Computer Science is no more about computers than astronomy is about telescopes

– E. W. Dijkstra.

University of Alberta

Improved approximation algorithms for Min-Max Tree Cover, Bounded Tree Cover, Shallow-Light and Buy-at-Bulk k-Steiner Tree, and $(k, 2)$-Subgraph

by

Mohammad Reza Khani

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

To my mother
Whom my calmness coming from

Abstract

In this thesis we provide improved approximation algorithms for the Min-Max k-Tree Cover, Bounded Tree Cover and Shallow-Light k-Steiner Tree, $(k, 2)$-subgraph problems.

In Chapter 2 we consider the Min-Max k-Tree Cover (MMkTC). Given a graph $G = (V, E)$ with weights $w : E \to \mathbb{Z}^+$, a set $T_1, T_2, \dots, T_k$ of subtrees of G is called a tree cover of G if $V = \bigcup_{i=1}^{k} V(T_i)$. In the MM$k$TC problem we are given graph G and a positive integer k and the goal is to find a tree cover with k trees, such that the weight of the largest tree in the cover is minimized. We present a 3-approximation algorithm for MMkTC improving the two different approximation algorithms presented in [7, 46] with ratios 4 and $(4 + \epsilon)$. The problem is known to have an APX-hardness lower bound of $\frac{3}{2}$ [125].

In Chapter 3 we consider the Bounded Tree Cover (BTC) problem. In the BTC problem we are given a graph G and a bound λ and the goal is to find a tree cover with minimum number of trees such that each tree has weight at most λ. We present a 2.5-approximation algorithm for BTC, improving the 3-approximation bound in [7].

In Chapter 4 we consider the Shallow-Light k-Steiner Tree (SLkST) problem. In the bounded-diameter or shallow-light k-Steiner tree problem , we are given a graph $G = (V, E)$ with terminals $T \subseteq V$ containing a root $r \in T$, a cost function $c : E \to \mathbb{Q}^+$, a length function $\ell : E \to \mathbb{Q}^+$, a bound $L > 0$ and an integer $k \geq 1$. The goal is to find a minimum c-cost r-rooted Steiner tree containing at least k terminals whose diameter under ℓ metric is at most L. The input to the Buy-at-Bulk k-Steiner tree problem (BBkST) is similar: graph $G = (V, E)$, terminals $T \subseteq V$, cost and length functions $c, \ell : E \to \mathbb{Q}^+$, and an integer $k \geq 1$. The goal is to find a minimum total cost r-rooted Steiner tree H containing at least k terminals, where the cost of each edge e is $c(e) + \ell(e) \cdot f(e)$ where $f(e)$ denotes the number of terminals whose path to root in H contains edge e. We present a bicriteria $(O(\log^2 n), O(\log n))$-approximation for SLkST: the algorithm finds a k-Steiner tree of diameter at most $O(L \cdot \log n)$ whose cost is at most $O(\log^2 n \cdot \textsc{opt}^*)$ where $\textsc{opt}^*$ is the cost of an LP relaxation of the problem. This improves on the algorithm of [66] (APPROX'06/Algorithmica'09) which had ratio $(O(\log^4 n), O(\log^2 n))$. Using this, we obtain an $O(\log^3 n)$-approximation for BBkST, which improves upon the $O(\log^4 n)$-approximation of [66]. Finally, we show our approximation algorithm for BBkST implies

approximation factors for some other network design problems.

In Chapter 5 we consider the problem of finding a minimum cost 2-edge-connected subgraph with at least k vertices, which is introduced as the $(k, 2)$-subgraph problem in [94] (STOC'07/SICOMP09). This generalizes some well-studied classical problems such as the k-MST and the minimum cost 2-edge-connected subgraph problems. We give an $O(\log n)$-approximation algorithm for this problem which improves upon the $O(\log^2 n)$-approximation of [94].

Acknowledgements

I would like to thank my supervisor Mohammad R. Salavatipour, who is the other author for all the results in this thesis. I am also grateful to him for his through review of the text, and suggestions regarding to the presentation of this work.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Problems considered and motivations

In several real world applications we are facing optimization problems in which we have to optimize (*e.g.* minimize) the cost of doing a task. In this thesis we study four different optimization problems motivated by their corresponding real world applications. We model each problem by a combinatorial optimization problem on a weighted graph. All of our combinatorial optimization problems are known to be NP-hard. Thus, instead of trying to find the optimum solution we seek for a solution that can be computed efficiently (*i.e.* polynomial time) while its cost is also guaranteed to be within a certain multiplicative factor of the optimum solution. More precisely, we try to find an α-approximation algorithm for our problems which gives a solution in a time polynomially bounded by the size of input whose cost is not worse than α times the optimum solution. Clearly, the closer α is to 1 the better the algorithm is. In this thesis, we improve the approximation ratios for four different problems.

Min-Max k-Tree Cover (MMkTC): Consider the problem of "Nurse station location" [46] in which a hospital wants to assign all its patients to a set of k nurses. Each nurse has to visit all her assigned patients every morning and return back to her station. Moreover, all the patients should be within a reasonable distance from their assigned nurse's station. The task is to find k locations for the nurses and distribute all the patients among them such that the distance travelled every morning by the nurse who travels the most is minimized.

We model the hospital as an undirected weighted graph $G = (V, E)$ in which each room in the hospital corresponds a node in the graph. Two nodes u and v are connected via an edge e with cost $c(e)$ if the corresponding rooms for u and v are close to each other in the hospital and the average travel time between them is $c(e)$. The coverage area for each nurse is modeled as a subtree in G covering all the nodes whose corresponding rooms have her patients. The optimization task is to find a set of k-subtrees that together cover all the nodes of G such that the cost of the most expensive one is minimized. The modeled combinatorial optimization problem is called Min-Max k-Tree Cover (MMkTC) and is proved to be NP-hard.

The previously known best approximation ratio for MMkTC was a 4-approximation algorithm due to Ravi *et al.* [46] and Arkin *et al.* [7]. In this thesis we provide a 3-approximation algorithm for MMkTC in Chapter 2. This result is published in [76].

Bounded Tree Cover (BTC): Consider the following scenario for the nurse station location problem introduced for MMkTC. The hospital wants to have a maximum bound λ on the coverage area of each nurse, *i.e.* the cost of each subtree has to be upper bounded by λ. The task is to find the minimum number of nurses required, such that the cost of the subtrees assigned to them do not exceed λ and together, they cover all the nodes.

In the Bounded Tree Cover (BTC) problem we are given an undirected weighted graph $G = (V, E)$ along with a bound λ, the task is to find the minimum number of subtrees in G such that the union of their nodes is V and the cost of none of them exceeds λ. This problem also has other applications in vehicle routing problems. For example, suppose graph G represents a map of locations in which each node is a customer needing a special service. We have vehicles of bounded fuel tank which can travel at most λ kilometers. The task is to find the minimum number of vehicles and assign the customers to them such that each vehicle can travel to all of its assigned customers and return back to its initial position. As finding tours for the vehicles is hard we usually estimate them with trees.

The best approximation algorithm for BTC before this work was due to Arkin *et al.* [7] with approximation ratio of 3. We give a 2.5-approximation algorithm for this problem in Chapter 3, which is published [76].

Shallow Light k-Steiner Tree (SLkST) and Buy-at-Bulk k-Steiner Tree (BBkST): Imagine a broadcast station (server) has to broadcast multimedia data to at least k of its customers. We refer to server, customers, and other intermediate transmitters as nodes in the network. Communication connections which have a communication delay can be established between each pair of nodes in the network with a cost of establishment. The task is to create a minimum cost network, containing the server and at least k customers such that the total delay seen by any customer is at most a given bound.

The corresponding graph theory problem is known as Shallow Light k-Steiner Tree (SLkST). In SLkST we are given the network as an undirected graph $G = (V, E)$ which has a cost $c(e)$ and a delay $l(e)$ on each edge e, a delay bound L, and integer k, a subset $T \subseteq V$ of terminals (customers), and a server $r \in T$. The objective is to find a subtree containing r and at least k terminals of T (at least $k-1$ terminals other than r) such that its cost regarding to the c metric is minimized and each terminal is not farther than L from r with respect to the l metric in the subtree.

The best previous result on SLkST was an $(O(\log^4 n), O(\log^2 n))$-bicriteria approximation algorithm [66]. The algorithm gives a tree in which each terminal is at most $O(\log^2 n) \cdot L$ away from the root and whose cost is at most $O(\log^4 n)$ times the optimum solution with bound L. In Chapter 4 we improve this result by presenting an $(O(\log^2 n), O(\log n))$-bicriteria approximation algorithm.

In Chapter 4 we also show how our result for SLkST can improve the approximation factor for the Buy-at-Bulk k-Steiner Tree (BBkST) problem. In BBkST we are given an undirected graph $G = (V, E)$ with a monotone nondecreasing cost function f_e for each $e \in E$, a set of terminals $T \subseteq V$ with demand δ_i for each $v_i \in T$, a root $r \in T$, and a positive integer k. The objective is to find a subtree H that contains r and at least $k-1$ other terminals from T and route all their demands from r such that $\sum_{e \in H} f(\delta_e)$ is minimized where δ_e is the total demand routed over edge e. The best previous approximation factor for BBkST was an $O(\log^3 n \cdot D)$-approximation in [66] which we improve to an $O(\log^2 n \cdot D)$-approximation algorithm where D is the total demand.

Note that we consider the most general form of cost scheme over the edges. This scheme can represent several realizations in the real world, *i.e.* cases where the cost for establishing and maintaining the cables between two nodes differ from one place to another, or the cost of cables capable of routing more loads are greater than the smaller cables (see *e.g.* Chapter 4). These results appear in [77].

$(k, 2)$-subgraph: Designing a reliable communication network which can continue to route the demands even if some of its connection edges are broken is an important problem in the network design. Reliable networks are also important for transshipment of crucial supplies. We model the network with a graph G, in which the transmitters are represented as nodes in G and cost of establishing a connection between each pair of transmitters are represented as the cost of its corresponding edge in G. We say a network is reliable if after deleting any edge it remains connected, *i.e.* G is 2-edge-connected.

We consider a problem called, $(k, 2)$-subgraph, in which for a given weighted graph G, and a positive integer k, we have to find a minimum cost subgraph which is 2-edge-connected and has at least k nodes of G. The best previous result was an $O(\log^2 n)$-approximation [94], which we improve to $O(\log n)$-approximation in Chapter 5.

1.2 Background

This section is mainly designated to introduce some notations used throughout the thesis. We start with defining a few graph theoretical concepts, then the formal definitions related to the approximation algorithm is given. After that we give an elementary introduction to linear programming, and finally we finish the section by giving the best approximation factor for the set cover problem which we use later in Chapter 5.

1.2.1 Graph theory

In this thesis we represent a graph G as an ordered pair $(V(G), E(G))$ in which $V(G)$ is the set of nodes and $E(G)$ is the set of edges. We simply show $V(G)$ as V and $E(G)$ as E when G is clear from the context. We show each edge $e \in E$ as (u, v) to specify that u and v are the *end-point*s of e. We consider only undirected graphs in this thesis. Thus, as G is undirected the existence of (u, v) implies the existence of (v, u) and vice versa. If G is a *weighted graph* then each edge $e = (u, v)$ has a cost shown as $c(e)$ or $c(u, v)$. If G is unweighted we assume $c(e) = 1, \ \forall e \in E$. In the following we explain some graph theory concepts used in this thesis:

- **Walk:** A walk in a graph G is a sequence of nodes $v_1, v_2, \ldots, v_k$ such that for each $1 \leq i \leq k-1$, edge (v_i, v_{i+1}) exists in G. The cost of the walk is $\sum_{i=1}^{k-1} c(v_i, v_{i+1})$.

- **Tour:** A tour $(v_1, v_2, \ldots, v_k)$ in G is a closed walk, in which $v_1 = v_k$.

- **Path:** Path is a walk $(v_1, v_2, \ldots, v_k)$, in which each v_i is distinct. The *shortest path* between u and v is a path in which the first node is u and the last node is v with the minimum cost.

- **Cycle:** Cycle is a tour with distinct nodes except $v_1 = v_k$.

- **Diameter:** Diameter of a weighted graph $G = (V, E)$ is $max_{u,v \in V} d(u, v)$ where $d(u, v)$ denotes the cost of the shortest path between u and v.

- **Tree:** Tree is a connected acyclic graph. It is easy to check that a tree T has exactly $|V(T)| - 1$ edges and there is a unique path connecting each pair of nodes in T.

- **Metric graph:** We call a complete graph G metric if the cost of edges satisfy the triangle inequality. More precisely, for any triple $u, v, w \in V$, $c(u, v) + c(v, w) \geq c(u, w)$. Usually graphs that model real world applications are metric. This comes from the fact that to go from a node u to another node v, one can take a shortest path between them. This model of distance measure is referred to as the shortest path metric completion of a graph. Graph $\hat{G}$ is called the shortest path metric completion of G if it has the same set of nodes as G and there is an edge between $\hat{u}, \hat{v} \in V(\hat{G})$ with cost $\hat{c}(\hat{u}, \hat{v}) = d(u, v)$ if u and v are connected in G with a shortest path with cost $d(u, v)$. It is easy to check that a shortest path metric completion of any graph is a metric and looking for the shortest tour covering all the nodes in G is equivalent to searching for the shortest cycle in $\hat{G}$ covering all the nodes.

- **Tree Cover:** A set $T_1, T_2, \ldots, T_k$ of subtrees of G is called a tree cover of G if every vertex of V appears in at least one T_i $(1 \leq i \leq k)$, i.e. $V = \bigcup_{i=1}^{k} V(T_i)$. Note that the trees in a tree-cover are not necessarily edge-disjoint (thus may share vertices too).

- **Matching:** A matching is a subset of edges $M \subseteq E$ such that no two edges in M share an endpoint. The cost of a matching M is $\sum_{e \in M} c(e)$. A maximum matching in G is a matching with maximum number of edges. A perfect matching is a matching with exactly $\frac{|V|}{2}$ edges. A min-cost maximum matching is a maximum matching with minimum cost and a min-cost perfect matching is a perfect matching with minimum cost.

- **λ-edge-connected graph:** A λ-edge-connected graph is a connected graph in which deleting any $\lambda - 1$ edges does not make it disconnected. It is easy to check that in a λ-edge-connected graph there are at least λ edge-disjoint paths between any pair of nodes.

1.2.2 Approximation algorithms

A decision problem Π is a problem whose answer is either "yes" or "no". Note that the description of the problem can suitably be presented as a string in the binary alphabet $\Sigma = \{0, 1\}$; so that each decision problem Π can be viewed as a language L_Π which consists of all the strings representing "yes" instances of Π. We say that algorithm *alg* can decide L_Π (or *solve* the problem Π) if for any $x \in \Sigma^*$ it can decide whether $x \in L_\Pi$ or $x \notin L_\Pi$. We refer to the number of bits in the string x as the input size which is denoted by $|x|$.

Suppose $DTIME(t)$ denotes class of problems that can be solved in deterministic time $O(t)$. Similarly ZPTIME(t) denotes class of problems that can be solved using a randomized algorithm whose expected running time is $O(t)$. In randomized algorithms we assume that they have access to a source of random bits.

Let $poly(n) = \cup_{k \geq 0} n^k$. The class of polynomial time solvable problems (or polynomial time decidable languages) P is defined as $P = \cup_{p \in poly(n)} DTIME(p)$ where n is the size of input. A *pseudo-polynomial time algorithm* is an algorithm that runs in time polynomial in the numeric value of the input. A *quasi-polynomial time algorithm* is an algorithm that solves the problem in $DTIME(n^{poly(\log n)})$. From now on, we refer to an algorithm as a *polynomial algorithm* if it runs in a time polynomial in the size of input.

A language (or problem) L is in NP if there exists a polynomial time algorithm M called *verifier* such that [122]:

- if $x \in L$ then there is a certificate (or solution) $y \in \Sigma^*$ with $|y| \in poly(|x|)$ such that $M(x, y)$ accepts x.

- if $x \notin L$ then for any $y \in \Sigma^*$ which $|y| \in poly(|x|)$, $M(x, y)$ rejects x.

An instance I of an *NP-optimization problem* consists of: (1) A set of feasible solutions $S(I)$, (2) a polynomial time computable function *Obj* that for a given $s \in S(I)$ assigns it a non-negative rational number. The objective is specified as either maximization or minimization which is finding a solution $s \in S(I)$ with minimum or maximum $Obj(s)$.

We focus on minimization problems in this thesis. We refer to an optimal solution (OPT) as a feasible solution with minimum value (OPT). For each NP-optimization problem we can define the corresponding decision problem by giving a bound B on its optimal value, as a result the decision problem will be a pair (I, B) and it is a "yes" instance if the optimal value is less than or equal to B and "no" otherwise. We can extend the NP-hard problems to the optimization problems if their corresponding decision problems are NP-hard.

Let $\delta : \mathbb{Z}^+ \rightarrow \mathbb{Q}^+$ be a function with $\delta \geq 1$ which maps the input size for an NP-optimization problem to a rational number. We say that algorithm $\mathcal{A}$ is a *δ-approximation algorithm* for the NP-optimization problem Π if for each instance I of Π, $\mathcal{A}$ produces a feasible solution $s \in S(I)$ such that $Obj(s) \leq \delta(|I|).\textsc{opt}(I)$. Note that in general the approximation factor can be dependant on the input size. As an example, a *poly-logarithmic-approximation algorithm* is an approximation algorithm with $\delta \in O(poly(\log n))$ where n is the size of input. By an γ-hardness factor based on a certain complexity assumption (such as $P \neq NP$) for an NP-optimization problem we mean that we cannot find an approximation algorithm with ratio better than γ unless that assumption is false.

For some algorithms, δ can be independent of the input size which means the approximation ratio will not grow for bigger problem instances. Clearly, the closer δ is to 1, the better approximation algorithm we have. Note that for NP-hard optimization problems we cannot achieve 1-approximation algorithms unless $P = NP$. Instead we try to find a $(1+\epsilon)$-approximation algorithm for a small positive ϵ. Interestingly enough, some NP-hard optimization problems admit $(1+\epsilon)$-approximation algorithm for an arbitrarily small but fixed $\epsilon > 0$. An algorithm $\mathcal{A}$ for an NP-optimization problem Π is called a *polynomial time approximation scheme (PTAS)* if for any given instance I and a fixed constant ϵ as input, $\mathcal{A}$ outputs a solution s such that $Obj(s) \leq (1+\epsilon).\textsc{opt}$ and $\mathcal{A}$ runs in $poly(|I|)$. Note that in a PTAS the runtime should only be polynomial in the size of input. If we require $\mathcal{A}$ to be polynomial in the size of I and $\frac{1}{\epsilon}$ then $\mathcal{A}$ is said to be a *fully polynomial time approximation scheme (FPTAS)* which is a remarkable approximation algorithm.

In this thesis we say a problem is *APX-hard* [10] to imply that there is no PTAS for it unless $P = NP$, in other words there is a c-hardness factor for it for a fixed constant $c > 1$.

Let Π_1 and Π_2 be two NP-optimization problems. By an *approximation factor preserving reduction* from Π_1 to Π_2, we loosely mean that if there is an α-approximation algorithm for Π_2 there is also an α-approximation algorithm for Π_1. More formally [122], this reduction consists of two polynomial time computable functions $f : \Sigma^* \rightarrow \Sigma^*$ and $g : \Sigma^* \rightarrow \Sigma^*$ such that:

- for any instance I_1 of Π_1 , $I_2 = f(I_1)$ is an instance of Π_2 such that $\mathrm{OPT}_{\Pi_2}(I_2) \leq$

$\text{OPT}_{\Pi_1}(I_1)$.

- for any solution t of I_2, $s = g(I_1, t)$ is a solution to I_1 such that $Obj_{\Pi_1}(I_1, s) \leq Obj_{\Pi_2}(I_2, t)$.

It is easy to see that an α-approximation algorithm for Π_2 along with this reduction result in an α-approximation algorithm for Π_1.

We can generalize the single criterion optimization problems to bicriteria optimization problems. An (A, B, S)-bicriteria optimization problem has two minimization objective A and B, and a feasible solutions set S. The problem specifies a budget L on the second objective and seeks to minimize the first objective. In other words, it seeks for a feasible solution $s \in S$ in which the cost of s under the second criterion (say $Obj_B(s)$) is not greater than L such that cost of s under the first criterion (say $Obj_A(s)$) is minimized.

An (α, β)-*Bicriteria approximation algorithm* for (A, B, S) is an algorithm which finds a feasible solution s in which $Obj_B(s)$ is at most $\beta \cdot L$ and whose cost is at most α times an optimum solution OPT where $Obj_A(\text{OPT})$ is minimized and $Obj_B(\text{OPT})$ is bounded L.

As an example, suppose we are given an undirected graph $G = (V, E)$, with two cost functions c and ℓ on each edge, a bound L, and a set of terminals $T \subseteq V$. An (α, β)-bicriteria approximation algorithm for (total cost, diameter, Steiner tree) is an algorithm which finds a Steiner tree H over the terminals T whose diameter (under ℓ metric) is at most $\beta \cdot L$ and whose cost is at most α times the value of an optimum solution under the function c with diameter bound L. In Chapter 4 we study this problem under the name of shallow light Steiner tree.

1.2.3 Linear Programming

Integer Programming (IP) is one of the famous NP-hard optimization problems that can model several other problems. A general form of an IP is as follows.

$$\begin{array}{ll} \min & c^T x \\ \text{subject to} & \\ & Ax \geq b \\ & x \in \{0, 1\}, \end{array}$$

where c is a vector with size n (c^T is its transpose) which we call the objective function or vector, A is an $m \times n$ constraints matrix and x is an integer vector with n variables for the optimization. Clearly we cannot solve IPs generally unless $P = NP$.

