of vertices in V.

Figure 2.4: (a) shows the original graph and (b) shows the resulting graph after splitting-off (su, sv) pair.

Chapter 3

Planar Directed Steiner Tree

In this chapter, we study Question 1 posed in the introduction. We show DST on planar graphs is easier to approximate than in general graphs by giving an $O(\log k)$-approximation for DST on planar graphs. Since DST on general graphs does not have an approximation factor better than $O(\log^{2-\varepsilon} n)$ for any $\varepsilon > 0$ unless $NP \subseteq \text{ZTIME}\,(n^{\text{polylog}\,(n)})$ [35], where ZTIME(t) is the class of problems that can be solved by probabilistic algorithms whose expected running time is t with zero error probability. This makes a clear distinction between the complexity of DST on planar graph and general graphs.

3.1 Introduction and outline

Let us first recall the definition of 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, 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 $r - t$ directed path (dipath for short) 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. Recall the definition of arborescence from Section 2.1. Throughout, we let $n := |V|$ and $k := |X|$.

Building on a height-reduction technique of Calinescu [8] and Zelikovsky [71], Charikar et al. give a recursive greedy algorithm that currently provides

the best approximation for DST which is an $O(k^{\varepsilon})$-approximation for any constant $\varepsilon > 0$ [11] and also an $O(\log^3 k)$-approximation in $O(n^{\text{polylog}(k)})$ time (quasi-polynomial time). The latter was recently improved by Grandoni, Laekhanukit, and Li [32], who give a quasi-polynomial time $O(\frac{\log^2 k}{\log\log k})$-approximation factor for DST. They also provide a matching lower bound in that no asymptotically-better approximation is possible even for quasi-polynomial time algorithms, unless either the PROJECTION GAMES CONJECTURE fails to hold or $NP \subseteq \text{ZPTIME}(2^{n^{\delta}})$ for some $0 < \delta < 1$.

Ghuge and Nagarajan [28] studied a variant of DST called the SUBMODULAR 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 [32] for DST.

It is known that UNDIRECTED STEINER TREE on planar graphs is NP-hard [27], and since DST is a generalization of UNDIRECTED STEINER TREE, DST on planar graphs is NP-hard as well. Prior to our work in this chapter, there was no concrete result for DST on planar graphs. It is worth noting that Demaine, Hajiaghayi, and Klein [14] show that if one takes a standard flow-based relaxation for DST in planar graphs and further constrains 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. However, the resulting relaxation is non-convex and, to date, we do not know how to compute a low-cost, non-crossing flow in polynomial time for DST instances on planar graphs.

Our approach is based on the planar separators presented by Thorup [68][1] which states given an undirected graph G with n vertices, one can find a "well-structured" subgraph F such that each connected component of $G \setminus F$ has at most $\frac{n}{2}$ vertices. Well-structured separators are useful in enabling divide-and-conquer approach for some problems, such as MAXIMUM INDEPENDENT SET and PEBBLING [53]. We note that our approach was inspired by a recent work of Cohen-Addad [13] who uses similar separator to design QP-

[1] As stated in [68] this separator theorem was implicitly proved in [52].

TASes for k-MST and UNDIRECTED STEINER TREE on planar graphs. He also develops a new separator to deal with these problems in minor-free graphs.

We show the separator theorem of Thorup can be used to obtain a simple logarithmic approximation algorithm for planar DST.

Theorem 10. *There is a $O(\log k)$-approximation for planar* DIRECTED STEINER TREE *where k is the number of terminals.*

We remark that it is trivial to generalize our algorithm to the node-weighted setting of DST in planar graphs. That is, to instances where Steiner nodes $v \in V \setminus (X \cup \{r\})$ have costs $c_v \geq 0$ and the goal is to find the cheapest S of Steiner nodes such that the graph $G[\{r\} \cup X \cup S]$ contains an $r - t$ dipath for each $t \in X$. Clearly, node-weighted DST generalizes edge-weighted DST even in planar graphs settings since we can subdivide an edge with cost c_e and include this cost on the new node. In general graphs, edge-weighted DST generalizes node-weighted DST because a node v with cost c_v can be turned into two nodes v^+, v^- connected by an edge (v^+, v^-) with cost c_v; edges entering v now enter v^+ and edges exiting v now exit v^-. But this operation does not preserve planarity, and it is easy to find examples where this results in a non-planar graph.

We can extend our result to the multi-rooted case. In MULTI-ROOTED DIRECTED STEINER TREE (MR-DST), instead of one root, we are given multiple roots $r_1, \ldots, r_R$ and the set of terminals $X \subseteq V \setminus \{r_1, \ldots, r_R\}$. The goal here is to find a minimum cost $F \subseteq E$ such that every terminal is reachable from one of the roots using only edges in F.

Note that MR-DST on general graphs is equivalent to DST by adding an auxiliary root node r and adding edges (r, r_i) for $1 \leq i \leq R$ with zero cost. However, this reduction does not preserve planarity. We provide two approximation factors for MR-DST. We obtain our results for MR-DST by constructing a "well-structured" separator for the multi-rooted case.

Theorem 11. *There is a polynomial time $O(R + \log k)$-approximation algorithm for planar* MULTI-ROOTED DIRECTED STEINER TREE, *where R is the number of*

roots and k is the number of terminals.

Theorem 12. *There is an $O(\log k)$-approximation for planar* MULTI-ROOTED DIRECTED STEINER TREE *with running time $n^{O(R \cdot \log k)}$.*

We note if $R = \text{polylog}\, k$, then the second result recovers the $O(\log k)$-approximation factor for the single rooted case albeit the running time is quasi-polynomial.

3.2 Notation and some basic facts

In this section, we recall some notation from Preliminaries chapter and some facts that will be used later in the chapter.

For convenience, we allow our input graphs to contain multiple directed edges between two nodes. All directed paths (dipath for short) are simple. Fix a digraph $G = (V, E)$ with edge costs $c_e \geq 0$ for all $e \in E$. We identify a dipath P by its corresponding sequence vertices, i.e., $P = v_1, \ldots, v_a$ where $(v_i, v_{i+1}) \in E$ for all $1 \leq i \leq a-1$, and we say P is a $v_1 - v_a$-dipath. The *start* and *end* vertices of P are v_1 and v_a, respectively. For a subgraph H of G, recall the cost of a H is $\text{cost}_c(H) := \sum_{e \in E(H)} c_e$.

We say a vertex v is *reachable* from u if there is a dipath from u to v. We denote by $d_c(u, v)$ the cost of a shortest dipath from u to v, in particular, $d_c(u, u) = 0$. The *diameter* of a digraph is defined as the maximum $d_c(u, v)$ for all $u \neq v$ where v is reachable from u. For both $d_c(.)$ and $\text{cost}_c(.)$ we drop the subscript c if the edge costs is clear from the context. For a subset $S \subseteq V$ and a vertex v, we define $d(S, v) := \min_{u \in S}\{d(u, v)\}$.

Recall $G[S]$ is the *induced subgraph* of G on the subset of vertices S. A *weakly connected component* of G is a connected component of the undirected graph obtained from G by ignoring the orientation of the edges. The *indegree* of a vertex v with respect to $F \subseteq E$ is the number of edges in F oriented towards v.

A *branching* $T = (V_T, E_T)$ rooted at r in G, is a (not necessarily spanning) subgraph of G such that $r \in V_T$ and T is a directed tree oriented away from

r. Note arborescences are branchings that span all the vertices. A *breadth first search* (BFS) branching B_G rooted at r is a branching including all nodes reachable from r where the dipath from r to any vertex v on B_G is a shortest dipath from r to v according to metric $d_c(.,.)$.

For two disjoint subsets of vertices $S, T \subseteq V$ denote by $\delta(S,T)$ the set of edges with one endpoint in S and the other endpoint in T (regardless of the orientation).

Our algorithm is based on planar separators described by Thorup [68].

Theorem 13 (Lemma 2.3 in [68])**.** *Let $G = (V, E)$ be a connected and undirected planar graph with non-negative vertex weights, and let T be a spanning tree rooted at a vertex $r \in V$. In linear time, one can find three vertices v_1, v_2, and v_3 such that the union of vertices on paths P_i between r and v_i in $V(T)$ for $i = 1, 2, 3$ forms a separator of G, i.e., every connected component of $G \setminus (P_1 \cup P_2 \cup P_3)$ has at most half the weight of G, where the weight of a subset of vertices is the sum of the weights of vertices in the subset.*

An immediate consequence of the above result is that given a directed graph and a spanning BFS branching rooted at r instead of a spanning tree, one can obtain a separator consisting of three shortest dipaths each starting at r.

Corollary 14 (Directed separator)**.** *Let $G = (V, E)$ be a planar digraph with edge costs $c_e \geq 0$ for all $e \in E$, and non-negative vertex weights such that every vertex $v \in V$ is reachable from r. Given a vertex $r \in V$, in polynomial time, we can find three shortest dipaths P_1, P_2, and P_3 each starting at r such that every weakly connected component of $G \setminus (P_1 \cup P_2 \cup P_3)$ has at most half the weight of G.*

Consider an instance I of DST (or MR-DST in general). Throughout this chapter, we create subinstances from I by contracting a subset of edges F in G. Whenever, we create a subinstance I' we let the edge cost for the subinstance to be the natural restriction of c to G/F, i.e., if e is in both $E(G)$ and $E(G/F)$ then e has cost c_e in I' and if e is in $E(G/F)$ but not in $E(G)$, then its cost

in I' is set to be the cost of the corresponding edge in $E(G)$, see 2.1 for the definition of "corresponding edge".

Let $I = \big(G = (V, E), c, \{r_1, \ldots, r_R\}, X\big)$ be an instance of MR-DST on planar graphs where G is a planar digraph, $c_e \geq 0$ for all $e \in E$ is the edge costs, $\{r_1, \ldots, r_R\}$ are the roots, and $X \subseteq V \setminus \{r_1, \ldots, r_R\}$ is the set of terminals. By losing a small factor in the approximation guarantee, one can assume in an instance of MR-DST that all the costs are positive integers and $d\big(\{r_1, \ldots, r_R\}, v\big)$ is polynomially bounded by n for all $v \in V$. The proof is standard but we include it for completeness.

Lemma 15 (Polynomial bounded distances)**.** *For any constant $\varepsilon > 0$, if there is an α-approximation for MR-DST instances in planar graphs where all edges have positive integer costs and $d_c(r, v) \leq \frac{|X| \cdot |V|}{\varepsilon} + |V|$ for each $v \in V$, then there is an $(\alpha \cdot (1 + \varepsilon))$-approximation for general instances of MR-DST in planar graphs.*

Proof. Let $\Delta := \max_{t \in X} \Big\{ d\big(\{r_1, \ldots, r_R\}, t\big) \Big\}$, i.e., Δ is the maximum distance from any root to a terminal. Let opt_I be the optimal value of instance I. Then, $\Delta \leq \mathrm{opt}_I \leq k \cdot \Delta$.

If $\Delta = 0$, then $\mathrm{opt}_I = 0$ and the collection of all shortest dipaths from the roots to the terminals is a solution of cost 0. So we assume $\Delta > 0$.

We can safely remove any edge e having $c_e > k \cdot \Delta$ and any Steiner node v (along with its incident edges) having $d(\{r_1, \ldots, r_R\}, v) > k \cdot \Delta$ since no optimal solution of I uses e or v. Since we have only deleted elements of G, it remains planar.

Define a new edge costs $c'_e := \lceil c_e \cdot \frac{n}{\varepsilon \cdot \Delta} \rceil$ and form the instance $I' = (G, c', \{r_1, \ldots, r_R\}, X)$. Note for any shortest dipath P starting at root r_i, we have

$$\mathrm{cost}_{c'}(P) \leq \sum_{e \in P} c'_e \leq \sum_{e \in P} (c_e \cdot \frac{n}{\varepsilon \cdot \Delta} + 1) \leq \mathrm{cost}_c(P) \cdot \frac{n}{\varepsilon \cdot \Delta} + n \leq \frac{n \cdot k}{\varepsilon} + n,$$

where the last inequality follows because all the distances from the root have length at most $k \cdot \Delta$. So the length of all shortest dipaths starting at r in I' are bounded by $O(\frac{n^2}{\varepsilon})$.

Let $\text{opt}_{I'}$ be the optimal value of instance I'. A similar calculation as before shows $\text{opt}_{I'} \leq \frac{n}{\varepsilon \cdot \Delta} \cdot \text{opt}_I + n$.

Let F be an α-approximate solution for I'. Then, we have

$$\begin{aligned}
\text{cost}_c(F) &\leq \frac{\varepsilon \cdot \Delta}{n} \cdot \text{cost}_{c'}(F) \\
&\leq \frac{\varepsilon \cdot \Delta}{n} \cdot \alpha \cdot \text{opt}_{I'} \\
&\leq \frac{\varepsilon \cdot \Delta}{n} \cdot \alpha \cdot (\frac{n}{\varepsilon \cdot \Delta} \cdot \text{opt}_I + n) \\
&\leq \alpha \cdot \text{opt}_I + \alpha \cdot \varepsilon \cdot \Delta \\
&\leq \alpha \cdot (1 + \varepsilon) \cdot \text{opt}_I,
\end{aligned}$$

where the first inequality follows because $c'_e \geq c_e \cdot \frac{n}{\varepsilon \cdot \Delta}$ and the last inequality holds because $\text{opt}_I \geq \Delta$. □

3.3 Planar DST

In this section we prove Theorem 10. Fix an instance $I = \big(G = (V, E), c, r, X\big)$ of DST on planar graphs that satisfies the assumptions in Lemma 15 for, say, $\varepsilon = 1/2$. Let $n := |V|$ and $k := |X|$. Furthermore, fix an optimal solution OPT for this instance and let opt denote its cost. So the distance of every vertex from r is at most $O(n \cdot k)$.

Our algorithm recursively constructs smaller subinstances based on a branching (as a separator) and disjoint subsets of vertices (as the weakly connected components after removing the separator). The following is a more formal definition of these subinstances.

Definition 16 (Induced subinstances)**.** *Let $I = (G = (V, E), c, r, X)$ be an instance of DST on planar graphs. Let T be a branching rooted at r, and let $C_1, \ldots, C_h$ be the weakly connected components of $G \setminus T$. The subinstances of DST induced by tuple $(G, T, C_1, \ldots, C_h)$ are defined as follows. Let G_{contract} be the graph obtained from G by contracting T into singleton vertex r. For each C_i where $1 \leq i \leq h$ we construct instance $I_{C_i} := \big(G_{C_i}, c, r, C_i \cap X\big)$ where $G_{C_i} := G_{\text{contract}}[C_i \cup \{r\}]$. See Figure 3.1.*

Given solutions $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_h$ for the subinstances induced by $(G, T, C_1, \ldots, C_h)$, one can naturally consider the corresponding subset of edges of $E(T) \cup \mathcal{F}_1 \cup \mathcal{F}_2 \cup \ldots \cup \mathcal{F}_h$ in G and it is easy to see this forms a feasible solution for instance I. We formalize this in the next lemma.

Lemma 17 (Merged solution)**.** *Consider the subinstances I_{C_i} for $1 \leq i \leq h$ as defined in Definition 16. Let $\mathcal{F}_{C_i}$ be a solution for I_{C_i}. Let $\mathcal{F} \subseteq E(G)$ be the corresponding edges of $E(T) \cup (\bigcup_{i=1}^{h} \mathcal{F}_{C_i})$ in G. Then, $\mathcal{F}$ is a feasible solution for instance I and furthermore* $\text{cost}(\mathcal{F}) = \text{cost}(T) + \sum_{i=1}^{h} \text{cost}(\mathcal{F}_{C_i})$. *See Figure 3.1 for an example.*

Proof. The furthermore part is obvious so we prove that $\mathcal{F}$ is feasible for I. Consider a terminal node $t \in C_i$. Since $\mathcal{F}_i$ is feasible for I_{C_i}, then there is a dipath P from r to t. Let (r, v) be the first edge on P and let (u, v) be the corresponding edge to (r, v) in $E(G)$. Then, we must have $u \in V(T)$ as $\delta(C_i, C_j) = \emptyset$ for all $1 \leq i \neq j \leq h$. So we can go from r to u in T, then take the edge (u, v) and then go from v to t in $\mathcal{F}_{C_i}$. Since all these edges are present in $\mathcal{F}$ and t is an arbitrary terminal, $\mathcal{F}$ is a feasible solution for I. □

We first present a high-level idea of a simple $O(\log k)$-approximation that runs in quasi-polynomial time and then with a little extra work, we can make it run in polynomial time with a small loss in the approximation guarantee.

3.3.1 Warm-up: An overview of a quasi-polynomial time approximation

The algorithm is simple. Fix an optimal solution OPT with cost opt. First guess opt. Note by Lemma 15, opt is polynomial in n and integral so there are polynomially many guesses. Then, we remove all vertices such that their distance from r is more than our guessed value (this is the preprocessing step). For the purpose of separating into subinstances with balanced weight, we let the weight of each terminal to be 1 and the rest of vertices have zero weight. Apply Corollary 14 and let P_1, P_2, and P_3 be the resulting shortest dipaths each starting at r. Note that $\text{cost}(P_i) \leq \text{opt}$ for $i = 1, 2, 3$ because of the

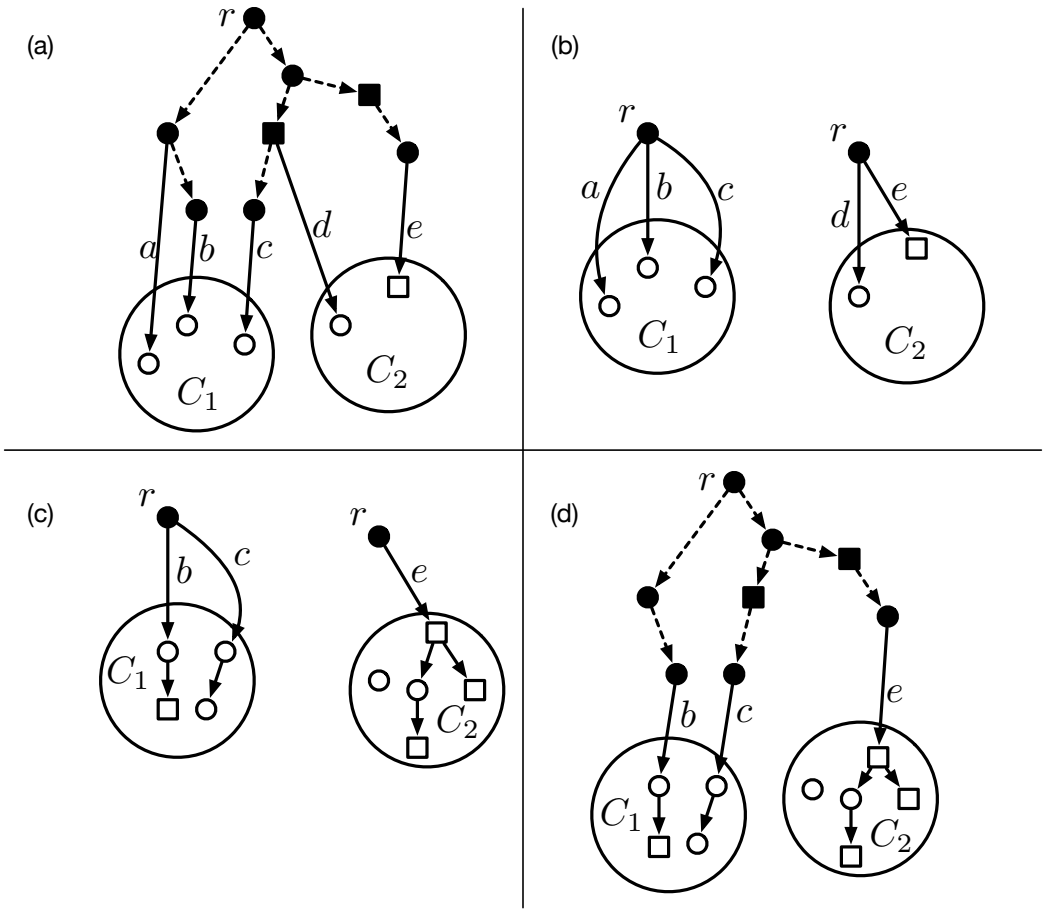


Figure 3.1: Throughout, squares are terminals and circles are Steiner nodes or the root node r. In (a) the separator is shown with dashed edges and solid vertices. The weakly connected components of $G \setminus T$ are shown as circles denoted by C_1 and C_2. Note that we did not show any edge directed from C_1 or C_2 into the separator because we can safely remove these edges. In (b) the subinstances I_{C_1} and I_{C_2} induced by (G, T, C_1, C_2) are depicted. In (c), the solutions for each subinstances are shown. Finally, (d) shows how to merge the solutions in (c) to get a solution for the original instance. Note that leaf nodes are not necessarily terminals. One could prune them as a post-processing step, but that is not required by our algorithm.

preprocessing step. Let $T := P_1 \cup P_2 \cup P_3$, then T is a branching rooted at r. Let C_i for $1 \leq i \leq h$ be the weakly connected components of $G \setminus T$. Note that $|T \cap C_i| \leq \frac{k}{2}$ for all i's. Then, we recursively solve the subinstances induced by $(G, T, C_1, \ldots, C_h)$ (see Definition 16), and finally return the corresponding solution of $E(T) \cup \bigcup_{i=1}^{h} \mathcal{F}_{C_i}$ in G. When the number of terminals in a subinstance becomes one, we can solve the problem exactly by finding the shortest dipath between the root and the only terminal.

Note that each recursive call reduces the number of terminals by half and each recursive call spawn up to k many subinstances. Furthermore, The guess work for each instance is polynomial in n. So the total number of recursive calls is bounded by $n^{O(\log k)}$. Since each time we apply the separator result on an instance I, we buy a branching (union of up to three dipaths) of cost at most $3 \cdot \text{opt}$, and since the *total* cost of optimal solutions across all of the resulting subinstances I_{C_i} is at most opt, a simple induction on the number of terminals shows the final cost is within $(3 \cdot \log k + 1) \cdot \text{opt}$. A slight improvement of the running time can be made by guessing opt within a constant factor (thus only making $O(\log n)$ guesses since all distances are integers bounded by a polynomial in n), but the size of the recursion tree would still be $O(\log n)^{O(\log k)}$ which is still not quite polynomial.

In the next section, we show how to avoid the above aggressive guessing which gives us the polynomial running time. We remark there are some similarities between our algorithm with the one presented in [28] for quasi-polynomial time algorithm for SUBMODULAR TREE ORIENTEERING in the sense that both need to guess some value (in our case opt and in their case the budget) for the subproblems and performing this guess naively is too slow. However, the approaches to overcoming this barrier are different.

3.3.2 The polynomial time algorithm

The idea here is similar to the quasi-polynomial time algorithm; however, instead of guessing the diameter of an optimal arborescence for each instance, we keep an estimate of it. Our recursive algorithm tries two different re-

cursive calls: (1) divide the current estimate by half and recur, or (2) buy a separator and divide the instance into smaller instances and recur on these instances using the current estimate as the current estimate passed to each smaller instance.

The rationale behind this idea is that if the estimate is close to the optimal value, then our separator is "cheap" compared to optimal value so (2) is "good progress" otherwise we make the estimate smaller so (1) is "good progress". The key idea here that leads to polynomial time is that we do not "reset" our guess for the optimal solution cost in each recursive call since we know that if our guess is correct for the current instance, then it is an upper bound for the optimal solution cost in each subinstance.

As we mentioned at the beginning, the algorithm is recursive. The input to the algorithm is a tuple $(I, \widetilde{\text{opt}})$ where $\widetilde{\text{opt}}$ is an estimate of opt. The algorithm computes two solutions and takes the better of the two. One solution is by a recursive call to $(I, \frac{\widetilde{\text{opt}}}{2})$ and the other one is obtained by applying Corollary 14 to get smaller subinstances, and solve each subinstance recursively, and merge the solutions as described in Lemma 17. See Algorithm 2 for the pseudocode.

By Lemma 15, we can assume the edge costs are positive integers; hence, $\text{opt} \geq 1$. So if $\widetilde{\text{opt}} < 1$, then the output of $\text{DST}(I, \widetilde{\text{opt}})$ is infeasible. The algorithm will terminate since each recursive call either halves $\widetilde{\text{opt}}$ or halves the number of terminals.

3.3.3 Analysis

In this section, we analyze the cost and the running time of Algorithm 2.

Lemma 18 (Cost and running time)**.** *Consider an instance* $I = \big(G = (V, E), c, r, X\big)$ *and a pair* $(I, \widetilde{\text{opt}})$*. Let* ℓ *and* $\mathcal{O}$ *be non-negative integers such that* $|X| \leq 2^{\ell}$ *and* $\widetilde{\text{opt}} \leq 2^{\mathcal{O}}$*. If* $\widetilde{\text{opt}} \geq \text{opt}$ *where* opt *is the optimal value of* I*, then* $\text{DST}(I, \widetilde{\text{opt}})$ *returns a solution with cost at most* $(6 \cdot \ell + 1) \cdot \text{opt}$*. Furthermore, the total number of recursive calls made by* $\text{DST}(I, \widetilde{\text{opt}})$ *and its subsequent recursive calls is at most* $|X| \cdot 2^{2 \cdot \ell + \mathcal{O}}$*.*

Algorithm 2 $\mathrm{DST}(I, \widetilde{\mathrm{opt}})$

Input: $I := \big(G = (V, E), c, r, X\big)$ and an estimate $\widetilde{\mathrm{opt}}$.