A common strategy to find a good approximation ratio for several problems is to find a suitable IP for the problem and relax it to a *Linear Programming (LP)* problem as shown below.

$$\begin{array}{ll} \min & c^T x \\ \text{subject to} & \\ & Ax \geq b \\ & x \geq 0 \end{array}$$

A feasible solution to an LP is any real vector x satisfying $Ax \geq b$. An LP is called feasible if it has at least one feasible solution. Since every feasible solution for an IP is a feasible solution to the corresponding LP, it is easy to see that the optimal value of the LP is a lower bound for its corresponding IP.

Assume Π is an NP-optimization problem. For each instance $I \in \Pi$ let $\text{OPT}_{IP}(I)$ be the optimum value for the corresponding IP of I and $\text{OPT}_{LP}(I)$ be the optimum value of the LP relaxation. We call $\sup_I \frac{\text{OPT}_{IP}(I)}{\text{OPT}_{LP}(I)}$ the *integrality gap* of the LP formulation of Π. As a result, the optimal value of the LP is a good estimation of the optimal value of I if the integrality gap is small. Finding a lower bound, an upper bound, or the exact value of integrality gap of an LP relaxation of an NP-optimization problem is usually an interesting question.

As an example, we can take an optimal solution to the LP and round its variables to some suitable integers such that the rounded values are feasible in the corresponding IP and the objective value for the integer variables is T. Assume $T \leq \delta \cdot \text{OPT}_L$, since $opt_L \leq opt_I$ we have $T \leq \delta \cdot \text{OPT}_I$ which is a δ-approximation for the IP problem. This technique is referred to as *LP rounding*.

A *Basic Feasible Solution (BFS)* is a feasible solution that cannot be written as a convex combination of two other feasible solutions. Another characterization of BFSs is that columns A_i in which $x_i \neq 0$ are linearly independent. BFSs are important for us as they sometimes can be rounded to an integer solution without losing too much in the objective value. Moreover, if an LP is feasible then for each objective vector c in which the optimal value of LP is bounded, there is at least one BFS that optimizes the LP.

The previous form for an LP is called primal form, there is a dual formulation for each primal LP defined as follow:

$$\begin{array}{ll} \text{max} & b^T y \\ \text{subject to} & \\ & A^T y \leq c \\ & y \geq 0 \end{array}$$

The following theorem is a useful fact about the feasible solutions of primal and dual forms.

Theorem 1 *(Weak duality theorem [122]) If $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_m)$ are feasible solutions for the primal and dual program, respectively, then:*

$$\sum_{j=1}^{n} c_j x_j \geq \sum_{i=1}^{m} b_i y_i$$

Note that from the weak duality theorem, we can conclude that every feasible solution to the dual-LP is a lower bound for the primal-LP. This theorem holds tightly when both dual and primal formulations have a finite optimum value which is more precisely stated in Theorem 2.

Theorem 2 *(LP-duality theorem [122]) The primal program has finite optimum iff its dual has finite optimum. Moreover if $x^* = (x_1^*, \ldots, x_n^*)$ and $y^* = (y_1^*, \ldots, y_n^*)$ are optimal solutions for the primal and dual programs respectively then:*

$$\sum_{j=1}^{n} c_j x_j^* = \sum_{i=1}^{m} b_i y_i^*$$

LPs can be solved in polynomial time. One way of doing this is to use *ellipsoid algorithm* [59]. It can be shown that if there is a polynomial time algorithm that checks whether a given candidate solution is feasible and if not finds a violated constraint, then one can optimize the LP using the ellipsoid algorithm. Such an algorithm that finds a violated constraint (if there is any) is called a *separation oracle*. As a result, any LP that has a separation oracle can be solved in polynomial time. This fact is specially useful when there are exponentially many constraints but there is a polynomial time separation oracle. We use this fact in Chapter 4.

1.2.4 The set cover problem

We briefly explain the best approximation algorithm for the set-cover problem as an example in the field of approximation algorithms. Set-cover is one of the central problems in the field and its technique is usually used in the covering problems (we also use the set cover analysis in Chapter 4). The formal definition of the set cover problem is as follow.

Definition 1 set cover [122]: *Given a universal set U of n elements, a collection of subsets of U, $\mathcal{S} = \{S_1, \ldots, S_k\}$, and a cost function $c : \mathcal{S} \to \mathbb{Q}^+$, find a minimum cost sub-collection of $\mathcal{S}$ that covers all the elements of U.*

It is easy to see that the definition of set cover problem is general and contains as a special case several other problems such as vertex cover, edge cover, tree cover, etc. In the following we present a simple greedy algorithm and prove its approximation ratio to be $H_n = \frac{1}{n} + \frac{1}{n-1} + \ldots + 1 \approx \ln n$ [74, 96, 42]. Interestingly enough, it is essentially the best ratio one can hope for. More formally, if there is a $((1-\epsilon) \cdot \ln n)$-approximation algorithm for the set cover problem for any constant $\epsilon > 0$ then $NP \subseteq DTIME(n^{O(\log \log n)})$ [48].

Let OPT be the total cost of an optimal solution OPT to the problem. The idea behind the algorithm is simple: at each step i select the subset with the best *density* (defined below) and continue doing this until all the elements are covered (see *i.e.* Figure 1.1 [122]). Let C_i be the set of covered elements before step i. Then the density of each subset S_j is $\frac{c(S_j)}{|S_j - C_i|}$. Because at each step i, OPT covers all the elements in $U - C_i$ there is a set with density at most $\frac{\mathrm{OPT}}{|U - C_i|}$ (**fact 1**).

The following theorem is the key theorem in the set cover analysis and is a basis for analysis of several other covering problems, we also use this theorem later in Chapter 5.

Inputs U, $\mathcal{S}$
Output: A sub-collection $\mathcal{T}$ covering all elements of U.

1. $C_1 \leftarrow \emptyset$, $\mathcal{T} \leftarrow \emptyset$.
2. While $C \neq U$ do
3. $i \leftarrow 1$
   (a) $i \leftarrow i + 1$
   (b) Find the set S with the minimum density
   (c) Let the density be $\alpha = \frac{c(S)}{|S-C|}$
   (d) $\mathcal{T} \leftarrow \mathcal{T} + S$
   (e) for each $e \in S - C$ set $price(e) = \alpha$.
   (f) $C_{i+1} \leftarrow C_i \cup S$
4. Output $\mathcal{T}$

Figure 1.1: Greedy algorithm for the set cover problem

Note that from the fact 1, we know that $f(n)$ in Theorem 3 for our algorithm is 1 which results the H_n-approximation for the set cover problem.

Theorem 3 *If an algorithm alg for the set cover problem at each step i adds a set with density at most $f(n) \cdot \frac{\text{OPT}}{|U-C_i|}$ where $f(n) : \mathbb{Z}^+ \to \mathbb{Q}^+$ is a function, then alg is a $(f(n) \cdot H_n)$-approximation algorithm for the set cover problem.*

Proof. Let the *price* of an element to be the average cost in the subset covering it for the first time, conversely, we assume that when we pick a set, its cost is distributed among the newly covered elements. Number the elements of U according to the step they enter into the cover, break ties arbitrarily. Let the numbering be $e_1, \ldots, e_n$. Consider the step i where e_k enters C_{i+1}. By definition of density and price we know that $price(e_k) = f(n) \cdot \frac{\text{OPT}}{|U-C_i|}$. Since at least all the $n - k + 1$ elements after e_k are not in C_i, $price(e_k)$ is at most $f(n) \cdot \frac{\text{OPT}}{n-k+1}$.

As we distribute the cost of each set in the $\mathcal{T}$ between all of its uncovered elements the total cost of $\mathcal{T}$ is at most $price(e_1) + \ldots + price(e_n) = f(n) \cdot (\frac{\text{OPT}}{n} + \frac{\text{OPT}}{n-1} + \ldots + \frac{\text{OPT}}{1}) = f(n) \cdot \text{OPT} \cdot H_n$. ■

1.3 Outline of thesis

In Chapter 2 we study the MMkTC problem. In that chapter (Section 2.3) we prove some lemmas which we use in the rest of the chapter and in Chapter 3. We give a 3-approximation algorithm for the MMkTC problem. In Chapter 3, we obtain a 2.5-approximation algorithm for the BTC problem. In Chapter 4, we study the SLkST problem and the BBkST problem. We give an $(O(\log^2 n), O(\log n))$-bicriteria approximation factor for the SLkST problem

(Section 4.4). In Section 4.3, we prove that our result for the SLkST problem implies an $O(\log^3 n)$-approximation ratio for the BBkST problem. We also show our results improve the approximation ratios for some related problems (Section 4.5). Chapter 5 is the last chapter of the thesis in which we give an $O(\log n)$-approximation algorithm for the $(k, 2)$-subgraph problem. In each chapter we review the previous works and possible future lines of research related to the problem(s) of the chapter in separate sections.

Chapter 2

Minimizing maximum k-tree cover

The study of problems in which the vertices of a given graph are needed to be covered with special subgraphs, such as trees, paths, or cycles, with a bound on the number of subgraphs used or their weights has attracted a lot of attention in Operations Research and Computer Science community. Such problems arise naturally in many applications such as vehicle routing and network design problems. As an example, in a vehicle routing problem with min-max objective, we are given a weighted graph $G = (V, E)$ in which each node represents a client. The goal is to dispatch a number of service vehicles to service the clients and the goal is to minimize the largest client waiting time, which is equivalent to minimizing the total distance traveled by the vehicle which has traveled the most. Observe that the subgraph traveled by each vehicle is a walk that can be approximated with a tree.

Min-max routing problems are part of an active body of research in the literature and have several applications (see e.g.[27, 46, 7, 105, 125] and the references there).

2.1 Problem Formulation

In this chapter we consider the Min-Max k-Tree Cover Problem (MMkTC). A problem named "Nurse station location", was the main motivation in [46] to study MMkTC. In the nurse station location problem, a hospital wants to assign all its patients to k nurses. Each nurse visits all its assigned patients every morning. The problem is to find a station for each nurse and assign all the patients to them such that the last completion time is minimized.

More formally, suppose we are given an undirected graph $G = (V, E)$ and a weight function $w : E \to \mathbb{Z}^+$.

Definition 2 *In the Min-Max k-tree Cover problem (MMkTC) we are given the weighted graph G and a positive integer k and the goal is to find a tree cover with k trees, which we call a k-tree cover, such that the weight of the largest tree in the cover is minimized where* *the weight of a tree T_i is $W(T_i) = \sum_{e \in T_i} w(e)$.*

The main result of this chapter is the following theorem.

Theorem 4 *There is a polynomial time 3-approximation algorithm for the MMkTC problem, for any arbitrary small $\epsilon > 0$.*

This improves upon the 4-approximation algorithms of [7, 46]

2.2 Related Works

Even et al. [46] and Arkin et al. [7] give two different 4-approximation algorithms for MMkTC. [46] also gives a 4-approximation algorithm for the rooted version of MMkTC in which k nodes are given in the input and each tree in a k-tree cover has to be rooted at one of them. It is shown that MMkTC is APX-hard in [125], specifically a hardness factor of $\frac{3}{2}$ is provided.

Nagamochi and Okada [103] give a $(3 - \frac{2}{k+1})$-approximation algorithm for MMkTC when all the trees have to be rooted at a given vertex r. They also give a $(2 - \frac{2}{k+1})$-approximation algorithm for MMkTC when the underlying metric is a tree and a $(2+\epsilon)$-approximation algorithm for MMkTC when the underlying metric is a tree and each tree has to be rooted at a certain vertex r.

Andersson *et al.* [2] consider the problem of "balanced partition of minimum spanning tree" in which for a given set of n points in the plane, the objective is to partition them into k sets such that the largest minimum spanning tree for each set is minimized. They give a $(2+\epsilon)$-approximation algorithm for this problem when $k \geq 3$ and a $(\frac{4}{3}+\epsilon)$-approximation ratio when $k = 2$.

In addition to trees, covering graphs with other objects, such as tours, paths, and stars are studied in the literature. Frederickson et al. [52] studied three Min-Max objective (the objective is to minimize the maximum tour) problems: the k-Traveling Salesman problem (k-TSP), the k-Stacker Crane problem (k-SCP), and the k-Chinese Postman Problem (k-CPP). In k-TSP the objective is to cover all the nodes with k tours, in k-CPP the objective is to cover all the edges with k tours, and in k-SCP the objective is to cover some specified directed edges with k tours. They give a $(\alpha + 1 - \frac{1}{k})$-approximation algorithm for each of these problems where α is the best approximation ratio for the corresponding single person problem. They do this by finding a best achievable solution for one person instance and splitting it into k balanced tours. They also give a $\frac{9}{5}$-approximation algorithm for the single SCP. Note that 1-CPP is polynomially solvable although its k-person version is NP-complete [52, 115], and the best current approximation factor for TSP is $\frac{3}{2}$ [40] (in the unweighted case there is a 1.461-approximation algorithm due to [101]).

Averbakh *et al.* [11] consider the k-TSP problem with min-max objective where the underlying metric is a tree, they give an $\frac{k+1}{k-1}$-approximation algorithm for this problem.

Arkin *et al.* [7] give a 3-approximation algorithm for the min-max path cover problem. Xu *et al.* [126] consider a similar problem with extra service cost at each node. They consider three variation of this problem: (i) all the paths have to start from a root, (ii) all the paths have to start from any of a given subset of nodes, and (iii) all the paths can start from any node. They give approximation factors of 3, $(4+\epsilon)$, and $(5+\epsilon)$ and hardness factors of $\frac{4}{3}$, $\frac{3}{2}$, and $\frac{3}{2}$ for these variations, respectively.

Another problem related to k-TSP is called k-Traveling Repairman Problem (KTR) in which instead of minimizing the total lengths of the tour the objective function is to minimize the total latency of the nodes where the latency of each node is the distance travelled (time elapsed) before visiting that node for the first time. The case of $k=1$ is known as the minimum latency problem. The best known approximation algorithm for $k=1$ is 3.59 due to [31] and the best known approximation for KTR is $2(2+\alpha)$ [38] where α is the best approximation ratio for the problem of finding minimum tree spanning k nodes a.k.a k-MST (see also [75] and the references there).

Online vehicle routing problems are also considered in the literature, for a survey see [71]. Other different variations of min-max objective vehicle routing problems are also studied in the literature (see e.g. [7, 104, 105, 95, 65]).

2.3 Preliminaries

For a connected subgraph $H \subseteq G$ by tree weight of H we mean the weight of a minimum spanning tree (MST) of H and denote this value by $W_T(H)$. Note that this is different from the weight of H, i.e. $W(H)$ which is the sum of weights of *all* the edges of H.

In every solution to either the MMkTC or BTC (to be seen in Chapter 3) problem, we can replace every edge (u,v) of a tree in the cover with the shortest path between u,v in the graph without increasing the cost of the tree and the solution still remains feasible. Therefore, without loss of generality, if the input graph is G and $\tilde{G}$ is the shortest-path metric completion of G, we can assume that we are working with the complete graph $\tilde{G}$. Any solution to $\tilde{G}$ can be transformed into a feasible solution of G (for MMkTC or BTC) without increasing the cost (we can replace back the paths in G representing the edges in $\tilde{G}$).

The following lemma will be useful in our algorithms for both the MMkTC and BTC problems.

Lemma 1 *Suppose $G=(V,E)$ is a graph which has a k-tree cover $\mathcal{T}=\{T_1,\ldots,T_k\}$, with maximum tree weight of λ and let $\lambda' \leq \lambda$ be a given parameter. Assume we delete all the* *edges e with $w(e) > \lambda'$ (call them heavy edges) and let the resulting connected components be $C_1, \ldots, C_p$. Then $\Sigma_{i=1}^{p} W_T(C_i) \leq k\lambda + (k-p)\lambda'$.*

Proof. Let $G' = \bigcup_{i=1}^{p} C_i$ be the graph after deleting the heavy edges. Each tree in $\mathcal{T}$ might be broken into a number of subtrees (or parts) after deleting heavy edges; let $\mathcal{T}'$ denote the set of these broken subtrees, $|\mathcal{T}'| = k'$, and n_i be the number of trees of $\mathcal{T}'$ in component C_i. The total weight of the subtrees in $\mathcal{T}'$ is at most $k\lambda - (k'-k)\lambda'$, since the weight of each tree in $\mathcal{T}$ is at most λ and we have deleted at least $k'-k$ edges from the trees in $\mathcal{T}$ each having weight at least λ'. In each component C_i we use the cheapest $n_i - 1$ edges that connect all the trees of $\mathcal{T}'$ in C_i into one spanning tree of C_i. The weight of each of these added edges is no more than λ' and we have to add a total of $k'-p$ such edges (over all the components) in order to obtain a spanning tree for each component C_i. Thus, the total weight of spanning trees of the components C_i's is at most $k\lambda - (k'-k)\lambda' + (k'-p)\lambda' = k\lambda + (k-p)\lambda'$. ∎

Through our algorithms we may need to break a large tree into smaller trees that cover (the vertices of) the original tree, are edge-disjoint, and such that the weight of each of the smaller trees is bounded by a given parameter. We use the following lemma which is implicitly proved in [46] (in a slightly weaker form) in the analysis of their algorithm.

Lemma 2 *Given a tree T with weight $W(T)$ and a parameter $\beta > 0$ such that all the edges of T have weight at most β, we can edge-decompose T into trees $T_1, \ldots, T_k$ with $k \leq \max(\lfloor \frac{W(T)}{\beta} \rfloor, 1)$ such that $W(T_i) \leq 2\beta$ for each $1 \leq i \leq k$.*

Proof. The idea is to "split away" (defined below) trees of weight in interval $[\beta, 2\beta)$ until we are left with a tree of size smaller than 2β. This process of "splitting away" is explained in [46]. We bring it here for the sake of completeness. Consider T being rooted at an arbitrary node $r \in T$. For every vertex $v \in T$ we use T_v to denote the subtree of T rooted at v; for every edge $e = (u, v)$ we use T_e to denote the subtree rooted at u which consists of T_v plus the edge e. Subtrees are called light, medium, or heavy depending on whether their weight is smaller than β, in the range $[\beta, 2\beta)$, or $\geq 2\beta$, respectively. For a vertex v whose children are connected to it using edges $e_1, e_2, \ldots, e_l$ splitting away subtree $T' = \bigcup_{i=a}^{b} T_{e_i}$ means removing all the edges of T' and vertices of T' (except v) from T and putting T' in our decomposition. Note that we can always split away a medium tree and put it in our decomposition and all the trees we place in our decomposition are edge-disjoint. So assume that all the subtrees of T are either heavy or light. Suppose T_v is a heavy subtree whose children are connected to v by edges $e_1, e_2, \ldots$ such that all subtrees $T_{e_1}, T_{e_2}, \ldots$ are light (if any of them is heavy we take that subtree). Let i be the smallest index such that $T' = \bigcup_{a=1}^{i} T_{e_a}$ has weight at least β. Note that T' will be medium as all T_{e_j}'s are light. We split away T' from T and repeat the process until there is no heavy subtree of T (so at the end the left-over T is either medium or light).

If $W(T) \leq 2\beta$ then we do not split away any tree (since the entire tree T is medium) and the theorem holds trivially. Suppose the split trees are $T_1, T_2, \ldots, T_d$ with $d \geq 2$ with $W(T_i) \in [\beta, 2\beta)$ for $1 \leq i < d$. The only tree that may have weight less than β is T_d. Note that in the step when we split away T_{d-1} the total weight of the remaining tree was at least 2β, therefore we can assume that the average weight of T_{d-1} and T_d is not less than β. Thus, the average weight of all T_i's is not less than β which proves that d cannot be greater than $\lfloor \frac{W(T)}{\beta} \rfloor$. ■

2.4 A 3-approximation algorithm for MMkTC

In this section we prove Theorem 4. Before describing our algorithm we briefly explain the $(4+\epsilon)$-approximation algorithm of [46]. Suppose that the value of the optimum solution to the given instance of MMkTC is $\textsc{opt}$ and let $\lambda \geq \textsc{opt}$ be a value that we have guessed as an upper bound for $\textsc{opt}$. The algorithm of [46] will either produce a k-tree cover whose largest tree has weight at most 4λ or will declare that $\textsc{opt}$ must be larger than λ, in which case we adjust our guess λ. So assume we have guessed a value λ such that $\lambda \geq \textsc{opt}$.

For simplicity, let us assume that G is connected and does not have any edge e with $w(e) > \lambda$ as these clearly cannot be part of any optimum k-tree cover. Let T be a MST of G and $\mathcal{T} = \{T_1, \ldots, T_k\}$ be an optimum k-tree cover of G. We can obtain a spanning tree of G from $\mathcal{T}$ by adding at most $k-1$ edges between the trees of $\mathcal{T}$. This adds a total of at most $(k-1)\lambda$ since each edge has weight at most λ. Thus, $W(T) \leq \sum_{i=1}^{k} W(T_i) + (k-1)\lambda \leq (2k-1)\lambda$. Therefore, by Lemma 2 if we start from a MST of G, say T, and we split away trees of size in $[2\lambda, 4\lambda)$ then we obtain a total of at most $(2k-1)\lambda/2\lambda \leq k$ trees each of which has weight at most 4λ. In reality the input graph might have edges of weight larger than λ. First, we delete all such edges (called heavy edges) as clearly these edges cannot be part of an optimum solution. This might make the graph disconnected. Let $\{G_i\}_i$ be the connected components of the graph after deleting these heavy edges and let T_i be a MST of G_i. For each component G_i the algorithm of [46] splits away trees of weight in $[2\lambda, 4\lambda)$. Using Lemma 2 one can obtain a k_i-tree cover of each G_i with $k_i \leq \max(W_T(G_i)/2\lambda, 1)$ with each tree having weight at most 4λ. A similar argument to the one above shows (Lemma 3 in [46]) that $\sum_i (k_i + 1) \leq k$. One can do a binary search for the smallest value of λ with $\lambda \geq \textsc{opt}$ which yields a polynomial 4-approximation.

Now we describe our algorithm. As said earlier, we work with the metric graph $\tilde{G}$. We use OPT to denote an optimal solution and $\textsc{opt}$ to denote the weight of the largest tree in OPT. Similar to [46] we assume we have a guessed value λ for $\textsc{opt}$ and present an algorithm which finds a k-tree cover with maximum tree weight at most 3λ if $\lambda \geq \textsc{opt}$. Moreover, if the algorithms fails to produce a solution with the objective value 3λ then $\lambda < \textsc{opt}$. Having this algorithm we can do binary search for λ to find the optimum value ($\textsc{opt}$) for λ which results a 3-approximation algorithm, that runs in time polynomial in input size.

First, we delete all the edges e with $w(e) > \lambda/2$ to obtain graph G'. Let $C_1, \ldots, C_\ell$ be the components of G' whose tree weight (i.e. the weight of a MST of that component) is at most λ (we refer to them as *light components*), and let $C_{\ell+1}, \ldots, C_{\ell+h}$ be the components of G' with tree weight greater than λ (which we refer to as *heavy components*). The general idea of the algorithm is as follows: For every light component we do one of the following three: find a MST of it as one tree in our tree cover, or we decide to connect it to another light component with an edge of weight at most λ in which case we find a component with MST weight at most 3λ and put that MST as a tree in our solution, or we decide to connect a light component to a heavy component. For heavy components (to which some light components might have been attached) we split away trees with weight in $[\frac{3}{2}\lambda, 3\lambda)$. We can show that if this is done carefully, the number of trees is not too big. We explain the details below.

For every light component C_i let $w_{min}(C_i)$ be the minimum edge weight (in graph $\tilde{G}$) between C_i and a heavy component if such an edge exists with weight at most λ, otherwise set $w_{min}(C_i)$ to be infinity. We might decide to combine C_i with a heavy component (one to which C_i has an edge of weight $w_{min}(C_i)$). In that case the tree weight of that heavy component will be increased by $A(C_i) = W_T(C_i) + w_{min}(C_i)$. The following lemma shows how we can cover the set of heavy components and some subset of light components with a small number of trees whose weight is not greater than 3λ.

Lemma 3 *Let $L_s = \{C_{l_1}, \ldots, C_{l_s}\}$ be a set of s light-components with bounded $A(C_i)$ values. If $\sum_{1\le i\le s} A(C_{l_i}) + \sum_{\ell+1\le i\le \ell+h} W_T(C_i) \le x - h\frac{\lambda}{2}$, then we can cover all the nodes in the heavy-components and in components of L_s with at most $\lfloor \frac{2x}{3\lambda} \rfloor$ trees with maximum tree weight no more than 3λ.*

Proof. First we find a MST in each heavy component and in each component of L_s, then we attach the MST of each C_{l_i} to the nearest spanning tree found for heavy components. As we have h heavy components, we get a total of h trees, call them $T_1, \ldots, T_h$. From the definition of $A(C_{l_j})$, the total weight of the constructed trees will be:

$$\sum_{i=1}^{h} W(T_i) = \sum_{1\le j\le s} A(C_{l_j}) + \sum_{\ell+1\le i\le \ell+h} W_T(C_i) \le x - h\frac{\lambda}{2}, \tag{2.1}$$

where the last inequality is by the assumption of lemma. Now to each of the h constructed trees we will apply the procedure of Lemma 2 with $\beta = \frac{3}{2}\lambda$ to obtain trees of weight at most 3λ. This gives at most $\sum_{1\le i\le h} \max(\lfloor \frac{2W(T_i)}{3\lambda} \rfloor, 1)$ trees. To complete the proof of lemma it is sufficient to prove the following:

$$\sum_{1\le i\le h} \max(\lfloor \frac{2W(T_i)}{3\lambda} \rfloor, 1) \le \lfloor \frac{2x}{3\lambda} \rfloor. \tag{2.2}$$

Consider T_i for an arbitrary value of i. If T_i has been split into more than one tree, by Lemma 2 we know that the amortized weight of the split trees is not less than $\frac{3}{2}\lambda$. If T_i is not split, as T_i contains a spanning tree over a heavy component, $W(T_i) \geq \lambda$. Thus every split tree has weight at least $\frac{3}{2}\lambda$ except possibly h trees which have weight at least λ. Therefore, if the total number of split trees is r, they have a total weight of at least $r\frac{3}{2}\lambda - h\frac{\lambda}{2}$. Using Equation (2.1), it follows that r cannot be more than $\lfloor \frac{2x}{3\lambda} \rfloor$. ∎

Before presenting the algorithm we define a graph H formed according to the light components.

Definition 3 *For two given parameters a, b, graph H has $\ell + a + b$ nodes: ℓ (regular) nodes $v_1, \ldots, v_\ell$, where each v_i corresponds to a light component C_i, a dummy nodes called* null *nodes, and b dummy nodes called* heavy *nodes. We add an edge with weight zero between two regular nodes v_i and v_j in H if and only if $i \neq j$ and there is an edge in $\tilde{G}$ with length no more than λ connecting a vertex of C_i to a vertex of C_j. Every null node is adjacent to each regular node v_i ($1 \leq i \leq \ell$) with weight zero. Every regular node $v_i \in H$ whose corresponding light component C_i has finite value of $A(C_i)$ is connected to every heavy node in H with an edge of weight $A(C_i)$. There are no other edges in H.*

In the following we show that algorithm MMkTC (Figure 2.1) finds a k-tree cover with maximum tree weight at most 3λ, if $\lambda \geq$ OPT. Before showing this, let see how this fact implies Theorem 4. If $\lambda \geq$ OPT, Algorithm 2.1 will find a k-tree cover with maximum tree weight at most 3λ. If $\lambda <$ OPT the algorithm may fail or may provide a k-tree cover with maximum weight at most 3λ which is also a true 3-approximation. As OPT can be at most $\sum_{e \in E} w(e)$, by a binary search in the interval $[0, \sum_{e \in E} w(e)]$, we can find a λ for which our algorithm will give a k-tree cover with bound 3λ and for $\lambda - 1$ the algorithm will fail. Thus, for this value of λ, we get a 3-approximation factor. This completes the proof of Theorem 4.

As a result throughout the whole proof we assume $\lambda \geq$ OPT. In order to bound the maximum weight of the cover with 3λ we need to use the optimal k-tree cover. Consider an optimal k-tree cover OPT; so each $T \in$ OPT has weight at most λ. First note that every tree $T \in$ OPT can have at most one edge of value larger than $\lambda/2$; therefore each $T \in$ OPT is either completely in one component C_i or has vertices in at most two components, in which case we say it is broken. If T is broken it consists of two subtrees that are in two components (we refer to the subtrees as *broken subtree or part* of T) plus an edge of weight $> \lambda/2$ connecting them; we call that edge the *bridge edge* of T. We characterize the optimal trees in the following way: a tree $T \in$ OPT is called light (heavy) if the entire tree or its broken subtrees (if it is broken) are in light (heavy) components only, otherwise if it is broken and has one part in a light component and one part in a heavy component then we

Inputs: $G(V, E)$, k, λ
Output: A set S which is a k-tree cover with maximum tree size 3λ.

1. Build $\tilde{G}$ which is the shortest-path metric completion of G and then delete all edges with weight more than $\frac{\lambda}{2}$; let $C_1, \ldots, C_{\ell+h}$ be the set of ℓ light and h heavy components created.
2. For $a : 0 \to \ell$
   (a) For $b : 0 \to \ell$
      i. $S \leftarrow \emptyset$
      ii. Construct H (as described in Definition 3) with a null nodes and b heavy nodes.
      iii. Find a perfect matching with the minimum cost in H; if there is no such perfect matching continue from Step 2a ,
      iv. Attach each light-component C_i to its nearest heavy component (using the cheapest edge in $\tilde{G}$ between the two) if v_i is matched to a *heavy* node in the matching
      v. Decompose all the heavy components and the attached light components using Lemma 3 and add the trees obtained to S
      vi. If a vertex v_i is matched to a null node, add a MST of C_i to S.
      vii. For every matching edge between two regular nodes v_i and v_j join a MST of C_i and a MST of C_j using the cheapest edge among them (in G) and add it to S.
      viii. If $|S| \leq k$ then return S.
3. return failure

Figure 2.1: MMkTC Algorithm

call it a bad tree. We denote the number of light trees, heavy trees, and bad trees of OPT by k_ℓ, k_h, and k_b; therefore $k_\ell + k_h + k_b = k$. We say that a tree $T \in \text{OPT}$ is incident to a component if the component contains at least one vertex of T (see Figure 2.2).

We define multi-graph $H' = (V', E')$ similar to how we defined H except that edges of H' are defined based on the trees in OPT. V' consists of ℓ vertices, one vertex v'_i for each light component C_i. For each light tree $T \in \text{OPT}$, if T is entirely in one component C_i we add a loop to v'_i and if T is broken and is incident to two light components C_i and C_j then we add an edge between v'_i and v'_j. So the total number of edges (including loops) is k_ℓ. There may be some isolated nodes (nodes without any edges) in H', these are nodes whose corresponding light components are incident to only bad trees. Suppose M is a maximum matching in H' and let U be the set of vertices of H' that are not isolated and are not saturated by M. Because M is maximal, every edge in $E' \setminus M$ is either a loop or is an edge between a vertex in U and one saturated vertex. Therefore:

$$|M| + |U| \leq k_\ell. \tag{2.3}$$

Note that for every node v'_i (corresponding to a light component C_i) which is incident to

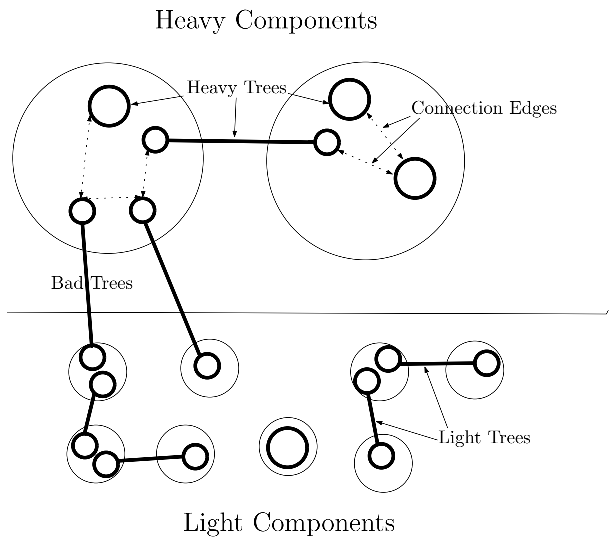

Figure 2.2: Structure of G after deleting edges with length greater than $\frac{\lambda}{2}$. Each thin circle corresponds to a component and each solid circle corresponds to an optimum tree or a broken subtree (part) of an optimum tree.

a bad tree, that bad tree has a bridge edge (of weight at most λ) between its broken subtree in the light component (i.e. C_i) and its broken subtree in a heavy component. Therefore:

Lemma 4 *For every light component C_i which is incident to a bad tree, and in particular if v_i' is isolated, $A(C_i)$ is finite.*

We define the excess weight of each bad tree as the weight of its broken subtree in the light component plus the bridge edge. Let W_{excess} be the total excess weights of all bad trees of OPT. Note that W_{excess} contains $\sum_{v_i \text{ is isolated}} A(C_i)$, but it also contains the excess weight of some bad trees that are incident to a light component C_i for which v_i is not isolated. Thus:

$$W_{excess} \geq \sum_{v_i \text{ is isolated}} A(C_i). \tag{2.4}$$

Only at Steps 2(a)v, 2(a)vi, and 2(a)vii the algorithm adds trees to S. First we will show that each tree added to S has weight at most 3λ. At step 2(a)v, according to Lemma 3, all the trees will have weight at most 3λ. At Step 2(a)vi, as C_i is a light component its MST will have weight at most λ. At Step 2(a)vii, the MST of C_i and C_j are both at most λ, and as v_i and v_j are connected in H there is an edge with length no more than λ connecting C_i and C_j; thus the total weight of the tree obtained is at most 3λ. Hence, every tree in S has weight no more than 3λ. The only thing remain is to show that the algorithm will eventually finds a set S that has no more than k trees. We show that in the iteration at which $a = |U|$ and b is equal to the number of isolated nodes in H': $|S| \leq k$.

Lemma 5 *The cost of the minimum perfect matching computed in step 2(a)iii is no more than* W_{excess}.

Proof. Consider the iteration at which $a = |U|$ and b is the number of isolated nodes in H'. In this case, we can find a perfect matching in the following way: for every vertex $v'_i \in U$, $v_i \in H$ can be matched to a null node in H, for every isolated node $v'_i \in H'$, $v_i \in H$ can be matched to a heavy node in H (note that $A(C_i)$ is finite by Lemma 4), for all other vertices $v'_i \in H'$, v'_i is saturated by M, so the corresponding $v_i \in H$ can be matched according to the matching M. Note that the cost of this matching is $\sum_{v_i \text{ is isolated}} A(C_i)$ which is no more than W_{excess} by Equation (2.3). Since we find a minimum perfect matching in step (2(a)iii). ■

Note that the number of trees added to S at step (2(a)vii) is $|M|$ and the number of trees added at step (2(a)vi) is $|U|$. Thus the total number of trees added to S at these two steps is at most $|M| + |U| \le k_\ell$ by Equation (2.3). The weight of the minimum perfect matching found in (2(a)iii) represents the total weight we add to the heavy components in step (2(a)iv). By Lemma 5, we know that the added weight is at most W_{excess}. In Lemma 6 we bound the total weight of heavy components and the added extra weight of matching by $(k_h + k_t) * \frac{3}{2}\lambda - h\frac{\lambda}{2}$. Using Lemma 3 we know that we can cover them by at most $k_h + k_b$ trees. Thus the total number of trees added to S is at most $k_\ell + k_h + k_b = k$.

The following lemma will bound the weight of the heavy components and W_{excess}.

Lemma 6 $\sum_{\ell+1 \le i \le \ell+h} W_T(C_i) + W_{excess} \le (k_h + k_b) * \frac{3}{2}\lambda - h\frac{\lambda}{2}$, *if* $\lambda \ge$ OPT.

Proof. Again, we assume that $\lambda \ge$ OPT. We show a possible way to form a spanning tree for each heavy component plus the light components attached to it. Then we bound the total weight of these spanning trees.

We can make a spanning tree over a heavy component C_i by connecting all the trees and broken subtrees of the optimum solution that are in that component by adding edges of weight at most $\lambda/2$ between them since each edge in C_i has weight at most $\lambda/2$ (see Figure 2.2). Therefore, the tree weight of a heavy component can be bounded by the weight of optimal trees or broken subtrees inside it plus some edges to connect them. Suppose p trees of the heavy trees are broken and q of them are completely inside a heavy component; note that $p + q = k_h$. The rest of broken subtrees in heavy components are from bad trees. So overall we have $2p + q + k_b$ trees or broken subtrees in all the heavy components. Each of the q heavy trees that are not broken contribute at most $q\lambda$ to the left hand side. Those p heavy trees that are broken contribute at most $p\lambda/2$ to the left hand side since each of them has an edge of weight more than $\lambda/2$ that is deleted and is between heavy components. By definition of W_{excess}, we can assume the contribution of all bad trees to the left hand side is at most $k_b\lambda$. Thus, the total weight of edges e such that e belongs to an optimal tree and also belongs to a heavy component or is part of W_{excess} (i.e. the broken part of a bad tree plus its bridge edge) is at most $(p + q + k_b)\lambda - p\frac{\lambda}{2}$.

Overall we have $2p + q + k_b$ trees or broken subtrees in all the heavy components. In order to form a spanning tree in each heavy component we need at most $2p+q+k_b-h$ edges connecting the optimal trees and broken subtrees in the heavy components, since we have h heavy components. Since each edge in a component has weight at most $\frac{\lambda}{2}$, the total weight of these edges will be at most $(2p+q+k_b-h)\frac{\lambda}{2}$. Therefore, the total weight of spanning trees over all heavy components plus W_{excess} will be at most $(p+q+k_b)\lambda-p\frac{\lambda}{2}+(2p+q+k_b-h)\frac{\lambda}{2} = (k_h + k_b) * \frac{3}{2}\lambda - h\frac{\lambda}{2}$. ■

We end this section by analyzing the running time of our algorithm. As we discussed earlier, the binary search for finding $\lambda \geq \text{OPT}$ is done in the interval $[0, \sum_{e\in E} w(e)]$, so the binary search takes at most $O(\log(\sum_{e\in E} w(e)))$. At each step of the binary search we call the algorithm shown in Figure 2.1. In Step 1 of the algorithm, we can build the metric completion of the graph in $O(|V|^3)$ operations using all pairs shortest path algorithm of Floyds-Warshall. Step 2(a)iii is the most time consuming step in Loop 2 in which we find a perfect matching on a graph with at most 3ℓ nodes. The perfect matching can be found using the algorithm of [102] in $O(\ell^{2.376})$. As a result, Loop 2 takes at most $O(\ell^{4.376})$ operations. By noting that ℓ can be at most $|V|$ the total run time of our algorithm is $O(\log(\sum_{e\in E} w(e))|V|^{4.376})$ which is polynomial in the size of the inputs.

2.5 Future Works

Finding an approximation algorithm and a hardness of approximation with the same ratio for a problem is the ultimate goal in study of approximation algorithms. However, often this is too optimistic and one hopes to close the gap between the upper bound (approximation ratio) and the lower bound (hardness factor) as much as possible. As discussed in Section 2.2 the best hardness factor for MMKTC is $\frac{3}{2}$ [125] which still has a large gap from its best current approximation factor 3.

The algorithm for min-max path cover [7] also uses a similar scheme: (1) deleting edges with weight greater than λ, (2) finding a tree in each induced component, (3) doubling the edges of each tree, (4) making a TSP tour for each tree with double edges, and (5) splitting the TSP tours into k paths. We believe similar technique of deleting edges with weight greater than $\lambda/2$ and using an appropriate matching may lead to a better approximation factor for this problem.

Algorithms for covering graphs with tours, trees, paths, *etc.* with min-max objective use the simple technique of spliting the solution of single person instance into k balanced subgraphs. It is an interesting open question if there is a better applicable techinque such as LP-based algorithms for these types of problems.

Chapter 3

Bounded tree cover

In the previous chapter we looked at the problem of covering the vertices of a graph with a given class of graphs (namely trees) while minimizing the maximum weight of them. The motivation for these types of problems is coming from the fact that one can model a group of customers needing special service as nodes in the graph and the subgraphs represent servicing agent. Suppose a company does not want to have a customer serviced later than a target deadline. A related problem is Bounded Tree Cover in which we are given an upper bound on the size of each tree and our goal is to minimize number of trees in the cover.

As an example, in a bounded vehicle routing problem, we are given a weighted graph $G = (V, E)$ in which each node represents a client. The goal is to dispatch a number of service vehicles to service the clients such that each client has to be served before a specified time bound λ. The goal is to minimize the total number of required vehicles. Observe that the subgraph travelled by each vehicle is a walk that can be approximated with a tree. Such problems arise naturally in many applications such as vehicle routing and network design problems.

3.1 Problem Formulation

In this section we study the problem of Bounded Tree Cover (BTC). More formally, suppose we are given an undirected graph $G = (V, E)$ and a weight function $w : E \rightarrow \mathbb{Z}^+$.

Definition 4 *In the BTC problem, we are given the weighted graph G and a parameter λ and the goal is to find a tree cover with minimum number of trees such that the weight of every tree in the cover is at most λ, where the weight of a tree T_i is $W(T_i) = \sum_{e \in T_i} w(e)$.*

In this chapter we improve the approximation ratio for BTC. Specifically, we prove the following theorem in Section 3.4

Theorem 5 *There is a polynomial time 2.5-approximation algorithm for the BTC problem.*

This improves upon the 3-approximation algorithm of [7].

3.2 Related Works

In the bin packing problem we are given a set of items $\{i_1, \ldots, i_n\}$ each of which has a weight $w(i_j)$, the objective is to pack all the items into bins with size B such that the number of bins used is minimized and the total weight of items in each bin does not exceed B. The bin packing problem is APX-hard and has a hardness of $\frac{3}{2} - \epsilon$ for any $\epsilon > 0$ [122]. BTC is APX-hard even in the case when G is a weighted tree with height one by an easy reduction from the bin packing problem. The reduction is as follows. For a given instance of bin packing problem with item set $\{i_1, \ldots, i_n\}$ and bound B, build the corresponding BTC instance consisting of a weighted graph with node set $V = \{r\} \cup \{v_1, \ldots, v_n\}$ such that every node v_j is connected to r by an edge of weight $w(i_j)$ and $\lambda = B$. It is easy to see that an α-approximation algorithm for the BTC instance gives a corresponding solution for the bin packing instance with similar approximation ratio and vice-versa. Thus, the BTC problem is at least as hard as the bin packing problem.

Nagarajan *et al.* [105] consider distance constrained vehicle routing problem in which the objective is to cover an undirected graph with the minimum number of tours with bounded lengths. They give an $O(\log n)$-approximation for the general metric graphs and a 2-approximation for tree metrics.