Output: A feasible solution for instance I or output infeasible.

if $\widetilde{\mathrm{opt}} < 1$ or $d(r,t) > \widetilde{\mathrm{opt}}$ for some terminal $t \in X$ **then**
  return `infeasible`
else if $|X| = 1$ **then**
  Let $\mathcal{F}$ be the shortest dipath from r to the only terminal in X.
else
  $\mathcal{F}_1 \leftarrow \mathrm{DST}(I, \frac{\widetilde{\mathrm{opt}}}{2})$, if $\mathcal{F}_1$ is `infeasible` solution then set $\mathrm{cost}(F_1) \leftarrow \infty$.
  Remove all vertices v with $d(r,v) > \widetilde{\mathrm{opt}}$. {This is the preprocessing step.}
  Apply Corollary 14 to obtain a branching T consists of up to 3 shortest dipaths starting at r. Let $C_1, \ldots, C_h$ be the weakly connected components of $G \setminus T$. Let I_{C_i} be the i-th subinstance induced by $(G, T, C_1, \ldots, C_h)$ for $i = 1, \ldots, h$.
  for $i = 1, \ldots, h$ **do**
    $\mathcal{F}'_i \leftarrow \mathrm{DST}(I_{C_i}, \widetilde{\mathrm{opt}})$
  $\mathcal{F}_2 \leftarrow E(T) \cup (\bigcup_{i=1}^{h} \mathcal{F}'_i)$, if any $\mathcal{F}'_i$ is `infeasible` then set $\mathrm{cost}(\mathcal{F}_2) \leftarrow \infty$.
  if both $\mathrm{cost}(\mathcal{F}_1)$ and $\mathrm{cost}(\mathcal{F}_2)$ are ∞ **then**
    return `infeasible`
  $\mathcal{F} \leftarrow \arg\min\{\mathrm{cost}(\mathcal{F}_1), \mathrm{cost}(\mathcal{F}_2)\}$
return $\mathcal{F}$.

Proof. First we analyze the cost of the output solution. If $\ell = 0$ then we solve I exactly so the statement holds. So for the rest of the proof we assume $\ell \geq 1$. We proceed by induction on $\ell + \mathcal{O} \geq 1$.

We assume $\widetilde{\text{opt}} \leq 2 \cdot \text{opt}$, otherwise we have $\text{cost}(\text{DST}(I, \widetilde{\text{opt}})) \leq \text{cost}(\text{DST}(I, \frac{\widetilde{\text{opt}}}{2})) \leq (6 \cdot \ell + 1) \cdot \text{opt}$ by induction where the last inequality holds because $\log \frac{\widetilde{\text{opt}}}{2} \leq \log(\widetilde{\text{opt}}) - 1$.

Let $\mathcal{F}$ be the solution returned by $\text{DST}(I, \widetilde{\text{opt}})$. Since $\text{cost}(\mathcal{F}) \leq \text{cost}(\mathcal{F}_2)$, it suffices to prove $\text{cost}(\mathcal{F}_2) \leq (6 \cdot \ell + 1) \cdot \text{opt}$. Let $\mathcal{F}'_i = \text{DST}(I_{C_i}, \widetilde{\text{opt}})$ for $i = 1, \ldots, h$ be the solutions constructed recursively for the subinstances. Note that each I_{C_i} for $i = 1 \ldots, h$ has at most $2^{\ell-1}$ terminals and $\text{opt}_{I_{C_i}} \leq \widetilde{\text{opt}}$ where $\text{opt}_{I_{C_i}}$ is the optimal value of I_{C_i}. By the induction hypothesis, we conclude

$$\text{cost}(\mathcal{F}'_i) \leq (6 \cdot (\ell - 1) + 1) \cdot \text{opt}_{I_{C_i}} \leq 6 \cdot \ell \cdot \text{opt}_{I_{C_i}}, \text{ for } i = 1, \ldots, h \tag{3.1}$$

Note that T is the union of up to three shortest dipaths and because of the preprocessing step, each shortest dipath starting at r has cost at most $\widetilde{\text{opt}} \leq 2 \cdot \text{opt}$. So the following holds:

$$\text{cost}(T) \leq 3 \cdot \widetilde{\text{opt}} \leq 6 \cdot \text{opt}\,. \tag{3.2}$$

Combining (3.1) and (3.2) we get:

$$\begin{aligned}
\text{cost}(\mathcal{F}) &= \text{cost}(T) + \sum_{i=1}^{h} \text{cost}(\mathcal{F}'_i) \\
&\leq \text{cost}(T) + \sum_{i=1}^{h} 6 \cdot \ell \cdot \text{opt}_{I_{C_i}} \\
&\leq 6 \cdot \text{opt} + 6 \cdot \ell \cdot \sum_{i=1}^{h} \text{opt}_{I_{C_i}} \\
&\leq 6 \cdot \text{opt} + 6 \cdot \ell \cdot \text{opt} \\
&= (6 \cdot \ell + 1) \cdot \text{opt},
\end{aligned}$$

where the first equality follows from Lemma 17, the first and the second inequalities follow from (3.1) and (3.2), respectively, and finally the last inequal-

ity follows from the fact that $\sum_{i=1}^{h} \mathrm{opt}_{I_{C_i}} \leq \mathrm{opt}$ as the restriction of OPT on each G_{C_i} is a feasible solution for I_{C_i} and G_{C_i}'s are edge-disjoint.

Next, we analyze the number of recursive calls $R(\ell, \mathcal{O})$ in $\mathrm{DST}(I, \widetilde{\mathrm{opt}})$. We prove by induction on $\ell + \mathcal{O}$ that $R(\ell, \mathcal{O}) \leq |X| \cdot 2^{2 \cdot \ell + \mathcal{O}}$. If $\ell = 0$, then there is no recursive call. So suppose $\ell \geq 1$. Let $X_i := |X \cap C_i| \leq \frac{|X|}{2}$ be the number of terminals in subinstance I_{C_i} and let ℓ_i be the smallest integer where $|X_i| \leq 2^{\ell_i}$. Since the number of terminals in the subinstances are halved, we have $\ell_i \leq \ell - 1$ for all $1 \leq i \leq h$. So we can write

$$\begin{aligned}
R(\ell, \mathcal{O}) &= 1 + R(\ell, \mathcal{O} - 1) + \sum_{i=1}^{h} R(\ell_i, \mathcal{O}) \\
&\leq 1 + |X| \cdot 2^{2 \cdot \ell + \mathcal{O} - 1} + \sum_{i=1}^{h} |X_i| \cdot 2^{2 \cdot \ell_i + \mathcal{O}} \\
&\leq 1 + |X| \cdot 2^{2 \cdot \ell + \mathcal{O} - 1} + 2^{2(\ell - 1) + \mathcal{O}} \cdot \sum_{i=1}^{h} |X_i| \\
&\leq 1 + |X| \cdot 2^{2 \cdot \ell + \mathcal{O} - 1} + 2^{2 \cdot \ell + \mathcal{O} - 2} \cdot |X| \\
&= 1 + |X| \cdot 2^{2 \cdot \ell + \mathcal{O} - 1} + (2^{2 \cdot \ell + \mathcal{O} - 1} - 2^{2 \cdot \ell + \mathcal{O} - 2}) \cdot |X| \\
&= 1 + |X| \cdot 2^{2 \cdot \ell + \mathcal{O}} - |X| \cdot 2^{2 \cdot \ell + \mathcal{O} - 2} \\
&\leq |X| \cdot 2^{2 \cdot \ell + \mathcal{O}},
\end{aligned}$$

where the first inequality follows from the induction hypothesis, the second inequality comes from the fact that $\ell_i \leq \ell - 1$, the third inequality holds because $\sum_{i=1}^{h} |X_i| \leq |X|$, and the last inequality follows from the fact that $|X| \geq 1$ and $\ell \geq 1$. □

Proof of Theorem 10. By Lemma 15, we can assume all the shortest dipaths starting at the root are bounded by $\mathrm{poly}(n, \varepsilon)$ by losing a $(1 + \varepsilon)$ multiplicative factor in the approximation guarantee for any $\varepsilon > 0$. So we assume properties of Lemma 15 holds for the rest of the proof.

Let Δ be the maximum distance from the root to any terminal. Let $\widetilde{\mathrm{opt}} := k \cdot \Delta \leq \mathrm{poly}(n)$. We find a solution by calling $\mathrm{DST}(I, \widetilde{\mathrm{opt}})$. Applying Lemma 18 with $\widetilde{\mathrm{opt}} := k \cdot \Delta$, $\ell := \lceil \log k \rceil \leq \log k + 1$ and $\mathcal{O} := \lceil \log \widetilde{\mathrm{opt}} \rceil$ guarantees the solution has cost at most $(6 \cdot (\log k + 1) + 1) \cdot \mathrm{opt}$.

For the running time of Algorithm 2, we have by Lemma 18 that the number of recursive calls is at most $k \cdot 2^{2\cdot\ell+\mathcal{O}} = O(k^4 \cdot \Delta)$. So the total number of recursive calls is $\text{poly}(n)$ (recall $k \cdot \Delta = \text{poly}(n)$). The running time within each recursive call is also bounded by $\text{poly}(n)$ so the algorithm runs in polynomial time. □

3.4 Multi-rooted planar DST

The algorithm for the multi-rooted case is similar to Algorithm 2. Recall, the goal is to find a minimum cost $F \subseteq E$ such that every terminal is reachable from one of the roots in F. Note we are not requiring that all the roots to be in F.

To design an algorithm for multi-rooted instances, we need analogous versions of the separator, how we define the subinstances, and how we merge the solutions of smaller subinstances to get a solution for the original instance.

We start by a generalization of branching in the single rooted case to multiple roots.

Definition 19 (Multi-rooted branching)**.** *Given a digraph $G = (V, E)$, R vertices $r_1, \ldots, r_R$ designated as roots. We say a subgraph T of G is a multi-rooted branching if it satisfies the following properties:*

1. *There are vertex-disjoint branchings $T_{i_1}, \ldots, T_{i_q}$ rooted at $r_{i_1}, \ldots, r_{i_q}$, respectively, and a subset of edges $F \subseteq E \setminus \left(\bigcup_{j=1}^{q} E(T_{i_j})\right)$, where the endpoints of each edge in F belong to $\bigcup_{j=1}^{q} V(T_{i_j})$, such that $T = F \cup (\bigcup_{j=1}^{q} T_{i_q})$.*

2. *T is weakly connected and has no cycle (in the undirected sense).*

If a multi-rooted branching T covers all the vertices in G, then we say T is a multi-rooted arborescence for G. See Figure 3.2 for an example.

Fix an instance $I = (G, c, \{r_1, \ldots, r_R\}, X)$ of MR-DST on planar graphs. Next, we present subinstances induced by a multi-rooted branching and

bunch of disjoint subsets analogous to Definition 16. Intuitively, given a multi-rooted branching T, and each weakly connected component C of $G \setminus T$, we create a subinstance I_C as follows: contract T into a vertex r_T and remove all vertices except r_T and vertices in C. The edge cost is the restriction of the original edge cost to this graph, the roots are r_T and all the roots in C, and the terminals are the original terminal nodes in C. We denote this subinstance by $I_C := \left(G_C, c, \{r_T\} \cup \left(C \cap \left(\{r_1, \ldots, r_R\} \setminus \{r_{p_1}, \ldots, r_{p_q}\}\right)\right), C \cap X\right)$ where $G_C := G_{\text{contract}}[C \cup \{r_T\}]$.

Definition 20 (Induced subinstances, multi-rooted)**.** *Let $T = F \cup (\bigcup_{j=1}^{q} T_{p_j})$ be a multi-rooted branching in G where T_{p_j} is a branching rooted at r_{p_j} for $1 \leq j \leq q$. In addition, let $C_1, \ldots, C_h$ be the weakly connected components of $G \setminus T$. The subinstances of multi-rooted DST induced by tuple $(G, T, C_1, \ldots, C_h)$ are defined as follows: let G_{contract} be the graph obtained from G by contracting T into a singleton vertex called r_T. For each C_i where $1 \leq i \leq h$ we construct instance I_{C_i} as explained before.*

The following is analogous to Lemma 17 for merging solution in the multi-rooted case.

Lemma 21 (Merged solutions, multi-rooted)**.** *Let $T = F \cup (\bigcup_{j=1}^{q} T_{p_j})$ be a multi-rooted branching in G. Consider the subinstances I_{C_i} for $1 \leq i \leq h$ as defined in Definition 20, and let $\mathcal{F}_{C_i}$ be a solution for I_{C_i}. Let $\mathcal{F} \subseteq E(G)$ be the corresponding edges in $(E(T) \setminus F) \cup (\bigcup_{i=1}^{h} \mathcal{F}_{C_i})$. Then, $\mathcal{F}$ is a feasible solution for instance I and furthermore* $\text{cost}(\mathcal{F}) = \text{cost}(T \setminus F) + \sum_{i=1}^{h} \text{cost}(\mathcal{F}_{C_i})$.

Proof. The furthermore part follows directly from the definition of $\mathcal{F}$. We prove $\mathcal{F}$ is feasible for I.

Consider a terminal t. If $t \in V(T)$, then $t \in V(T_{p_j})$ for some $1 \leq j \leq q$ (recall the vertices in T is the union of the vertices in all the branchings T_{p_j}'s) so t is reachable from r_{p_j}, the root of T_{p_j}, in $\mathcal{F}$. Suppose $t \in C_i$ for some $1 \leq i \leq h$. If t is reachable from a root other than r_T in $\mathcal{F}_{C_i}$ then we are done because the same dipath exists in $\mathcal{F}$. So we suppose not and let P be

the dipath in $\mathcal{F}_{C_i}$ from r_T to t. Let (u, v) be the corresponding edge to (r_T, v) in G. Note that $u \in V(T_{p_j})$ for some $1 \leq j \leq q$ because $\delta(C_s, C_{s'}) = \emptyset$ for $1 \leq s \neq s' \leq h$. Hence, t is reachable from r_{p_j}, the root of T_{p_j}, in $\mathcal{F}$ as $E(T_{p_j}) \subseteq \mathcal{F}$. □

Finally, we present our separator for the multi-rooted case.

Lemma 22 (A structured separator, multi-rooted)**.** *Let $A_1, \ldots, A_R$ be a vertex-disjoint branchings rooted at $r_1, \ldots, r_R$ in G, respectively. There is a multi-rooted branching $T = F \cup (\bigcup_{i=1}^{R} T_i)$, where T_i could possibly be empty (i.e., with no vertices) such that the following hold:*

(a) T_i is either empty or is a subtree of A_i rooted at r_i that consists of the union of up to four dipaths each starting at r_i.

(b) Let $C_1, \cdots, C_h$ be the weakly connected components of $G \setminus T$. Then, each subinstance I_{C_i} induced by $(G, T, C_1, \ldots, C_h)$ has at most $\frac{|X|}{2}$ terminals for $1 \leq i \leq h$.

(c) Let $\mathcal{F}_i$ be a solution to subinstance I_{C_i} for $1 \leq i \leq h$. Then, the corresponding solution $(E(T) \setminus F) \cup (\bigcup_{i=1}^{h} \mathcal{F}_i)$ in G is feasible for I with cost exactly $\text{cost}(T \setminus F) + \sum_{i=1}^{h} \text{cost}(\mathcal{F}_i)$.

Proof. Figure 3.2 helps to visualize this proof.

Since G is weakly connected, there is a subset of edges F' in G such that $T' := F' \cup (\bigcup_{i=1}^{R} A_i)$ is a multi-rooted arborescence of G (spans all the vertices) and the endpoints of edges in F are in $\bigcup_{i=1}^{R} V(A_i)$. Make T' rooted at an arbitrarily chosen root, say r_1. Apply Theorem 13 with terminal vertices having weight 1 and the rest of vertices having weight 0, and T' as the spanning tree (in the undirected sense). This gives three paths P_1, P_2, and P_3 in T' each with starting vertex r_1 such that every weakly connected component C_i of $G \setminus (P_1 \cup P_2 \cup P_3)$ has at most $\frac{|X|}{2}$ terminals for $1 \leq i \leq h$. Note, these three paths do not necessarily follow the directions of the edges.

Fix A_i for some $1 \leq i \leq R$ and a path $P_j := (r_1 = v_1), v_2, \ldots, v_N$ for $1 \leq j \leq 3$. Let a and b (possibly $a = b$) be the smallest and the largest indices, respectively, such that v_a and v_b are in $V(A_i)$. We claim the subpath $P_{[a,b]} := v_a, v_{a+1}, \ldots, v_b$ is a subgraph of A_i. Suppose not, so there must be two indices $a \leq a' < b' \leq b$ such that $v_{a'}, v_{b'} \in V(A_i)$ and $v_{a'+1}, v_{a'+2}, \ldots, v_{b'-1} \notin V(A_i)$. Let $P^{a'}_{A_i}$ and $P^{b'}_{A_i}$ be the paths from r_i to a' and b' in $V(A_i)$, respectively. So $P^{a'}_{A_i} \cup P^{b'}_{A_i} \cup P_{[a',b']}$ forms a cycle in T', a contradiction. Furthermore, for $j = 1, 2, 3$ let v_j be the closest vertex to r_1 on P_j (in terms of edge hops) that is in A_i as well (if exists). Then, $v_1 = v_2 = v_3$ as otherwise we have a cycle in T' because all P_j's start at r_1.

For each $1 \leq i \leq R$ and $1 \leq j \leq 3$, we mark the nodes with smallest and largest indices in P_j that are in A_i. We proved above, that the number of these marked vertices in each A_i is at most 4. Furthermore, $(P_1 \cup P_2 \cup P_3) \cap A_i$ is a subgraph of the union of dipaths from r_i to each marked vertices in A_i for all $1 \leq i \leq h$.

We construct our multi-rooted branching T as follows: let T_i be the union of (up to four) shortest dipaths from r_i to the marked vertices in A_i. Let $F := E\big(P_1 \cup P_2 \cup P_3\big) \setminus \big(\bigcup_{i=1}^{R} E(T_i)\big)$ which is the subset of edges whose endpoints are in different $V(A_i)$'s, i.e., $F \subseteq F'$. Let $T := F \cup \big(\bigcup_{i=1}^{R} T_i\big)$. Note that for A_i's with no marked vertices, T_i is empty (with no vertices not even r_i). Since T is a multi-rooted branching that contains $P_1 \cup P_2 \cup P_3$ as a subgraph, every weakly connected components of $G \setminus T$ has at most $\frac{|X|}{2}$ terminals. This finishes the proof of parts (a) and (b).

Property (c) follows from Lemma 21 and the fact that the conditions in Lemma 21 are satisfied. □

In the statement of Lemma 22 we did not specify what are the arborescences A_i's rooted at r_i. We construct two different such arborescences which one proves Theorem 11 and the other proves Theorem 12.

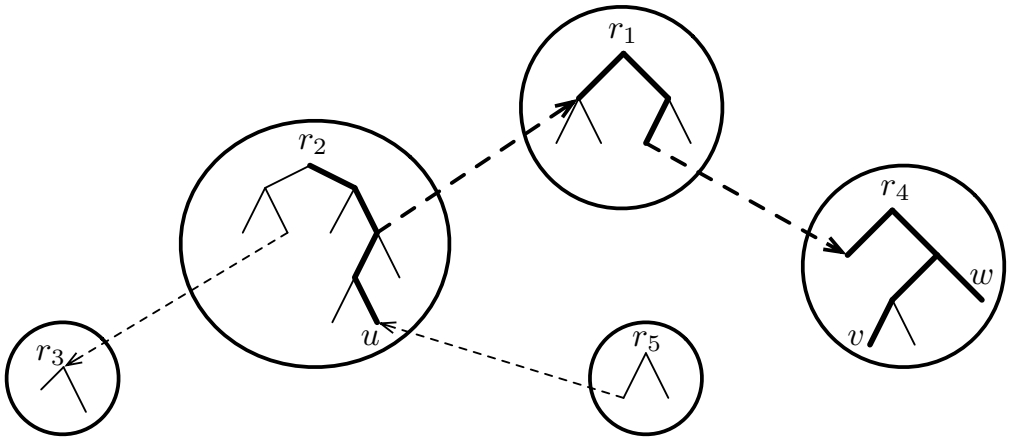


Figure 3.2: A depiction of the multi-rooted separator in an instance with $R = 5$ roots. The solid edges (thick and thin) are the branchings A_i for $i = 1, \ldots, R$. The dashed edges are F'. After applying Theorem 13 to this tree (in the undirected sense), we get three vertices depicted as u, v, w. The marked vertices, as in the proof of Lemma 22, are u, v, w and the endpoints of thick dashed edges. Then, the separator constructed in the lemma contains dipaths from the root node to the marked vertices in each A_i plus the thick dashed edges.

3.4.1 Proof of Theorem 11

In order to prove Theorem 11, we utilizes our separator result, Lemma 22. However, in order to do that we need to specify how to construct the arborescences A_i's in the statement of the lemma. We do this in the following way.

Given an instance I with roots $r_1, \ldots, r_R$, temporarily add an auxiliary node r and add edges (r, r_i) for all $1 \leq i \leq R$ with zero cost (it might destroy the planarity). Run a shortest path algorithm as usual rooted at r. Then, remove r and all the edges incident to r. The result is a vertex-disjoint BFS branching $A_1, A_2, \ldots, A_R$ rooted at $r_1, \ldots, r_R$. Note that for every $v \in V(A_i)$, v is closest to r_i than any other roots, i.e., the dipath from r_i to v has cost $d\big(\{r_1, \ldots, r_R\}, v\big)$.

The algorithm for the multi-rooted version is the same as Algorithm 2 with the following two tweaks: (1) in the preprocessing step we remove all the vertices v where $d\big(\{r_1, \ldots, r_R\}, v\big) > \widetilde{\mathrm{opt}}$, and (2) instead of Corollary 14 we apply Lemma 22 with above constructed A_i's to obtain the subinstances.

Next, we analyze the cost and the running time of this algorithm.

Lemma 23 (Cost and running time, multi-rooted)**.** *Consider an instance* $I = \big(G = (V, E), w, \{r_1, \ldots, r_R\}, X\big)$ *and a pair* $(I, \widetilde{\text{opt}})$*. Let* ℓ *and* $\mathcal{O}$ *be non-negative integers such that* $|X| \leq 2^{\ell}$ *and* $\widetilde{\text{opt}} \leq 2^{\mathcal{O}}$*. If* $\widetilde{\text{opt}} \geq \text{opt}$ *where* opt *is the optimal value of* I*, then* $\text{DST}(I, \widetilde{\text{opt}})$ *returns a solution of cost at most* $\big(8 \cdot (R + \ell) + 1\big) \cdot \text{opt}$ *and the number of recursive calls is at most* $|X| \cdot 2^{2 \cdot \ell + \mathcal{O}}$*.*

Proof. The proof of the number of recursive calls is exactly the same as in the proof of Lemma 18. So we turn to proving the bound on the cost of the returned solution.

The proof is by induction on $R + \ell + \mathcal{O}$. As in the proof of Lemma 18, we only need to focus on the case that $\widetilde{\text{opt}} \leq 2 \cdot \text{opt}$ and show that $\text{cost}(\mathcal{F}_2) \leq \big(8 \cdot (R + \ell) + 1\big) \cdot \text{opt}$.

Let $T = F \cup (\bigcup_{i=1}^{R} T_i)$ be the separator obtained from Lemma 22. Suppose T contains R' many of the roots. Then, exactly R' many of T_i's are non-empty. By Lemma 22 (a) and the fact that A_i's are BFS branchings, we have that each non-empty T_i is consists of up to four shortest dipaths rooted at r_i so $\text{cost}(T_i) \leq 4 \cdot \widetilde{\text{opt}}$ because of the preprocessing step plus the fact that $\widetilde{\text{opt}} \leq 2 \cdot \text{opt}$, we conclude

$$\text{cost}(T \setminus F) \leq 8 \cdot R' \cdot \text{opt}\,. \tag{3.3}$$

Since T contains R' many roots, each subinstance I_{C_i} induced by $(G, T, C_1, \ldots, C_h)$ has at most $R - R' + 1$ many roots for $1 \leq i \leq h$. Furthermore, by Lemma 22 (b) each I_{C_i}'s has at most $\frac{|X|}{2} \leq 2^{\ell - 1}$ many terminals. So by induction hypothesis, for $i = 1, \ldots, h$ we have

$$\text{cost}(\mathcal{F}_{C_i}) \leq \Big(8 \cdot \big((R - R' + 1) + \ell - 1\big) + 1\Big) \cdot \text{opt}_{I_{C_i}} \leq \big(8 \cdot (R - R' + \ell) + 1\big) \cdot \text{opt}_{I_{C_i}}\,. \tag{3.4}$$

Using Lemma 22 (c), the bounds in (3.3) and (3.4) we have

$$\begin{aligned}
\text{cost}(\mathcal{F}) &\leq \text{cost}(T \setminus F) + \sum_{i=1}^{h} \text{cost}(\mathcal{F}_{I_{C_i}}) \\
&\leq 8 \cdot R' \cdot \text{opt} + \big(8 \cdot (R - R' + \ell) + 1\big) \cdot \sum_{i=1}^{h} \text{opt}_{I_{C_i}} \\
&\leq 8 \cdot R' \cdot \text{opt} + \big(8 \cdot (R - R' + \ell) + 1\big) \cdot \text{opt} \\
&= \big(8 \cdot (R + \ell) + 1\big) \cdot \text{opt},
\end{aligned}$$

where the third inequality follows from the fact that $\sum_{i=1}^{h} \text{opt}_{I_{C_i}} \leq \text{opt}$ as the restriction of OPT on each G_{C_i} is a feasible solution for I_{C_i} and G_{C_i}'s are edge-disjoint.. □

Proof of Theorem 11. Note both of the tweaks in Algorithm 2 are implementable in polynomial time. The proof has exactly the same structure as in the proof of Theorem 10 with the difference that we use Lemma 23 here instead of Lemma 18. □

3.4.2 Proof of Theorem 12

The proof is similar to the proof of Theorem 11. First we need to specify how do we construct the branching A_i's in the statement of Lemma 22.

Fix an optimal solution OPT with cost opt. By Lemma 15, we have opt $\leq 2 \cdot n^4$ (substituting $\frac{1}{n}$ for ε). For each $1 \leq i \leq R$, let A_i^* be the branching rooted at r_i in OPT. Note A_i^* could consists of only r_i. Let B_i be our guess for the budget (cost) of A_i^* for each $1 \leq i \leq R$. Note for each B_i, there are at most $2 \cdot n^4$ possibilities. We construct the required branching A_i's for Lemma 22 as follows.

Preprocessing step: given budgets $B_1, \ldots, B_R$, we only keep vertices that are reachable from r_i within distance B_i for some $1 \leq i \leq R$ and remove the vertex otherwise. With abuse of notation we denote the resulting graph by $G = (V, E)$ again.

Next, we assign each vertex in V (after preprocessing step) to a unique root node.

$$\Phi : V \setminus \{r_1, \ldots, r_R\} \to \{1, \ldots, R\}$$
$$\Phi(v) := \arg\max_{1 \le i \le R} \{B_i - d(r_i, v)\},$$

in the case of tie, Φ chooses the lower index. We define $\Phi^{-1}(i) := \{v \in V : \Phi(v) = i\}$. The important property of this mapping is as follows.

Lemma 24. *For each $1 \le i \le R$, there is a BFS branching rooted at r_i in $G[\{r_i\} \cup \Phi^{-1}(i)]$ where the metric on $G[\{r_i\} \cup \Phi^{-1}(i)]$ is based on the restriction of edge cost c to edges in $E[\{r_i\} \cup \Phi^{-1}(i)]$.*

Proof. It suffices to prove if $\Phi(v) = i$ then every internal node of any shortest dipath from r_i to v is mapped to i under Φ as well. Let P be a shortest dipath from r_i to v. Consider an arbitrary internal vertex u on P. Note that $P_{[r_i,u]}$ is a shortest dipath from r_i to u and similarly $P_{[u,v]}$ is a shortest dipath from u to v. Since $\Phi(v) = i$, for any $1 \le j \ne i \le R$ we can write

$$B_j - d(r_j, u) - d(u, v) \le B_j - d(r_j, v) \le B_i - d(r_i, v) = B_i - d(r_i, u) - d(u, v)$$
$$\Rightarrow B_j - d(r_j, u) \le B_i - d(r_i, v),$$

where the first inequality holds because of triangle inequality and the second inequality holds because $\Phi(v) = i$. Note that if the inequalities above hold with equality then it must be that $i < j$ since $\Phi(v) = i$. Therefore, $\Phi(u) = i$.

Therefore, a shortest dipath from r_i to a vertex in $\Phi^{-1}(i)$ lies completely in $G[\{r_i\} \cup \Phi^{-1}(i)]$, as desired. □

Now we are ready to construct the branchings A_i's to use in Lemma 22. For each i, let A_i be the BFS branching in $G[\{r_i\} \cup \Phi^{-1}(i)]$ whose existence is guaranteed by Lemma 24.

We assume after preprocessing step above, the graph is weakly connected, otherwise the algorithm below can be applied to each weakly connected components separately.

Here we present the idea behind the algorithm and why it proves Theorem 12. For each i, guess B_i (i.e., the cost of A_i^* in OPT). Assume, we guessed B_i's correctly. Then, we construct the branchings A_i's based on B_i's as described above and apply Lemma 22 to get smaller subinstances (i.e., the number of terminals are halved in each subinstance). However, note that $\sum_i B_i = \text{opt}$ for the correct guess; hence the cost of the separator is $O(\sum_{i=1}^{R} B_i) = O(\text{opt})$. So we paid $O(\text{opt})$ and reduce the instance to smaller subinstances. This happens $O(\log k)$ many times and hence the $O(\log k)$-approximation. It is easy to show that the running time is $n^{O(R \cdot \log k)}$. See Algorithm 3 for a pseudocode.

Algorithm 3 $MR - \text{DST}(I)$

Input: An instance of $MR - \text{DST}$ $I := \big(G = (V, E), c, r_1, \ldots, r_R, X\big)$.
Output: A feasible solution $\mathcal{F}$ for instance I or output infeasible.

if $|X| = 1$ **then**
  Let $\mathcal{F}$ be the shortest dipath from $\{r_1 \ldots, r_R\}$ to the only terminal in X.
for all R-tuple $(B_1, \ldots, B_R)$ where $B_i \in [0, 2 \cdot n^4]$ is an integer **do**
  $\mathcal{F}_{B_1,\ldots,B_R} \leftarrow \emptyset$. $\{\mathcal{F}_{B_1,\ldots,B_R}$ will be the solution for I given the guess B_i's as the cost of A_i^*'s in OPT.$\}$
  if a terminal node is removed in the preprocessing step **then**
    $\text{cost}(\mathcal{F}_{B_1,\ldots,B_R}) \leftarrow \infty$.
  else
    Apply Lemma 22 with arborescences A_i's constructed using B_i's as described above to obtain subinstances I_{C_i}'s.
    For all subinstances I_{C_i} let $\mathcal{F}_i' \leftarrow MR - \text{DST}(I_{C_i})$.
    if for any subinstance I_{C_i}, $\mathcal{F}_i'$ is infeasible **then**
      $\text{cost}(\mathcal{F}_{B_1\ldots,B_R}) \leftarrow \infty$.
    else
      Let $\mathcal{F}_{B_1\ldots,B_R}$ be the solution for I obtained by combining the solutions of the subinstances and the separator according to Lemma 22 part (c).
if for all R-tuple $(B_1, \ldots, B_R)$, $\text{cost}(\mathcal{F}_{B_1,\ldots,B_R})$ is ∞ **then**
  return `infeasible`.
else
  $\mathcal{F} \leftarrow \arg\min_{(B_1,\ldots,B_R)} \{\text{cost}(\mathcal{F}_{B_1,\ldots,B_R})\}$
  return $\mathcal{F}$.

We analyze Algorithm 3 which in turn proves Theorem 12.

Proof of Theorem 12. First we show the guaranteed cost factor mentioned in the theorem.

Fix OPT and its cost opt. Since we take the minimum cost solution over all possibilities of the R-tuples $B_1, \ldots, B_R$, in the following we assume B_i is exactly the cost of the branching A_i^* rooted at r_i in OPT. Note that $\sum_{i=1}^{R} B_i = \text{opt}$. In order to bound the cost of $\mathcal{F}$ output by Algorithm 3, we need to bound the cost of the separator constructed in Lemma 22. Recall, each A_i is a branching rooted at r_i whose diameter is at most B_i. Also part (a) of the lemma says the separator consists of up to four dipath rooted at r_i in A_i for all $1 \leq i \leq R$. So overall, the cost of the separator is at most $4 \cdot \sum_{i=1}^{R} B_i = 4 \cdot \text{opt}$. Let $\mathcal{F}_i'$ be the solution for the subinstance I_{C_i}.

We prove $\text{cost}(\mathcal{F}) \leq 4 \cdot \text{opt}$ by induction on k, the number of terminals. Note that for $k = 1$ we solve this instance exactly so the base case holds.

$$\begin{aligned}
\text{cost}(\mathcal{F}) &\leq 4 \cdot \text{opt} + \sum_i \text{cost}(\mathcal{F}_i') \\
&\leq 4 \cdot \text{opt} + 4 \cdot \sum_i \log \frac{k}{2} \cdot \text{opt}_{I_{C_i}} \\
&\leq 4 \cdot \text{opt} + 4 \cdot (\log k - 1) \cdot \text{opt} \\
&= 4 \cdot \log k \cdot \text{opt},
\end{aligned}$$

where the first inequality follows from part (c) in Lemma 22, the second inequality holds because the number of terminals is halved in each subinstance together with the induction hypothesis, and finally the last inequality follows from the fact that the subinstances I_{C_i}'s are disjoint so $\sum_i \text{opt}_{I_{C_i}} \leq \text{opt}$.

Next, we bound the running time of the algorithm. Let ℓ be an integer such that $|X| \leq 2^\ell$ and let $f(\ell)$ be the number of recursive calls in the algorithm when the number of terminals is at most 2^ℓ, the number of vertices is at most n, and the number of roots is at most R. By induction on ℓ, we prove $f(\ell) \leq n^{8 \cdot R \cdot \ell}$. If $\ell = 0$ so when we have one terminal, then we solve the problem exactly with no recursive call. So the base case holds. Since we have

at least two terminals, we assume $n \geq 2$ in the following.

$$\begin{aligned} f(\ell) &\leq 1 + \sum_{R-tuples\ (B_1,\ldots,B_R)} \sum_i f(\ell - 1) \\ &\leq 1 + n^{6\cdot R} \cdot n \cdot n^{8\cdot R\cdot \ell - 8\cdot R} \\ &= 1 + n^{8\cdot R\cdot \ell + 1 - 2\cdot R} \\ &\leq 1 + n^{8\cdot R\cdot \ell - R} \\ &\leq n^{8\cdot R\cdot \ell}, \end{aligned}$$

where the second sum in the first expression is over all the subinstance created given a tuple $(B_1, \ldots, B_R)$ which is at most n. The second inequality follows from the fact that the number of R-tuples is at most $(1 + 2 \cdot n^4)^R \leq n^{6\cdot R}$ for $n \geq 2$ and the induction hypothesis. The third inequality follows because $R \geq 1$, and the last inequality holds because $n \geq 2$.

Setting $\ell := \lceil k \rceil$ and the fact the running time within each recursive is bounded by polynomial in n, we have the desired running time of $n^{O(R\cdot \log k)}$.

□

We remark the same idea mentioned in the warm-up section to reduce the running time of the proposed algorithm slightly, works here as well by losing a bit in the approximation factor. More precisely, we can round up the cost of each branching in OPT to the closest power of $1 + \varepsilon$ for any $\varepsilon > 0$. Then, the number of R-tuples to check is at most $O(\log^R_{1+\varepsilon} n)$. So the running time will be $(\log n)^{O(R\cdot \log k)}$, and the approximation factor will be at most $4(1 + \varepsilon) \cdot (\log k + 1)$.

Chapter 4

Planar Quasi-bipartite Directed Steiner Tree

In the last chapter we saw an $O(\log k)$-approximation for DST on planar graphs. In this chapter, we further restrict ourselves to quasi-bipartite instances of DST on planar graphs and more generally graphs excluding a fixed minor, and show a constant factor approximation for these instances.

4.1 Introduction and outline

Another well-studied special case of UNDIRECTED STEINER TREE, besides planar instances, is *quasi-bipartite* 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 [60] in order to study the bidirected-cut relaxation of the UNDIRECTED STEINER TREE problem: this is exactly (**Primal-LP**) where we regard both directions of an undirected edge as separate entities. Feldmann et al. [19] studied UNDIRECTED 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 [7]. Currently, the best approximation in quasi-bipartite instances of UNDIRECTED STEINER TREE is $\frac{73}{60}$-approximation [30].

Naturally, researchers have considered quasi-bipartite instances of DST. Hibi and Fujito [36] presented an $O(\log |X|)$-approximation algorithm for

this case. Assuming $P \neq NP$, this result asymptotically matches the lower bound $(1 - o(1)) \cdot \ln |X|$; this lower bound comes from the hardness of SET COVER [15], [18] and the fact that quasi-bipartite DST generalizes the SET COVER problem. Friggstad, Könemann, and Shadravan [22] showed 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 LP up to a constant.

More recently, Chan et al. [10] 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.

For general instances of DST, methods based on linear programming have been less successful. Zosin and Khuller [73] 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 [51] provided an example showing the integrality gap of this LP is at least polynomial in n. On the positive side, [63] 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 LP with integrality gap $O(\ell \cdot \log |X|)$. This was extended to the LP-based Sherali-Adams and Lovász-Schrijver hierarchies by [21].

We consider the cut-based relaxation (**Primal-LP**) for DST, which is equivalent to the flow-based relaxation considered in [51], [73]; the flow-based relaxation is an extended formulation of (**Primal-LP**). Let $\delta^{in}(S)$ be the set of

directed edges entering a set $S \subseteq V$,

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

Note that if $|X| = 1$ (the s,t-SHORTEST PATH problem) or $X \cup \{r\} = V$ (the ARBORESCENCE problem), the extreme points of (**Primal-LP**) are integral, see [57] and [16] respectively.

Primal-Dual techniques for Steiner tree problems

Consider the NODE-WEIGHTED STEINER TREE (NWST) problem which is similar to undirected UNDIRECTED STEINER TREE except the cost is on the Steiner vertices instead of edges and can also be viewed as a special case of DST. Guha et al. [34] presented a primal-dual algorithm with approximation guarantee of $O(\ln n)$ which is asymptotically tight since NWST also generalizes set cover. Könemann, Sadeghian, and Sanità [44] gave an $O(\log n)$-approximation via primal-dual framework for a generalization of NWST called NODE-WEIGHTED PRIZE COLLECTING STEINER TREE[1].

Demaine, Hajiaghayi, and Klein [14] 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 [31] (see also 2.3) they showed a 6-approximation and further they extended their result to minor-free graphs. Later Moldenhauer [56] 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 [9] for quasi-bipartite instances of UNDIRECTED STEINER TREE. They introduced a new "simplex-embedding" LP relaxation and their primal-dual scheme raises dual variables with different rates. It is worth noting that although they obtain upper bound for the

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

integrality gap of the so-called *bidirected-cut relaxation* (BCR) of quasi-bipartite instances of UNDIRECTED STEINER TREE, the algorithm and the simplex-embedding LP relaxation itself are valid only in the undirected setting.

Our contributions

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, see Section 2.3) 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 25. *There is an $O(r \cdot \sqrt{\log r})$-approximation algorithm for quasi-bipartite DST on 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 such instances of DST.*

Remark 26. *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 27. *There is a* 20*-approximation algorithm for quasi-bipartite DST on planar graphs. Moreover, the algorithm gives an upper bound of* 20 *on the integrality gap of* (**Primal-LP**) *for such instances of DST.*

We also verify that UNDIRECTED 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 28. UNDIRECTED 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 quasi-bipartite DST on planar graphs is *NP*-hard as well.

The outline

In Section 4.2, we state some definition and notation where we use throughout this chapter. In Section 4.3 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 4.4 we present our primal-dual algorithm and in Section 4.5 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 Section 4.5.4 and prove Theorems 25 & 27. In the last section, we prove the hardness result (Theorem 28).

4.2 Notation and some basic facts

In this chapter, graphs are simple directed graphs unless stated otherwise. By simple we mean there are no parallel edges[2]. 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 we fix a directed graph $G = (V, E)$, edge costs $c_e \geq 0$ for all $e \in E$, a root node r, a set of terminals $X \subseteq V \setminus \{r\}$, and there is no edge between any two Steiner nodes. We denote the optimal value for this DST instance by opt.

For an edge $e = (u, v)$, we call u the ***tail*** and v the ***head*** of e. By ***SCCs*** of $F \subseteq E$ we mean the strongly connected components of (V, F) that contain 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. By ***height*** of a vertex u in tree 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, the algorithm, and the analysis rely on the concept of ***active sets***, so we define them here.

Definition 29 (Violated set)**.** *Given any 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 30 (Active set)**.** *Given any 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.

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

Definition 31 (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-bipartite graphs, active moat have a rather "simple" structure. Our algorithm will leverage the following properties.

Lemma 32. *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 vertices 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.*

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 $B \subseteq A \setminus \{t'\}$ 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 33 (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 34 (Thomason 2001 [67])**.** *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 bound on the number of edges in planar graphs. The following is the consequence of Euler's formula that will be useful in our tighter analysis for quasi-bipartite, planar graphs.

Lemma 35. *Let $G = (V, E)$ be a bipartite planar graph with no parallel edges. Then, $|E| \leq 2 \cdot |V|$.*

4.3 Standard primal-dual algorithm and a bad example

Given a DST instance with $G = (V, E)$ as the input graph, edge costs $c_e \geq 0$ for all $e \in E$, the root node $r \in V$, and $X \subseteq V - \{r\}$ as the terminal set, we define $\mathcal{S} := \{S \subsetneq V : r \notin S, \text{ and } S \cap X \neq \emptyset\}$. We consider the dual of (**Primal-LP**).

$$\begin{aligned} \text{maximize:} \quad & \sum_{S \in \mathcal{S}} y_S && && \textbf{(Dual-LP)} \\ \text{subject to:} \quad & \sum_{\substack{S \in \mathcal{S}: \\ e \in \delta^{in}(S)}} y_S \leq && c_e \quad \forall e \in E \\ & y \geq && 0 \end{aligned}$$

In Section 2.3, we presented the standard primal-dual algorithm that solves the ARBORESCENCE problem on any directed graph, this algorithm is due to Edmonds [16]. 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 variables 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 30) 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 4.1) that shows the total growth of the dual variables is 2 while the optimal value is $k+1$ 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 4.1), 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

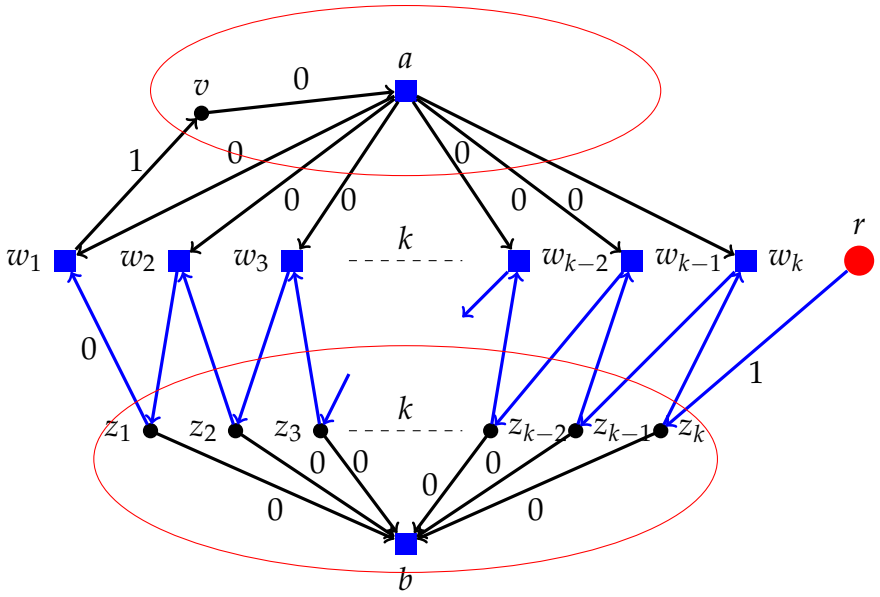


Figure 4.1: This is an example to show why a standard primal-dual algorithm fails. The square vertices are terminals. The downward blue edges (i.e., (w_i, z_{i-1})'s for $2 \leq i \leq k$) have cost 1, the upward blue edges (i.e., (z_i, w_i)'s for $1 \leq i \leq k$) have cost 0. The cost of the black edges are shown in the picture. Note any feasible solution contains all the blue edges and the cost of an optimal solution is $k + 1$. However, the total dual variables that are grown using a standard primal-dual algorithm is 2 (both the bottom and top moats raises their dual variable to 1 and stop growing).

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 tie-breaking criteria we mention below is satisfied.

4.4 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:

$$\begin{aligned} \text{maximize:} \quad & \sum_{S \in \mathcal{S}} y_S && \text{(\textbf{Dual-LP-Modified})} \\ \text{subject to:} \quad & \sum_{\substack{S \in \mathcal{S}: \\ e \in \delta^{in}(S)}} y_S \leq 2 \cdot c_e \quad \forall e \in E \\ & y \geq 0 \end{aligned}$$

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.

Now we, informally, describe our algorithm, see Algorithm 4 for the detailed description. Recall the definition of active moats (Definition 30).

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 33) 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 36 (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,*

- *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 4.2 for an illustration of the above definition.

Intuition behind this definition: When $e = (u, v)$ is a killer edge with respect to an active moat A, then 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 4.2, 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 4.2, 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.

Now we can state our algorithm in details, see Algorithm 4. Note that the purchased edge e_ℓ at iteration ℓ enters some active moat at iteration ℓ.

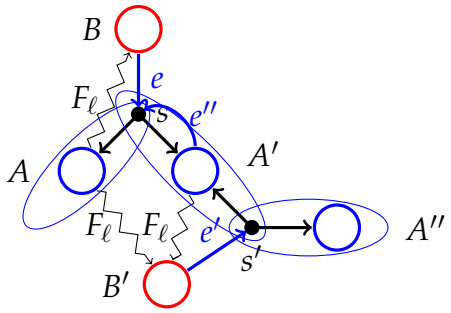


Figure 4.2: Above is a part of a graph at the beginning of iteration ℓ in the algorithm. F_ℓ denotes the set F at this iteration. The circles are SCCs in (V, F_ℓ). Blue circles are inside some active moats shown with ellipses. The black dots s and s' are Steiner nodes. The black edges and the zigzag paths are in F_ℓ. The edges e, e', and e'' have not been purchased yet (i.e., $e, e', e'' \notin F_\ell$). Since C_A is a subset of an active moat namely $A \cup B \cup \{s\}$ with respect to $F_\ell \cup \{e\}$, 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 $F_\ell \cup \{e'\}$-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 $F_\ell \cup \{e'\}$.

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

Definition 37 (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 38. *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 tie-breaking 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 4 on the bad example in Figure 4.1. 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 (w_1, z_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) 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.

4.5 The analysis

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 4. We define $\overline{F}_{\text{Killer}}$ to be the set of edges in $\overline{F}$ that was purchased as killer edge[3]. 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, $\mathcal{A}_\ell$ 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

[3]See Definition 37.

Algorithm 4 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 root node r, and a set of terminal $X \subseteq V \setminus \{r\}$.
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}
$\ell \leftarrow 0$. {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 $\varepsilon \geq 0$ such that the following holds:
(a) for every antenna edge e we have $b_e^{\text{Ant}} + \sum_{\substack{A \in \mathcal{A}: \\ e \in \delta^{in}(A)}} \varepsilon \leq c_e$.
(b) for every non-antenna edge e we have $b_e^{\text{Exp}} + \sum_{\substack{A \in \mathcal{A}: \\ e \text{ is expansion} \\ \text{with resp. to } A}} \varepsilon \leq c_e$.
(c) for every non-antenna edge e we have $b_e^{\text{Killer}} + \sum_{\substack{A \in \mathcal{A}: \\ e \text{ is killer with} \\ \text{resp. to } A}} \varepsilon \leq c_e$.
Increase the dual variables y^* corresponding to each active moat by ε.
for every antenna edge e **do**
$b_e^{\text{Ant}} \leftarrow b_e^{\text{Ant}} + \sum_{\substack{A \in \mathcal{A}: \\ e \in \delta^{in}(A)}} \varepsilon$.
for every non-antenna edge e **do**
$b_e^{\text{Exp}} \leftarrow b_e^{\text{Exp}} + \sum_{\substack{A \in \mathcal{A}: \\ e \text{ is expansion} \\ \text{with resp. to } A}} \varepsilon$.
$b_e^{\text{Killer}} \leftarrow b_e^{\text{Killer}} + \sum_{\substack{A \in \mathcal{A}: \\ e \text{ is killer with} \\ \text{resp. to } A}} \varepsilon$.
pick any single edge $e_\ell \in \cup_{A \in \mathcal{A}} \delta^{in}(A)$ with one of (a)-(c) being tight (break ties arbitrarily).
$F \leftarrow F \cup \{e_\ell\}$.
update $\mathcal{A}$ based on the minimal violated sets with respect to F.
$\ell \leftarrow \ell + 1$.
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**
$\overline{F} \leftarrow \overline{F} \setminus \{e_i\}$.
return $\overline{F}$

(**Dual-LP-Modified**) constructed in the course of the algorithm. We use the following notation throughout the analysis.

Definition 39. *Fix an iteration ℓ. For any $A \in \mathcal{A}_\ell$, let*

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

in other words, $\Delta^\ell_{\text{Killer}}(A)$ 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 $\Delta^\ell_{\text{Exp}}(A)$.