Another closely related problem is the orienteering problem. In the orienteering problem we are given a weighted graph G and a bound B, the objective is to find a walk that visits (or services) as many nodes as possible within the time bound B. The approximation ratio for this problem on undirected graphs has been improved from 4 [23] to 3 [15] and finally to $(2+\epsilon)$ [35]. For the directed case, the best approximation ratio is $O(\log^2 n)$ due to [35, 104], the best previously known approximation ratio is a quasi-polynomial time (see Section 1.2) algorithm with approximation guarantee of $\log n$ [36].

In another setting, there might be a deadline $D(v)$ on each node v meaning that the client should be served by that time. This problem is known as deadline-TSP. Deadline-TSP can be generalized to the case when each client has to be served between the time interval $[R(v), D(v)]$. This problem is known as vehicle routing with time windows. Bansal *et al.* [15] give an $O(\log n)$-approximation for the deadline-TSP and extend it to the time window with an $O(\log^2 n)$-approximation guarantee. Chekuri and Pal [36] give an $O(\log n)$ approximation for the time window but their algorithm runs in quasi-polynomial time. Frederickson *et al.* [51] consider the special case when the time windows are unit length and give a constant factor approximation for it and they also consider some other special variations.

In addition to trees there are some other variations of bounded graph covering studied in the literature [7, 127].

3.3 A 2.5-approximation algorithm for BTC

Given an instance of BTC consisting of a graph G and a bound λ on the tree sizes we use OPT to denote an optimum solution and $k = \text{OPT}$ denote the number of trees in OPT. As before, we can assume we are working with the shortest-path metric completion graph $\tilde{G} = (V, E)$. Our algorithm for this problem is similar to the algorithm for MMkTC, although the analysis is different (we use the preliminaries introduced in Section 2.3 in this chapter) . The overall structure of the algorithm is as follows. We delete all the edges with weight greater than $\lambda/4$ in $\tilde{G}$ to obtain graph G'. Let $C_1, \ldots, C_\ell$ be the components of G' whose weight is at most $\lambda/4$, called *light components*, and $C_{\ell+1}, \ldots, C_{\ell+h}$ be the components with weight greater than $\lambda/4$ which we refer to as *heavy components*. We define $A(C_i)$, the tree of a light component C_i plus the weight of attaching it to a heavy component as in Section 2: it is the weight of minimum spanning tree of C_i, denoted by $W_T(C_i)$, plus the minimum edge weight that connects a node of C_i to a node in a heavy component if such an edge e exists (in $\tilde{G}$) such that $W_T(C_i) + w(e) \leq \lambda$; otherwise $A(C_i)$ is set to infinity. The proof of the following lemma is identical to that of Lemma 3 with $\frac{3}{2}\lambda$ replaced with $\frac{1}{2}\lambda$.

Lemma 7 *Let $L_s = \{C_{l_1}, \ldots, C_{l_s}\}$ be a set of s light-components with bounded $A(C_i)$ values. If $\sum_{1 \leq i \leq s} A(C_{l_i}) + \sum_{\ell+1 \leq i \leq \ell+h} W_T(C_i) \leq x - h\frac{\lambda}{4}$, then we can cover all the nodes in the heavy components and in components of L_s with at most $\lfloor \frac{2x}{\lambda} \rfloor$ trees with maximum tree weight no more than λ.*

Before presenting the algorithm we define a graph $H = (L, F)$ formed according to the light components similar to the way we defined it in the MMkTC problem.

Definition 5 *For two given parameters a, b, graph H has $\ell + a + b$ nodes: ℓ (regular) nodes $v_1, \ldots, v_\ell$, where each v_i corresponds to a light component C_i, a dummy nodes called* null *nodes, and b dummy nodes called* heavy *nodes. We add an edge with weight zero between two regular v_i and v_j in H if and only if $i \neq j$ and there is an edge e between C_i and C_j in $\tilde{G}$ such that $W_T(C_i) + W_T(C_j) + w(e) \leq \lambda$. Every null node is adjacent to each regular node v_i ($1 \leq i \leq \ell$) with weight zero. Every regular node $v_i \in H$ whose corresponding light component C_i has finite value of $A(C_i)$ is connected to every heavy node in H with an edge of weight $A(C_i)$. There are no other edges in H.*

Theorem 5 follows from the following theorem.

Theorem 6 *Algorithm BTC (Figure 3.1) finds a k'-tree cover with maximum tree cost bounded by λ, such that $k' \leq 2.5\text{OPT}$ and runs in time $O(|V|^{4.376})$.*

Proof. It is easy to check that in all three steps 2(a)v, 2(a)vi, and 2(a)vii the trees found have weight at most λ: since each is either found using Lemma 7 (Step 2(a)v), or is a MST

Input: $G(V, E)$, λ
Output: A set S containing k'-tree cover with maximum tree cost λ in which $k' \leq 2.5\textsc{opt}$.

1. Take $\tilde{G}$ to be the metric completion of G and delete edges with length more than $\frac{\lambda}{4}$ to form graph G' with components $C_1, \ldots, C_{\ell+h}$
2. For $a : 0 \to \ell$
   (a) For $b : 0 \to \ell$
      i. $S_{a,b} \leftarrow \emptyset$
      ii. Build graph H according to Definition 5 with a null nodes and b heavy nodes.
      iii. Find a perfect matching with the minimum cost in H, if there is no such perfect matching continue from Step 2a
      iv. Attach each light component C_i to its nearest heavy component if v_i is matched to a heavy node
      v. Decompose all the heavy components and the attached light components as explained in Lemma 7 and add the trees obtained to $S_{a,b}$
      vi. If a node v_i is matched to a null node, add MST of C_i to $S_{a,b}$.
      vii. For every matching edge between v_i and v_j consider the cheapest edge e between C_i and C_j (in $\tilde{G}$) and add a minimum spanning trees of $C_i \cup C_j \cup \{e\}$ to $S_{a,b}$.
3. return set $S_{a,b}$ with the minimum number of trees.

Figure 3.1: BTC Algorithm

of a light component (Step 2(a)vi), or is the MST of two light components whose total weight plus the shortest edge connecting them is at most λ (Step 2(a)vii). So it remains to show that for some values of a, b, the total number of trees found is at most $2.5\textsc{opt}$.

First note that if matching M found in Step 2(a)iii assigns nodes $v_{l_1}, \ldots, v_{l_b}$ to heavy nodes and has weight W_M then $\sum_{1 \leq i \leq b} A(C_{l_i}) = W_M$. Let W_h denote the total tree weight of heavy components, i.e. $W_h = \sum_{\ell+1 \leq i \leq \ell+h} W_T(C_i)$. Then the number of trees generated using Lemma 7 in Step 2(a)v is at most $\lfloor \frac{2(W_M + W_h + h\lambda/4)}{\lambda} \rfloor$, and the number of trees generated in Steps 2(a)vi and 2(a)iii is exactly $(\ell - b + a)/2$; so we obtain a total of at most $\lfloor \frac{2(W_M + W_h + h\lambda/4)}{\lambda} \rfloor + (\ell - b + a)/2$ trees.

Lemma 8 *There exist $0 \leq a' \leq n$ and $0 \leq b' \leq n$ such that if H is built with a' null nodes and b' heavy nodes then H has a matching M' such that if Algorithm BTC uses M' then each tree generated has weight at most λ and the total number of trees generated will be at most* $2.5\textsc{opt}$.

For now assume Lemma 8 is correct (the proof is presented at Section 3.4) This lemma is sufficient to complete the proof for correctness of the algorithm as follows. Consider an iteration of the algorithm in which $a = a'$ and $b = b'$. Suppose that the minimum perfect matching that the algorithm finds in this iteration is M with weight W_M. Since $W_M \leq$

$W_{M'}$, the total number of trees generated in Step 2(a)v is at most $\lfloor \frac{2(W_M+W_h+h\lambda/4)}{\lambda} \rfloor \leq \lfloor \frac{2(W_{M'}+W_h+h\lambda/4)}{\lambda} \rfloor$. Furthermore, the number of trees generated in Steps 2(a)vi and 2(a)vii is exactly $(\ell - b' + a')/2$, so we obtain a total of at most $\lfloor \frac{2(W_M+W_h+h\lambda/4)}{\lambda} \rfloor + (\ell - b + a)/2$ trees. This together with the fact that $W_M \leq W_{M'}$ and Lemma 8 shows that we get at most 2.5OPT trees using M.

Now, we prove that the running time of our algorithm is in $O(|V|^{4.376})$. In Step 1 of the algorithm, we can build the metric completion of the graph in $O(|V|^3)$ operations using all pairs shortest path algorithm of Floyds-Warshall. Step 2(a)iii is the most time consuming step in the Loop 2 in which we find a perfect matching on a graph with at most 3ℓ nodes. The perfect matching can be found using the algorithm of [102] in $O(\ell^{2.376})$. As a result, Loop 2 takes at most $O(\ell^{4.376})$ operations. By noting that ℓ can be at most $|V|$ the total run time of our algorithm is $O(|V|^{4.376})$.

■

3.4 Proof of Lemma 8

In this section we prove Lemma 8. We use the structure of OPT in order to determine a', b' as well as the matching M'. We do not give an explicit value for a', b', instead we start with $a' = b' = 0$ and we define the edges we add to M' instead. For every two nodes of H that we pair (i.e. every edge we place in M') if that edge involves a null node (or a heavy node) we increase a' (or b') accordingly. In other words, we add a new null node (or heavy node) to H whenever we need to use a new copy of a null node (or heavy node). At the end, a' will be the total number of null nodes we used in our matching M' and b' will be the number of heavy nodes we used.

We call every tree in OPT an optimum tree. We say an optimum tree T is incident to a component C_i if C_i contains at least one node of T. Note that each optimum tree can be incident to at most 4 components as each edge deleted had weight more than $\lambda/4$. Let F be the set of light components which are incident to only one optimum tree. So each such component contains only one tree or broken subtree of a tree. We add matching edges to M' in 5 steps (described below) and also characterize the optimum trees into types. Initially $M' = \emptyset$, and we start with the optimum trees of first type and match some pairs of nodes in H based on the definition of Type 1 and add them to M'; then in Step 2 we define optimum trees of Type 2 and add all the matching edges that they define into M', and so on. Whenever we need to match a node v_i to a node v_j where v_j is already matched to another node in M' (in one of the previous steps) we use a new null node and match v_i to the null node (instead of v_j).

Step 1: Type 1 trees

An optimum tree is Type 1 if it is incident to only light components, say $C_{x_1}, \ldots, C_{x_p}$ (with

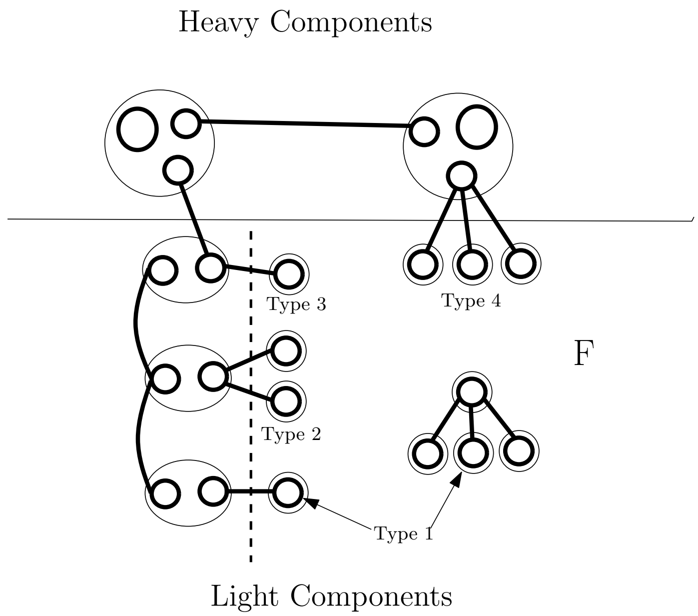

Figure 3.2: Each thin circle shows a component and each solid circle shows a broken subtree of an optimum tree; the solid lines show bridge edges that are deleted and were connecting broken subtrees of optimum trees

$p \leq 4$) which satisfy at least one of the following: i) $p \leq 2$, in which case we match each of v_{x_1} and v_{x_2} (if it is not already matched) to a new null node and add these (at most) two edges to M', or ii)$p = 3$ and at least two of $v_{x_1}, v_{x_2}, v_{x_3}$ are adjacent in H, say v_{x_1}, v_{x_2}, then we add the edge $v_{x_1}v_{x_2}$ to M' and match v_{x_3} with a null node, or iii) $p = 4$ and there are two independent edges among these four nodes in H, say v_{x_1}, v_{x_2} are adjacent and v_{x_3}, v_{x_4} are adjacent, then we add these two edges to M'. So, for each Type 1 optimum tree, we generate at most a total of two trees in Steps 2(a)vi and 2(a)vii (each corresponding to a matching edge described above) that together cover all the nodes of the components $C_{x_1}, \ldots, C_{x_p}$. Note that each of the trees generated this way has weight at most λ by definition of edges of H. In Step 1 we add all possible matching edges to M' by considering all Type 1 optimum trees before going to the next step.

Step 2: Type 2 trees

Every optimum tree T that is not Type 1 and is incident to an even number (specifically 2 or 4) of the light components in F is Type 2. We claim that the nodes of H corresponding to these light components are all adjacent (with edges of weight zero). So we can match them arbitrarily with at most two edges, we add these (at most) two edges to M'. The reason is each of these components contains only the nodes of T (because they are in F, so cannot be incident with any other optimum tree). Therefore all these components belong to the same optimum tree T; so for any two such components, say C_i and C_j, there is a path P connecting two nodes of them in T such that $W_T(C_i) + W_T(C_j) + W(P) \leq \lambda$. Since we are working with the complete graph $\tilde{G}$, there is an edge $e \in \tilde{G}$ between C_i and C_j with $W_T(C_i) + W_T(C_j) + w(e) \leq \lambda$, so v_i, v_j are adjacent in H. In Step 2 we place all the matching edges generated by Type 2 trees into M' before going to the next step.

Step 3: Type 3 trees

Every optimum tree T that is not Type 1 or 2 and has following properties is Type 3: T is incident to an odd number of light components $C_{x_1}, \ldots, C_{x_p}$ (p is specifically 1 or 3) in F and at least one light component C_y not in F such that the broken subtree of T in C_y is connected to the broken subtree of one of C_{x_i}'s, say C_{x_1}, with an edge e_T of T (which is now deleted).

Suppose that T is Type 3, and $p = 3$ (the case that T is incident with only one light component in F is easier and is dealt with below). First note that since each of $C_{x_1}, C_{x_2}, C_{x_3}$ belongs to the same optimum tree (namely T), similar arguments as in case of Type 2, show that nodes $v_{x_1}, v_{x_2}, v_{x_3}$ are all adjacent in H. Without loss of generality assume e_T connects the broken subtree of T in C_y to the one in C_{x_1}. We claim in that $v_{x_1}v_y$ is an edge in H as well. More specifically $W_T(C_{x_1}) + W_T(C_y) + w(e_T) \leq \lambda$. The following claim implies this:

Claim 1 $W_T(C_{x_1}) + W_T(C_y) + w(e_T) \leq \lambda$.

Proof. Since T is not Type 1, C_{x_1} and C_y cannot be the only components to which T is incident (otherwise, each of C_{x_1} and C_y would be matched to null nodes as in Type 1). Therefore, there is at least one other component that has a broken subtree of T, and there is at least one other edge of T, call e', (which is deleted now) connecting that broken subtree to C_{x_1} or to C_y in $\tilde{G}$. Note that $w(e') > \lambda/4$ and $W_T(C_y) \leq \lambda/4$. Therefore, $W_T(C_{x_1}) + W_T(C_y) + w(e_T) \leq W_T(C_{x_1}) + w(e') + w(e_T) \leq \lambda$ since all these are parts of T. ■

Hence we can pair v_{x_1} with v_y and pair v_{x_2} with v_{x_3} and add these two edges to M'. Note that again the tree generated by each of these pairs has weight at most λ. If T is Type 3 and is incident to only one light component in F, say C_{x_1} then we pair v_{x_1} with v_y as above and add only one edge to M'. We consider all Type 3 optimum trees and add the corresponding matching edges to M' before going to consider the next step..

Step 4: Type 4 trees

Every optimum tree T that is not Type 1, 2, or 3 and has following properties is Type 4: T is incident to an odd number of light components $C_{x_1}, \ldots, C_{x_p}$ (p is specifically 1 or 3) in F and at least one heavy component C_y such that the broken subtree of T in C_y is connected to the broken subtree in one of $C_{x_1}, \ldots, C_{x_p}$ with an edge e_T of T (which is now deleted).

Suppose that optimum tree T is Type 4 and $p = 3$ (again the case that T is incident with only one light component in F is easier). Arguments similar to the case of Type 3 show that nodes $v_{x_1}, v_{x_2}, v_{x_3}$ (corresponding to $C_{x_1}, C_{x_2}, C_{x_3}$) are all adjacent in H. Without loss of generality let assume e_T connects the broken subtree of T in C_y to the one in C_{x_1}. In this case, the weight of the broken subtree of T in C_{x_1}, plus the weight of e_T is no more than λ (as they are all part of T); in particular $W_T(C_{x_1}) + w(e_T) \leq \lambda$. This implies $A(C_{x_1}) \leq \lambda$ and so v_{x_1} is adjacent to heavy nodes in H. In this case we pair v_{x_1} with a (not already matched) heavy node in H and pair v_{x_2} with v_{x_3} and add these two edges to M'. Note that as argued before, the tree generated by the matching edge v_{x_2}, v_{x_3} has weight at most λ. Also, since we use Lemma 7 for each heavy component together with the light components attached to it, the weight of each tree generated from the heavy components (and their assigned light components) is at most λ.

Step 5: The rest of light components

This step completes the description of M'. Before starting this step, we explain why all the nodes in H corresponding to components in F are saturated by M' before this step. We show that if T has at least one broken subtree in a component in F then T is either Type 1, 2, 3, or 4, therefore all the components in F containing a broken subtree of T are matched in M'. We consider the following three cases for T: (1) If T is incident at only components in F then it is Type 1, (2) If T is incident at even number of components in F then it is Type 2, (3) If T is incident at odd number of components in F then, as it is not Type 1, T is incident at some other components not in F. As T is connected, there is an edge e_T which connects a broken subtree of T in a component in F to a broken subtree of T to a component C_y not in F. If C_y is a light component then T is Type 3 and if C_y is a heavy component then T is Type 4. So, all nodes corresponding to light components in F are already matched. In Step 5, if there is any light component that is not matched so far, each of them is incident with at least two optimum trees. In this step we match the corresponding node (in H) of each of these light components to a null node and these edges are added to M'.

Now we prove that the total number of trees generated by matching M' is at most 2.5OPT. Let N_1 denote the number of matching edges added to M' in Step 1, Y denote the number of matching edges added to M' in Steps 2 to 5 that does not involve a heavy node, and N_4 denote the number of trees generated by applying Lemma 7 to the matching edges added to M' in Step 4 that involves a heavy node. Note that the total number of trees generated in the final solution based on matching M' is $N_1 + Y + N_4$. We use OPT_1 to denote the number of optimum trees of Type 1, and $\text{OPT}_{rest} = \text{OPT} - \text{OPT}_1$ to denote the number of other optimum trees. Our goal is to show $N_1 + Y + N_4 \leq 2.5\text{OPT}$, more specifically we show: $N_1 + Y + N_4 \leq 2\text{OPT}_1 + 2.5\text{OPT}_{rest}$. It is easy to see that for every optimum tree of Type 1, we add at most 2 edges to M' in Step 1 (and therefore at most 2 trees in the final solution). Therefore, $N_1 \leq 2\text{OPT}_1$. In the rest we show that $Y + N_4 \leq 2.5\text{OPT}_{rest}$. We prove the following claim.

Claim 2 *Suppose we add Y edges to M' in Steps 2 to 5 that do not involve a heavy node and have matched light components $C_{l_1}, \dots, C_{l_s}$ to heavy nodes in Step 4. Then:*

(i) The union of the Y trees that are generated in the final solution based on the matching edges added to M' in Steps 2 to 5 that do not involve a heavy node contains at least $2Y$ broken subtrees of the optimum trees that are not Type 1.

(ii) $\sum_{i=1}^{s} A(C_{l_i}) + \sum_{\ell+1 \le i \le \ell+h} W_T(C_i) \le \frac{5}{4}\lambda \cdot \text{OPT}_{rest} - \frac{\lambda}{2} \cdot Y - h\frac{\lambda}{4}$.

First, let us show how we can complete the proof of lemma using this claim. Using Lemma 7 and part (ii) of Claim 2, the total number of trees generated based on matching edges added to M' at Step 4 that involves a heavy node is bounded by: $N_4 \le \lfloor 2.5\text{OPT}_{rest} - Y \rfloor$. Also, the total number of matching edges that do not involve a heavy node (and therefore the corresponding number of trees in the final solution) generated at Steps 2 to 5 is Y. So we get $Y + N_4 \le \lfloor 2.5\text{OPT}_{rest} - Y \rfloor + Y \le 2.5\text{OPT}_{rest}$, as wanted. Now it only remains to prove Claim 2.

Proof of Claim 2:

Part (i): We show that for every tree generated in the final solution based on matching edges added to M' in Steps 2 to 5 that do not involve a heavy node, there are two distinct broken subtrees of the optimum trees. To show this, we assign two broken subtrees for every such tree generated in the final solution such that each broken subtree is assigned to at most one tree of final solution.