Let $\Delta^\ell_{\text{Ant}}(A) := \{e \in \overline{F}_{\text{Ant}} : e \in \delta^{in}(A) \text{ and } A \text{ is active in iteration } \ell\}$. Finally, we define

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

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

Suppose we want to show that the performance guarantee of Algorithm 4 is $2 \cdot \alpha$ for some $\alpha \geq 1$, it suffices to show the following: for any iteration ℓ we have

$$\sum_{S \in \mathcal{A}_\ell} |\Delta^\ell(S)| \leq \alpha \cdot |\mathcal{A}_\ell|. \tag{4.1}$$

Once we have (4.1), then the $2 \cdot \alpha$-approximation follows easily:

$$\sum_{e \in \overline{F}} c_e = \sum_{e \in \overline{F}_{\text{Killer}}} \sum_\ell \sum_{\substack{S \in \mathcal{A}_\ell: \\ e \in \Delta^\ell_{\text{Killer}}(S)}} \varepsilon_\ell + \sum_{e \in \overline{F}_{\text{Exp}}} \sum_\ell \sum_{\substack{S \in \mathcal{A}_\ell: \\ e \in \Delta^\ell_{\text{Exp}}(S)}} \varepsilon_\ell + \sum_{e \in \overline{F}_{\text{Ant}}} \sum_\ell \sum_{\substack{S \in \mathcal{A}_\ell: \\ e \in \Delta^\ell_{\text{Ant}}(S)}} \varepsilon_\ell \tag{4.2}$$

$$= \sum_\ell \varepsilon_\ell \cdot \sum_{S \in \mathcal{A}_\ell} |\Delta^\ell(S)| \tag{4.3}$$

$$\leq \alpha \cdot \sum_\ell |\mathcal{A}_\ell| \varepsilon_\ell \tag{4.4}$$

$$= \alpha \cdot \sum_{S \subseteq V \setminus \{r\}} y^*_S \tag{4.5}$$

$$\leq 2 \cdot \alpha \cdot (\textit{optimal value of } \textbf{(Dual-LP)}) \tag{4.6}$$

$$= 2 \cdot \alpha \cdot (\textit{optimal value of} \textbf{(Primal-LP)}) \tag{4.7}$$

$$\leq 2 \cdot \alpha \cdot \text{opt}, \tag{4.8}$$

where the first equality follows from the algorithm, the second equality is just an algebraic manipulation, (4.3) follows from (4.1). Equality (4.5) follows

from the fact that we uniformly increased the dual variables corresponding to active moats by ε_ℓ at iteration ℓ, (4.6) follows from feasibility of $\frac{y^*}{2}$ for (**Dual-LP**), and (4.7) follows from strong duality theorem for linear programming (see Theorem 2).

It remains to show (4.1) holds. Consider iteration ℓ. Using the bound on the total degree of nodes in G (using minor-free properties) to show (4.1), it suffices to bound the number of edges in $\overline{F}_{\text{Ant}} \cup \overline{F}_{\text{Killer}} \cup \overline{F}_{\text{Exp}}$ that are being paid by some active moat at iteration ℓ, by $O(|\mathcal{A}_\ell|)$ (where the constant in the O-notation depends on α and r, the size of the excluding minor). 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 $A \in \mathcal{A}_\ell$, there is at most one antenna edge in $\overline{F}$ that enters A, this is proved in Section 4.5.1. 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 4.5.2. 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 Section 4.5.3. Finally, in Section 4.5.4 we put all the bounds we obtained together and derive our approximation factors.

4.5.1 Counting the number of antenna edges in an iteration

Fix an iteration ℓ. Recall F_ℓ denotes the set F at iteration ℓ, and $\mathcal{A}_\ell$ 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 $|\mathcal{A}_\ell|$. We do this in the next lemma.

Lemma 40. *At the beginning of each iteration ℓ, we have* $\sum_{A \in \mathcal{A}_\ell} |\Delta^\ell_{\text{Ant}}(A)| \leq |\mathcal{A}_\ell|$.

Proof. Suppose an active moat $A \in \mathcal{A}_\ell$ is paying toward at least two antenna edges $e = (u, v)$ and $f = (u', v')$ that are in $\overline{F}$. Let C_A be the SCC part of A. Note that since e and f are antenna edges, u and u' are Steiner nodes. Together with the fact that the graph is quasi-bipartite, the heads v and v' are terminals and therefore contained in C_A. Since all the edges in C_A are bought before e and f, one of e or f should have been pruned in the deletion phase, a contradiction. Hence, $|\Delta^\ell_{\text{Ant}}(A)| \leq 1$ which implies the desired bound. □

4.5.2 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 $e_\ell = (u, v)$ be the edge that was added to F_ℓ at iteration ℓ. If $e_\ell = (u, v)$ 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 $e_\ell = (u, v)$ is an antenna edge, then for every active moat A such that $e_\ell \in \delta^{in}(A)$ and C_A is **not** in any active moat with respect to $F_\ell \cup \{e_\ell\}$, 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,

[4]It is possible, e_ℓ is bought as an expansion edge but kills some alive terminals. For example, in Figure 4.2 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 $e_\ell = (u, v) \in \delta^{in}(A)$ 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.

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 41. *Fix an iteration ℓ. We define*

$$\overline{F}^{\ell}_{\text{Killer}} := \{e \in \overline{F}_{\text{Killer}} : \exists A \in \mathcal{A}_\ell \; s.t. \; e \in \Delta^{\ell}_{\text{Killer}}(A)\},$$

in other words, $\overline{F}^{\ell}_{\text{Killer}}$ 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 42. *At the beginning of each iteration ℓ, we have $|\overline{F}^{\ell}_{\text{Killer}}| \leq |\mathcal{A}_\ell|$.*

Proof. As shown above, every killer edge e is assigned to a terminal t_e that was alive until e was added to F. Thus, at iteration ℓ all the edges in $\overline{F}_{\text{Killer}} \setminus F_\ell$ correspond to a terminal that is alive at this iteration. Since there is a one-to-one correspondence between alive terminals and active sets, the number of edges in $\overline{F}_{\text{Killer}} \setminus F_\ell$ is at most $|\mathcal{A}_\ell|$. The lemma follows by noticing that $\overline{F}^{\ell}_{\text{Killer}} \subseteq \overline{F}_{\text{Killer}} \setminus F_\ell$. □

Note that the above lemma does not readily bound $\sum_{A \in \mathcal{A}_\ell} |\Delta^{\ell}_{\text{Killer}}(A)|$ against $|\mathcal{A}_\ell|$ which is required to prove inequality (4.1). We need the properties of minor-free graphs to do so. In the next section we prove a similar bound for expansion edges and then using the properties of the underlying graph, we demonstrate our approximation guarantee.

4.5.3 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 $\overline{F} \cup F_\ell$ and contract all SCCs[6] of (V, F_ℓ). Then, we construct an auxiliary

[6]Recall that we do NOT call a singleton Steiner node that is a 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.

tree that highlights the role of expansion edges to the connectivity of active moats. Then, 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.

We fix an iteration ℓ for this section. First let us recall some notation and definition that we use extensively in this section.

- $\overline{F}$ is the output solution of the algorithm.
- $F_\ell \subseteq E$ 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).
- $\mathcal{A}_\ell$ is the set of active moats with respect to F_ℓ (see Definition 30). Recall each $A \in \mathcal{A}_\ell$ consists of an SCC (with respect to edges in F_ℓ) and a bunch of Steiner nodes. Denote by C_A the SCC part of A.

We define an analogue of Definition 41 for expansion edges.

Definition 43. *Fix an iteration ℓ. Then, we define*

$$\overline{F}^{\ell}_{\text{Exp}} := \{e \in \overline{F}_{\text{Exp}} : \exists A \in \mathcal{A}_\ell \text{ s.t. } e \in \Delta^{\ell}_{\text{Exp}}(A)\},$$

in other words, $\overline{F}^{\ell}_{\text{Exp}}$ is the set of all expansion edges in $\overline{F} \setminus F_\ell$ 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 44. *At the beginning of each iteration ℓ of the algorithm, we have $|\overline{F}^{\ell}_{\text{Exp}}| \leq 2 \cdot |\mathcal{A}_\ell|$.*

Sketch of the proof

We start by giving a sketch of the proof of Lemma 44. Consider the subgraph $F_\ell \cup \overline{F}$ 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 $\mathcal{A}_\ell$, 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 $\overline{F}^{\ell}_{\text{Exp}}$. Note that every non root vertex in $V(H)$ is either labeled active/inactive, or it is a Steiner node. Lemma 44 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 45. *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 to construct 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 4.3 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 4.3 (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 (see Claim 46). Then, we "cut" the subtree rooted at B_2 and "paste" it under A_3 as shown in Figure 4.3(c). It is easy to verify that now every expansion edge is good

with respect to the arborescence in Figure 4.3(c). We formalize this "cut and paste" procedures in Algorithm 5 and prove the output of the algorithm is an arborescence with the property that every expansion edge is good. At the end of this section, given an arborescence that every expansion edge is good, we show there is a rather natural charging scheme that proves Lemma 44.

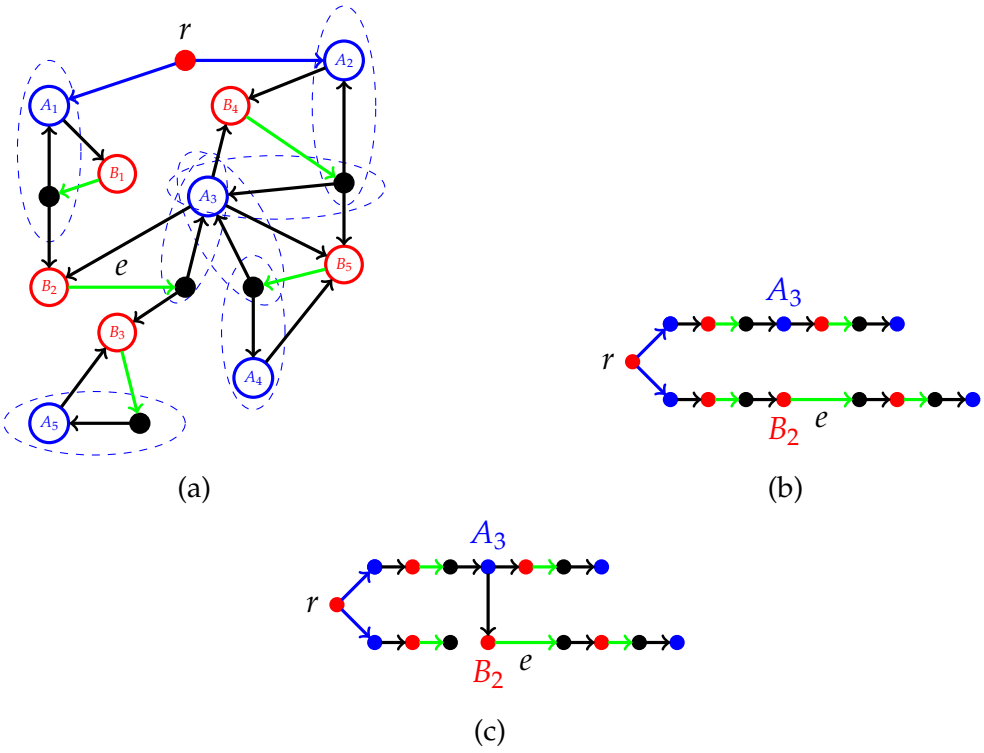


Figure 4.3: (a) shows part of the subgraph $F_\ell \cup \overline{F}$ of G, in particular, the SCCs of (V, F_ℓ) are shown with circles but the nodes inside SCCs are not shown for simplicity. The blue SCCs are inside some active moats shown with dashed ellipses. Contracting all the SCCs result in the graph H discussed before. Black edges are in F_ℓ, blue edges are in $\overline{F} \setminus F_\ell$, and green edges are in $\overline{F}^{\ell}_{\text{Exp}}$. In (b), we have a shortest path arborescence rooted at r where the cost of edges is one if it is green and zero otherwise. 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.

Detailed proof

Our arguments use the following observations about edges being paid as expansion edges in this iteration.

Claim 46. *Let $e = (u, v) \in \overline{F}^{\ell}_{\text{Exp}}$, then $u \in X$, $v \in A$ for some $A \in \mathcal{A}_\ell$, and there is a F_ℓ-path from C_A to u. Furthermore, the SCC of (V, F_ℓ) that contains u is not contained in any active moat in $\mathcal{A}_\ell$.*

Proof. Since e is an expansion edge with respect to A by Definition 36 there exists $A' \subsetneq V$ that is active with respect to $F_\ell \cup \{e\}$. Since e is a non-antenna edge, u must be a terminal. Furthermore, $u \neq r$ because A' is active so $u \in X$. By Lemma 32, the SCC part $C_{A'}$ of A' contains both u and all vertices in C_A, hence there is a dipath in $F_\ell \cup \{e\}$ from C_A to u. However, notice that this dipath cannot contain e, thus the path is actually a F_ℓ-path. Finally, since there is a F_ℓ-path from C_A to u, the SCC B of F_ℓ that contains u is not a violated set and therefore no active moat in $\mathcal{A}_\ell$ contains B. □

Recall the definition of graph H. We state a couple of facts about this graph which will be useful later.

Claim 47. *For every inactive vertex v in H, there is a F_ℓ-path from either r or an active vertex to v.*

Proof. Let v be the contraction of SCC B. Consider all SCCs in (V, F_ℓ) that B is reachable from via a F_ℓ-path and pick such SCC C that is not reachable from any other SCCs of (V, F_ℓ), it is easy to see that either $C = \{r\}$ or C is inside an active moat and therefore, v is reachable from the active vertex that is the contraction of C. □

Claim 48. *For every expansion edge $e = (u, v) \in E(H)$, u must be inactive and v is either active or a Steiner node.*

Proof. Let $e' = (u', v') \in \overline{F}^{\ell}_{\text{Exp}}$ be the corresponding edge to e. By Claim 46, $u' \in X$ and the SCC B in (V, F_ℓ) that contains u' is not a subset of any active moat in $\mathcal{A}_\ell$. Therefore, u is the contraction of such SCC B and so it is labeled inactive. Again by Claim 46, $v' \in A$ for some $A \in \mathcal{A}_\ell$. If v' is not a Steiner node (and therefore v is not a Steiner node) then $v' \in C_A$ and v is the contraction of C_A and so it is labeled active. □

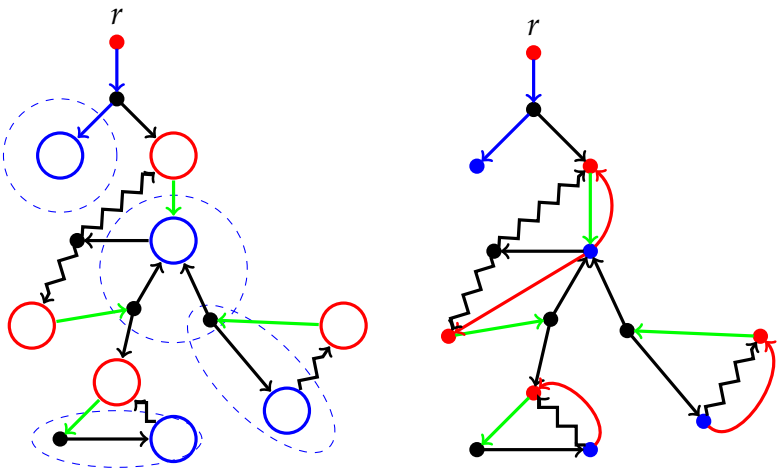


Figure 4.4: The left picture shows the subgraph $F_\ell \cup \overline{F}$. The SCCs of (V, F_ℓ) is shown with circles and the blue ones are inside some active moats shown with dashed ellipses at iteration ℓ. Zigzag paths and black edges are in F_ℓ, blue edges are in $\overline{F} \setminus F_\ell$, and green edges are in $\overline{F}^\ell_{\text{Exp}}$. The right picture shows H_{aux} constructed from $F_\ell \cup \overline{F}$. The red edges are the auxiliary edges.

To simplify the exposition, we use the following auxiliary graph instead of H in proving the main lemma of this section. With abuse of notation, we say a dipath in H is a F_ℓ-path if its corresponding edges are in F_ℓ.

Definition 49 (Auxiliary graph H_{aux})**.** *For every expansion edge $e = (u, v)$ in H and every active vertex w in H such that there is a F_ℓ-path from w to u, add an auxiliary edge (w, u)*[7]*. Set the cost of each expansion edge to 1 and the rest of the edges (including the auxiliary edges) have cost 0. Denote this graph by H_{aux}.*

See Figure 4.4 for an illustration of H_{aux}. Given a subset $T \subseteq E(\text{H}_{\text{aux}})$, we say $e \in \overline{F}^\ell_{\text{Exp}}$ is in T if its corresponding expansion edge in $E(\text{H}_{\text{aux}})$ is in T. For the rest of this section, when we talk about arborescence we mean an arborescence rooted at r that is a subgraph of H_{aux} and every active/inactive vertices in $V(\text{H}_{\text{aux}})$ is reachable from r in this arborescence. Following are two properties of arborescences that will be useful.

Lemma 50. *Let T be an arborescence rooted at r in H_{aux}. Then, we have*

[7]We might create parallel edges but since at the end we work with arborescence, the parallel edges do not matter. Also the resulting graph might destroy the minor-free property; however, we do not need any property of minor-free graphs in this section.

a. *Every edge in $\overline{F}^{\ell}_{\text{Exp}}$ is in T as well. And*

b. *For every expansion edge $e = (u, v)$ in T, either v is active or the subtree T_v of T rooted at v contains an active vertex.*

Proof. Proof of part (a): note that all edges in $\overline{F}^{\ell}_{\text{Exp}}$ are present in H_{aux}. Suppose $e \in \overline{F}^{\ell}_{\text{Exp}}$ that is not in T. Replace every auxiliary edge with its corresponding F_ℓ-path in T. Note that the resulting subgraph H' is a subgraph of H (recall H is the contracted graph obtained from $F_\ell \cup \overline{F}$) and every active/inactive vertex is still reachable from r in H'. Replace every active/inactive vertex in H' by its corresponding contracted SCC, this is a subgraph of $(\overline{F} \cup F_\ell) \setminus \{e\}$ and every terminal is reachable from r. Therefore, $(F_\ell \cup \overline{F}) \setminus \{e\}$ is a feasible solution for the DST instance. Since e was added to F after all edges in F_ℓ, in the deletion phase we should have pruned e, a contradiction with the fact that $e \in \overline{F}$.

Proof of part (b): suppose not. Then v is a Steiner node and every vertex in the subtree rooted at v is either inactive or a Steiner node. If it is inactive then by Claim 47 there must be a F_ℓ-path from either an active vertex or r to it. Add these F_ℓ-path for all inactive vertices in T_v. With the same argument as in part (a) we conclude that (u, v) (i.e., its corresponding edge in $\overline{F}^{\ell}_{\text{Exp}}$) should have been pruned, a contradiction. □

Denote by $\overline{T}$ the shortest path tree in H_{aux} rooted at r. In the following we show how to turn $\overline{T}$ to an arborescence such that every expansion edge is a good expansion edge (with respect to the resulting arborescence). Once we have that, we can provide a charging argument that proves the main lemma of this section (i.e., Lemma 44). We use the following algorithm for modifying $\overline{T}$, note that this is for the analysis and our primal-dual algorithm does not use this.

Algorithm 5 Modifying $\overline{T}$

Input: A shortest path tree $\overline{T}$ of $\mathrm{H}_{\mathrm{aux}}$.
Output: A tree T^* rooted at r such that every active/inactive vertex of $\mathrm{H}_{\mathrm{aux}}$ is reachable from r in T^* and every expansion edge is a good expansion edge.

$\mathcal{L} \leftarrow \emptyset$. {This is the set of edges to be added to $\overline{T}$ at the end.}
Let Γ be the set of all **bad expansion edges** with respect to $\overline{T}$ (cf. Definition 45).
while $\Gamma \neq \emptyset$ **do**
  pick an arbitrary edge $e = (u, v) \in \Gamma$. Let w be an active vertex such that $(w, u), (v, w) \in E(\mathrm{H}_{\mathrm{aux}})$ cf. Lemma 51. Then
  $\mathcal{L} \leftarrow \mathcal{L} \cup \{(w, u)\}$.
  update Γ by removing all the expansion edges incident to u from Γ. {this makes sure that we add only one edge to $\mathcal{L}$ whose head is u}
$\overline{T} \cup \mathcal{L}$ is a DAG (cf. Lemma 54), so by Claim 55 there exists a subset of edges of $E(\overline{T})$ such that its removal makes $\overline{T} \cup \mathcal{L}$ an arborescence rooted at r. Call the resulting arborescence T^*.
return T^*.

We show Algorithm 5 works correctly by a series of lemmas.

Lemma 51. *For every bad expansion edge $(u, v) \in E(\overline{T})$, there exists an active vertex w such that $(w, u), (v, w) \in E(\mathrm{H}_{\mathrm{aux}})$.*

Proof. Note that v is a Steiner node, otherwise (u, v) is a good expansion edge of type 2 with respect to $\overline{T}$. Let (u', v) be the corresponding edge to (u, v) in $F_\ell \cup \overline{F}$. Claim 46 implies $u' \in B$ for some SCC B of (V, F_ℓ), $v \in A \setminus C_A$ for some $A \in \mathcal{A}_\ell$, and there is a F_ℓ-path from C_A to B. Let w be the contraction of C_A in H. Note that u is the contraction of B in H. Therefore, there is a F_ℓ-path from w to u and hence there is an auxiliary edge (w, u) in $\mathrm{H}_{\mathrm{aux}}$. The claim follows by noting that there is an edge whose tail is v and enters C_A; hence (v, w) is in $\mathrm{H}_{\mathrm{aux}}$ as well. □

The above claim proves that the while loop in Algorithm 5 work correctly. Before we prove $\overline{T} \cup \mathcal{L}$ is a DAG, we need two helper claims.

Claim 52. *For any edge $(w, u) \in \mathcal{L}$, we have*

$$\mathrm{Elevel}_{\overline{T}}(u) \leq \mathrm{Elevel}_{\overline{T}}(w) \leq \mathrm{Elevel}_{\overline{T}}(u) + 1.$$

Proof. Let (u, v) be the bad expansion edge that caused us to add (w, u) to $\mathcal{L}$ in Algorithm 5. By Claim 51 we have $(w, u), (v, w) \in E(\mathrm{H}_{\mathrm{aux}})$. Also considering that $\overline{T}$ is a shortest path tree finishes the proof. □

For the next claim we use the following notation. Note that edges in $\mathcal{L}$ will not form a dipath of length greater than 1 because the the edges in $\mathcal{L}$ are oriented from an active vertex to an inactive vertex. So for any dipath P in $\overline{T} \cup \mathcal{L}$ beginning with an edge in $\mathcal{L}$, we write $P = v_1, \mathcal{L}, v_2, \overline{T}, v_3, \mathcal{L}, ..., \overline{T}, v_k$ where $(v_i, v_{i+1}) \in \mathcal{L}$ for odd i and the subpath $P_{[v_i, v_{i+1}]}$ uses only edges in $\overline{T}$ for even i.

Claim 53. *Let $k \geq 3$ be odd and let $P = v_1, \mathcal{L}, v_2, \overline{T}, v_3, \mathcal{L}, ..., \overline{T}, v_k$ be a dipath such that v_i is active for odd i and inactive for even i. Then* $\mathrm{Elevel}_{\overline{T}}(v_k) \geq \mathrm{Elevel}_{\overline{T}}(v_1) + \frac{k-1}{2}$.

Proof. We prove it by induction. Let $k = 3$ (i.e., $P = v_1, \mathcal{L}, v_2, \overline{T}, v_3$). Since $(v_1, v_2) \in \mathcal{L}$ it must be the case that there is a bad expansion edge (with respect to $\overline{T}$) whose tail is v_2; together with the fact that v_3 is active and it is in the subtree rooted at v_2, we have

$$\begin{aligned} \mathrm{Elevel}_{\overline{T}}(v_3) &\geq \mathrm{Elevel}_{\overline{T}}(v_2) + 2 \\ &\geq \mathrm{Elevel}_{\overline{T}}(v_1) + 1, \end{aligned}$$

where the last inequality follows by applying Claim 52 to $(v_1, v_2) \in \mathcal{L}$.

Now suppose the claim holds for k and we prove it for $k + 2$. So the dipath is $P = v_1, \mathcal{L}, v_2, \overline{T}, ..., \overline{T}, v_k, \mathcal{L}, v_{k+1}, \overline{T}, v_{k+2}$.

$$\begin{aligned} \mathrm{Elevel}_{\overline{T}}(v_{k+2}) &\geq \mathrm{Elevel}_{\overline{T}}(v_{k+1}) + 2 \\ &\geq (\mathrm{Elevel}_{\overline{T}}(v_k) - 1) + 2 \\ &\geq \mathrm{Elevel}_{\overline{T}}(v_1) + \frac{k-1}{2} + 1 \\ &= \mathrm{Elevel}_{\overline{T}}(v_1) + \frac{k+1}{2}, \end{aligned}$$

where the first inequality follows because there is a bad expansion edge whose tail is v_{k+1} and v_{k+2} is active and it is in the subtree rooted at v_{k+1},

the second inequality follows from applying Claim 52 to $(v_k, v_{k+1}) \in \mathcal{L}$, and the last inequality follows from the induction hypothesis. □

Next we prove the statements after the while loop in Algorithm 5 works correctly.

Lemma 54. *After the while loop in Algorithm 5, $\overline{T} \cup \mathcal{L}$ is a DAG.*

Proof. Recall that edges in $\mathcal{L}$ will not form a dipath of length greater than 1; hence, any dicycle in $\overline{T} \cup \mathcal{L}$ always alternate between a dipath in $\overline{T}$ and an edge in $\mathcal{L}$. By reordering the alternation, we denote a dicycle in $\overline{T} \cup \mathcal{L}$ by $C = v_1, \mathcal{L}, v_2, \overline{T}, v_3, \mathcal{L}, ..., \overline{T}, v_{k-1}, \mathcal{L}, v_k, \overline{T}, v_1$ where v_i is active for odd i and even otherwise. Note that k is even.

It is easy to see $k \neq 2$. Otherwise we have a dicycle $C = v_1, \mathcal{L}, v_2, \overline{T}, v_1$ which implies there is a bad expansion edge (with respect to $\overline{T}$) whose tail is v_2 together with the fact that v_1 is an active vertex in $\overline{T}_{v_2}$ we must have $\text{Elevel}_{\overline{T}}(v_1) \geq \text{Elevel}_{\overline{T}}(v_2) + 2$ (otherwise all expansion edges whose tail is v_2 are good expansion edges). On the other hand, since $(v_1, v_2) \in \mathcal{L}$ by Claim 52 we have $\text{Elevel}_{\overline{T}}(v_1) \leq \text{Elevel}_{\overline{T}}(v_2) + 1$, a contradiction.

For the sake of contradiction, we assume there is a dicycle $C = v_1, \mathcal{L}, v_2, \overline{T}, ..., \overline{T}, v_{k-1}, \mathcal{L}, v_k, \overline{T}, v_1$, where v_i is active for odd i and inactive otherwise, furthermore we assume $k \geq 4$. By applying Claim 53 to $v_1, \mathcal{L}, v_2, \overline{T}, ..., \overline{T}, v_{k-1}$, we get $\text{Elevel}_{\overline{T}}(v_1) + 1 \leq \text{Elevel}_{\overline{T}}(v_{k-1})$, and by applying Claim 52 to $(v_{k-1}, v_k) \in \mathcal{L}$ we have $\text{Elevel}_{\overline{T}}(v_{k-1}) - 1 \leq \text{Elevel}_{\overline{T}}(v_k)$. Together, we see $\text{Elevel}_{\overline{T}}(v_1) \leq \text{Elevel}_{\overline{T}}(v_k)$. On the other hand, since $(v_{k-1}, v_k) \in \mathcal{L}$ there is a bad expansion edge whose tail is v_k, and the fact that v_1 is an active vertex in $\overline{T}_{v_k}$, it must be that $\text{Elevel}_{\overline{T}}(v_k) + 2 \leq \text{Elevel}_{\overline{T}}(v_1)$ which is a contradiction. □

Next, we show that $\overline{T} \cup \mathcal{L}$ can be turned into an arborescence by removing a unique subset of edges of $E(\overline{T})$. To do so we use the following generic claim.

Claim 55. *Let $T = \big(V(T), E(T)\big)$ be an arborescence, and let $L = \{(u_1, v_1), ..., (u_k, v_k)\}$ be a collection of edges such that $u_i, v_i \in V(T)$ and $(u_i, v_i) \notin$*

$E(T)$ for all $1 \leq i \leq k$. Furthermore, $v_i \neq v_j$ for $i \neq j$. If $T \cup L$ is a DAG, then there is a unique set of edges $B \subseteq E(T)$ of size k such that $(T \cup L) \setminus B$ is an arborescence.

Proof. We prove this by induction on the size of L. The base case (i.e., when we have one edge in L) is easy to see. Suppose it is true when $|L| \leq k$ now we prove it for $|L| = k+1$. Let $L' \subsetneq L$ be a subset of size k. Since $T \cup L$ is a DAG so is $T \cup L'$ and hence by induction hypothesis there is a unique $B' \subseteq E(T)$ such that $T' := (T \cup L') \setminus B'$ is an arborescence rooted at r. Let $\{e\} = L \setminus L'$, since $T' \cup \{e\}$ is a subgraph of $T \cup L$, we know $T' \cup \{e\}$ is a DAG too and again by induction hypothesis there is an edge $e' \in E(T')$ such that $(T' \cup \{e\}) \setminus \{e'\}$ is an arborescence. Since $(T' \cup \{e\}) \setminus \{e'\}$ is an arborescence, e' and e must have the same heads (otherwise the head of e has indegree 2 in $(T' \cup \{e\}) \setminus \{e'\}$). The inductive step follows by noticing that e' cannot be in L because otherwise it contradicts the fact that the heads of edges in L are disjoint; hence, $e' \in E(T)$. Let $B := B' \cup \{e'\} \subseteq E(T)$. Note $|B| = k+1$. Then, we have $(\overline{T} \cup L) \setminus B = (T' \cup \{e\}) \setminus \{e'\}$ which is an arborescence. □

Note that $\overline{T} \cup \mathcal{L}$ satisfies all the conditions of Claim 55 so the line after the while loop in the algorithm works correctly.

Remark 56. *The edges in B in Lemma 55 are the edges of $E(T)$ whose head is one of vertices $v_1, ..., v_k$.*

Finally, we show that every expansion edge is a good expansion edge (recall Definition 45) with respect to T^* to finish the correctness of Algorithm 5.

Lemma 57. *Every expansion edge is a good expansion edge with respect to T^*.*

Proof. Note that for a bad expansion edge $e = (u, v)$ in $\overline{T}$ since there is an edge $(w, u) \in \mathcal{L}$ in T^* where w is active, e is a good expansion edge of type 1 with respect to T^*.

Next we show that when we are removing edges from $\overline{T}$ to make $T^* = \overline{T} \cup \mathcal{L}$ a DAG, we do not make a good expansion edge becomes bad in T^*.

By Remark 56, we remove $(x, y) \in \overline{T}$ if and only if there exists an edge in $\mathcal{L}$ whose head is y.

case 1. If $e = (u, v)$ is a good expansion edge of type 1 in $\overline{T}$. So there is an active vertex w in $\overline{T}$ such that the dipath $P_{w,u}$ in $\overline{T}$ from w to u does not have any expansion edge. Furthermore, if there is an expansion edge whose tail is on $P_{w,u}$, then that expansion edge is of type 1. Hence, there is no edge in $\mathcal{L}$ whose head is in $P_{w,u}$ and so $P_{w,u}$ is in T^* as well and e is a good expansion edge of type 1 in T^*.

case 2. If $e = (u, v)$ is a good expansion edge of type 2 in $\overline{T}$. So there is an active vertex w in the subtree of $\overline{T}$ rooted at u such that the dipath $P_{u,w}$ in $\overline{T}$ from u to w has at most one expansion edge (it could be that $w = v$). Pick the one that is closest (in terms of edge hops) to u. Then all the expansion edges whose tail is on $P_{u,w}$ is of type 2. Therefore, there is no edge in $\mathcal{L}$ whose head is in $P_{u,w}$ and so $P_{u,w}$ is in T^* as well and e is a good expansion edge of type 2 in T^*.

□

Finally, we can state the proof of the main lemma of this section.

Proof. (of Lemma 44) Note that in order to prove Lemma 44, it suffices to show the number of expansion edges in T^* is at most twice the number of active vertices in T^*. We prove this based on a token argument. Assign **two** tokens to every active vertex. We show that the number of expansion edges is at most the number of tokens. We do this via the following charging scheme.

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 $1 \leq i \leq k$ 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 $1 \leq i \leq k$ 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.

Here we prove by induction on the height l, that case (i) and case (ii) works correctly.

Consider the following base case: let u be a vertex and let $(u, v_1), ..., (u, v_k)$ be the only expansion edges in T^*_u. Then, by Lemma 50(b), for every $1 \leq i \leq k$ there is an active vertex in $T^*_{v_i}$ and so it has two unused tokens. Therefore, both cases (i) and (ii) work in the base case.

Now consider a vertex u and assume case (i) and case (ii) are correct for all vertices (except u) in T^*_u whose is the tail of an expansion edge. We show it is correct for u as well.

Proof for case (i): Suppose u falls into case (i). So each (u, v_i) for $1 \leq i \leq k$ is of type 1. If there is no expansion edge in $T^*_{v_i}$ then by Lemma 50(b) there is an active vertex in $T^*_{v_i}$ and has two unused tokens. So now assume there is an expansion edge in $T^*_{v_i}$ and pick the one $f_i = (x_i, y_i)$ whose tail is closest to v_i (break the ties arbitrarily). If f_i is of type 2, then by induction hypothesis $T^*_{x_i}$ has one unused token (when we processed x_i) and since by the choice of f_i there is no expansion edge on the dipath P_{v_i,x_i} in T^*; hence this token is unused at this iteration as well. If f_i is of type 1, then there is an active vertex z on P_{v_i,x_i} and has two tokens. Again we note that the tokens of z are unused since there is no expansion edge on $P_{v_i,z}$.

So we proved for each (u, v_i) where $1 \leq i \leq k$ there is at least one unused token in $T^*_{v_i}$, as desired.

Proof for case (ii): Suppose u falls into case (ii). So each (u, v_i) for $1 \leq i \leq k$ is of type 2. With the exact same argument as in case (i), we can show that for each $1 \leq i \leq k$ there is (at least) one unused token in $T^*_{v_i}$. So we just need to show an extra unused token in T^*_u.

Since (u, v_i)'s are of type 2, there must be an active vertex w such that $\text{Elevel}_{T^*}(u) \leq \text{Elevel}_{T^*}(w) \leq \text{Elevel}_{T^*}(u) + 1$. Pick such w with smallest Elevel. If $\text{Elevel}_{T^*}(w) = \text{Elevel}_{T^*}(u)$ then w has two tokens and these tokens are different than the ones in $T^*_{v_i}$ because w is not in $T^*_{v_i}$'s. Furthermore, the tokens of w are unused because there is no expansion edge on the dipath $P_{u,w}$ in T^*.

So let us assume $\text{Elevel}_{T^*}(w) = \text{Elevel}_{T^*}(u) + 1$. There are two cases to consider:

- w is in $T^*_{v_j}$ for some $1 \leq j \leq k$. Note that there is no expansion edge on $P_{v_j,w}$. Therefore, among the two tokens of w, one could be assigned to (u, v_j) as before and the other one will be unused when we are processing u so this would be the extra unused token we wanted.

- w is not in $T^*_{v_j}$ for any $1 \leq j \leq k$. So there is one expansion edge (x, y) on $P_{u,w}$. By the choice of w (with smallest Elevel), (x, y) must be of type 2 (otherwise there is an active vertex w' on $P_{u,x}$ whose Elevel is strictly smaller than w, a contradiction with the choice of w). Therefore, x has one unused token when x was processed. Since there is no expansion edge on $P_{u,x}$, this token is unused at this iteration as well. Finally, since x is not in $T^*_{v_i}$ for $1 \leq i \leq k$ this unused token is the extra token, as desired.

□

4.5.4 Putting everything together

Fix an iteration ℓ. We use Lemmas 42 & 44 and the properties of graph G to bound $\sum_{A \in \mathcal{A}_\ell} |\Delta^\ell_{\text{Killer}}(A) \cup \Delta^\ell_{\text{Exp}}(A)|$. 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 58. *Fix an iteration ℓ and an active moat $A \in \mathcal{A}_\ell$. There is at most one edge in $\Delta^\ell_{\text{Killer}}(A) \cup \Delta^\ell_{\text{Exp}}(A)$ whose head is in C_A.*

Proof. Suppose there are two edges e and f in $\Delta^\ell_{\text{Killer}}(A) \cup \Delta^\ell_{\text{Exp}}(A)$ that enter C_A. Since e and f are bought later than all the edges in C_A, we should have pruned one of e or f in the deletion phase. □

Consider the graph $F_\ell \cup \overline{F}$. 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 $\overline{F}^\ell_{\text{Killer}} \cup \overline{F}^\ell_{\text{Exp}}$. Then, for each $A \in \mathcal{A}_\ell$ 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[8]. 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 59. *For each active moat $A \in \mathcal{A}_\ell$, we have*

$$|\Delta^\ell_{\text{Killer}}(A) \cup \Delta^\ell_{\text{Exp}}(A)| \leq |\delta^{in}_{G'}(C_A)| + 1. \tag{4.9}$$

Proof. Consider an active moat A and let v be a Steiner node in $A \setminus C_A$. First note that the indegree of vertices in $\overline{F}$ is at most 1 therefore there is at most one edge $e \in \overline{F}^\ell_{\text{Killer}} \cup \overline{F}^\ell_{\text{Exp}}$ that enters v. Secondly, we note that by Lemma 32 there is at least one edge in F_ℓ from v to C_A and we kept one such edge in G'; so the contribution of e to the LHS of (4.9) is accounted for in the RHS.

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

Finally, by Claim 58 at most one killer/expansion edge enters C_A and the contribution of this edge is accounted for by the plus one in the RHS. □

Next, using Lemmas 42 & 44 we bound the number of vertices in G'.

Claim 60. $|V(G')| \leq 4 \cdot |\mathcal{A}_\ell|$.

Proof. The set $V(G')$ is consist of C_A's for some active moat A and bunch of Steiner nodes. Note that we kept a Steiner node s if there is (exactly) one edge in $\overline{F}^\ell_{\text{Killer}} \cup \overline{F}^\ell_{\text{Exp}}$ that enters s. Therefore, $|V(G')|$ is at most $|\mathcal{A}_\ell| + |\overline{F}^\ell_{\text{Killer}}| + |\overline{F}^\ell_{\text{Exp}}|$. The bound follows from Lemmas 42 & 44. □

Finally, we prove Theorems 25 & 27.

Proof. (of Theorem 25) Since G is K_r-minor free so is G'. So we can write

$$\begin{aligned}\sum_{A \in \mathcal{A}_\ell} \left|\Delta^\ell_{\text{Killer}}(A) \cup \Delta^\ell_{\text{Exp}}(A)\right| &\leq \sum_{A \in \mathcal{A}_\ell} \left(|\delta^{in}_{G'}(C_A)| + 1\right) \\ &= |E(G')| + |\mathcal{A}_\ell| \\ &\leq O(r \cdot \sqrt{\log r}) \cdot 4 \cdot |\mathcal{A}_\ell| + |\mathcal{A}_\ell| \\ &= O(r \cdot \sqrt{\log r})|\mathcal{A}_\ell|,\end{aligned} \tag{4.10}$$

where the inequality follows from Claim 59 and the second inequality follows from Claim 60 together with Theorem 34.

Next we show (4.1) holds for $\alpha = O(r \cdot \sqrt{\log r})$.

$$\begin{aligned}\sum_{A \in \mathcal{A}_\ell} |\Delta^\ell(A)| &= \sum_{A \in \mathcal{A}_\ell} |\Delta^\ell_{\text{Killer}}(A) \cup \Delta^\ell_{\text{Exp}}(A)| + \sum_{A \in \mathcal{A}_\ell} |\Delta^\ell_{\text{Ant}}(A)| \\ &\leq O(r \cdot \sqrt{\log r})|\mathcal{A}_\ell| + |\mathcal{A}_\ell| \\ &= O(r \cdot \sqrt{\log r})|\mathcal{A}_\ell|,\end{aligned}$$

where inequality follows from inequality (4.10) and Lemma 40.

As we discussed at the beginning of Section 4.5 that if (4.1) holds for α then we have a $(2 \cdot \alpha)$-approximation algorithm. Hence, Algorithm 4 is an $O(r \cdot \sqrt{\log r})$-approximation for DST on quasi-bipartite, K_r-minor free graphs. □

Proof. (of Theorem 27) The proof of Theorem 27 is exactly the same as proof of Theorem 25 except instead of $O(r \cdot \sqrt{\log r})$ in (4.10) we have 2 because G'

is a bipartite planar graph, see Lemma 35. Now we can write

$$\sum_{A \in \mathcal{A}_\ell} \left|\Delta^\ell_{\text{Killer}}(A) \cup \Delta^\ell_{\text{Exp}}(A)\right| \leq 9 \cdot |\mathcal{A}_\ell|,$$

and

$$\sum_{A \in \mathcal{A}_\ell} |\Delta^\ell(A)| \leq 10 \cdot |\mathcal{A}_\ell|.$$

Therefore, (4.1) holds for $\alpha = 10$ and hence we have a 20-approximation algorithm, as desired. □

4.6 NP-hardness

In this section we prove Theorem 28. We reduce from the NP-complete problem CONNECTED VERTEX COVER (CVC) on planar graphs. Here, we are given a planar graph $G = (V, E)$ and a positive integer k. The goal is to decide if there is a vertex cover $S \subseteq V$ (i.e., every edge has at least one of its endpoint in S) such that $|S| \leq k$ and $G[S]$ is connected. This problem is known to be NP-complete, see Lemma 2 in [27].

Our reduction from CVC on planar graphs to UNDIRECTED STEINER TREE on quasi-bipartite planar graphs is similar to the reduction showing UNDIRECTED STEINER TREE problem is NP-hard on general graphs from [39]. Let $(G = (V, E),\ k)$ be an instance of CVC where G is planar. Subdivide every edge $e \in E$ by a terminal vertex x_e and call the resulting graph G', which is also planar. Let $X := \{x_e :\ \forall e \in E\}$ be the set of terminals and $V(G)$ is the set of Steiner nodes in G'.

Lemma 61. *G' has a Steiner tree of size $k + |E(G)| - 1$ if and only if G has a connected vertex cover of size k.*

Proof. Suppose G has a connected vertex cover S of size k. Then, $G'[S \cup X]$ is connected and therefore it has a spanning tree T where $|E(T)| = |S \cup X| - 1 = |E(G)| + k - 1$.

Now let T' be a tree that spans X in G' and $|E(T')| = |E(G)| + k - 1$. Since $|X| = |E(G)|$, we have $|V(T') \setminus X| = k$. Define $S := V(T') \setminus X$; we

show that S is a connected vertex cover for G. The fact that it is a vertex cover is clear because for every edge $e \in E(G)$ at least one of its endpoint is in $V(T') \setminus X$. Consider $u, v \in S$. Since $u, v \in V(T')$, there is a path $P = u, x_{e_1}, w_1, x_{e_2}, w_2, ..., w_{\ell-1}, x_{e_\ell}, v$ in T'. Note that the path $u, w_1, w_2, ..., w_{l-1}, v$ is in $G[S]$. So we showed $G[S]$ is a connected subgraph of G and $|S| = k$, as desired. □

This completes the proof of Theorem 28.

Chapter 5

Bounded-Degree Traveling Salesman Problem

5.1 Overview and the basics

In this chapter we study Question 2 stated in the introduction.

Let us start by recalling the definition of the problems we consider in this chapter. An input to BOUNDED-DEGREE SUBSET TRAVELING SALESMAN problem (BDSTSP) is an undirected graph $G = (V, E)$ with edge costs $c_e \geq 0$ for all $e \in E$, a subset of terminals $X \subseteq V$ we are to visit ($|X| \geq 2$), and even integer bounds $b_v \geq 0$ for all nodes $v \in V$. The goal is to find a minimum-cost closed walk (tour) $\mathcal{Q}$ spanning all terminals such that $d_{\mathcal{Q}}(v) \leq b_v$ (i.e., the number of edges in the multiset $\mathcal{Q}$ incident to v is at most b_v). Note b_v should be thought of as a degree bound, thus the tour should pass through v at most $\frac{b_v}{2}$ times. We call a special case of BDSTSP where $X = V$, the BOUNDED-DEGREE TRAVELING SALESMAN problem (BDTSP).

All graphs in this chapter are undirected and may be multi-graphs and all subsets of edges may be multi-subsets, we adopt this convention now so we do not have to use the prefix *multi* on every set or graph. In particular, when we discuss the degree of a vertex with respect to a set of edges or with respect to a graph, we mean its degree if we count all edges with the same multiplicity that they appear in the set/graph. However, all subsets of vertices will be actual sets: each vertex will be in the set at most once. The cost is considered with multiplicity, e.g. if edge e is used k times its contribution

to the cost is $k \cdot c_e$.

All of our algorithms are based on linear-programming relaxations: if the LPs are not feasible then we report there is no feasible solution. Otherwise, we will find an $(\alpha, +d)$-approximate solution: the cost of the tour will be at most α times the value of the LP relaxation and will visit each node at most $\frac{(b_v+d)}{2}$ times (i.e., the degree of the tour at v will be at most $b_v + d$). We note that if there is a feasible solution, then the LP relaxations we use will be feasible and will have value at most the optimum solution value.

Recall for $F \subseteq E$, $odd(F)$ is the set of all vertices with odd degree with respect to F, i.e., $odd(F) = \{v \in V(F) : \ d_F(v) \text{ is odd}\}$. Given a (sub)graph H, for brevity, we use $d_H(v)$ and $odd(H)$ instead of $d_{E(H)}(v)$ and $odd(E(H))$, respectively. Recall $\chi(F)$ is the characteristic vector of the edges in F. We sometime use notation $|A| = odd$ which means $|A| \equiv 1 \pmod 2$, similarly we define the notation $|A| = even$. Since the graphs in this chapter are undirected, we use uv to denote an edge between u and v.

The results and Techniques

As a warm up, in Section 5.2 we present a simple $(\frac{3}{2}, +4)$-approximation algorithm for BDTSP (i.e. if $X = V$).

Theorem 62 (BDTSP)**.** *There is a $(\frac{3}{2}, +4)$-approximation algorithm for BDTSP.*

Since a feasible solution is an Eulerian graph and b_v's are even, if there is an approximation algorithm whose degree violation is better than additive factor of 2, then this algorithm can decide the Hamiltonian cycle problem. Hence, assuming $P \neq NP$, the additive factor of 2 violation on degree bounds is necessary. Furthermore, the same integrality gap example for Held-Karp relaxation where the degree bound on every vertex is 2, see Figure 5.1, shows the integrality gap of the natural LP formulation (**BDTSP-LP**) is at least $(\frac{4}{3}, +2)$, meaning any tour obtained by rounding a feasible solution of (**BDTSP-LP**) has cost at least $\frac{4}{3}$ times the LP optimum and violates the degree bound of at least one vertex by at least $+2$.

Throughout this chapter we heavily use some classic results in combi-

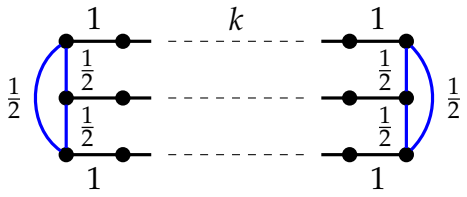


Figure 5.1: This is the graph $G = (V, E)$ that shows the $(\frac{4}{3}, +2)$ integrality gap of the natural LP for BDTSP (**BDTSP-LP**). All vertices are terminals, the cost of blue edges is zero and the cost of black edges is 1. The LP value on blue edges are $\frac{1}{2}$, and 1 on all the other edges. Also $b_v = 2$ for all $v \in V$. Note that the cost of the LP is $3 \cdot k$ and satisfies all the degree bounds. However, in any integer solution we must cross one of the path of length k at least twice. Therefore, any integer solution will violate the degree constraint by at least an additive factor of 2 and its cost is at least $4 \cdot k$ which give the desired integrality gap.

natorial optimization that we discuss here briefly. Recall the definition of the BOUNDED-DEGREE STEINER TREE problem (BD-Steiner-TP) from Section 2.5. Lau and Singh [49] showed the following result about the natural LP relaxation of this problem. Let $W \subseteq V$, be the subset of vertices with degree constraints (so vertices in $V \setminus W$ can have arbitrary large degrees). The following is the natural LP relaxation for BD-Steiner-TP.

$$\begin{aligned}
&\text{minimize:} && \sum_{e \in E} c_e \cdot x_e && && \textbf{(BDST-LP)}\\
&\text{subject to:} && x(\delta(v)) \leq && b_v \quad \forall v \in W && (5.1)\\
& && x(\delta(S)) \geq && 1 \quad \forall S \neq X,\ S \cap X \neq \emptyset && (5.2)\\
& && x \geq && 0 && (5.3)
\end{aligned}$$

Theorem 63 (Theorem 1.1 in [49])**.** *Let $\overline{x}$ be a feasible solution to* (**BDST-LP**)*. Then, in polynomial time, one could get a Steiner tree T of cost at most $2 \cdot \text{cost}(\overline{x})$ and $d_T(v) \leq b_v + 3$ for all $v \in W$.*

Next, recall the classical Y-join problem in Section 2.6. Like Steiner tree, one could generalize Y-join problem to include degree bounds. In an instance of BOUNDED-DEGREE Y-JOIN, in addition to Y-JOIN input, there are integer degree bounds b_v for all $v \in V$ where b_v is odd if and only if $v \in Y$. Here is a

natural LP relaxation for this problem.

$$\begin{aligned}
\text{minimize:} \quad & \sum_{e \in E} c_e \cdot x_e && && \textbf{(BD-}Y\textbf{-join LP)} \\
\text{subject to:} \quad & x(\delta(v)) \leq && b_v \quad \forall v \in V && (5.4) \\
& x(\delta(S)) \geq && 1 \quad \forall S \subsetneq V,\ |S \cap Y| = odd && (5.5) \\
& x \geq && 0 && (5.6)
\end{aligned}$$

The first set of constraints are called *degree constraints* and the second set of constraints are *odd-cut constraints*.