For every optimum tree T of Type 2, each edge e added to matching M' in Step 2 is between two components in F each of which contains exactly one broken subtree of T; therefore the corresponding tree in the final solution generated based on e contains the broken parts of T in those two components of F. We assign those two broken subtrees to the tree generated. Also, every light component that is considered in Step 5 is not in F, i.e. it is incident with at least two optimum trees and so has at least two broken subtrees of two different optimum trees (that are not Type 1). Therefore, the tree generated at the final solution for each light component in Step 5 contains at least two broken subtrees of optimum trees that are not Type 1, we assign those two broken subtrees to the tree generated. Now we consider the matching edges added in Step 3 and 4 that do not involve a heavy node. If the matching edge $e \in M'$ corresponds to two components in F then similar to the case of Step 2, the tree generated in the final solution based on e contains the broken subtrees defined by the two components in F; we assign those two parts to the tree generated based on e. So the only remaining case is when we have a Type 3 tree T and it has a broken subtree in a component in F, say C_{x_1}, and another broken subtree in a light component not in F, say C_y (in Step 3). Note that C_{x_1} (since is in F) by definition has only one broken subtree and that is of tree T. Also, C_y has at least one broken subtree of T even if node $v_y \in H$ (corresponding to C_y) was matched to a different node (because C_y also had a broken subtree for a different optimum tree T'). So regardless of whether v_{x_1} is matched to v_y or to a null node, we can assign the two broken subtrees of T (one in C_{x_1} and one in C_y) to the tree generated based this matching edge in M'.

part (ii): To prove this part, suppose T is a Type 4 optimum tree as in Step 4 which has two broken subtrees, one subtree in the light component $C_{x_1} \in F$, denote it by T_{x_1} (note that there is no other subtree in C_{x_1}), and one subtree in the heavy component C_y, denote it by T_y, and there is an edge $e_T \in T$ that connects a node of T_{x_1} to a node of T_y. Recall that e_T was deleted as $W(e_T) > \lambda/4$. For the purpose of analysis, we merge component C_{x_1} (which consists of the nodes of T_{x_1}) with the heavy component C_y by adding the edge e_T back; this will merge the two broken subtrees T_{x_1} and T_y into one subtree. Also, by doing this, the weight of a MST in the new heavy component increases by $A(C_{x_1})$ only. If there are multiple optimum trees of Type 4 which have a broken subtree in C_y we merge them all with C_y. More generally, we do this merge operation for all the light trees $C_{l_1}, \ldots, C_{l_s}$ that are matched with heavy nodes in Step 4 and we let $C'_{\ell+1}, \ldots, C'_{\ell+h}$ be the set of new modified heavy components after these merge operations. Note that:

$$\sum_{i=1}^{s} A(C_{l_i}) + \sum_{\ell+1 \leq i \leq \ell+h} W_T(C_i) = \sum_{\ell+1 \leq i \leq \ell+h} W_T(C'_i).$$

Now we prove that $\sum_{\ell+1 \leq i \leq \ell+h} W_T(C'_i) \leq \frac{5}{4}\lambda \cdot \textsc{opt}_{rest} - \frac{\lambda}{2} \cdot Y - h\frac{\lambda}{4}$. Let p denote the number of (new) broken subtrees of the $\textsc{opt}_{rest}$ optimum trees that are not Type 1; Using part (i), at least $2Y$ of these parts are covered by (i.e. are contained in) the Y trees generated using the matching edges of Steps 2 to 5 that do not involve a heavy node. Therefore, the remaining at most $p - 2Y$ broken subtrees are in the modified heavy components. The total weight of optimum trees that are not Type 1 is at most $\lambda \cdot \textsc{opt}_{rest}$. Out of these trees at least $p - \textsc{opt}_{rest}$ edges are deleted even after merge operations that built modified heavy components, since we have a total of p broken subtrees. Therefore, the total weight of these p broken subtrees is at most $\lambda \cdot \textsc{opt}_{rest} - \frac{\lambda}{4} \cdot (p - \textsc{opt}_{rest})$ and all of these are inside the modified heavy components. By an argument similar to that of proof of Lemma 1, to make a spanning tree in each modified heavy component, the total weight of edges that need to be added between the broken subtrees inside the heavy components is at most $(p - 2Y - h)\frac{\lambda}{4}$. Therefore, the total weight of (MST's of) the modified heavy components is bounded by:

$$\lambda \cdot \textsc{opt}_{rest} - \frac{\lambda}{4} \cdot (p - \textsc{opt}_{rest}) + (p - 2Y - h)\frac{\lambda}{4} \leq \frac{5}{4}\lambda \cdot \textsc{opt}_{rest} - \frac{\lambda}{2} \cdot \lambda - h \cdot \frac{\lambda}{4}.$$

3.5 Future Works

The hardness factor explained in Section 3.2 is through an easy reduction to the case of trees with height 1. It is quite possible to find a better hardness factor by exploiting the fact that the graph can be a general graph.

The algorithm for bounded path cover [7] also uses a similar scheme: (1) deleting edges with weight greater than λ, (2) finding a tree in each induced component, (3) doubling the edges of each tree, (4) making a TSP tour for each tree with double edges, and (5) splitting the TSP tours into paths that are not violating the bound. We believe similar technique of deleting edges with weight greater than $\lambda/2$ and using an appropriate matching could lead to a better approximation factor this problem.

Chapter 4

Buy-at-bulk and shallow-Light k-Steiner Tree

In the Steiner tree problem we are given an undirected graph $G = (V, E)$ with non-negative edge costs in which the vertices are partitioned into two sets, *terminals* and *Steiner* nodes. The goal is to find a tree with minimum cost containing all the terminal nodes. This problem is one of the classical and fundamental problems in Theoretical Computer Science and Operations Research and studied intensively [122]. The problem has a wide range of application such as: design of VLSI, optical and wireless communication systems, transportation, and distribution networks [70]. Another importance of the Steiner tree problem is that it appears as a subproblem or a special case of many other problems such as Steiner Forest [57], Prize-Collecting Steiner tree [6], Virtual Private Network [44], Single-Sink Rent-or-Buy [45, 62], Connected Facility Location [45, 121], and Single-Sink Buy-At-Bulk [58, 62] among others.

The problem of designing a network capable of broadcasting multimedia (both video and audio) data in a multicast (simultaneous transmission of data to multiple destinations) environment is an important problem in the real world applications [97, 39, 50, 21, 87, 81]. A communication network can be modeled by a graph in which transmitters are represented by the vertices and the edges represent the connections between them. There are cost and delay of connection assigned to each edge. The amount of buffer space or channel bandwidth used is typically refered to as construction cost and combination of the propagation, transmission, and queuing delays is the delay cost. Constructing a tree with two optimization criteria is a common task in the design of a network in such a multicast environment [87]. The first criteria is the longest waiting time for the receivers which can be modeled by the diameter of the tree or the longest distance to the root. The second criteria is to minimize the total cost of constructing the tree. Thus, we can model the problem as Shallow-Light Steiner Tree (SLST) or Bounded Diameter Steiner Tree(BDST), in which we want to cover the terminal nodes with a minimum cost tree whose diameter is not greater than a given bound. Such multi-criteria network design problems have also applications in information retrieval [26] and VLSI designs (see [128, 97] and the references).

Network optimization problems with multiple cost functions, such as buy-at-bulk network design problems, have been studied extensively because of their applications. These problems can model, among others, situations where every edge e (link) can be either purchased at a fixed price $c(e)$ or rented at a price $r(e)$ per amount of flow (or load). The selected edges are required to provide certain bandwidth to satisfy certain demands between nodes of the graph. So if an edge is rented and there is a flow of $f(e)$ on that edge the cost for that edge will be $r(e) \cdot f(e)$ whereas if the edge is purchased, the cost will be $c(e)$ regardless of the flow. It can be shown that this problem and some other variations can be modeled using buy-at-bulk network design.

4.1 Problem formulations

In this chapter we study a general problem called Shallow-Light k-Steiner Tree (SLkST) in which instead of covering all the terminal nodes covering only k nodes is sufficient. This problem is defined formally below:

Definition 6 *In the SLkST problem we are given an undirected graph $G = (V, E)$, a cost function $c : E \rightarrow \mathbb{Q}^+$, a length function $\ell : E \rightarrow \mathbb{Q}^+$, a subset $T \subseteq V$ called terminals which includes a root node r, and a positive bound L. The goal is to find a Steiner tree over terminals T and rooted at r containing at least $k-1$ other terminals such that the cost of the tree (under c metric) is minimized while the diameter of the tree (under ℓ metric) is at most L.*

The main result of this chapter is the following theorem [77]:

Theorem 7 *There is a polynomial time $(O(\log^2 n), O(\log n))$-approximation for SLkST. More specifically, the algorithm finds a k-Steiner tree of diameter at most $O(L \cdot \log n)$ whose cost is at most $O(\text{OPT}^* \cdot \log^2 n)$ where OPT^* is the cost of an LP relaxation of the problem.*

Another closely related class of network design problems are Buy-at-Bulk network design problems. In this chapter we are specifically interested at Buy-at-Bulk k Steiner Tree (BBkST) defined below:

Definition 7 *Suppose we are given an undirected graph $G = (V, E)$, a set of terminals $T \subseteq V$ including root r, a sub-additive monotone non-decreasing cost function $f_e : \mathbb{Q}^+ \rightarrow \mathbb{Q}^+$ for each edge e, and positive demand values $\{\delta_i\}_i$, one for each $t_i \in T$. In the BBST problem the goal is to find an Steiner tree rooted at r to route the demands from terminals to the root which minimizes the sum of costs of the edges, where the cost of each edge e is $f_e(\delta(e))$ where $\delta(e)$ is the total demand routed over edge e. This is also refered to as single-sink* *buy-at-bulk problem. Similar to SLkST, one can generalize the BBST problem by having an extra parameter $k \geq 1$ in the input and a feasible solution is an r-rooted Steiner tree which contains at least k terminals (instead of all of the terminals). This way, we obtain the Buy-at-Bulk k-Steiner Tree (BBkST) problem.*

In this chapter we obtain the following result for BBkST:

Theorem 8 *There is an $O(\log^2 n \cdot \log D)$-approximation for BBkST, where D is the sum of demands.*

4.2 Related works

Multi-Objective Shortest Path (MOSP) problem is among the first two cost functions problems studied in the literature. In MOSP problem we are given an undirected graph $G = (V, E)$, two specified nodes $s, t \in V$, and two cost functions c and l over the edges and the goal is to find the shortest path between s and t with respect to c such that their distance with regard to l is not more than a given upper bound L. There is a FPTAS for this problem [67, 123]. Similar to MOSP, the constrained minimum spanning tree problem is defined over a graph with two cost functions c and l. The objective is to find a spanning tree in which total cost with respect to c is minimized and its total length with respect to l is not more than a given bound L. Ravi *et al.* [109] give a FPTAS for this problem.

Marathe *et al.* [97] develop a framework for two criteria approximation algorithms (Section 1.2). They studied different bi-criteria network design problems. The objective is to find a Steiner tree to minimize one criteria subject to a constraint on the second criteria. The criteria they considered are (i) total cost of the Steiner tree, (ii) diameter of the graph, or (iii) the maximum degree of the graph. They provide the following table in which rows indicate the first criteria and columns indicate the second one (some results are from other works). Each cell represents the best approximation factor for the problem of finding a Steiner tree to minimize the first criteria (mentioned in the row) subject ot a given bound on the second criteria (mentioned in the column).

Cost Measures	(i) Degree	(ii) Diameter	(iii) Total Cost
(i) Degree	$(O(\log n), O(\log n))$	$(O(\log^2 n), O(\log n))$ [108]	$(O(\log n), O(\log n))$ [110]
(ii) Diameter	$(O(\log^2 n), O(\log n))$ [108]	$(1+\gamma, 1+\frac{1}{\gamma})$	$(O(\log n), O(\log n))$
(iii) Total Cost	$(O(\log n), O(\log n))$ [110]	$(O(\log n), O(\log n))$	$(1+\gamma, 1+\frac{1}{\gamma})$

Figure 4.1: Bicriteria approximation algorithms for Steiner trees with different criteria. n is the number of nodes in the graph and γ is a sufficiently small real value

Koneman *et al.* [83] consider the problem of degree-bounded minimum diameter spanning tree in which the goal is to find a spanning tree with minimum diameter subject to a bound B on the maximum degree of all the nodes in graph. They give an $O(\sqrt{\log_B n})$-approximation algorithm for this problem with no violation on the degree bound.

Problem of minimum-cost low-degree spanning tree and its variations are studied extensively in the literature [53, 84, 85, 82, 30, 112, 56, 120]. Koneman [82] gives a spanning tree whose maximum degree is in $O(B + \log(n))$ with the cost of at most a constant larger than the optimum degree-B-bounded spanning tree. This result is later improved by Goemans [56], his algorithm gives a spanning tree that violates the degree bounds by only 2 units and cost of at most the optimum solution. Finally, Singh *et al.* [120] give an algorithm which violates the bound by only 1 unit and cost of at most the optimum solution. Their algorithm is esentially the best one can hope for, moreover, it works for the general case when every vertex has an upper bound and lower bound on its degree.

Meyerson [98] studies the online version of shallow-light Steiner tree in which the Steiner nodes may change during the time, and the algorithm may add edges to the tree accordingly, but the diameter of the tree should always be bounded by the given constant B and the cost is to be minimized. He gives an online algorithm for maintaining a tree whose diameter is kept in $O(B \log n)$ and its cost is at most $O(\log^2 n \cdot C^*)$ where C^* is the cost of optimal off line solution.

Kortsarz and Peleg [88] consider the problem of shallow-light Steiner tree for the case when there is no second cost function and the diameter of the graph is simply the number of edges between the two farthest apart pair. They give an $O(\log n)$-approximation (the diameter bound will not be violated) for the case when the diameter bound is a constant integer. This special problem is proved to be $(\ln n)$-hard even for the case when the diameter bound is 4 [17]. A similar question is studied in [1] in which the cost function is metric and they give an $O(\log n)$ approximation for the general diameter bound.

Basov and Vainshtein [19] consider graphs with $k \geq 2$ different nonnegative costs associated with each edge e and a cost function $c : \mathbb{R}^k \to \mathbb{R}_+$, trying to find a minimum-cost edge subset with a certain property such as paths, spanning trees, cuts, joins, etc. They proved these problems are weakly NP-hard and give simple approximation algorithms for them. For other results related to SLST and bicriteria network design problems see [79, 111, 91, 92].

In the multi-commodity buy-at-bulk problem we are given p source-sink pairs of terminals $\{s_i, t_i\}_{i=1}^p$ each with a demand δ_i. A subset of edges E' is feasible if for every $1 \leq i \leq p$ there is a s_i, t_i-path in $G' = (V, E')$. The goal is to minimize $\sum_{e \in E'} c(e) + \sum_i \delta_i \cdot \mathrm{dist}_{G'}(s_i, t_i)$ where the distance is with respect to length function ℓ. This model is referred to as *cost-distance*. Later in Section 4.3 we show that with a small constant factor loss in the approximation factor any approximation algorithm for this model can be extended to the general case where every edge has a function f_e.

In the uniform version of buy-at-bulk all the values along the edges are the same, i.e. $c(e) = c(e')$ and $\ell(e) = \ell(e')$, for all $e, e' \in E$ (we refer to the version we defined as non-uniform). The uniform multi-commodity buy-at-bulk has an $O(\log n)$-approximation [14, 18, 47]. There are constant-factor approximations for the single-sink uniform case and some other special cases [60, 63, 64, 93]. Meyerson *et al.* . [99] give a randomized $O(\log n)$-approximation for the (non-uniform) BBST and this was derandomized in [34] using an LP formulation. For the (non-uniform) multi-commodity version Chekuri *et al.* [32] give the first polylogarithmic approximation with ratio $O(\log^4 n)$. In [86] this is improved to $O(\log^3 n)$ if all the demands are polynomial in n. Some generalizations of these problems to higher connectivity are considered in [5, 61].

For hardness of approximation, Andrews [3] shows that unless NP $\subseteq$ ZPTIME $(n^{\mathrm{polylog}\, n})$ the buy-at-bulk multicommodity problem has no $O(\log^{1/2-\epsilon} n)$-approximation algorithm for any $\epsilon > 0$. For the BBST Chuzhoy *et al.* [41] show that the problem cannot be approximated better than $\Omega(\log\log n)$ unless NP $\subseteq$ DTIME$(n^{\log\log\log n})$.

The BBkST and SLkST problems generalize some classic problems such as Steiner tree and k-MST. In the k-MST problem we are given an undirected graph G and the objective is to find a minimum cost tree that covers at least k nodes of G. The k-MST problem [13, 24, 55] is the special case of SLkST when $L = \infty$ and also the bounded diameter spanning tree problem, studied in [68], is the special case when costs are zero. Also, the SLST problem studied in [97] is a special case of SLkST with $k = |T|$. Even the $k = |T|$ special case is NP-hard and also NP-hard to approximate within a factor better than $c \log n$ for some universal constant c [16].

Buy-at-bulk problems and their special cases have been studied through a long line of papers in the Operation Research and Computer Science communities after the problem was introduced by Salman *et al.* [116] (see e.g. [4, 5, 14, 28, 33, 60, 63, 64, 66, 86, 93, 99]).

4.3 Reduction from Buy-at-Bulk Steiner tree to shallow-light Steiner tree

In this section we show how to prove Theorem 8 from Theorem 7. First we show that the general definition of buy-at-bulk Steiner tree problem given in Section 4.1, with a function f_e for each edge e, is equivalent (with a small constant factor loss in the approximation) to the *cost-distance* formulation: The input is the same except that instead of function f_e for every edge e, we have two metric functions on the edges: $c : E \to \mathbb{Q}^+$ is called cost and $\ell : E \to \mathbb{Q}^+$ is called length. The cost of a feasible solution H is defined as: $\sum_{e\in H} c(e) + \sum_i \delta_i \cdot L(t_i)$, where $L(t_i)$ is the length (w.r.t ℓ) of the r, t_i-path in H.

It is easy to see that this formulation is a special case of buy-at-bulk since a linear function (defined based on c and ℓ) is also a sub-additive, non-decreasing and monotone function.

It turns out that an α-approximation for the cost-distance version implies a $(2\alpha + 2\epsilon)$-approximation algorithm for the buy-at-bulk version too for a fixed integer ϵ (see [4, 33, 99]). We bring its proof from [33] for the sake of completeness. We approximate the function f_e for each e by a collection of simple piece-wise linear functions of the form $a + b \cdot x$. We replace the edge e by a group of parallel edges with a linear cost functions. More precisely, given a function $f : \mathbb{Q}^+ \to \mathbb{Q}^+$, and a fixed $\epsilon \geq 0$, for integer $i \geq 0$ let $g_i : \mathbb{Q}^+ \to \mathbb{Q}^+$ be a linear function defined as $g_i(x) = f(a^i) + f(a^i)/a^i \cdot x$ where $a = (1 + \epsilon)$. It can be verified that if f is monotone, nondecreasing and sub-additive then for all $x \geq 1$, $\frac{a}{2+\epsilon} min_i g_i(x) \leq f(x) \leq min_i g_i(x)$.

As a result, an $O(\log^2 n \cdot \log D)$-approximation for cost-distance formulation of BBkST is also an $O(\log^2 n \cdot \log D)$-approximation for BBkST . For simplicity, we focus on the two cost function (cost-distance) formulation of buy-at-bulk from now on.

In general there are other models for the cost function in buy-at-bulk problems defined in [66]:

Model A. The *unique cost* model: In this model every edge e has either a buy cost $b(e)$ or a rent cost $r(e)$. For buy edges we have to pay $b(e)$ and for rent edges we have to pay $r(e) \cdot f(e)$.

Model B. The *rent or buy* model: In this model every edge e has both buy cost and rent cost and we can decide whether to buy this edge at cost $b(e)$ or to rent it at cost $r(e) \cdot f(e)$.

Model C. The *rent and buy* or *cost-distance* model: In this model every edge has both buy cost and rent cost and we have to pay $b(e) + c(e) \cdot r(e)$ for every edge that is going to be used.

Hajiaghayi *et al.* [66] show that all the above three models are in fact equivalent, i.e. a graph with a cost model from one of the above models can be transformed to another model with a polynomial time algorithm. Moreover this transformation is approximation factor preserving (see Section 1.2).

The only previous result for SLkST was [66] which had ratio $(O(\log^4 n), O(\log^2 n))$. This was obtained by applying the following theorem iteratively:

Theorem 9 *[66] There is a polynomial time algorithm that given an instance of the SLkST problem with diameter bound L returns a $\frac{k}{8}$-Steiner tree with diameter at most $O(\log n \cdot L)$ and cost at most $O(\log^3 n \cdot \text{OPT})$, where* OPT *is the cost of an optimum shallow-light k-Steiner tree with diameter bound L.*

Then a set-cover type analysis (see Section 1.2) yields an $(O(\log^4 n), O(\log^2 n))$-approximation for SL$k$ST. We should point out that this theorem was the main ingredient in a greedy type $O(\log^4 n)$-approximation for multi-commodity buy-at-bulk in [32, 33] as well. In [66], the following lemma was also proved:

Lemma 9 *[66] Suppose we are given an approximation algorithm for the SLkST problem which returns a solution with at least $\frac{k}{8}$ terminals and has diameter at most $\alpha \cdot L$ and cost at most $\beta \cdot \text{OPT}$. Then we can obtain an approximation algorithm for the BBkST problem such that given an instance of BBkST in which all demands $\delta_i = 1$ and a given parameter $M \geq \text{OPT}$ (where OPT is the optimum cost of the BBkST instance) returns a solution of cost at most $O((\alpha + \beta) \log k \cdot M)$.*

The corollary of this lemma, Theorem 9, and a binary search to find a close enough value for M was an $O(\log^4 n)$-approximation for the BBkST for unit demand instances; this can also be extended to an $O(\log^3 n \cdot \log D)$-approximation for general demands where $D = \sum_t \delta_t$ in [66]. Using Theorem 7 and Lemma 9 we can obtain Theorem 8 which improves the result of [66] for BBkST by a $\log n$ factor.