Maybe a less known fact in combinatorial optimization is that (**BD-Y-join LP**) is integral as well. This result is a consequence of a more general theorem in [64]. However, this connection is not obvious.

Theorem 64 (Theorem 36.8 in [64])**.** *The feasible region of* (**BD-Y-join LP**) *is integral, if and only if, b_v is odd if $v \in Y$ and even otherwise for all $v \in V$. In other words, any extreme point optimal solution of* (**BD-Y-join LP**) *is a minimum cost Y-join while respects the degree bounds b_v's.*

The proof of Theorem 36.8 in [64], which the above result is derived from, is quite complicated. Instead, we present rather a standard iterative rounding algorithm to prove Theorem 64. For the sake of better readability, we postpone the proof to the end of the chapter, see Section 5.4.

Now we have all the tools to talk about the first result of this chapter. The proof of Theorem 62 is a straightforward adaptation of Wolsey's analysis [70] of the Christofides-Serdyukov algorithm for TSP so we sketch it here to discuss our techniques. Let x^* be an optimal solution for the natural LP formulation of BDTSP. Step (1): it is easy to see $\frac{x^*}{2}$ is feasible for (**BDST-LP**) when terminal nodes are the entire set of vertices. From this, we obtain a spanning tree T of cost at most $\sum_{e \in E} c_e \cdot x^*_e$ whose degree on vertex v is at most $\frac{b_v}{2} + 3$. Step (2): fix the degree parities of T using Y-join polytope augmented with degree bounds at most $\frac{b_v}{2} + 1$ (depending on the parity of v's degree in the tour) and show $\frac{x^*}{2}$ is feasible for (**BD-Y-join LP**).

Remark 65. *Notice that we did not use the $+1$ algorithm for degree-bounded spanning trees by [66]. This is because dividing x by 2 is only guaranteed to satisfy the weaker cut-based LP relaxation for spanning trees.*

Theorem 64 and its proof serve as introduction to the framework we use in this chapter. The main results of this chapter is about different approximation/violation trade-offs for BDSTSP.

Theorem 66. *There is a $(\frac{5}{3}, +4)$-approximation algorithm for BDSTSP.*

Theorem 67. *There is a $(\frac{13}{8}, +6)$-approximation algorithm for BDSTSP.*

Theorem 68. *There is a $(\frac{3}{2}, +8)$-approximation algorithm for BDSTSP.*

In each of these, we first adapt step (1) from BD-Steiner-TP discussed above, compute a Steiner tree (instead of spanning tree) T using $\frac{x^*}{2}$ as a fractional solution to (**BDST-LP**). However, step (2) is not applicable since $\frac{x^*}{2}$ might not be feasible for (**BD-Y-join LP**).

To prove Theorem 66, it is easy to show that $\frac{1}{3} \cdot (\chi(T) + x^*)$ is feasible for (**BD-Y-join LP**) where $\chi(T)$ is the characteristic vector of T which yields $(\frac{5}{3}, +4)$-approximation factor[1]. In order to improve the cost factor, we first augment the natural LP for BDSTSP with non-trivial constraints asserting the number of cut edges in a Steiner cuts (a cut that contains only Steiner nodes) should be at least the degree of any Steiner node in the cut. Then, we modify the iterative rounding algorithm of [49] using splitting off techniques by Mader to obtain a more "structured" Steiner tree. Namely, some Steiner nodes are designated *dangerous* because they have low fractional degree in our LP solution: our modification ensures dangerous nodes will have even degree in the resulting tree. Finally, we show how this Steiner tree helps us to obtained a better bounded-degree Y-join to fix the degree parity of odd-degree vertices.

[1]Interestingly, this is basically the same fractional join from [3] that could be formed to analyze Hoogeveen's TSP-Path algorithm.

5.2 Bounded-Degree TSP (Warm Up!)

In this section we prove Theorem 62. Fix an instance of BDTSP: $G = (V, E)$, edge cost $c_e \geq 0$ for all $e \in E$, even degree bound $b_v \geq 0$ for all $v \in V$. The following is a natural LP formulation for this problem. For each edge e, there is a variable x_e indicating whether e is in the solution or not. Note that in an optimal solution for the problem, we might need to pick an edge twice but not more than twice since, otherwise, one can reduce its occurrence by two and retain connectivity.

$$\begin{aligned}
&\text{minimize:} && \sum_{e \in E} c_e \cdot x_e && && \textbf{(BDTSP-LP)}\\
&\text{subject to:} && x(\delta(S)) \geq 2 && \forall\, \emptyset \neq S \subsetneq V && (5.7)\\
& && x(\delta(v)) \leq b_v && \forall\, v \in V && (5.8)\\
& && 0 \leq x_e \leq 2 && \forall\, e \in E && (5.9)
\end{aligned}$$

One can separate the constraints using a minimum-cut algorithm, so we can find an optimal solution (or determine (**BDTSP-LP**) is infeasible) in polynomial time using ellipsoid method. If the LP is infeasible, we report there is no feasible solution and terminate. Otherwise, we proceed as follows.

The algorithm is very similar to Wolsey's analysis of Christofides-Serdyukov algorithm. First we compute a spanning tree T using an optimal solution to (**BDTSP-LP**) and then we fix the degree parities using the $odd(T)$-join polytope. However, we need to respect (approximately) the degree bounds. For a node v, let $b'(v, T)$ be the smallest integer at least $\frac{b_v}{2}$ whose parity is the same as $|d_T(v)|$: note $b'(v, T) \in \{\frac{b_v}{2}, \frac{b_v}{2} + 1\}$.

Algorithm 6 $(\frac{3}{2}, +4)$-approximation algorithm for BDTSP

Input: Graph $G = (V, E)$ with edge costs $c_e \geq 0$ for every $e \in E$ and even degree bounds b_v for every $v \in V$.
Output: A tour that spans V.

Compute an optimal solution x^* of (**BDTSP-LP**).
Let T be a Steiner tree (in this case spanning tree) obtained from applying Theorem 63 with input $G = (V, E)$, edge cost c_e for $e \in E$, $X := V$, degree bounds $\frac{b_v}{2}$ for every $v \in V$ and $\overline{x} := \frac{x^*}{2}$.
Let $odd(T)$ be the set of vertices with odd degrees with respect to T. Compute an $odd(T)$-join J in G using Theorem 64 with degree bounds $b'(v, T)$ for all $v \in V$.
Output a closed spanning walk in $T \cup J$.

We show Algorithm 6 works correctly and this proves Theorem 62.

Proof. (of Theorem 62) Note $\overline{x}$ is feasible for (**BDST-LP**) where $X := V$ and degree bounds $\frac{b_v}{2}$ for every $v \in V$. Hence, by Theorem 63 we have $\text{cost}(T) \leq 2 \cdot \text{cost}(\overline{x}) = \text{cost}(x^*)$. Furthermore, $d_T(v) \leq \frac{b_v}{2} + 3$.

Consider vertex v in the graph, note that $\overline{x}(\delta(v)) \leq \frac{b_v}{2}$. By using degree bounds $b'(T, v)$ for $v \in V$, we ensure (**BD-Υ-join LP**) is integral and that $\overline{x}$ is a feasible solution. Thus, by Theorem 64 J has cost at most $\text{cost}(\overline{x}) = \text{cost}(\frac{x^*}{2})$ and $d_J(v) \leq \frac{b_v}{2} + 1$ for every $v \in V$.

Putting the bounds on T and J together we have an Eulerian subgraph $T \cup J$ with cost $\frac{3}{2} \cdot \text{cost}(x^*)$ and $d_{T \cup J}(v) \leq b_v + 4$. □

5.3 Bounded-Degree Subset TSP

In this section, we prove our main results, i.e., Theorems 66, 67 and 68. Fix an instance of BDSTSP: $G = (V, E)$, edge costs $c_e \geq 0$ for each $e \in E$, a set of terminals $X \subseteq V$, and an even integer degree bound $b_v \geq 0$ on each vertex $v \in V$. We refer to vertices in $V \setminus X$ as *Steiner* nodes.

The algorithm for BDSTSP is to find a "good" Steiner tree T that spans the terminals and then fix the degree parity using $odd(T)$-join. However, finding a "good" $odd(T)$-join is not as trivial as it was for BDTSP since $\frac{x^*}{2}$ may no longer be feasible for $odd(T)$-join polytope since some cuts S involving

only Steiner nodes may have very low $x^*(\delta(S))$. Nevertheless, for all the approximation factors in this section, we show combining x^* and T itself with appropriate ratios is sufficient to construct a "good" (fractional) solution for the $odd(T)$-join polytope with degree constraints. We start with a simple application of this idea by proving Theorem 66.

We begin with the natural LP for BDSTSP. As before, we assume the LP has an optimal solution, otherwise the BTSTSP instance has no feasible solution.

$$\begin{aligned}
\text{minimize:} \quad & \sum_{e \in E} c_e \cdot x_e && && \textbf{(BDSTSP-Natural-LP)} \\
\text{subject to:} \quad & x(\delta(S)) \geq && 2 \quad \forall S \neq X,\ S \cap X \neq \emptyset && (5.10) \\
& x(\delta(v)) \leq && b_v \quad \forall v \in V && (5.11) \\
& 0 \leq x_e \leq && 2 \quad \forall e \in E && (5.12)
\end{aligned}$$

Proof of of Theorem 66. Let x^* be an optimal solution to (**BDSTSP-Natural-LP**). Since $\frac{x^*}{2}$ is feasible for (**BDST-LP**) where degree bounds are $\frac{b_v}{2}$ for every $v \in V$, we obtain a Steiner tree T of cost at most $2 \cdot \text{cost}(\frac{x^*}{2}) = \text{cost}(x^*)$ and $d_T(v) \leq \frac{b_v}{2} + 3$, see Theorem 63. Furthermore, we iteratively prune leaf nodes that are Steiner nodes so all leaves of T are terminals. With abuse of notation, we denote the resulting tree by T.

Next, we show that $\overline{y} := \frac{\chi(T)}{3} + \frac{x^*}{3}$ is feasible for (**BD-$\mathcal{Y}$-join LP**) when $odd(T)$ is the set of odd degree vertices and the RHS of the degree constraints are either $\frac{b_v}{2} + 1$ or $\frac{b_v}{2} + 2$ (whichever has the same parity as $d_T(v)$). Note that by definition of $\overline{y}$, and the fact that $d_T(v) \leq \frac{b_v}{2} + 3$, $\overline{y}$ respects the degree constraints in (**BD-$\mathcal{Y}$-join LP**). Now consider a cut S that contains a terminal. Then T crosses the cut at least once and $x^*(\delta(S)) \geq 2$ so $\overline{y}(\delta(S)) \geq 1$.

Now consider an odd-cut S, i.e. $|S \cap odd(T)| = odd$, that contains only Steiner nodes. Since $\sum_{v \in S} d_T(v) = 2 \cdot |E_T[S]| + |\delta_T(S)|$ and $\sum_{v \in V} d_T(v)$ is odd, we must have $|\delta_T(S)| = odd$. We claim that $|\delta_T(S)| > 1$, otherwise $|\delta_T(S)| = 1$ means S contains a leaf node which is impossible since (the pruned version) of T has only terminals as leaf nodes. Therefore, $|\delta_T(S)| \geq 3$ and by definition of $\overline{y}$ we have $\overline{y}(\delta(S)) \geq 1$, as desired.

So we have proved $\overline{y}$ is feasible for (**BD-Y-join LP**). By Theorem 64, there is an $odd(T)$-join J of cost at most $\text{cost}(\overline{y}) = \frac{1}{3} \cdot \text{cost}(T) + \frac{1}{3} \cdot \text{cost}(x^*) \leq \frac{2}{3} \cdot \text{cost}(x^*)$ and $d_J(v) \leq \frac{b_v}{2} + 2$ for all $v \in V$. Finally we output a closed walk in subgraph $\mathcal{Q} := T \cup J$. Note $\text{cost}(\mathcal{Q}) \leq \frac{5}{3} \cdot \text{cost}(x^*)$ and $d_{\mathcal{Q}}(v) \leq b_v + 5$ for all $v \in V$. Since $\mathcal{Q}$ is an Eulerian graph and b_v's are even, it must be that $d_{\mathcal{Q}}(v) \leq b_v + 4$ for all $v \in V$. This finishes the proof of Theorem 66. □

To improve on the approximation factor of Theorem 66 we consider a slight strengthening of (**BDSTSP-Natural-LP**) for BDSTSP. We first make an observation about the structure of an optimal solution and then we add a set of constraints based on this observation.

We use the following definition throughout this section. Let v be a Steiner node, we say S is a v, X-cut if $v \in S \subseteq V \setminus X$.

Lemma 69. *There exists an optimal solution $\mathcal{Q}^*$ such that for any Steiner node $\overline{v}$ and any $\overline{v}, X$-cut S, we have $|\delta_{\mathcal{Q}^*}(S)| \geq d_{\mathcal{Q}^*}(\overline{v})$.*

Proof. Among all optimal solutions, let $\mathcal{Q}^*$ be one with the minimum number of edges. For this proof, every degree or cut is with respect to $\mathcal{Q}^*$ unless stated otherwise. We show $|\delta_{\mathcal{Q}^*}(S)| \geq d_{\mathcal{Q}^*}(\overline{v})$ for every $\overline{v}$ and any $\overline{v}, X$-cut S. Suppose otherwise, that for $\overline{v}$ there is some $\overline{v}, X$-cut S with $|\delta_{\mathcal{Q}^*}(S)| < d_{\mathcal{Q}^*}(\overline{v})$. We take S to be a minimum-cardinality $\overline{v}, X$-cut.

Let $k := |\delta(S)|$. Note $\sum_{v \in S} d(v) = 2 \cdot |E_{\mathcal{Q}^*}[S]| + |\delta(S)|$ and since all vertices have even degree, k must be even. We say $u \in S$ is a *boundary node* with respect to S if $\delta(u) \cap \delta(S) \neq \emptyset$.

Contract $V \setminus S$ to a single vertex and call it t. Since S is a minimum cardinality $\overline{v}, t$-cut, there are k edge-disjoint simple paths $P_1, ..., P_k$ from $\overline{v}$ to t, in particular, for every boundary node $u \in S$, $|\delta(u) \cap \delta(S)|$ of the paths $P_1, ..., P_k$ have u as their second-last node (just before t).

Construct a graph G' obtained from $\mathcal{Q}^*$ as follows. Remove all the edges in $E_{\mathcal{Q}^*}[S] \setminus \cup_{i=1}^{k} P_i$ which is non-empty as we assumed $d_{\mathcal{Q}^*}(v) > k$ and then remove all the isolated vertices. Note that $d_{G'}(\overline{v}) = k$ which is even. Also for every boundary vertex $u \in S$, $|\delta(u) \cap \delta(S)|$ many of P_i's contain u, so the

degree of boundary vertices is even as well. The vertices inside S except $\overline{v}$ are internal vertices of some paths so their degrees are even. Finally, degree of the vertices outside of S did not change, so their degree is even as well. Hence, G' is Eulerian.

We claim G' connects all the terminals, which would contradict the minimality of $\mathcal{Q}^*$. Hence, $|\delta_{\mathcal{Q}^*}(S)| \geq d_{\mathcal{Q}^*}(\overline{v})$, as desired. It is easy to prove the claim. Consider two terminals x and x'. If a $x - x'$ path in $\mathcal{Q}^*$ does not use any vertex in S then this path exists in $E(G')$. So suppose every $x - x'$ paths in $\mathcal{Q}^*$ crosses S, let u and u' (possibly $u = u'$) be the two boundary vertices (with respect to S) on a $x - x'$ path in $\mathcal{Q}^*$. Note that there is a path between u and $\overline{v}$, and a path between u' and $\overline{v}$ in $E(G')$. Therefore, x and x' are connected in $E(G')$. □

We use Lemma 69 to get a slightly stronger LP relaxation for BDSTSP.

$$\begin{aligned}
\text{minimize:} \quad & \sum_{e \in E} c_e \cdot x_e & & & \textbf{(BDSTSP-LP)} \\
\text{subject to:} \quad & x(\delta(S)) \geq & 2 \qquad & \forall S \neq X,\ S \cap X \neq \emptyset & (5.13) \\
& x(\delta(v)) \leq & b_v \qquad & \forall v \in V & (5.14) \\
& x(\delta(v)) \leq & x(\delta(S)) \qquad & \forall S \subseteq V \setminus X, \forall v \in S & (5.15) \\
& 0 \leq x_e \leq & 2 \qquad & \forall e \in E & (5.16)
\end{aligned}$$

Constraint (5.15) is valid because there is an optimal solution that has this property according to Lemma 69. Furthermore, this constraint can be separated by computing a minimum weight v, X-cut (with respect to weight x on the edges) for every Steiner node v. Let x^* be an optimal solution to this LP.

Here we discuss how to get a more well-structured Steiner "tree"[2] T using a slightly modified version of the algorithm in [49] for computing a degree-bounded Steiner tree, which in turn, results in a cheaper $odd(T)$-join. An ingredient we use here is the splitting-off procedure from Section 2.7.

[2]We put tree in the quotation as we will see later that T might contain cycles to allow the degree parity of some Steiner nodes to be even, so T might not be a proper tree.

The tweak in the algorithm of [49] for BD-Steiner-TP is to completely "split-off" a predetermined subset of Steiner nodes when the algorithm decides to drop the degree constraint corresponding to these Steiner nodes. This will ensure that we get a feasible solution for BD-Steiner-TP (not necessarily a tree) such that all the Steiner nodes in the predetermined subset have even degree (counting with multiplicities). More precisely, we use the following lemma.

Lemma 70. *Given an instance $\big(G = (V, E), X, c, b\big)$ of BD-Steiner-TP, a feasible solution $\overline{x}$ of* (**BDST-LP**) *and a set $A \subseteq V \setminus X$ where $b_v = 1$ for all $v \in A$, there is a polynomial time algorithm that computes a feasible solution T of BD-Steiner-TP such that*

1. $\text{cost}(T) \leq 2 \cdot \text{cost}(\overline{x})$.
2. $d_T(v) \leq b_v + 3$ *for all* $v \in V \setminus A$.
3. $d_T(v) \leq b_v + 7$ *for all* $v \in A$.
4. $d_T(v)$ *is even for all* $v \in A$.
5. $|\delta_T(S)|$ *is even for any* $S \subseteq A$.
6. *Let $T^{(m)}$ be a minimal (inclusion-wise) subset of edges of T such that $T^{(m)}$ is still a feasible solution for the BD-Steiner-TP instance. Then, $T^{(m)}$ satisfies property 1 and $d_T(v) \leq b_v + 3$ for all $v \in V$.*

We defer the proof of the lemma until the end and instead we show how to use it to prove our results about BDSTSP.

5.3.1 Proof of Theorem 67

Here is our algorithm for BDSTSP and we will show a $(\frac{13}{8}, +6)$-approximation guarantee for this algorithm.

For the rest of this section, let x^* be an optimal solution for (**BDSTSP-LP**) computed in step (a) and $\overline{x} := \frac{x^*}{2}$. Let A be the subset of Steiner nodes and

Algorithm 7 $(\frac{13}{8}, +6)$-approximation algorithm for BDSTSP

Input: Graph $\big(G = (V, E), X, c, b\big)$.
Output: A closed walk $\mathcal{Q}$ in G that spans X.

(a) Compute an optimal solution x^* of (**BDSTSP-LP**). Let $A := \{v \in V \setminus X : x^*(\delta(v)) < 2\}$, and let $b'_v := \frac{b_v}{2}$ if $v \notin A$ and $b'_v := 1$ if $v \in A$.
(b) Apply Lemma 70 with instance $\big(G = (V, E), X, c, b'\big)$, feasible solution $\overline{x} := \frac{x^*}{2}$, and set A to obtain a Steiner tree T with properties 1-5 in the lemma.
(c) Compute a Steiner tree $T^{(m)}$ using T according to property 6 of Lemma 70.
(d) Compute a minimum cost $odd(T^{(m)})$-join J such that $d_J(v) \leq b'_v + 3$ for all $v \in V$ (cf. Lemma 71).
(e) Output a closed walk $\mathcal{Q}$ in $T^{(m)} \cup J$ that spans all the terminals.

b'_v's the degree bounds constructed based on x^* in step (a). Also let T and $T^{(m)}$ be the Steiner trees computed in steps (b) and (c) of the algorithm, respectively. Note that $\overline{x}$ is feasible for (**BDST-LP**) when the degree bounds are according to b'. Combining this fact and Lemma 70 we have:

$$\text{cost}(T^{(m)}) \leq \text{cost}(T) \leq 2 \cdot \text{cost}(\overline{x}) = \text{cost}(x^*). \tag{5.17}$$

Lemma 71. *There is an $odd(T^{(m)})$-join J with cost at most $\frac{5}{8} \cdot \text{cost}(x^*)$ and $d_J(v) \leq \frac{b_v}{2} + 3$ for all $v \in V$.*

Proof. We claim $y := \frac{\chi(T)}{4} + \frac{3 \cdot x^*}{8}$ is feasible for (**BD-Y-join LP**) when the set of odd degree vertices is $odd(T^{(m)})$ and the RHS of degree constraint is at most $b'_v + 3$ for all $v \in V$ (in fact this fractional solution is feasible when degree bounds are $\frac{b_v}{2} + 2$ but similar to the proof of Theorems 62 and 66, we might need to consider $\frac{b_v}{2} + 3$ for some vertices as (**BD-Y-join LP**) is integral if and only if the parity of the degree bounds match the parity of the degree of vertices in an integral solution). We prove the claim after we show how the lemma follows from this claim. By Theorem 64, there is an integral $odd(T^{(m)})$-join J whose cost is at most

$$\text{cost}(y) = \text{cost}(\frac{1}{4} \cdot \chi(T)) + \text{cost}(\frac{3}{8} \cdot x^*) \leq \frac{1}{4} \cdot \text{cost}(x^*) + \frac{3}{8} \cdot \text{cost}(x^*) \leq \frac{5}{8} \cdot \text{cost}(x^*),$$

where the first inequality follows from (5.17). Furthermore, $d_J(v) \leq b'_v + 3 \leq \frac{b_v}{2} + 3$.

So it remains to prove the claim. As (5.18) shows, y satisfies the degree constraints of (**BD-Y-join LP**) when the degree bound is at most $b'_v + 3$ for all $v \in V$.

$$y(\delta(v)) = \frac{1}{4} \cdot d_T(v) + \frac{3}{8} \cdot x^*(\delta(v)) \leq \frac{1}{4} \cdot (b'_v + 7) + \frac{3}{8} \cdot b_v \leq \frac{b_v}{2} + 3, \quad (5.18)$$

where the first inequality follows from properties 2 & 3 of Lemma 70.

Next we show cut constraints in (**BD-Y-join LP**) hold under solution y. Consider a subset $S \subseteq V$ such that $|S \cap odd(T^{(m)})| = odd$. There are three cases to consider:

- Case 1: If $S \cap X \neq \emptyset$. Then, $x^*(\delta(S)) \geq 2$ and since T is connected we have $|\delta_T(S)| \geq 1$ which implies $y(\delta(S)) \geq 1$.

- Case 2: If $S \cap X = \emptyset$ and $S \cap (V \setminus A) \neq \emptyset$. Then, there is a Steiner node $s \in S$ such that $s \notin A$. By definition of set A we have $x^*(\delta(s)) \geq 2$. By constraint (5.15) in (**BDSTSP-LP**) this implies $x^*(\delta(S)) \geq 2$ as well. Again since T is connected, we have $|\delta_T(S)| \geq 1$ and this implies $y(\delta(S)) \geq 1$.

- Case 3: If $S \subseteq A$. Since (5.19) holds for any subset of vertices and $|S \cap odd(T^{(m)})| = odd$, the LHS of (5.19) is odd and so is $|\delta_{T^{(m)}}(S)|$. Furthermore, if $|\delta_{T^{(m)}}(S)| = 1$ then we can remove S from $T^{(m)}$ and still have a feasible solution, contradicting the inclusion-wise minimality of $T^{(m)}$. Hence $|\delta_{T^{(m)}}(S)| \geq 3$. So we have

  $$|\delta_T(S)| \geq |\delta_{T^{(m)}}(S)| \geq 3.$$

  On the other hand, since $S \subseteq A$, by property 5 Lemma 70 we have $|\delta_T(S)| = even$; together with above inequality we get $|\delta_T(S)| \geq 4$. Therefore, $y(\delta(S)) \geq 1$ in this case as well.

□

Now the proof of Theorem 67 follows easily.

Proof of Theorem 67. Since $T^{(m)} \cup J$ is an Eulerian subgraph, there is a closed walk Q in it that spans X. By (5.17) we have $\text{cost}(T^{(m)}) \leq \text{cost}(x^*)$. By Lemma 71 $\text{cost}(J) \leq \frac{5}{8} \cdot \text{cost}(x^*)$ and this implies $\text{cost}(T^{(m)} \cup J) \leq \frac{13}{8} \cdot \text{cost}(x^*)$, as desired.