4.4 $(O(\log^2 n), O(\log n))$-approximation algorithm for shallow-light Steiner Tree

In this section we prove Theorem 7. To prove this theorem we combine ideas from all of [20, 33, 34, 86]. We first show that the algorithm of Marathe et al. [97] for SLST actually finds a solution with diameter at most $O(L \cdot \log |T|)$ whose cost is at most $O(\text{OPT}^* \cdot \log |T|)$, where OPT^* is the cost of a natural LP-relaxation, so we give a stronger bound (based on an LP relaxation) for the cost of their algorithm. This is based on ideas of [34] which gives a deterministic version of algorithm of [99] for BBST. Then we use an idea in [86] to write an LP for SLkST and use a trick in [20] for rounding this LP.

First we show that the algorithm of [97] in fact bounds the integrality gap of a natural LP relaxation for the SLST problem too. Recall that the instance of SLST consists of a graph $G = (V, E)$ with costs $c(e)$, lengths $\ell(e)$, terminal set $T \subseteq V$ including a node r. The goal is to find a Steiner tree H over T with minimum $\sum_{e \in H} c(e)$ such that the diameter w.r.t. ℓ function is at most L. First, let us briefly explain the algorithm of [97] for SLST. Denote the given instance of SLST by $\mathcal{I}$ and define graph F over terminals and the root as below. For every pair of terminals $u, v \in T$, let $b(u, v)$ be the (approximate) lowest c-cost path between them whose length (under ℓ) is no more than L (there is an FPTAS for computing the value of $b(u, v)$ [67]); let the cost of the edge between (u, v) in F be cost of $b(u, v)$. Lemma 10 is a known fact about the optimum solution $\mathcal{I}$.

Lemma 10 *[80] In the optimum solution $\mathcal{I}$, there is a pairing of the terminals (except possibly one if the number of them is odd) such that the unique paths connecting the pairs in the optimum are all edge-disjoint*

Proof. Consider a pairing which minimizes the number of overlapping edges of the connecting paths. We claim that in this pairing the connecting paths are edge-disjoint. Suppose for contradiction two connecting paths for the pairs (u, u') and (v, v') overlap over an edge xy. Without loss of generality suppose the connecting path for (u, u') consists of P_{ux},xy, and $P_{yu'}$ where P_{ab} represents the path connecting a to b. Similarly assume the connecting path for (v, v') consists of P_{vx}, xy, and $P_{yv'}$. Now, we pair (u, v) with a path selected from the edges in P_{ux} and P_{vx} (note that both P_{ux} and P_{vx} can share some edges), and pair (u', v') with a path selected from the edges of $P_{u'y}$ and $P_{v'y}$. It is easy to check that the number of overlapping edges drops by at least one, which is a contradiction. ■

It follows from this lemma that, the total cost of these connecting paths is at most the value of optimum solution, denoted by OPT, and the length of each of them is at most L. We consider a minimum cost maximum matching in F whose cost is at most $(1+\epsilon)$OPT. We find a minimum cost maximum matching in F and let us say terminals $\{u_i, v_i\}_i$ are paired. We pick one of the two (arbitrarily), say u_i and remove v_i from the terminal set; let this new instance be $\mathcal{I}'$. Clearly the cost of optimum solution on $\mathcal{I}'$, denoted by OPT$'$, is at most OPT (as the original solution is still feasible for the new instance $\mathcal{I}'$). Also, for any solution of $\mathcal{I}'$, we can add the paths defined by $b(u_i, v_i)$ to connect v_i to u_i. This gives a solution to instance $\mathcal{I}$ of cost at most OPT$' + (1+\epsilon)$OPT and the diameter increases by at most L. We can do this repeatedly for $O(\log |T|)$ iterations until $|T| = 1$, since each time the number of terminals drops by a constant factor.

Remark: A similar algorithm was designed in [99] to obtain an $O(\log n)$-approximation for BBST problem. Then an LP-based algorithm was presented by Chekuri *et al.* [34] to derandomize the algorithm of [99] for BBST.

We use the same approach as in [34] to bound the integrality gap of SLST. This LP relaxation is a flow-based LP (like those used in [33, 34]). We use an idea of [86] which only considers bounded lengths flow paths. For each terminal $t \in T$ let $\mathcal{P}_t$ be the set of all paths of length at most L from t to r in G. We assume that the terminals are at distinct nodes (we can enforce this by attaching some dummy nodes with edge cost and length equal to zero to the original nodes). Therefore, $\mathcal{P}_t$ and $\mathcal{P}_{t'}$ are disjoint. For every edge e we have an indicator variable x_e which indicates whether edge e belongs to the tree H or not. For each path $p \in \bigcup_t \mathcal{P}_t$, $f(p)$ indicates whether path p is used to connect a terminal to the root.

It is easy to check that the answer to following integer program is actually an optimum solution to the SLST:

We can relax the integer constraints (3) in order to make it an LP as follows:

$$
\begin{array}{llrcll}
\textbf{IP-SLST} & \min & \sum_e c(e)\cdot x_e & & & \\
& \text{s.t.} & \sum_{p\in P_t|e\in p} f(p) & \leq & x_e & \forall e\in E,\;\; t\in T \quad (1)\\
& & \sum_{p\in\mathcal{P}_t} f(p) & \geq & 1 & t\in T \quad (2)\\
& & x_e, f(p) & \in & \{0,1\} & \forall e\in E,\;\; p\in\cup_t\mathcal{P}_t \quad (3)
\end{array}
$$

$$
\begin{array}{llrcll}
\textbf{LP-SLST} & \min & \sum_e c(e)\cdot x_e & & & \\
& \text{s.t.} & \sum_{p\in P_t|e\in p} f(p) & \leq & x_e & \forall e\in E,\;\; t\in T \quad (4)\\
& & \sum_{p\in\mathcal{P}_t} f(p) & \geq & 1 & t\in T \quad (5)\\
& & x_e, f(p) & \geq & 0 & \forall e\in E,\;\; p\in\cup_t\mathcal{P}_t \quad (6)
\end{array}
$$

Let (x^*, f^*) be an optimal solution to LP-SLST with cost $\textsc{opt}^*$. Note that this LP has exponentially many variables, however we do not need to solve this LP; instead we only show the algorithm of [97] finds a solution whose cost is bounded by $O(\log|T|)$ factor of $\textsc{opt}^*$. Define graph $F = (V(F), E(F))$ over terminals T and r as above, i.e. the cost of edge $e = (u, v) \in E(F)$ for two terminals $u, v \in V(F)$ will be the cost of $(1+\epsilon)$-approximate minimum c-cost u, v-path of length at most L computed using algorithm of [67].

We show that the cost of algorithm of [97] is at most $O(\textsc{opt}^* \cdot \log|T|)$ while the diameter is at most $O(L \cdot \log|T|)$. The proof of following lemma is analogous of Lemma 2.1 in [34].

Lemma 11 *Graph F contains a matching M of size at least $|T|/3$ whose cost is at most $(1+\epsilon)\textsc{opt}^*$.*

Proof. The structure of the proof is as follow. We show that the optimal value of the dual form of LP-SLST in G is not less than the optimal value of a dual LP for min-cost perfect matching defined in graph F. Therefore, by LP duality theorem, the value of LP-SLST ($\textsc{opt}^*$) is equal to the value of its dual and is greater than the value of min-cost perfect matching LP. Then we argue that from a basic feasible solution of the matching LP we can make a integral matching whose cost is not greater than the min-cost perfect matching LP's value and has at least $|T|/3$ edges. Taking into consideration that graph F is built with edges that are $(1+\epsilon)$ approximation of the actual values, we conclude that M costs at most $(1+\epsilon)\textsc{opt}^*$.

Consider the following LP for the min-cost perfect matching (MMP) in graph F, along with its dual (MMD) in which $b^*(u, v)$ represents the optimal minimum c-cost (u, v)-path of length at most L found by [67]'s algorithm.

$$
\begin{array}{rcll|rcll}
\multicolumn{4}{l|}{\textbf{MMP}} & \multicolumn{4}{l}{\textbf{MMD}}\\
\multicolumn{4}{c|}{\min \sum_{(u,v)\in E(F)} b^*(u,v)x(u,v)} & \multicolumn{4}{c}{\max \sum_{u\in V(F)} y(u)}\\
\sum_{v\in V(F)} x(u,v) & = & 1 & \forall u\in V(F) & y(u)+y(v) & \leq & b^*(u,v) & \forall u,v\in E(F)\\
x(u,v) & \geq & 0 & \forall (u,v)\in E(F) & y(u) & \geq & 0 & \forall u\in V(F)
\end{array}
$$

We show that the optimal solution of dual LP for SLST (D-SLST) has value at least as big as the optimal value of MMD which implies the optimal value of MMP is not greater than OPT* using LP duality [118].

The LP D-SLST is the following:

D-SLST:

$$\begin{array}{rcllr} \max & \sum_{t\in T}\alpha_t & & & \\ \sum_{t\in T}\beta_e^t & \leq & c(e) & e\in E & (7)\\ \alpha_t-\sum_{e\in p}\beta_e^t & \leq & 0 & t\in T, p\in\mathcal{P}_t & (8)\\ \alpha_t,\beta_e^t & \geq & 0 & e\in E, t\in T & (9) \end{array}$$

Let y_t^* be an optimal solution for MMD and $d(u,v)$ be the shortest path between u and v with regard to cost function c in G. We make a ball B_t of radius y_t^* around each $t\in T$ in G. More formally, let B_t be a set containing all the nodes v with $d(v,t)\leq y_t^*$ and the edges $e=(u,v)$ which at least one of $d(u,t)<y_t^*$ or $d(v,t)<y_t^*$ is true. Let $g^t(e)$ be the fraction of edge $e=(u,v)$ contained in ball B_t, in other words $g^t(e)=min\{\frac{y_t^*-min\{d(u,t),d(v,t)\}}{c(e)},1\}$.

Define $\hat{\beta}_e^t=g^t(e)\cdot c(e)$ and $\hat{\alpha}_t=y_t^*$. In the following we prove that $\hat{\beta}$ and $\hat{\alpha}$ is a feasible solution to D-SLST. It is clear that $\hat{\beta}$ and $\hat{\alpha}$ do not violate constraints (9). The main observation here is that balls $\{B_t\}_{t\in T}$ are disjoint as we have $y(u)+y(v)\leq b^*(u,v),\ \forall(u,v)\in E(F)$ in MMD. This observation directly shows that constraints (7) are not violated. Note that r is also in $V(F)$ so the ball B_r is also disjoint from the other balls. As a result, each path $p\in\mathcal{P}_t$ consists of at least one part in B_t and one part in B_r, therefore p is longer than the radius of B_t which makes constraints (8) be tight. Thus, $\hat{\alpha}$ and $\hat{\beta}$ are feasible solution to D-SLST with value at least $\sum_{u\in V(F)}y_u^*$ and hence D-SLST is at least as big as MMD.

Now we show how to find an integral matching containing at least $|T|/3$ nodes. Notice that there is no odd-set constraints in MMP which makes it integral (the integral LP with odd set constraints is known as Edmond's matching polytope). It is well known that in a basic feasible solution to MMP all $x(u,v)$ are in the set $\{0,\frac{1}{2},1\}$ and the edges with value $\frac{1}{2}$ make odd cycles [118]. This can be proved from the fact that any basic feasible solution cannot be written as convex combination of two other feasible solutions.

Let x^* be a basic feasible solution to MMP. We add all the edges e with $x^*(e)=1$ to M. Moreover, from each odd cycle O, it is easy to see that we can add at least $\frac{|O|}{3}$ of its edges to M such that the total cost of added edges is less than $\sum_{e\in O}x^*(e)\cdot c(e)$ taking into account that $x^*(e)=\frac{1}{2}$ for all $e\in O$. Therefore, M has at least $\frac{V(F)}{3}$ edges whose cost is not more than the MMP's value. As we showed that the value of MMP is not greater than OPT* and as we are able to find $b^*(u,v)$ for each edge of F with accuracy $1+\epsilon$, the proof of lemma follows. ■

Suppose we have a matching M as above with cost C_M. For every pair of terminals u_i,v_i matched by M pick one of the two as the hub for connecting both of them to r and remove the other one from T. Let OPT′ be the LP cost of the new instance. Note that the terminals for the new instance are a subset of the terminals in the original one, therefore the current solution (x^*, f^*) is still feasible for the LP defined for the terminals in the new instance; therefore $\textsc{opt}' \leq \textsc{opt}^*$. Also, the cost of routing all the terminals that were deleted to their hubs is at most $C_M \leq (1+\epsilon)\textsc{opt}^*$. Notice that the number of terminals ($|T_{i+1}|$) for the new instance is at most $\lceil \frac{2}{3} T_i \rceil$ of the previous instance (if there are only two terminals or one terminal we can just connect them to r using algorithm of [67]). Doing this iteratively, at each iteration we drop the number of terminals by a factor of at least $\frac{2}{3}$, so overall we repeat this process for $O(\log |T|)$ times. As the cost increases by $\textsc{opt}^*$ and diameter increases by L at each iteration, we obtain a solution whose cost is at most $O(\textsc{opt}^* \cdot \log |T|)$ and the diameter of the solution is at most $O(L \cdot \log |T|)$.

Now we prove Theorem 7. Our algorithm is based on rounding a natural LP relaxation of the problem. Before presenting the LP we explain how we preprocess the input. We first guess a value $\textsc{opt}'$ such that $\textsc{opt} \leq \textsc{opt}' \leq 2\textsc{opt}$. We show that for a guessed value of $\textsc{opt}'$ the solution returned by the algorithm satisfies the bounds (i.e. cost of the final tree is $O(\textsc{opt}' \cdot \log |T|))$ if $\textsc{opt}' \geq \textsc{opt}$. On the other hand if the algorithm fails then $\textsc{opt}' < \textsc{opt}$. Thus in order to find $\textsc{opt}'$ we can do binary search between zero and the largest possible value of $\textsc{opt}$ (e.g. $\sum_{e \in E} c(e)$), and according to output of the algorithm we can adjust our guess for $\textsc{opt}'$.

We define $V' \subseteq V$ to be the set of vertices v such that v has a path p to r with $c(p) \leq \textsc{opt}'$ and length at most L again using the algorithm of [67]. Clearly, every vertex of any optimum solution must belong to V'. We can safely delete all the vertices of $V \setminus V'$; so let G be the new graph after pre-processing. The following LP is similar to LP-SLST, except that we have an indicator variable y_t for every terminal.

$$
\begin{array}{llrclll}
\textbf{LP-SL}k\textbf{ST} & \min & \sum_e c(e) \cdot x_e & & & & \\
& \text{s.t.} & \sum_{p \in P_t | e \in p} f(p) & \leq & x_e & \forall e \in E, \quad t \in T & (10) \\
& & \sum_{p \in \mathcal{P}_t} f(p) & \geq & y_t & t \in T & (11) \\
& & \sum_{t \in T} y_t & \geq & k & & (12) \\
& & y_t & \leq & 1 & t \in T & (13) \\
& & x_e, f(p) & \geq & 0 & \forall e \in E, \quad p \in \cup_t \mathcal{P}_t & (14)
\end{array}
$$

If we replace y_t in the constraints (11) with 1 and drop constraints (12) and (13) (and remove y_t variables) then we obtain the LP-SLST. Our rounding algorithm is similar to those in [33, 20] for two completely different problems (density version of Buy-at-Bulk Steiner tree in [33] and k-ATSP tour in [20]). In particular we use the approach of [20] to avoid losing an extra $O(\log n)$ factor in the ratio by giving a direct algorithm for rounding the LP-SLkST instead of reducing the problem to the density version.

Since we need to solve this LP let's briefly say how we can do that although LP-SLkST has an exponential number of variables. First we write its dual as follows (note that instead of each constraint in primal we have a variable in the dual and instead of each variable in the primal we have a constraint in the dual):

$$\begin{array}{llrcll}
\textbf{LPD-SL}k\textbf{ST} & \max & k\gamma - \sum_{t\in T}\lambda^t & & & \\
& \text{s.t.} & \sum_{t\in T}\alpha^t(e) & \leq & C(e) & \forall e\in E \\
& & \beta^t - \sum_{e\in p}\alpha^t(e) & \leq & 0 & \forall t\in T, p\in\mathcal{P}_t \\
& & -\beta^t+\gamma & \leq & \lambda^t & t\in T \\
& & \alpha^t(e),\beta^t,\gamma,\lambda^t & \geq & 0 & \forall e\in E, \quad p\in\cup_t\mathcal{P}_t
\end{array}$$

There are exponentially many constraints in the dual LP but one can obtain an optimum feasible solution if one can give a separation oracle for it. It is easy to verify that a shortest-path algorithm gives a separation oracle for the dual LP. Moreover, from the dual solution we can obtain an optimal feasible solution for the primal form in which the number of variables that are non-zero is polynomially bounded. For the detailed explanation of the process see [59].

Suppose that (x^*, y^*, f^*) is an optimum feasible solution to LP-SLkST with value OPT^*. Our first step is to convert (x^*, y^*, f^*) to an approximate solution in which y_t values are of the form 2^{-i}, $0 \leq i \leq \lceil 3\log n\rceil$. Lemmas 12 and 14 are analogous of Lemma 9 and Theorem 10 in [20].

Lemma 12 *There is a feasible solution (x', y', f') to LP-SLkST of cost at most 4OPT^* such that each y'_t is equal to 2^{-i} for some $0 \leq i \leq \lceil 3\log n\rceil$.*

Proof. Let (x^*, y^*, f^*) be an optimal feasible solution to LP-SLkST. We set $x'_e = 4x^*_e$ for all $e \in E$ and $f'(p) = \min(4f^*(p), 1)$ for all $t \in T$ and $p \in \mathcal{P}_t$. For each $t \in T$ and i such that $1/2^i \leq y^*_t < 1/2^{i-1}$, if $i > \lceil 3\log(n)\rceil$ set $y'_t = 0$; otherwise, $y'_t = \min(1, 1/2^{i-2})$. It is easy to see that the cost of (x', y', f') is at most 4OPT^*. Also, the first constraint is satisfied. The second constraint is also satisfied since it is clearly satisfied if $f'(p) = 4f^*(p)$ for all $p \in \mathcal{P}_t$, and if this is not the case then at least one $f'(p) = 1$ which is at least as big as y'_t since $y'_t \leq 1$. So it only remains is to show that the last constraint is satisfied.

Let Y_0 be the set of terminals t for which $y^*_t > 0$ but $y'_t = 0$. These are the only terminals whose y value has decreased. Note that for each $t \in Y_0$: $y^*_t \leq 1/n^3$; so $\sum_{t\in Y_0} y^*_t \leq 1/n^2$. Let Y_1 be the set of terminals t with $y'_t = 1$. If $|Y_1| \geq k$, then the last constraint clearly holds. Otherwise, $|Y_1| \leq k-1$ which implies that $\sum_{t\notin Y_1} y^*_t \geq 1$ must be true; therefore $\sum_{t\notin Y_1\cup Y_0} y^*_t \geq 1 - 1/n^2 \geq 1/n^2 \geq \sum_{t\in Y_0} y^*_t$. Also, note that for each vertex $t \notin Y_0\cup Y_1$: $y'_t \geq 2y^*_t$. Thus, the amount $\sum_{t\in Y_0} y^*_t$ that is decreased in y' is compensated for by $\sum_{t\notin Y_0\cup Y_1} y'_t$ therefore the last constraint holds too. ■

Let T_i be the set of terminals with $y'_t = 2^{-i}$ and $k_i = |T_i|$, for $0 \leq i \leq \lceil 3\log n\rceil$. Note that $\sum_{i=0}^{\lceil 3\log n\rceil} 2^{-i}\cdot k_i \geq k$. Consider the instance of SLST defined over $T_i \cup \{r\}$. First observe that we can obtain a feasible solution (x'', f'') to LP-SLST over this instance of SLST of cost at most $2^{i+2}\cdot\text{OPT}^*$ in the following way: define $x''_e = 2^i \cdot x'_e$ for each edge

$e \in E$ and $f''(p) = 2^i \cdot f'(p)$ for each $t \in T_i$ and path $p \in \mathcal{P}_t$. The cost of this solution is $O(2^{i+2} \cdot \text{OPT}^*)$ since $x''_e = 2^{i+2} \cdot x^*_e$. Now since we proved the integrality gap of LP-SLST is $O(\log n)$, we obtain the following:

Lemma 13 *For each T_i, we can find a Steiner tree over $T_i \cup \{r\}$, rooted at r of total cost $O(2^{i+2} \cdot \text{OPT}^* \cdot \log n)$ and diameter $O(L \cdot \log n)$.*

Next we prove the following lemma.

Lemma 14 *For every $0 \leq i \leq \lceil 3 \log n \rceil$ and given a Steiner tree H_i over T_i with total cost $O(2^{i+2} \cdot \text{OPT}^* \cdot \log n)$ and diameter $O(L \cdot \log n)$ we can find a Steiner tree H'_i rooted at some $r_i \in T_i$ containing at least $\lceil k_i/2^i \rceil$ terminals of T_i of cost at most $O(\text{OPT}^* \cdot \log n)$ and diameter at most $O(L \cdot \log n)$.*

For now, let us assume this lemma and see how to complete the proof of Theorem 7. Suppose that H'_i is the Steiner tree promised by Lemma 14 which contains $\lceil k_i/2^i \rceil$ terminals of T_i and is rooted at a node r'_i. Let p_i be the minimum cost path from r'_i to r with length at most L (note that because of the pre-processing we did, such path p_i exists). Let $H''_i = H'_i \cup p_i$ and let $H = \bigcup_i H''_i$. Observe that H contains at least $\sum_{i=0}^{\lceil 3 \log n \rceil} 2^{-i} \cdot k_i \geq k$ terminals. Also, the total cost of H is at most $\sum_{i=0}^{\lceil 3 \log n \rceil} c(H''_i) \leq O(\text{OPT}^* \cdot \log^2 n)$. Since the diameter of each H''_i is at most $O(L \cdot \log n)$ (because diameter of H'_i is at most $O(L \cdot \log n)$ and we added a path p_i of length at most L to H'_i) and since all of H''_i's share the root r, the diameter of H is at most $O(L \cdot \log n)$ as well. This completes the proof of Theorem 7.

So it only remains to prove Lemma 14. If we are given Steiner tree H_i over T_i we use the following lemma with $\beta = \lceil k_i/2^i \rceil$ to edge-decompose H_i into trees $F_1, \ldots, F_d$ such that the number of terminals of each F_i is in $[\beta, 3\beta)$. It follows that $d = \Theta(2^i)$ and so by an averaging argument, at least one of F_i's has cost $O(\text{OPT}^* \cdot \log n)$. The proof of the following lemma is essentially the same as Lemma 2 in Chapter 2.

Lemma 15 *Given a rooted tree F containing a set of k terminals and given an integer $1 \leq \beta \leq k$ we can edge-decompose F into trees $F_1, \ldots, F_d$ with the number of terminals of each F_i in $[\beta, 3\beta)$, $1 \leq i \leq d$.*

Proof. If $k < 3\beta$ then F satisfies the statement of the theorem. The idea is to "split away" (defined below) trees whose number of terminals is in interval $[\beta, 2\beta)$ until we are left with one tree whose number of terminals is in $[\beta, 3\beta)$. ■

4.5 Relation to other network design problems

In this section we study how our result for BBkST can give better approximation factor for some other network design problems. In order to do this we present an approximation factor preserving reduction from them to the BBkST problem. The study of the reductions were observed by [99] for the BBST problem. Recall that [99] gives an $O(\log n)$-approximation algorithm for BBST. In this section we generalize those reductions to the case in which instead of covering all the terminals covering only k of them is sufficient.

4.5.1 Multicast tree design

In this problem we are given an undirected graph $G = (V, E)$ with cost and delays on the edges, a source node $s \in V$, and a subset of receivers $R \subseteq V$. The objective is to find a tree which minimizes the sum of edges' cost plus the sum of delay seen by every receiver. It is easy to see that this problem is an easy case of BBST where delays are represented by the length of the edges and all the demands are 1. This problem is studied in the network community [21, 43, 69, 81, 90, 106, 124], and some heuristics are given. As a result of this equivalence, [99] gives an $O(\log |R|)$ approximation for the general problem and our algorithm gives an $O(\log^3 n)$-approximation factor for the k-multicast tree design in which instead of covering R, covering at least k of them is sufficient.

4.5.2 Extended single-sink buy-at-bulk

In the Extended Single-Sink Buy-at-Bulk (ESSBB) problem [117], we are given an undirected graph $G = (V, E)$ in which every edge e has a length $l(e)$, a subset of terminals $T \subseteq V$ in which each terminal t_i has demand δ_i, a single-sink t, and a set of P pipes in which every pipe i has a cost c_i per unit of length and capacity u_i. The demands should be routed to t along a tree. The objective is to find a tree and buy pipes along its edges such that all the demands can be routed using the pipes and minimize the total cost for buying pipes. Similarly, we can generalize it to k-ESSBB problem if servicing k of the terminals is sufficient. It is assumed that P is in $O(poly(|V|))$.

For a given instance $\mathcal{I}$ with an optimum solution $T_{\mathcal{I}}$ with cost $\text{OPT}_{\mathcal{I}}$ of ESSBB we make a corresponding instance $\mathcal{I}'$ for BBST as follow. We replace each edge of the graph with edges $e_1, \ldots, e_P$ in which edge e_i has costs $(l(e)c_i, l(e)\frac{c_i}{u_i})$. This will be an instance of cost-distance BBST. We claim [14, 117, 99] that the cost of the optimum tree $T_{\mathcal{I}'}$ in $\mathcal{I}'$ is at most $2\text{OPT}_{\mathcal{I}}$, and as the cost of every tree in $\mathcal{I}'$ is more than its cost in $\mathcal{I}$ every α-approximation for $\mathcal{I}'$ is a 2α-approximation for $\mathcal{I}$.

To prove the claim, suppose $T_{\mathcal{I}}$ has d amount of flow on the edge e_i, thus it pays $l(e_i)c_i\lceil \frac{d}{u_i} \rceil$ in $\mathcal{I}$ and $l(e_i)c_i(1 + \frac{d}{u_i})$ on $\mathcal{I}'$. It is easy to see that the amount paid on $\mathcal{I}'$ is at most twice of the one in $\mathcal{I}$. Therefore, the optimum solution in $\mathcal{I}'$ has a cost at most twice of $\text{OPT}_{\mathcal{I}}$.

The $O(\log n)$-approximation algorithm of [99] implies an $O(\log n)$-approximation for the ESSBB problem and our algorithm for BBkST implies an $O(\log^3 n)$-approximation for the

k-single-sink buy-at-bulk problem. The best approximation factor for ESSBB problem has been improved down to a constant factor in a series of papers [14, 60, 64]. We are not aware of approximation algorithms for k-single-sink buy-at-bulk problem.

We can assume that instead of using a universal set of pipes for all the edges, a different subset of pipes is available for each edge. It is clear that the reduction mentioned above is still applicable for this instance, however all the previous results that did not use BBST assume that all types of the pipes are available for all the edges. This generalization is especially important in real world [99] when cost of buying, installing, and maintenance may vary in different places.

4.5.3 Priority Steiner tree

Priority Steiner Tree (PST) [29] generalizes the Steiner tree problem. In this problem we are given an undirected graph $G = (V, E)$, a set of terminals $T \subseteq V$, and a root $r \in V$. Each edge e has a priority $p(e)$ and a cost $c(e)$, and each terminal t has a priority $p(t)$. The goal is to connect every terminal $t \in T$ to r with a path in which every edge has a priority at most $p(t)$. The objective is to minimize the total cost of the network.

It is shown that PST is a special case of BBST [41, 99] by giving a reduction from it to the special instance of ESSBB introduced in Section 4.5.2 where each edge has its own types of pipes. We give its outline here. For a given instance of PST ($\mathcal{I}$), assume $C = max_e c(e)$, $min_e c(e) \geq 1$ by scaling, and there are q priority levels. For each node v in $\mathcal{I}$ with priority level i we have a node in the corresponding instance of ESSBB ($\mathcal{I}'$) with demand $\delta'_v = (nC)^{5(q-i)}$. We assume that for each edge e in $\mathcal{I}$ with priority level i, there is only one type of pipes with capacity $u'_e = (nC)^{5(q-i)+2}$, cost $c'(e) = c(e)$, and length $l'(e) = 1$ for the corresponding edge in $\mathcal{I}'$.

For an optimal tree s to $\mathcal{I}$ we can make a tree s' to $\mathcal{I}'$ by buying all the pipes corresponding to the edges of s. We show that s' is feasible to $\mathcal{I}'$. Suppose an edge e has priority level i, note that sum of the demands for all the terminals in s' whose corresponding terminals in s have priorities greater than or equal to i is at most $n\sum_{j \geq i}(nC)^{5(q-j)} \geq (nC)^{5(q-i)+2} = u'_e$. This shows that the pipe capacities in s' are not violated, thus s' is a feasible solution to $\mathcal{I}'$ with the same cost. Conversely, for a given optimal tree s' in $\mathcal{I}'$ we show that its corresponding tree s in $\mathcal{I}$ is feasible. The demand for a terminal with corresponding priority i is routed over the edges with corresponding priority at most i; otherwise, at least $(nC)^3$ pipes need to be purchased in s' for the violated edge which costs $(nC)^3$. However, we can buy all the edges with cost n^2C, therefore in the optimal solution in $\mathcal{I}'$ all the demands are routed through the edges with the valid corresponding priority. Thus, an optimal solution for ESSBB can be transformed to an optimal solution for PST.

Charikar *et al.* [29] present an $O(\log |T|)$-approximation algorithm for PST. As a result of the previous reduction, the algorithm of [99] implies a similar approximation guarantee and our algorithm give an $O(\log^3 n)$-approximation for k-PST in which servicing only k terminals is sufficient.

There are also approximation preserving reductions from facility location, but-at-bulk facility location, and multilevel facility location to BBST, but they have constant approximation factors with other approaches [99].

4.6 Future works

One of the most important questions for SLST and SLkST problems is to answer whether there exists a true approximation (*i.e.* not bicriteria) for these problems. Note that as described in Section 4.2 there are $O(\log n)$-approximation algorithms for unit costs and metric ℓ. The current approximation algorithms violate both the bound on L and the optimality of the cost with respect to c for general ℓ and c. Another major question is to find a bicriteria approximation with a constant ratio for one of the criteria.

The current approximation ratio for BBkST is achieved by using SLkST algorithm during which we lose another $O(\log n)$ factor. It is an interesting question whether the current $O(\log n)$-approximation fator of [99, 34] for BBST can be used to get a better approximation ratio for BBkST.

A hardness factor of $\Omega(\ln n)$ is known for SLST in the special case of unit cost ℓ with the bound of 4 [17]. An open question is whether there is a better hardness lower bound for general ℓ or not. It would be interesting to know if it is possible to find an $(O(\log n), O(1))$-approximation, or it is hard to obtain an approximation better than $\Omega(\log n)$ even if one is allowed to violate the length constraints by a constant factor.

Chapter 5

The $(k, 2)$-subgraph

A major line of research in network design problems has focused on problems with connectivity requirements. As an example, the famous minimum spanning tree problem is to design a minimum cost network with connectivity requirement of 1 between all vertices. This problem can be generalized to the edge-connectivity requirement of λ, which is motivated from the real world applications where a certain level of reliability is required for the network. In other words if a connection is lost between two nodes then the flow of data can be maintained through other paths. A well-known problem in this class is the minimum cost λ-edge-connected spanning subgraph problem in which the objective is to find a minimum cost subgraph which is λ-edge-connected and covers all the nodes. This problem is further generalized to the generalized Steiner network design in which there is a different connectivity requirement between each pair of nodes [122].

Another generalization for network design problems is to require covering at least k nodes instead of covering all the nodes in the graph. The input for these problems has an integer k, and the goal is to find a subgraph satisfying the connectivity requirements with a lower bound k on the total number of vertices. The most well-studied problem in this class is the minimum k-spanning tree problem, a.k.a. k-MST which is introduced in the previous chapter. Recall that in this problem we are seeking a minimum cost tree spanning at least k vertices.

5.1 Problem Formulation

A natural common generalization of both the k-MST problem and the minimum cost λ-edge-connected spanning subgraph problem is the (k, λ)-subgraph problem introduced in Lau *et al.* [94] which is defined formally below:

Definition 8 *In the (k, λ)-subgraph problem, we are given a (multi-)graph $G = (V, E)$ with a cost function $c : E \to \mathbb{Q}^+$, and a positive integer k. The goal is to find a minimum cost λ-edge-connected subgraph containing at least k vertices.*

We should point out that the cost function c is arbitrary (i.e. does not necessarily satisfy the triangle inequality). Furthermore, we are not allowed to take more copies of an edge than what is presented in the graph. In particular, if G is a simple graph the solution must be simple too.

In this chapter we focus on the case of $\lambda = 2$, and prove the following theorem:

Theorem 10 *There is an $O(\log n)$-approximation algorithm for the $(k, 2)$-subgraph problem.*

This improves the result of [94] for the $(k, 2)$-subgraph problem by an $O(\log n)$ factor.

5.2 Related works

The (k, λ)-subgraph problem contains some classical problems as special cases. For example, the $(k, 1)$-subgraph problem is the k-MST problem and $(|V|, \lambda)$-subgraph is simply asking for a minimum cost λ-edge-connected spanning subgraph.

The k-MST problem is well studied in the field of approximation algorithms. The first approximation algorithm for this problem has a ratio of $O(\sqrt{k})$ [113] which is improved to $O(\log^2 k)$ in [12] and then to $O(\log k)$ in [107], and finally down to a constant in [25]. After a series of papers [25, 54, 9] improving the constant factor, Garg [55] achieves a 2-approximation factor for k-MST which is the best ratio until now. For the special case when nodes are in Euclidean plane a PTAS is known [8, 100].

Another closely related problem is k-TSP in which the objective is to find a walk in the graph such that it covers at least k nodes and return back to the initial position. This problem has its motivation from the vehicle routing problems where serving at least k clients is sufficient [114, 23, 35]. Similar techniques as in k-MST work also for k-TSP. Currently [55] is the best approximation algorithm for it with an approximation factor of 2. k-TSP in directed graphs is referred to as k-Asymmetric TSP (k-ATSP). Bateni and Chuzhoy [20] give an $O(\log^2 n/\log\log n)$-approximation algorithm for the k-ATSP . They also give an $O(\log^2 n)$ for the k-stroll problem in which the start and end point of the walk are given in the input.

The best approximation factor for minimum cost λ-edge-connected spanning subgraph is 2 due to the famous result of Jain [72]. He actually gave an algorithm with the same approximation guarantee for a more general problem called generalized Steiner network problem. In the generalized Steiner network problem instead of universal connectivity requirement λ, there is a given connectivity requirement $r(u, v)$ for each pair u and v of nodes. Moreover, there is a bound on the number of copies we can select from each edge. Connectivity problems have many different variations and there are several approximation and hardness of approximation results for them (for a survey on these results see [89]).

For the $(k,2)$-subgraph problem, an $O(\log n \cdot \log k)$-approximation is presented in [94]. For the more general problem of requiring the k-subgraph to be 2-node-connected an $O(\log n \cdot \log k)$-approximation is presented in [37]. These are the best known approximation algorithms for the $(k,2)$-subgraph problem. In [61] using a different approach an $O(\log^3 n)$-approximation is given for the problem. For metric cost functions, Safari and Salavatipour [114] present an $O(1)$-approximation for (k,λ)-subgraph (the constant is very large though).

In the densest k-subgraph problem we are given a graph G and the goal is to find a subgraph with k vertices which has the maximum number of induced edges. It is proved in [94] that the minimum densest k-subgraph problem has a poly-logarithmic reduction to the (k,λ)-subgraph problem. More precisely the following theorem is proved:

Theorem 11 *[94] An α-approximation algorithm for the (k,λ)-subgraph problem (even for the unweighted case) for arbitrary λ implies an $(\alpha \log^2 k)$-approximation algorithm for the Densest k-Subgraph problem.*

The densest k-subgraph problem is considered to be an extremely difficult problem (the best approximation algorithm for it has ratio $O(n^{\frac{1}{4}+\epsilon})$ [22]). This along with Theorem 11 show that for general λ, (k,λ)-subgraph problem is a very hard problem too.

5.3 An $O(\log n)$-approximation algorithm for $(k,2)$-subgraph problem

In this section we prove Theorem 10. This is based on rounding an LP relaxation of the problem similar to the one presented in [94]. Again we use the trick of [20] to round this LP and use the ideas of [94] to prune a partial solution.

In fact (similar to the algorithm in [94]) our algorithm works for a slightly more general case in which along with the weighted graph $G=(V,E)$ and integer k we are also given a set of terminals $T \subseteq V$ and the goal is to find a minimum cost 2-edge-connected subgraph that contains at least k terminals. Since our algorithm is based on that of [94], let us briefly explain how their algorithm works. The algorithm of [94] is for the rooted version of the problem, in which we are given an extra parameter $r \in V$ in the input and the solution must contain root r. Since one can try every possible vertex as the root, we can reduce the un-rooted version to the rooted version as well. A partial solution is a 2-edge-connected subgraph containing the root and the density of a partial solution is the ratio of the total cost of the edges over the number of terminals it contains.

The algorithm of [94] is based on adding a good density partial solution that covers some new terminals iteratively until the number of terminals connected to r (covered) is at least k. They presented an $O(\log n)$-approximation for finding good density partial solutions using an LP rounding procedure and we have to repeat the procedure until the number of terminals covered is at least k. One has to be careful as in an iteration where we are looking to cover k' terminals (for some $k' \leq k$) it is possible to find a partial solution with much larger than k' terminals (and so the combined solution has much larger than k terminals). In that case the algorithm has to be able to prune the partial solution to obtain a good density solution with about k' terminals. Lau *et al.* [94] present an algorithm for this pruning step which we will use too. Lemma 3 shows the final approximation ratio for the algorithm would be $O(\log n \cdot \log k)$.

Lemma 16 *If at each iteration the algorithm of [94] (1) finds a* 2*-edge-connected subgraph with density at most* $O(\log n \cdot \text{OPT}_s)$ *where* OPT_s *is the best density among the densities of all the subgraphs over the uncovered terminals, and (2) the number of covered terminals does not exceed* k*, then the algorithm is an* $O(\log n \cdot \log k)$*-approximation algorithm.*

Proof. Since each partial solution contains r, the union of all the partial solutions are 2-edge-connected. A simple set-cover type analysis (*i.e.* applying the set cover algorithm introduced in Theorem 3 with $f(n) = \log n$) shows that the algorithm of [94] is an $O(\log n \cdot \log k)$-approximation algorithm. ■

Our algorithm will directly round an LP relaxation, instead of iteratively finding good density partial solutions. This is similar to the overall structure of the algorithm we presented for the SLkST. Note that, it is sufficient to find a solution in which every terminal has two edge-disjoint paths to r since in every 2-edge-connected graph each terminal has two edge-disjoint paths to the root r and every graph in which each terminal has two edge-disjoint paths to the root r is 2-edge-connected. Similar to [94] first we preprocess the graph by deleting the vertices that cannot be part of any optimum solution. Firstly, for every vertex v we find two edge-disjoint paths between v and r of minimum total cost, let us denote it by $d_2(v, r)$. For finding $d_2(v, r)$ we can look for a minimum cost flow with 2 units of flow between v and r [118]. Suppose we have guessed a value OPT' such that $\text{OPT} \leq \text{OPT}' \leq 2\text{OPT}$, where OPT is the value of optimum solution (Finding OPT' is similar to the binary search technique described in Chapter 4, see Figure 5.1 for more details). Clearly every vertex v with $d_2(v, r) > \text{OPT}'$ cannot be part of any optimum solution and can be safely deleted. We work with this pruned version of graph G. Our algorithm is guided by the solution of an LP relaxation of the problem. Consider the following LP relaxation which is similar to what is proposed by Lau *et al.*[94].

$$
\begin{array}{llrcll}
\textbf{LP-k2EC} & \min & \sum_e c(e) \cdot x_e & & & \\
& \text{s.t.} & x(\delta(U)) & \geq & 2y_v & U \subseteq V - \{r\}, v \in U \quad (1) \\
& & x(\delta(U)) - x_{e'} & \geq & y_v & U \subseteq V - \{r\}, v \in U, e' \in \delta(U) \quad (2) \\
& & \sum_{v \in T} y_v & \geq & k & \quad (3) \\
& & y_r & = & 1 & \quad (4) \\
& & y_t & \leq & 1 & \quad (5) \\
& & x_e, y_v & \geq & 0 & \forall e \in E, \quad v \in T
\end{array}
$$

There are two types of indicator variables, x_e for each $e \in E$ and y_v for each $v \in T$; for every subset $U \subseteq V$, $\delta(U)$ is the set of edges across the cut $(U, V - U)$. Constraints (1) and (2) guarantee 2-edge-connectivity to the root. Our algorithm solves this LP and then uses the solution to find an integral solution of cost at most $O(\log n)$ times the optimal value. In order to do that we merge ideas from [20] and [94].

As argued in [94] this LP is a relaxation of the $(k, 2)$-subgraph problem and we can find an optimum solution of this LP since there is a polynomial time separation oracle although there are exponentially many constraints

We run the following algorithm whose detailed steps are explained below.

Input: Graph $G = (V, E)$, terminal set $T \subseteq V$ with root r, and integer $k \geq 1$
Output: a 2-edge-connected subgraph containing at least k terminals including r

1. Guess a value of $\textsc{opt}'$ for optimum solution and run the following algorithm.
2. $U \leftarrow r$
3. Start from original graph G and remove all the vertices with $d_2(v, r) > \textsc{opt}'$
4. Solve LP-K2EC and let its solution be (x^*, y^*)
5. Obtain (x', y') from (x^*, y^*) according to Lemma 17
6. Let T_i be the set of terminals v with $y'_v = 2^{-i}$ plus the root, for $0 \leq i \leq \lceil 3\log(n) \rceil$
7. Find a 2-edge-connected subgraph H_i over $T_i \cup \{r\}$ with cost $O(2^i \cdot \textsc{opt}^*)$
8. From H_i, find a 2-edge-connected subgraph H'_i containing r and at least $\lceil |T_i|/2^i \rceil$ and at most $2\lceil |T_i|/2^i \rceil$ vertices of T_i of cost at most $O(\textsc{opt}^*)$ and add it to U; if failed for any i then double the guess for $\textsc{opt}'$ and start from Step 2.
9. Return U.

Figure 5.1: $(k, 2)$-Subgraph Algorithm (k2EC)

In the rest of this section we show that Algorithm K2EC finds a 2-edge-connected subgraph of cost $O(\log(n) \cdot \textsc{opt})$ for the $(k, 2)$-subgraph problem. First we provide the details of the steps of the algorithm. Assume we sort all the vertices v according to their $d_2(v, r)$ value and let L be the kth smallest value. It is easy to see that $L \leq \textsc{opt} \leq k.L$. So we can start with L as our guess for $\textsc{opt}'$; if the algorithm fails to return a feasible solution of cost at most $O(\textsc{opt}' \cdot \log n)$ then we double our guess $\textsc{opt}'$ and run the algorithm again. Note that in $O(\log k)$ many steps we will have a guessed value $\textsc{opt}'$ with $\textsc{opt} \leq \textsc{opt}' \leq 2\textsc{opt}$ and therefore all the vertices that are deleted surely cannot be part of an optimum solution. Let (x^*, y^*) be an optimum feasible solution to LP-k2EC with value $\textsc{opt}^*$. For Step 5 of K2EC we round y values of the LP following the schema in [20].