By property 6 of Lemma 70, $d_{T^{(m)}}(v) \leq b'_v + 3 \leq \frac{b_v}{2} + 3$ and by Lemma 71 we have $d_J(v) \leq \frac{b_v}{2} + 3$. So $d_{T^{(m)} \cup J}(v) \leq b_v + 6$ for all $v \in V$, as desired. □

5.3.2 Proof of Theorem 68

To prove Theorem 68, we modify Algorithm 7 slightly as follows. Remove step (c), in step (d) compute a minimum cost $odd(T)$-join J (instead of $odd(T^{(m)})$-join), and in step (e) output a closed walk in $T \cup J$ that spans all the terminals. Similar to Lemma 71, we have the following bound on the bounded-degree $odd(T)$-join LP.

Lemma 72. *There is an $odd(T)$-join J with cost at most $\frac{1}{2} \cdot \text{cost}(x^*)$ and $d_J(v) \leq \frac{b_v}{2} + 1$ for all $v \in V$.*

Proof. We just need to show $\frac{x^*}{2}$ is feasible for (**BD-$\mathcal{Y}$-join LP**) when the degree bound is either $\frac{b_v}{2}$ or $\frac{b_v}{2} + 1$ depending on the parity of $d_T(v)$. It is trivial that the degree constraints hold. So we show the cut constraints hold. Let $S \subsetneq V$ such that $|S \cap odd(T)| = odd$.

- Case 1: If $S \cap X \neq \emptyset$. Then, $x^*(\delta(S)) \geq 2$ which implies $\frac{x^*}{2}(\delta(S)) \geq 1$.
- Case 2: If $S \cap X = \emptyset$ and $S \cap (V \setminus A) \neq \emptyset$. Then, there is a Steiner node $s \in S$ such that $s \notin A$. By definition of set A we have $x^*(\delta(s)) \geq 2$. By constraint (5.15) in (**BDSTSP-LP**) this implies $x^*(\delta(S)) \geq 2$ as well which implies $\frac{x^*}{2}(\delta(S)) \geq 1$.
- Case 3: If $S \subseteq A$. Thus, we have $S \subseteq A$ and note that $d_T(v)$ is even for all $v \in A$ by property 4 of Lemma 70. This is a contradiction because we assumed $|S \cap odd(T)|$ is odd. So the constraints for $S \subseteq A$ are not present in the LP.

□

Proof of Theorem 68. By (5.17) we have $\text{cost}(T) \leq \text{cost}(x^*)$ and by Lemma 72 we have $\text{cost}(J) \leq \frac{1}{2} \cdot \text{cost}(x^*)$ which implies $\text{cost}(T \cup J) \leq \frac{3}{2} \cdot \text{cost}(x^*)$.

By properties 2 and 3 of Lemma 70, we have $d_T(v) \leq \frac{b_v}{2} + 7$ and by Lemma 72 we have $d_J(v) \leq \frac{b_v}{2} + 1$ for all $v \in V$. Thus, $d_{T \cup J}(v) \leq b_v + 8$, as desired. □

5.3.3 Proof of Lemma 70

We restate the lemma here for convenience.

Lemma 73. *Given an instance $\big(G = (V, E), X, c, b\big)$ of BD-Steiner-TP, a feasible solution $\overline{x}$ of* **(BDST-LP)** *and a set $A \subseteq V \setminus X$ where $b_v = 1$ for all $v \in A$, there is a polynomial time algorithm that computes a feasible solution T of BD-Steiner-TP such that*

1. $\text{cost}(T) \leq 2 \cdot \text{cost}(\overline{x})$.
2. $d_T(v) \leq b_v + 3$ *for all* $v \in V \setminus A$.
3. $d_T(v) \leq b_v + 7$ *for all* $v \in A$.
4. $d_T(v)$ *is even for all* $v \in A$.
5. $|\delta_T(S)|$ *is even for any* $S \subseteq A$.
6. *Let $T^{(m)}$ be a minimal (inclusion-wise) subset of edges of T such that $T^{(m)}$ is still a feasible solution for the BD-Steiner-TP instance. Then, $T^{(m)}$ satisfies property 1 and $d_T(v) \leq b_v + 3$ for all $v \in V$.*

The following is the result that allows us to tweak the iterative rounding algorithm of [49] for BD-Steiner-TP to ensure Steiner nodes in A have even degree in the resulting tree (recall the statement of Lemma 70). We use Mader theorem as the subroutine. However, just applying Mader theorem repeatedly when the connectivities are based on LP weight on edges does not run in polynomial time. To overcome this, we follow the idea of Post and Swamy

[58]. In [58] they work with the directed version of Mader theorem. Although everything will be translated to our setting in a straightforward fashion, we present the proof for completeness.

Lemma 74. *Let $\mathcal{I} = \big(G = (V, E), X, c, b\big)$ be an instance of the BD-Steiner-TP and let $\overline{x}$ be a feasible solution for* (**BDST-LP**) *of this instance. Let $s \in V$ be a Steiner node. Then, in polynomial time, one can obtain an instance of BD-Steiner-TP $\mathcal{I}' = \big(G' = (V \setminus \{s\}, E'), X, c', b\big)$ and a feasible solution x' for* (**BDST-LP**) *of this instance such that*

1. $\text{cost}_{c'}(x') \leq \text{cost}_c(x)$.
2. *An integral solution T' for $\mathcal{I}'$ can be transformed to an integral solution T for $\mathcal{I}$ whose cost is at most* $\text{cost}_{c'}(T')$, *$d_T(s)$ is even, and $d_T(v) = d_{T'}(v)$ for all $v \in V \setminus s$.*

Proof. First we show how to obtain x' and c' using Mader's theorem repeatedly; however, the running time of this procedure might be exponential. At the end, we show how to modify it so it runs in polynomial time.

Since $\overline{x}$ is rational and the number bits needed to represent it is polynomial in the size of the input, there is a positive integer Δ such that $\Delta \cdot \overline{x}_e$ is an even integer for all $e \in E$. We replace each edge e, with $\Delta \cdot \overline{x}_e$ copies of parallel edges. Note that degree of s is even and there is no cut-edge in the graph. By applying Mader's theorem repeatedly, we can split-off s completely. Denote the resulting graph by $G' = (V \setminus \{s\}, E')$. Finally, we define a new solution x' for the resulting graph as follows: $x'_e = \frac{\#\ copies\ of\ e}{\Delta}$ for all $e \in E'$. Note that by construction, x' respects the degree bounds and we preserve the connectivity between each pairs of nodes (except s), hence x' is feasible for (**BDST-LP**) corresponding to G'. The edge costs c' is defined naturally, i.e., for the existed edges in G, c and c' agree with each other and for a new introduced edge uv we set $c'_{uv} := c_{su} + c_{sv}$. Let $\mathcal{I}' := (G', X, c', b)$ be the resulting instance.

We show that $\text{cost}_{c'}(x') \leq \text{cost}_c(\overline{x})$. This follows from the fact that if we introduce a new edge uv, then its cost is $c_{su} + c_{sv}$ and we decrease the LP

weight on su and sv by the same amount we increase the LP value on uv. If uv exists in G we claim that $c_{su} + c_{sv} > c_{uv}$ because otherwise one can increase the LP value on su and sv and decrease the LP value on uv by the same amount and get a cheaper feasible solution, contradicting the fact that $\overline{x}$ is optimal.

Finally, for any integral solution T' for $\mathcal{I}'$ we construct an integral solution T for $\mathcal{I}$ by replacing every edge uv in T' that is not in G with the corresponding pair of edges su and sv (keep the edges with multiplicities) that were split-off. Note we might have more than 2 copies of an edge e incident to s in T. In this case, we reduce the occurrences of e as much as possible so that the resulting solution is feasible for $\mathcal{I}$, and the parity of the degree of the endpoints of e does not change. By definition of c' we have $\text{cost}_c(T) \leq \text{cost}_{c'}(T')$.

Note $\Delta \cdot \overline{x}_e$ might be exponential in terms of the size of the graph. Here we show by a straightforward adaptation of techniques in [58], we can construct x' efficiently.

Let $\overline{x}$ be the LP weight on the edges. Splitting-off a pair (su, sv) to the extent of $\alpha > 0$ means reducing $\overline{x}_{sv}$ and $\overline{x}_{sv}$ by α, and increasing $\overline{x}_{uv}$ by α such that all the connectivities between any pair of nodes (except pairs involving s) are preserved (the connectivity between two nodes u and v is defined as the minimum weight $u - v$ cut when edges have weight according to $\overline{x}$). We say we split of (su, sv) to the *maximum extent* if value α above is the maximum value possible.

Note that by the first procedure we explained earlier, there exists always a pair (su, sv) and value $\alpha > 0$ such that we can split of (su, sv) to the extent of α. We can find such pairs by brute force ($O(n^2)$ pairs) and find α by binary search. Note that it is possible $u = v$ which in that case it means reducing x_{su} by α.

The algorithm is simple. We find a pair of edges (su, sv) and the largest value $\alpha > 0$ such that we can split-off (su, sv) to the extent of α and repeat this procedure until there is no edge incident to s.

In the next claim, we show if we split-off a pair of edges to the maximum

extent, that pair never becomes splittable again which in turn implies a polynomial running time of above procedure.

Claim 75. *Consider an splittable pair (su, sv) according to Mader theorem (Theorem 9). If we split-off (su, sv) to the maximum extent, then (su, sv) will not become splittable again.*

Proof of Claim 75. This follows from Claim 3.1 in [20] which states the following:

Claim 76 (Claim 3.1 in [20])**.** *A pair (su, sv) is splittable if and only if there is no set Y such that $u, v \in Y$, $s \notin Y$, and there are two nodes $w \in Y$, $s \neq z \notin Y$ such that $|\delta_G(Y)| \leq \lambda_G(w, z) + 1$.*

Multiply $\overline{x}$ by a suitable integer Δ such that $\Delta \cdot \overline{x}_e$ is even for all $e \in E$. Replace each edge e by $\Delta \cdot \overline{x}_e$ many parallel edges in G.

Suppose we split as much copy of (su, sv) as possible. Then, there must be a set Y that satisfies the properties of Claim 76. Now assume we split of a pair of edges (e, f) incident to s and let G' be the resulting graph. We show that Y still satisfies the properties of Claim 76; hence, non of the copy of (su, sv) are splittable after splitting-off (e, f).

Note $s \notin Y$ so $|\delta_{G'}(Y)| \leq |\delta_G(Y)|$. Furthermore, since (e, f) is a splittable pair we have $\lambda_{G'}(w, z) = \lambda_G(w, z)$. Therefore, $|\delta_{G'}(Y)| \leq |\delta_G(Y)| \leq \lambda_G(w, z) = \lambda_{G'}(w, z) + 1$. Hence, Y satisfies the properties of Claim 76 in G' as well so (e, f) is not splittable in G'. □

By the above claim, once a pair is split-off to the maximum extend, that pair never becomes splittable again. Hence, after at most n^2 iterations of splitting-off, the algorithm terminates. Also within each iteration, we need to do a binary search to find the appropriate α which is polynomial. This finishes the proof of the lemma. □

Now we are equipped to finish the proof of Lemma 70.

Proof of Lemma 70. We run the $(2, +3)$-approximation of [49] with the following tweak. Whenever we drop the degree constraints corresponding to

a Steiner node v in A, we completely split-off v as well. After the tree is constructed, we restore any edges produced by the splitting-off procedure to the original set of edges and further prune some edges if necessary (i.e., if their multiplicity is > 2 after this restoration step)[3]. See Algorithm 8 for the complete description of the algorithm.

Properties 1 and 2 are trivial because of the approximation factor of Algorithm 8 and Lemma 74.

Property 3: we completely split-off a Steiner node $s \in A$ in Algorithm 8 when the algorithm decides to drop the degree constraint corresponding to s (see step b in Algorithm 8). Hence, we must have $d(s) \leq b_s + 3 \leq 4$ in the current graph in that iteration. Once we apply Lemma 74 (property 2) to obtain a solution for the current instance of BD-Steiner-TP (i.e., by putting s back) we might use these (up to) four edges incident to s multiple times; however, we do not need to use any edge more than twice (otherwise we can drop two copies of the edge and preserve both connectivity and parities) so the degree of s in the solution will be at most $8 = b_v + 7$ since $b_v = 1$ for $v \in A$. This proves property 3.

Property 4: the degree of a node $v \in A$ that was split-off in our iterative rounding algorithm is even simply because each edge used in T that was produced in the splitting-off procedure is then subdivided to re-integrate v so the degree of v is even. Further, any parallel edges that were pruned maintain the parity of the degree of v.

Property 5: for any set $S \subseteq V$ we have

$$\sum_{v \in S} d_T(v) = 2 \cdot |E_T[S]| + |\delta_T(S)|. \tag{5.19}$$

If $S \subseteq A$, by property 4 the LHS of (5.19) is even and therefore $|\delta_T(S)|$ must be even.

Property 6: by property 2, for all $v \in V \setminus A$ we have $d_T(v) \leq b_v + 3$. In $T^{(m)}$ we keep only one copy of each parallel edge so $d_T(v) \leq b_v + 3$ for all

[3]Note we might have edges with multiplicity 2 in this solution but nevertheless we call the solution a Steiner tree.

$v \in A$ (see the argument for property 3) and the resulting solution is still feasible which implies the last property for $T^{(m)}$. □

Algorithm 8 Iterative rounding algorithm of [49] for BD-Steiner-TP with small change in step (b)

Input: Graph $\big(G = (V, E), X, c, b\big)$, a subset of Steiner nodes A where $b_v = 1$ for all $v \in A$.
Output: A connected subgraph T of G that spans X such that $d_T(v) \leq b_v + 7$ is even for all $v \in A$ and $d_T(v) \leq b_v + 3$ for all $v \in V \setminus A$.

Initialize $T' \leftarrow \emptyset$ and $W \leftarrow V$ (W is the set of vertices with degree constraints present in the LP formulation).
while T' is not a feasible solution for BD-Steiner-TP **do**
(a) Compute an optimal extreme point solution x of (**BDST-LP**) and remove every edge e with $x_e = 0$.
(**b, with modification**). Removing a degree constraint: For every $v \in W$ with degree at most $b_v + 3$ in G, remove v from W. Furthermore, if $v \in A$, completely split-off v and compute an optimal solution for the resulting instance (cf. Lemma 74). Redefine G to be this new graph and x to be the new optimal solution after splitting-off procedure.
(c) Picking 1-edge: For each edge $e = uv$ with $x_e = 1$, add e to T, remove e from G, and decrease b_u, b_v by 1.
(d) Picking a heavy edge with no degree constraints: For each edge $e = uv$ with $x_e \geq \frac{1}{2}$ and $u, v \notin W$, add e to T' and remove e from G.
Updating the connectivity requirements: For every set $S \neq X$, $S \cap X \neq \emptyset$ update the RHS of constraint (2) in (**BDST-LP**) to be $\max\{1 - |\delta'_T(S)|, 0\}$.
Let T be the resulting Steiner tree obtained from T' by applying Lemma 74 repeatedly.

Note that the algorithm works correctly for any subset A of Steiner nodes (i.e., A does not need to contain only Steiner nodes v with $b_v = 1$). The performance guarantee of this algorithm follows from [49] and the fact that after splitting-off a Steiner node, by Lemma 74 we still have a feasible solution for the (**BDST-LP**) of the resulting instance with cost at most the cost of the original LP. So T' is a Steiner tree for the instance $\big(G' = (V \setminus A, E'), X, c', b\big)$. The fact that the Steiner tree T for the original instance obtained from T' has the desired properties of Lemma 70 follows from Lemma 74.

5.4 Proof of integrality of (BD-Y-join LP)

For convenience we restate the LP here.

$$\begin{aligned}
\text{minimize:} \quad & \sum_{e \in E} c_e \cdot x_e && && \textbf{(BD-}Y\textbf{-join LP)} \\
\text{subject to:} \quad & x(\delta(v)) \leq && b_v \quad \forall v \in V && (5.20) \\
& x(\delta(S)) \geq && 1 \quad \forall S \subsetneq V,\ |S \cap Y| = odd && (5.21) \\
& x \geq && 0 && (5.22)
\end{aligned}$$

In this section, we prove the this LP is integral. Note b_v is odd if and only if $v \in Y$. This will be important. In fact, without this condition Figure 5.2 shows the integrality gap of (**BD-Y-join LP**) could be as bad as $\frac{1}{2}$.

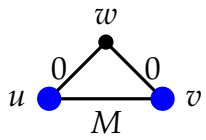


Figure 5.2: Let $Y = \{u, v\}$ and degree bound for all three vertices is 1. Note $w \notin Y$ but $b_w = 1$. Then, the optimal Y-join is the edge uv with cost M. However, setting $\frac{1}{2}$ on all the edges is a feasible solution for (**BD-Y-join LP**) with cost $\frac{M}{2}$. Therefore, the integrality gap of (**BD-Y-join LP**) is at least as bad as 2.

We present an iterative rounding algorithm that shows an extreme point of (**BD-Y-join LP**) is integral. The algorithm and its analysis are a simple adaptation of the iterative rounding algorithm for maximum matching presented in [48].

Assuming the while loop works correctly, it is easy to see any extreme point of (**BD-Y-join LP**) is integral: because of the updates we make after adding an edge e to J with $x_e = 1$, the vector x_{res} obtained from x by removing the entry corresponding to e is an extreme point of the updated (**BD-Y-join LP**). Therefore, by induction on $|E|$, we can assume x_{res} is integral which implies x is integral.

Algorithm 9 An iterative algorithm for bounded-degree Y-join problem

Input: Undirected graph $G = (V, E)$ with edge costs $c_e \geq 0$ for all $e \in E$, degree bounds $b_v \geq 0$ for all $v \in V$, and $Y \subseteq V$ where $|Y|$ even.
Output: A Y-join.

$J \leftarrow \emptyset$
while $E \neq \emptyset$ **do**
Find an optimal extreme point solution x to (**BD-Y-join LP**) defined on $G = (V, E)$. Find an edge $e = uv$ with either $x_e = 0$ or $x_e = 1$ (cf. Lemma 80). In the former case remove e, and in the latter case add e to J and do the following:
Update $Y \leftarrow Y \Delta \{u, v\}$. Update $E \leftarrow E \setminus \{e\}$. Update $b(v) \leftarrow b(v) - 1$, and $b(u) \leftarrow b(u) - 1$. Remove all the cut constraints for all sets S such that $|S \cap \{u, v\}| = 1$.
output J.

Next we prove the algorithm works correctly, i.e., there is an integral variable at each iteration. Before proving this, we need some properties of an extreme point solution of (**BD-Y-join LP**) which follows from the Rank Lemma (see Lemma 1) and standard uncrossing techniques.

Lemma 77 (Properties of an extreme point)**.** *Consider an extreme point x and suppose $0 < x_e < 1$ for all $e \in E$. Then, there exists a laminar family $\mathcal{L}$ of Y-odd sets and $W \subseteq V$ such that*

(i) For any $S \in \mathcal{L}$ we have $x(\delta(S)) = 1$ and for any $v \in W$ we have $x(\delta(v)) = b(v)$.

(ii) The vectors in $\{\chi(\delta(S)) : S \in \mathcal{L}\} \cup \{\chi(\delta(v)) : v \in W\}$ are linearly independent.

(iii) $|\mathcal{L}| + |W| = |E|$.

(iv) $E[S]$ is connected for each $S \in \mathcal{L}$.

Proof. Properties (i)-(iii) follow from a standard application of uncrossing technique and Rank Lemma; however, we present the proof here for completeness. We recommend reading Section 2.4 for reader unfamiliar with laminar families.

Let $\mathcal{F}$ be the collection of Y-odd sets S such that the corresponding constraint in the LP is tight with respect to x, i.e., $x(\delta(S)) = 1$. It is easy to show if A and B are both Y-odd sets, then either $A \cup B$ and $A \cap B$ are Y-odd sets or $A \setminus B$ and $B \setminus A$ are Y-odd sets. Another known fact is that $\delta(.)$ is a strong submodular function:

$$\delta(A) + \delta(B) \geq \delta(A \cup B) + \delta(A \cap B)$$
$$\delta(A) + \delta(B) \geq \delta(A \setminus B) + \delta(B \setminus A).$$

The proof of above inequalities are standard and can be found for example in Chapter 2 of [48].

Claim 78. *Let $A, B \in \mathcal{F}$. Then, either $A \cup B, A \cap B \in \mathcal{F}$ or $A \setminus B, B \setminus A \in \mathcal{F}$. Furthermore, in case $A \cup B, A \cap B \in \mathcal{F}$ then $\chi(\delta(A)) + \chi(\delta(B)) = \chi(\delta(A \cup B)) + \chi(\delta(A \cap B))$, and in the case $A \setminus B, B \setminus A \in \mathcal{F}$ then $\chi(\delta(A)) + \chi(\delta(B)) = \chi(\delta(A \setminus B)) + \chi(\delta(B \setminus A))$.*

Proof. Assume $A \cup B$ and $A \cap B$ are Y-odd sets (if not then $A \setminus B$ and $B \setminus A$ are Y-odd sets and the proof follows identically using these sets instead). From submodularity of $\delta(.)$ we have

$$\begin{aligned} 1 + 1 &= x(\delta(A)) + x(\delta(B)) \\ &\geq x(\delta(A \cup B)) + x(\delta(A \cap B)) \\ &\geq 1 + 1. \end{aligned}$$

So the above inequalities hold with equality. In particular, we conclude $x(\delta(A \cup B)) = 1$ and $x(\delta(A \cap B)) = 1$ which implies $A \cup B, A \cap B \in \mathcal{F}$. Also we have $x(\delta(A)) + x(\delta(B)) = x(\delta(A \cup B)) + x(\delta(A \cap B))$ which implies the furthermore part since $\delta(A \cup B) \cup \delta(A \cap B) \subseteq \delta(A) \cup \delta(B)$. □

Denote by $span(\mathcal{F})$ the vector space generated by the set of vectors $\{\chi(\delta(S)) : S \in \mathcal{F}\}$.

Claim 79. *Let $\mathcal{L}'$ be a maximal laminar subfamily of $\mathcal{F}$. Then $span(\mathcal{L}') = span(\mathcal{F})$.*

Proof. Suppose not. So $span(\mathcal{L}') \subsetneq span(\mathcal{F})$. For any $S \notin \mathcal{L}'$ define $intersect(S, \mathcal{L}')$ to be the number of sets in $\mathcal{L}'$ that are crossing S. Since $span(\mathcal{L}') \subsetneq span(\mathcal{F})$, there exists a set $A \in \mathcal{F}$ with $\chi(\delta(A)) \notin span(\mathcal{L}')$. Choose such set A with minimum $intersect(A, \mathcal{L}')$. Let $B \in span(\mathcal{L}')$ such that A and B are crossing (such B exists otherwise $\mathcal{L}' \cup \{A\}$ is laminar, contradicting the maximality of $\mathcal{L}$). By Claim 78 either $A \cup B, A \cap B \in \mathcal{F}$ or $A \setminus B, B \setminus A \in \mathcal{L}'$. Suppose the former case holds (the argument for the other case is identical). Note both $intersect(A \cup B, \mathcal{L}')$ and $intersect(A \cap B, \mathcal{L}')$ are smaller than $intersect(A, \mathcal{L}')$ because for example $A \cup B$ intersects the same sets that A intersects except B since $B \in \mathcal{L}'$. By the choice of A, we conclude $A \cup B, A \cap B \in \mathcal{L}'$. Applying Claim 78 we get

$$\chi(\delta(A)) + \chi(\delta(B)) = \chi(\delta(A \cup B)) + \chi(\delta(A \cap B))$$

which implies $\chi(\delta(A)) \in span(\mathcal{L}')$, a contradiction. □

Now properties (i)-(iii) follow easily. Take a maximal subset $\mathcal{L} \subseteq \mathcal{L}'$ such that $\chi(\delta(S))$'s for all $S \in \mathcal{L}$ are linearly independent. Take a maximal subset $W \subseteq V$ such that $x(\delta(v)) = 1$ for all $v \in W$ and $\{\chi(S) : S \in \mathcal{L} \text{ or } S \in W\}$ are linearly independent. Note Claim 79 implies $\{\chi(S) : S \in \mathcal{L} \text{ or } S \in W\}$ correspond to a maximal number of linearly independent tight constraints. So we can apply Rank Lemma and conclude $|\{\chi(S) : S \in \mathcal{L} \text{ or } S \in W\}| = |E|$. This finishes the proof of properties (i)-(iii).

We show that one can further modify the laminar family to obtain a laminar family with property (iv). Consider $S \in \mathcal{L}$ and suppose $E[S]$ is not connected. Since $|S \cap Y|$ is *odd*, there must be a connected component C of $E[S]$ such that $|C \cap Y|$ is *odd*. Because C is a Y-odd set we have $x(\delta(C)) \geq 1$. Also note since C is a connected component of $E[S]$ we have $\delta(C) \subseteq \delta(S)$. Putting these two facts together, we conclude C is a tight set. Now consider the laminar family $(\mathcal{L} \setminus \{S\}) \cup \{C\}$. Note $span((\mathcal{L} \setminus \{S\}) \cup \{C\}) = span(\mathcal{L})$. Repeating this procedure until there is no set in the laminar family that violates property (iv) finishes the proof. □

Now we are ready to prove the algorithm works correctly.

Lemma 80. *Given any extreme point x of* **BD-Y-join LP** *there must exist an edge e with $x_e = 0$ or $x_e = 1$.*

Proof. Suppose not. So we have $0 < x_e < 1$ for each $e \in E$. Let $\mathcal{L}$ be a laminar family and let W be a subset of V that satisfy properties (i)-(iv) in Lemma 77. We show a contradiction with property (iii) using a token-based argument. Let $\mathcal{L}' := \mathcal{L} \cup W$ be the extended laminar family. We assign one token to each edge in the support of x (i.e., edges with positive x-value). Then we distribute the tokens inductively among the sets in the laminar family such that each member of $\mathcal{L}'$ receives at least one token and we show there are some extra tokens left which shows the contradiction with the fact that $|E| = |\mathcal{L}'|$.

We use the natural forest structure that the laminar family $\mathcal{L}'$ imposes, recall that each component of this forest is a rooted tree. We use the following claim to redistribute the tokens of $E[S]$ among the laminar sets inside S. By property (iv) of Lemma 77 each child of S is a connected component in G so we can contract each child to a singleton and name the contracted vertex the same as the child's label. Denote the resulting graph G_S. We say an edge $e = (u, v)$ is an induced edge in S, if S is the smallest set in $\mathcal{L}$ that contains both u and v.

In the next claim, we show the number of induced edges in S is at least the number of children of S in the forest representation of $\mathcal{L}'$. Hence, we can redistribute the tokens of the induced edges in S to the children of S so that every child gets one token. Then, we show every root in the forest representation of $\mathcal{L}'$ gets one unassigned token as well, and there is at least one unassigned token left at the end.

Claim 81. *Let S be a tight set in $\mathcal{L}$. Then, $|E(G_S)| \geq |V(G_S)|$, i.e., the number of induced edges in S is at least the number of children of S in the forest representation of $\mathcal{L}'$.*

Proof. Let $R_1, ..., R_k$ be the children of S. By property (iv) of Lemma 77, G_S is a connected graph and if it is not a tree then $E(G_S) \geq k$ and we can assign

the tokens of these (at least) k edges in $E(G_S)$ to $R_1, ..., R_k$ and we are done. So suppose G_S is a tree.

$\sum_{i=1}^{k} x(\delta(R_i)) = x(\delta(S)) + 2E[G_S]$ and since $x(\delta(S)) = 1$, we must have $\sum_{i=1}^{k} x(\delta(R_i))$ is odd. Since G_S is a tree so it is a bipartite graph. Let V_1 and V_2 be the two parts of G_S and since $\sum_{i=1}^{k} x(\delta(R_i))$ is *odd*, w.l.o.g., assume $\sum_{R_i \in V_1} x(\delta(R_i)) \leq \frac{\sum_{i=1}^{k} x(\delta(R_i))-1}{2}$. From this inequality and the fact that $x(\delta(S)) = 1$ we get

$$\frac{\sum_{i=1}^{k} x(\delta(R_i)) - 1}{2} = x(E(G_S)) \leq \sum_{R_i \in V_1} x(\delta(R_i)) \leq \frac{\sum_{i=1}^{k} x(\delta(R_i)) - 1}{2}. \quad (5.23)$$

So all the inequalities in (5.23) must hold as equality. Therefore, $x(E(G_S)) = \sum_{R_i \in V_1} x(\delta(R_i))$ which implies $\chi(\delta(S)) = \sum_{R_i \in V_2} \chi(\delta(R_i)) - \sum_{R_i \in V_1} \chi(\delta(R_i))$ and this contradicts the linear independence of the characteristic vectors in $\mathcal{L}'$. This finishes the proof of the claim. □

We continue the proof of Lemma 80. Let $S' \in \mathcal{L}'$ and correspond to a non-root vertex in the forest representation. Let $S \in \mathcal{L}$ be the tight set that is the parent of S'. By Claim 81 we can assign a token of an induced edge in S to S'. Note that the tokens of the induced edges in S are only used for the children of S, therefore no token is assigned to more than one tight set/vertex.

Next, we need to show every root nodes of the forest representation of $\mathcal{L}'$ gets one unassigned token. Let $S_1, ..., S_k$ be the root nodes of the forest. Denote by G_R the graph obtained from contracting $E[S_i]$'s. Note that all the tokens of edges in $E(G_R)$ are unassigned so far by our token assignment to the non-root vertices.

Since $x_e < 1$, we have $|\delta(S_i)| \geq 2$. We show that at least one root node has degree at least 3. But first let us show if this holds then Lemma 80 follows. Since one non-root node has degree at least 3, we have $|E(G_R)| \geq k + 1$ which implies we can assign one token to each root node and still have at least

one token left unassigned. This implies $|E(G)| > |\mathcal{L}'|$, a contradiction with property (iii) of Lemma 77. Therefore, there must be an edge e with $x_e = 0$ or $x_e = 1$.

Now it remains to prove at least one root node has degree at least 3. Suppose not. Note $x(\delta(S_i)) = 1$ for all the root nodes otherwise there is a root node of degree at least 3. Since b_v is odd if and only if $v \in Y$, every S_i is a Y-odd set. Consider a connected component of G_R consisting of $S_{i_1}, \ldots, S_{i_\ell}$. Note ℓ must be even otherwise $\cup_{j=1}^{\ell} S_{i_j}$ is a Y-odd set so $x(\delta(\cup_{j=1}^{\ell} S_{i_j})) \geq 1$, a contradiction since $\cup_{j=1}^{\ell} S_{i_j}$ is a connected component of G_R. Hence, ℓ is even which implies $\sum_{i=1}^{\ell} (-1)^i \chi(\delta(S_i)) = 0$, a contradiction with the linear independence of characteristic vectors in $\mathcal{L}'$. □

Chapter 6

Conclusion

In this thesis we studied some problems in network design, namely DIRECTED STEINER TREE in different special cases and the TRAVELING SALESMAN problem in the presence of degree bounds. We designed approximation algorithms and gave upper bounds on the integrality gaps of the natural LP relaxations for some of the above problems. In this chapter, we discuss related open questions and possible further research directions.

6.1 DIRECTED STEINER TREE

In Chapter 3, we gave an $O(\log k)$-approximation for DST on planar graphs. However, there is no known lower bound to rule out a PTAS for this problem. Our first big open problem is to understand the approximability of planar DST better.

Question 3. *Better understanding of approximability of DST on planar graphs. More specifically, is there a PTAS or a constant factor approximation for DST on planar graphs?*

Even obtaining a QPTAS for planar DST is very interesting. One might try to use some of the ideas in the recent paper by Cohen-Addad [13] for UNDIRECTED STEINER TREE[1]. The high-level idea in [13] is the following. If the diameter of the graph is small compared to opt, then apply Thorup's

[1] Although there is a PTAS for UNDIRECTED STEINER TREE [6], Cohen-Addad provide a general framework to obtain QPTAS for many connectivity problems in minor-free graphs.

shortest path separator and obtain smaller subinstances. Otherwise, apply decomposition techniques (e.g. [2]) to obtain connected components with smaller diameter. The latter step requires some guessing which leads to quasi-polynomial running time. For planar DST, one could try the above two steps approach. The first step carries over easily; however, it is not trivial how to apply the the second step in the case of planar DST. Although there are results (e.g. [41]) that reduces the diameter of a planar directed graph by removing some edges, it does not give a "nice" decompositioning of the graph.

Another possible direction is to extend our result for DST on planar graphs to minor-free graphs. However, as pointed out in [1], [13], minor-free (undirected) graphs do not have shortest-path separators. In [13], Cohen-Addad bypassed this difficulty by designing a new separator called a *mixed separator* for undirected minor-free graphs. It is not clear that analogous separators exist in directed graphs. For example, the mixed separators in [13] are obtained, in part, by contracting certain paths. These paths are obtained using structural results in minor-free graphs [61] and it is not clear how to find analogous paths in the directed case.

Question 4. *What is the approximability of DST on graphs excluding a fixed minor?*

In Chapter 4, we showed if we further restrict the planar DST to quasi-bipartite instances, then we can get a constant factor approximation. Our result extends to minor-free graphs as well. Our approach was based on a primal-dual framework and it bounds the integrality gap of the classical LP relaxation for DST too. However, the primal-dual algorithm we presented here was rather "non-standard": we consider different buckets for an edge and fill these buckets with different rates. This leads to violation of dual constraints but we showed the violation can be bounded by a constant. This framework could have the potential to be applied to other problems.

Question 5. *What are the other applications of our primal-dual framework with*

multiple buckets presented in Chapter 4?

6.2 BOUNDED-DEGREE TRAVELING SALESMAN

In Chapter 5, we presented a $(\frac{5}{3}, +4)$, a $(\frac{13}{8}, +6)$ and a $(\frac{3}{2}, +8)$ approximations for the BOUNDED-DEGREE SUBSET TRAVELING SALESMAN problem. It would be interesting to see if there is any $O(1)$-approximation that violates the degree bounds by at most $+2$. On the other hand, a demonstration that any $O(1)$-approximation algorithm based on the natural LP relaxation requires a $+4$ violation on the degree bounds would be interesting as well.

Question 6. *Can we get an $(O(1), +2)$-approximation or show $+4$ on the degree violation is a lower bound using the natural LP relaxation of BDSTSP?*

Another interesting question to explore is a closely related problem to TSP but in the presence of degree bounds. Namely, the BOUNDED-DEGREE TRAVELING SALESMAN PATH problem (BDTSPP) where the input is an undirected graph $G = (V, E)$, a pair of vertices s and t, non-negative edge costs $c_e \geq 0$ for all $e \in E$, and non-negative integer degree bounds $b_v \geq 0$ for all $v \in V$. The goal is to find a cheapest walk spanning V which starts at s and finishes at t.

Given our approximation algorithms in Chapter 5, there is a similar result for BDTSPP. Let x^* be an optimal solution of (**BDTSPP-LP**), the natural LP relaxation for BDTSPP. Then, x^* is feasible for Bounded-degree Steiner tree polytope, and using [49] we get a spanning tree T with cost at most $2 \cdot \text{cost}(x^*)$ and $d_T(v) \leq b_v + 3$. We need to find a cheap join to fix the degree parity. One can show $\frac{x^*}{3} + \frac{\chi_T}{3}$ is feasible for the bounded-degree $(odd(T)\Delta\{s, t\})$-join polytope[2]. So overall, we have a solution $\mathcal{P}$ whose cost is at most $\frac{8}{3} \cdot \text{cost}(x^*)$ and $d_{\mathcal{P}}(v) \leq \frac{5}{3} \cdot b_v + 4$.

Note that we could not use $\frac{x^*}{2}$ to get a cheaper tree with less degree violation as we did for BDTSP because of constraint (6.1).

[2]This is essentially the same argument in [3] used to analyze Hoogeveen's TSP-path algorithm.

$$
\begin{aligned}
\text{minimize:} \quad & \sum_{e \in E} c_e \cdot x_e & & & \textbf{(BDTSPP-LP)} \\
\text{subject to:} \quad & x(\delta(S)) \geq & 2 \quad & \forall\, \emptyset \neq |S \cap \{s,t\}| = 0 \text{ or } 2,\ S \neq V & \\
& x(\delta(S)) \geq & 1 \quad & \forall\, |S \cap \{s,t\}| = 1 & (6.1) \\
& x(\delta(v)) \leq & b_v \quad & \forall\, v \in V & \\
& 0 \leq x_e \leq & 2 \quad & \forall\, e \in E &
\end{aligned}
$$

As before one could generalize BDTSPP so we only need to visit a certain subset of vertices in which we call the BOUNDED-DEGREE SUBSET TRAVELING SALESMAN PATH problem (BDSTSPP). The same approach for BDTSPP works here with the same approximation guarantee. So our last question is to investigate the approximability of these problems.

Question 7. *Can we do better than the trivial approximation algorithms for BDTSPP and BDSTSPP?*

References

[1] I. Abraham and C. Gavoille, "Object location using path separators," in *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, 2006, pp. 188–197.

[2] I. Abraham, C. Gavoille, D. Malkhi, and U. Wieder, "Strong-diameter decompositions of minor free graphs," in *Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures*, 2007, pp. 16–24.

[3] H.-C. An, R. Kleinberg, and D. B. Shmoys, "Improving Christofides' algorithm for the st path TSP," *Journal of the ACM (JACM)*, vol. 62, no. 5, pp. 1–28, 2015.

[4] M. Bateni, M. Hajiaghayi, and D. Marx, "Approximation schemes for Steiner forest on planar graphs and graphs of bounded treewidth," *Journal of the ACM (JACM)*, vol. 58, no. 5, pp. 1–37, 2011.

[5] M. Bern and P. Plassmann, "The Steiner problem with edge lengths 1 and 2," *Information Processing Letters*, vol. 32, no. 4, pp. 171–176, 1989.

[6] G. Borradaile, P. Klein, and C. Mathieu, "An o(n log n)-approximation scheme for Steiner tree in planar graphs," *ACM Transactions on Algorithms (TALG)*, vol. 5, no. 3, pp. 1–31, 2009.

[7] J. Byrka, F. Grandoni, T. Rothvoß, and L. Sanità, "Steiner tree approximation via iterative randomized rounding," *Journal of the ACM (JACM)*, vol. 60, no. 1, pp. 1–33, 2013.

[8] G. Calinescu and A. Zelikovsky, "The polymatroid Steiner problems," *J. Combonatorial Optimization*, vol. 33, no. 3, pp. 281–294, 2005.

[9] D. Chakrabarty, N. R. Devanur, and V. V. Vazirani, "New geometry-inspired relaxations and algorithms for the metric Steiner tree problem," *Mathematical programming*, vol. 130, no. 1, pp. 1–32, 2011.

[10] C.-H. Chan, B. Laekhanukit, H.-T. Wei, and Y. Zhang, "Polylogarithmic approximation algorithm for k-connected directed Steiner tree on quasi-bipartite graphs," *arXiv preprint arXiv:1911.09150*, 2019.

[11] M. Charikar, C. Chekuri, T.-Y. Cheung, *et al.*, "Approximation algorithms for directed Steiner problems," *Journal of Algorithms*, vol. 33, no. 1, pp. 73–91, 1999.

[12] N. Christofides, "Worst-case analysis of a new heuristic for the travelling salesman problem," Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, Tech. Rep., 1976.

[13] V. Cohen-Addad, "Bypassing the surface embedding: Approximation schemes for network design in minor-free graphs," in *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, 2022, pp. 343–356.

[14] E. D. Demaine, M. Hajiaghayi, and P. N. Klein, "Node-weighted Steiner tree and group Steiner tree in planar graphs," *ACM Transactions on Algorithms (TALG)*, vol. 10, no. 3, pp. 1–20, 2014.

[15] I. Dinur and D. Steurer, "Analytical approach to parallel repetition," in *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, 2014, pp. 624–633.

[16] J. Edmonds, "Optimum branchings," *Journal of Research of the national Bureau of Standards B*, vol. 71, no. 4, pp. 233–240, 1967.

[17] J. Edmonds and E. L. Johnson, "Matching, Euler tours and the chinese postman," *Mathematical programming*, vol. 5, pp. 88–124, 1973.

[18] U. Feige, "A threshold of ln n for approximating set cover," *Journal of the ACM (JACM)*, vol. 45, no. 4, pp. 634–652, 1998.

[19] A. E. Feldmann, J. Könemann, N. Olver, and L. Sanità, "On the equivalence of the bidirected and hypergraphic relaxations for Steiner tree," *Mathematical programming*, vol. 160, no. 1, pp. 379–406, 2016.

[20] A. Frank, "On a theorem of mader," *Discret. Math.*, vol. 101, pp. 49–57, 1992.

[21] Z. Friggstad, J. Könemann, Y. Kun-Ko, A. Louis, M. Shadravan, and M. Tulsiani, "Linear programming hierarchies suffice for directed Steiner tree," in *International Conference on Integer Programming and Combinatorial Optimization*, Springer, 2014, pp. 285–296.

[22] Z. Friggstad, J. Könemann, and M. Shadravan, "A logarithmic integrality gap bound for directed Steiner tree in quasi-bipartite graphs," in *15th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2016)*, R. Pagh, Ed., ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 53, Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016, 3:1–3:11, ISBN: 978-3-95977-011-8.

[23] Z. Friggstad and R. Mousavi, "A constant-factor approximation for quasi-bipartite directed Steiner tree on minor-free graphs," *arXiv preprint arXiv:2111.02572*, 2021.

[24] Z. Friggstad and R. Mousavi, "Bi-criteria approximation algorithms for bounded-degree subset TSP," in *33rd International Symposium on Algorithms and Computation (ISAAC 2022)*, Schloss Dagstuhl-Leibniz-Zentrum fur Informatik, 2022.

[25] Z. Friggstad and R. Mousavi, "An o (log k)-approximation for directed Steiner tree in planar graphs," *arXiv preprint arXiv:2302.04747*, 2023.

[26] T. Fukunaga, Z. Nutov, and R. Ravi, "Iterative rounding approximation algorithms for degree-bounded node-connectivity network design," *SIAM Journal on Computing*, vol. 44, no. 5, pp. 1202–1229, 2015.

[27] M. R. Garey and D. S. Johnson, "The rectilinear Steiner tree problem is NP-complete," *SIAM Journal on Applied Mathematics*, vol. 32, no. 4, pp. 826–834, 1977.

[28] R. Ghuge and V. 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*, SIAM, 2020, pp. 1039–1048.

[29] M. X. Goemans, "Minimum bounded degree spanning trees," in *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, IEEE, 2006, pp. 273–282.

[30] M. X. Goemans, N. Olver, T. Rothvoß, and R. Zenklusen, "Matroids and integrality gaps for hypergraphic Steiner tree relaxations," in *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, 2012, pp. 1161–1176.

[31] M. X. Goemans and D. P. Williamson, "The primal-dual method for approximation algorithms and its application to network design problems," *Approximation algorithms for NP-hard problems*, pp. 144–191, 1997.

[32] F. Grandoni, B. Laekhanukit, and S. Li, "O(log 2 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*, 2019, pp. 253–264.

[33] M. Grötschel, L. Lovász, and A. Schrijver, "The ellipsoid method and its consequences in combinatorial optimization," *Combinatorica*, vol. 1, pp. 169–197, 1981.

[34] S. Guha, A. Moss, J. Naor, and B. Schieber, "Efficient recovery from power outage," in *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, 1999, pp. 574–582.

[35] E. Halperin and R. Krauthgamer, "Polylogarithmic inapproximability," in *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, 2003, pp. 585–594.

[36] T. Hibi and T. Fujito, "Multi-rooted greedy approximation of directed Steiner trees with applications," in *International Workshop on Graph-Theoretic Concepts in Computer Science*, Springer, 2012, pp. 215–224.

[37] K. Jain, "A factor 2 approximation algorithm for the generalized Steiner network problem," *Combinatorica*, vol. 21, pp. 39–60, 2001.

[38] A. R. Karlin, N. Klein, and S. O. Gharan, "A (slightly) improved approximation algorithm for metric TSP," *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, 2021.

[39] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*, Springer, 1972, pp. 85–103.

[40] M. Karpinski and A. Zelikovsky, "New approximation algorithms for the Steiner tree problems," *Journal of Combinatorial Optimization*, vol. 1, no. 1, pp. 47–65, 1997.

[41] K.-i. Kawarabayashi and A. Sidiropoulos, "Embeddings of planar quasimetrics into directed l1 and polylogarithmic approximation for directed sparsest-cut," in *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, IEEE, 2022, pp. 480–491.

[42] L. G. Khachiyan, "A polynomial algorithm in linear programming," in *Doklady Akademii Nauk*, Russian Academy of Sciences, vol. 244, 1979, pp. 1093–1096.

[43] R. Khandekar, G. Kortsarz, and Z. Nutov, "On some network design problems with degree constraints," *Journal of Computer and System Sciences*, vol. 79, no. 5, pp. 725–736, 2013.

[44] J. Könemann, S. Sadeghian, and L. 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*, IEEE, 2013, pp. 568–577.

[45] B. H. Korte, J. Vygen, B. Korte, and J. Vygen, *Combinatorial optimization*. Springer, 2011, vol. 1.

[46] C. Kuratowski, "Sur le probleme des courbes gauches en topologie," *Fundamenta mathematicae*, vol. 15, no. 1, pp. 271–283, 1930.

[47] L. C. Lau, J. Naor, M. R. Salavatipour, and M. Singh, "Survivable network design with degree or order constraints," *SIAM Journal on Computing*, vol. 39, no. 3, pp. 1062–1087, 2009.

[48] L. C. Lau, R. Ravi, and M. Singh, *Iterative methods in combinatorial optimization*. Cambridge University Press, 2011, vol. 46.

[49] L. C. Lau and M. Singh, "Additive approximation for bounded degree survivable network design," *SIAM Journal on Computing*, vol. 42, no. 6, pp. 2217–2242, 2013.

[50] L. C. Lau and H. Zhou, "A unified algorithm for degree bounded survivable network design," *Mathematical Programming*, vol. 154, no. 1, pp. 515–532, 2015.

[51] S. Li and B. Laekhanukit, "Polynomial integrality gap of flow lp for directed Steiner tree," *arXiv preprint arXiv:2110.13350*, 2021.

[52] R. J. Lipton and R. E. Tarjan, "A separator theorem for planar graphs," *SIAM Journal on Applied Mathematics*, vol. 36, no. 2, pp. 177–189, 1979.

[53] R. J. Lipton and R. E. Tarjan, "Applications of a planar separator theorem," *SIAM journal on computing*, vol. 9, no. 3, pp. 615–627, 1980.

[54] A. Louis and N. K. Vishnoi, "Improved algorithm for degree bounded survivable network design problem," in *Scandinavian Workshop on Algorithm Theory*, Springer, 2010, pp. 408–419.

[55] W. Mader, "A reduction method for edge-connectivity in graphs," *Annals of discrete mathematics*, vol. 3, pp. 145–164, 1978.

[56] C. Moldenhauer, "Primal-dual approximation algorithms for node-weighted Steiner forest on planar graphs," *Information and Computation*, vol. 222, pp. 293–306, 2013.

[57] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[58] I. Post and C. Swamy, "Linear programming-based approximation algorithms for multi-vehicle minimum latency problems," in *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SIAM, 2014, pp. 512–531.

[59] H. J. Prömel and A. Steger, "A new approximation algorithm for the Steiner tree problem with performance ratio 5/3," *Journal of Algorithms*, vol. 36, no. 1, pp. 89–101, 2000.

[60] S. Rajagopalan and V. V. Vazirani, "On the bidirected cut relaxation for the metric Steiner tree problem.," in *SODA*, vol. 99, 1999, pp. 742–751.

[61] N. Robertson and P. D. Seymour, "Graph minors. xvi. excluding a non-planar graph," *Journal of Combinatorial Theory, Series B*, vol. 89, no. 1, pp. 43–76, 2003.

[62] G. Robins and A. Zelikovsky, "Tighter bounds for graph Steiner tree approximation," *SIAM Journal on Discrete Mathematics*, vol. 19, no. 1, pp. 122–134, 2005.

[63] T. Rothvoß, "Directed Steiner tree and the Lasserre hierarchy," *arXiv preprint arXiv:1111.5473*, 2011.

[64] A. Schrijver, *Combinatorial optimization: polyhedra and efficiency*. Springer Science & Business Media, 2003, vol. 24.

[65] A. Serdyukov, "O nekotorykh ekstremal'nykh obkhodakh v grafakh," *Upravlyayemyye sistemy*, vol. 17, pp. 76–79, 1978.

[66] M. Singh and L. C. Lau, "Approximating minimum bounded degree spanning trees to within one of optimal," *Journal of the ACM (JACM)*, vol. 62, no. 1, pp. 1–19, 2015.

[67] A. Thomason, "An extremal function for contractions of graphs," in *Mathematical Proceedings of the Cambridge Philosophical Society*, Cambridge University Press, vol. 95, 1984, pp. 261–265.

[68] M. Thorup, "Compact oracles for reachability and approximate distances in planar digraphs," *Journal of the ACM (JACM)*, vol. 51, no. 6, pp. 993–1024, 2004.

[69] K. Wagner, "About a property of plane complexes," *Mathematical Annals*, vol. 114, no. 1, pp. 570–590, 1937.

[70] L. A. Wolsey, "Heuristic analysis, linear programming and branch and bound," in *Combinatorial Optimization II*, Springer, 1980, pp. 121–134.

[71] A. Zelikovsky, "A series of approximation algorithms for the acyclic directed Steiner tree problem," *Algorithmica*, vol. 18, no. 1, pp. 99–110, 1997.

[72] A. Z. Zelikovsky, "An 11/6-approximation algorithm for the network Steiner problem," *Algorithmica*, vol. 9, no. 5, pp. 463–470, 1993.

[73] L. Zosin and S. Khuller, "On directed Steiner trees," in *SODA*, Citeseer, vol. 2, 2002, pp. 59–63.