Lemma 17 *There is a feasible solution (x', y') to LP-K2EC of cost at most $4\textsc{opt}^*$ such that all non-zero entries of y' belong to $\{2^{-i} | 0 \leq i \leq \lceil 3\log(n) \rceil\}$.*

Proof. The proof is very similar to that of Lemma 12. We set $x'_e = min(4x^*_e, 1)$ for all $e \in E$ and for all $v \in T$, select i such that $2^{-i} \leq y_v < 2^{-i+1}$, then if $i > \lceil 3\log(n) \rceil$ set $y'_v = 0$; otherwise, $y'_v = min(1, 2^{-i+2})$. It is easy to see that cost of (x', y') is at most $4\textsc{opt}^*$;

what remains is to show that (x', y') is a feasible solution to LP-K2EC. It is easy to see that Equations (8),(9),(11), and (12) are true for (x', y') as LHS is scaled at least as much as the RHS. Equation (10) is the only one to verify. As in the proof of Lemma 12, let Y_0 be the set of vertices v such that $y^*_v > 0$ but $y'_v = 0$. Note that $\sum_{v \in Y_0} y^*_v \leq 1/n^2$. These vertices are the only ones whose y value has decreased. Let Y_1 be the set of vertices v with $y'_v = 1$. If $|Y_1| \geq k$, then constraint (10) holds. Otherwise, $|Y_1| \leq k-1$ which implies $\sum_{v \notin Y_1} y^*_v \geq 1$, and therefore $\sum_{v \notin Y_1 \cup Y_0} y^*_v \geq 1 - 1/n^2 \geq \sum_{v \in Y_0} y^*_v$. Note also for each vertex $v \notin Y_0 \cup Y_1$, we know that $y'_v \geq 2y^*_v$. Thus, the amount $\sum_{v \in Y_0} y^*_v$ is compensated for with $\sum_{v \notin Y_0 \cup Y_1} y'_v$; therefore constraint (10) continues to hold.

■

Let T_i be the set of terminals with $y'_t = 2^{-i}$ and $k_i = |T_i|$, for $0 \leq i \leq \lceil 3 \log n \rceil$. Note that $\sum_{i=0}^{\lceil 3 \log n \rceil} 2^{-i} \cdot k_i \geq k$. Consider an instance of classical generalized Steiner network problem over terminals in $T_i \cup \{r\}$ with connectivity requirement 2 from every node in T_i to root. In the following lemma we show that we can compute a 2-edge-connected subgraph H_i over $T_i \cup \{r\}$ of cost at most $O(2^i \cdot \text{OPT}^*)$. This describes how to perform Step 7. The proof of this lemma is similar to Lemma 5.2 in [94].

Lemma 18 *In Step 7, For each* $0 \leq i \leq \lceil 3 \log n \rceil$*, we can find a 2-edge-connected subgraph* H_i *of cost at most* $2^{i+3} \cdot \text{OPT}^*$ *containing terminals* $T_i \cup \{r\}$.

Proof. In order to bound the cost of 2-edge-connected subgraph over $T_i \cup \{r\}$ we use the following natural LP for the special case of the generalized Steiner network problem in which all the connectivity requirements are 2:

$$
\begin{array}{llrcll}
\textbf{LP-2EC} & \min & \sum_e c(e) \cdot x_e & & & \\
& \text{s.t.} & x(\delta(U)) & \geq & 2 & U \subseteq V - \{r\}, U \cap T_i \neq \emptyset \quad (11) \\
& & 1 \geq x_e & \geq & 0 & \forall e \in E
\end{array}
$$

Jain [73] proved that the integrality gap of this LP is at most 2. Here, we show that after scaling (x', y'), we can find a feasible solution of LP-2EC over terminals $T_i \cup \{r\}$ of value at most $2^{i+2} \cdot \text{OPT}^*$. Using Jain's algorithm, we can then obtain an integer solution, i.e. a 2-edge-connected subgraph over $T_i \cup \{r\}$ of cost at most $2^{i+3} \cdot \text{OPT}^*$, which completes the proof of lemma.

Consider (x', y') obtained by Lemma 17 and define $\hat{x}_e = \min(1, 2^i \cdot x'_e)$. We will show that $\hat{x}$ is a feasible solution for LP-2EC, which clearly has cost at most $2^{i+2} \cdot \text{OPT}^*$ since $2^i \cdot x'_e = 2^{i+2} \cdot x^*_e$, thus as the LP-2EC selects the minimum over all the feasible solution its value is not greater than $2^{i+2} \cdot \text{OPT}^*$.

To verify that $\hat{x}$ is feasible for LP-2EC, take any set $U \subseteq V - r$ with $U \cap T_i \neq \emptyset$ and the corresponding constraint (11) in LP-2EC: $x(\delta(U)) \geq 2$. This has the corresponding constraint (8) in LP-k2EC $x(\delta(U)) \geq 2y_v$ for each $v \in U - \{r\}$. Suppose we define $\hat{x}_e =$

$\min\{1, 2^i \cdot x'_e\}$ and $\hat{y}_v = \min\{1, 2^i \cdot y'_v\}$. Note that for each $v \in T_i$: $\hat{y}_v = 1$. If all the edges $e \in \delta(U)$ have values $x'_e \le y'_v$ then after scaling we will have $\hat{x}(\delta(U)) \ge 2$ because the left hand side of $x(\delta(U)) \ge 2y_v$ is grown at least as much as the RHS is scaled. If there is at least one edge $e' \in \delta(U)$ with $x'_{e'} > y'_v$ then because of constraints (9) in LP-k2EC and since (x', y') is feasible, we have $x'(\delta(U)) - x'_{e'} \ge y'_v$. Thus after the scaling we still have $\hat{x}(\delta(U)) - \hat{x}_{e'} \ge 1$ because again the LHS is grown at least as much as the RHS. Also $\hat{x}_{e'} = 1$ because $\hat{y}_v = 1$ and $x'_{e'} > y'_v$; so $\hat{x}(\delta(U)) \ge 2$. This shows constraints (11) in LP-2EC are satisfied and so there is a feasible solution to LP-2EC with terminal set $T_i \cup \{r\}$ with cost at most $2^i\textsc{opt}^*$. ∎

In the following we show how to find subgraph H'_i in Step 8, which is 2-edge-connected, has root r, and has cost $O(\textsc{opt}')$, assuming that $\textsc{opt}' \ge \textsc{opt}$. Note that union of all H_i's ($0 \le i \le \lceil 3 \log n \rceil$) will be 2-edge-connected (since r is common in H'_i's), has at least k terminals, and has cost $O(\textsc{opt}' \cdot \log n)$. This will complete the proof of approximation ratio of the algorithm.

To show how to find a subgraph H'_i we use the same trick as in Section 5.1 of [94] for pruning a large good density solution to a smaller one. A nowhere-zero 6-flow in a directed graph $D = (V, A)$, is a function $f : A \rightarrow \mathbb{Z}_6$ such that we have flow conservation at every node (*i.e.* $f(\delta^{in}(v)) = f(\delta^{out}(v))$) and no edge gets f value of zero. If there is an orientation of an undirected graph H in which a nowhere-zero 6-flow can be defined we say H has a nowhere-zero 6-flow. Seymour [119] proved that every 2-edge-connected graph has a nowhere-zero 6-flow which can also be found in polynomial time. We obtain a multigraph $D = (H_i, A)$ from H_i by placing $f(e)$ copies of e with the direction defined by the flow. From Lemma 18 and the fact that we have at most 6 copies of each edge, the cost of D can be at most $6 \times 2^{i+3} \cdot \textsc{opt}^*$.

Note that D does not have directed cycle of length 2, therefore has an Eulerian walk. Start from r and build an Eulerian walk and partition the walk into segments $P_1, P_2, \ldots, P_\ell$ each of which includes $\lceil |T_i|/2^i \rceil$ terminals of H_i except possibly P_ℓ which can have between $\lceil |T_i|/2^i \rceil$ and $2\lceil |T_i|/2^i \rceil$ terminals. Thus, $\ell \ge \max(1, 2^{i-1})$ and so there is an index $1 \le q \le \ell$ such that the cost of path P_q is at most $6 \times 2^{i+2} \cdot \textsc{opt}^*/2^{i-1} = 48\textsc{opt}^*$. Let u, w be the endpoints of P_q and let Q^1_u and Q^2_u be the two edge-disjoint paths of $d_2(u, r)$ (in G) and Q^1_w and Q^2_w be the two edge-disjoint paths of $d_2(w, r)$ (again in G) of minimum total cost. Because of the preprocess step, the sum of costs of Q^1_u, Q^2_u, Q^1_w, and Q^2_w is at most $2\textsc{opt}'$. Let F_q be the simple graph in G defined by the edges of P_q and let $H'_i = F_q \cup Q^1_u \cup Q^2_u \cup Q^1_w \cup Q^2_w$. It follows that H'_i has cost at most $48\textsc{opt}^* + 2\textsc{opt}' \le 50\textsc{opt}'$. It only remains to show that H'_i is 2-edge-connected. By way of contradiction, suppose there is an edge e' such that $H'_i - e'$ has two components C_1 and C_2. Because of Q^1_u, Q^2_u, Q^1_w, and Q^2_w the two endpoints u and w are in the same component let say C_1. Since P_q is a directed walk from u to w and there is no cycle of size 2, there must be another edge $e'' \neq e'$ between C_1 and C_2 which goes in opposite direction of e', thus e' is not a cut edge.

5.4 Future works

A good line of research is to extend the algorithm for the $(k, 2)$-subgraph to the higher connectivitys. We are not aware of any attempt for approximating (k, λ)-subgraph for the special cases of $\lambda \geq 3$, the main difficulty in extending our algorithm to the $(k, 3)$-subgraph problem is pruning step *i.e.* Step 8 of our algorithm may not be done for the 3-edge-connected graphs efficiently. The constant factor for the metric case of (k, λ)-subgraph problem is large [114], finding an approximation algorithm with a small constant factor ratio is an interesting question.

There is a substantial gap between the best approximation factor for the densest k-subgraph problem ($O(n^{\frac{1}{4}+\epsilon})$ in [22]) and its hardness (It does not admit PTAS under various complexity theory assumptions [49, 78]). Improving either the approximation ratio or the hardness of densest k-subgraph is a major progress. Although there is a poly-logarithmic reduction to the densest k-subgraph problem for the (k, λ)-subgraph problem, theoretically it does not prove any hardness ratio since there is no large hardness factor for the densest k-subgraph. There are also no known hardness factor for the special cases of (k, λ)-subgraph where the graph is metric or where $\lambda = 2$.

Bibliography

[1] E. Althaus, S. Funke, S. Har-Peled, J. Konemann, E.A. Ramos, and M. Skutella. Approximating k-hop minimum-spanning trees. *Operations Research Letters*, 33(2):115–120, 2005.

[2] Mattias Andersson, Joachim Gudmundsson, Christos Levcopoulos, and Giri Narasimhan. Balanced partition of minimum spanning trees. In Peter Sloot, Alfons Hoekstra, C. Tan, and Jack Dongarra, editors, *Computational Science ICCS*, volume 2331 of *Lecture Notes in Computer Science*, pages 26–35. Springer Berlin / Heidelberg, 2002.

[3] M. Andrews. Hardness of buy-at-bulk network design. *Proc. of IEEE FOCS*, pages 115–124, 2004.

[4] M. Andrews and L. Zhang. Approximation algorithms for access network design. *Algorithmica (Preliminary version in* Proc. of IEEE FOCS *1998)*, 32(2):197–215, 2002.

[5] S. Antonakopoulos, C. Chekuri, B. Shepherd, and L. Zhang. Buy-at-bulk network design with protection. *Mathematics of Operations Research*, 36(1):71–87, 2011.

[6] Aaron Archer, MohammadHossein Bateni, Mohammad Taghi Hajiaghayi, and Howard J. Karloff. Improved approximation algorithms for prize-collecting steiner tree and tsp. In *IEEE Symposium on Foundations of Computer Science*, pages 427–436, 2009.

[7] Esther M. Arkin, Refael Hassin, and Asaf Levin. Approximations for minimum and min-max vehicle routing problems. *Journal of Algorithms*, 59:1–18, 2006.

[8] S. Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *Journal of the ACM (JACM)*, 45(5):753–782, 1998.

[9] S. Arora and G. Karakostas. A $2+\epsilon$; approximation algorithm for the k-MST problem. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 754–759. Society for Industrial and Applied Mathematics, 2000.

[10] G. Ausiello, P. Crescenzi, V. Kann, G. Gambosi, and A.M. Spaccamela. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer Verlag, 1999.

[11] I. Averbakh and O. Berman. $(p-1)/(p+1)$-approximate algorithms for p-traveling salesmen problems on a tree with minmax objective. *Discrete Applied Mathematics*, 75(3):201–216, 1997.

[12] B. Awerbuch, Y. Azar, A. Blum, and S. Vempala. Improved approximation guarantees for minimum-weight k-trees and prize-collecting salesmen. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages 277–283. ACM, 1995.

[13] B. Awerbuch, Y. Azar, A. Blum, and S. Vempala. New approximation guarantees for minimum-weight k-trees and prize-collecting salesmen. *SIAM Journal on Computing*, 28(1):254–262, 1999.

[14] Baruch Awerbuch and Yossi Azar. Buy-at-bulk network design. In *IEEE Symposium on Foundations of Computer Science*, pages 542–547, 1997.

[15] N. Bansal, A. Blum, S. Chawla, and A. Meyerson. Approximation algorithms for deadline-tsp and vehicle routing with time-windows. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 166–174. ACM, 2004.

[16] J. Bar-Ilan, G. Kortsarz, and D. Peleg. Generalized submodular cover problems and applications. *Theoretical Computer Science*, 250:179–200, 2001.

[17] Judit Bar-Ilan, Guy Kortsarz, and David Peleg. Generalized submodular cover problems and applications. *Theoretical Computer Science*, 250(1-2):179 – 200, 2001.

[18] Y. Bartal. On approximating arbitrary metrics by tree metrics. *Proc. of ACM STOC*, pages 161–168, 1997.

[19] Ivan Basov and Alek Vainshtein. Approximation algorithms for multi-parameter graph optimization problems. *Discrete Applied Mathematics*, 119(1-2):129 – 138, 2002.

[20] M.H. Bateni and J. Chuzhoy. Approximation algorithms for the directed k-tour and k-stroll problems. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 25–38, 2010.

[21] K. Bharath-Kumar and J. Jaffe. Routing to multiple destinations in computer networks. *Communications, IEEE Transactions on*, 31(3):343–351, 1983.

[22] A. Bhaskara, M. Charikar, E. Chlamtac, U. Feige, and A. Vijayaraghavan. Detecting high log-densities: an $o(n^{1/4})$-approximation for densest k-subgraph. In *Proceedings of the 42nd ACM symposium on Theory of computing*, pages 201–210. ACM, 2010.

[23] A. Blum, S. Chawla, D.R. Karger, T. Lane, A. Meyerson, and M. Minkoff. Approximation algorithms for orienteering and discounted-reward tsp. *SIAM Journal on Computing*, 37(2):653, 2008.

[24] Avrim Blum, R. Ravi, and Santosh Vempala. A constant-factor approximation algorithm for the k-MST problem. *Journal of Computer and System Sciences*, 58:101–108, 1999.

[25] Avrim Blum, R. Ravi, and Santosh Vempala. A constant-factor approximation algorithm for the k-MST problem. *Journal of Computer and System Sciences*, 58:101–108, 1999.

[26] A. Bookstein and S.T. Klein. Construction of optimal graphs for bit-vector compression. In *Proceedings of the 13th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 327–342. ACM, 1989.

[27] Ann Melissa Campbell, Dieter Vandenbussche, and William Hermann. Routing for relief efforts. *Transportation Science*, 42:127–145, May 2008.

[28] M. Charikar and A. Karagiozova. On non-uniform multicommodity buy-at-bulk network design. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 176–182. ACM, 2005.

[29] M. Charikar, J. Naor, and B. Schieber. Resource optimization in qos multicast routing of real-time multimedia. *Networking, IEEE/ACM Transactions on*, 12(2):340–348, 2004.

[30] K. Chaudhuri, S. Rao, S. Riesenfeld, and K. Talwar. What would Edmonds do? augmenting paths and witnesses for degree-bounded MSTs. *Approximation, Randomization and Combinatorial Optimization*, pages 26–39, 2005.

[31] Kamalika Chaudhuri, Brighten Godfrey, Satish Rao, and Kunal Talwar. Paths, trees, and minimum latency tours. In *IEEE Symposium on Foundations of Computer Science*, pages 36–45, 2003.

[32] C. Chekuri, M. Hajiaghayi, G. Kortsarz, and M. Salavatipour. Approximation algorithms for non-uniform buy-at-bulk network design problems. *Proc. of IEEE FOCS*, pages 677–686, 2006.

[33] C. Chekuri, M. Hajiaghayi, G. Kortsarz, and M. Salavatipour. Approximation algorithms for non-uniform buy-at-bulk network design. *SIAM J. on Computing*, 39(5):1772–1798, 2009.

[34] C. Chekuri, S. Khanna, and J. Naor. A deterministic approximation algorithm for the cost-distance problem. *Short paper in* Proc. of ACM-SIAM SODA., pages 232–233, 2001.

[35] C. Chekuri, N. Korula, and M. Pál. Improved algorithms for orienteering and related problems. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 661–670. Society for Industrial and Applied Mathematics, 2008.

[36] C. Chekuri and M. Pal. A recursive greedy algorithm for walks in directed graphs. In *Foundations of Computer Science, 2005. FOCS 2005. 46th Annual IEEE Symposium on*, pages 245–253. IEEE, 2005.

[37] Chandra Chekuri and Nitish Korula. Pruning 2-connected graphs. In *Foundations of Software Technology and Theoretical Computer Science*, pages 119–130, 2008.

[38] Chandra Chekuri and Amit Kumar. Maximum coverage problem with group budget constraints and applications. In *Approximation Algorithms for Combinatorial Optimization*, pages 72–83, 2004.

[39] C.H. Chow. On multicast path finding algorithms. In *Proceedings. Tenth Annual Joint Conference of the IEEE Computer and Communications Societies. Networking in the 90s., IEEE INFOCOM'91.*, pages 1274–1283. IEEE, 1991.

[40] N. Christofides. Worst case analysis of a new heuristic for the Traveling Salesman Problem. report 388, graduate school of industrial administration, 1976.

[41] J. Chuzhoy, A. Gupta, J.S. Naor, and A. Sinha. On the approximability of some network design problems. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 943–951. Society for Industrial and Applied Mathematics, 2005.

[42] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of operations research*, pages 233–235, 1979.

[43] M. Doar and I. Leslie. How bad is naive multicast routing? In *INFOCOM'93. Proceedings. Twelfth Annual Joint Conference of the IEEE Computer and Communications Societies. Net working: Foundation for the Future. IEEE*, pages 82–89. IEEE, 1993.

[44] Friedrich Eisenbrand and Fabrizio Grandoni. An improved approximation algorithm for virtual private network design. In *Symposium on Discrete Algorithms*, pages 928–932, 2005.

[45] Friedrich Eisenbrand, Fabrizio Grandoni, Thomas Rothvo, and Guido Schfer. Connected facility location via random facility sampling and core detouring. *Journal of Computer and System Sciences*, 76:709–726, 2010.

[46] G. Even, N. Garg, J. Konemann, R. Ravi, and A. Sinha. Covering graphs using trees and stars. *Operations Research Letters (Earlier version in Proceedings of APPROX 2003)*, 32(4):309–315, 2004.

[47] J. Fakcharoenphol, S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *Journal of Computer and System Sciences*, 69(3):485–497, 2004.

[48] U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.

[49] U. Feige. Relations between average case complexity and approximation complexity. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 534–543. ACM, 2002.

[50] A.J. Frank, L.D. Wittie, and A.J. Bernstein. Multicast communication on network computers. *IEEE software*, pages 49–61, 1985.

[51] Greg Frederickson and Barry Wittman. Approximation algorithms for the Traveling Repairman and Speeding Deliveryman Problems with Unit-Time Windows. In Moses Charikar, Klaus Jansen, Omer Reingold, and Jos Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, volume 4627 of *Lecture Notes in Computer Science*, pages 119–133. Springer Berlin / Heidelberg, 2007.

[52] Greg N. Frederickson, Matthew S. Hecht, and Chul E. Kim. Approximation algorithms for some routing problems. *Siam Journal on Computing*, 7:178–193, 1978.

[53] M. Fürer and B. Raghavachari. Approximating the minimum degree spanning tree to within one from the optimal degree. In *Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms*, pages 317–324. Society for Industrial and Applied Mathematics, 1992.

[54] N. Garg. A 3-approximation for the minimum tree spanning k vertices. In *Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on*, pages 302–309. IEEE, 1996.

[55] N. Garg. Saving an epsilon: a 2-approximation for the k-MST problem in graphs. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing (STOC)*, pages 396–402, 2005.

[56] Michel X. Goemans. Minimum bounded degree spanning trees. In *Foundations of Computer Science, 2006. FOCS '06. 47th Annual IEEE Symposium on*, pages 273 –282, oct. 2006.

[57] Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. In *Symposium on Discrete Algorithms*, pages 307–316, 1995.

[58] F. Grandoni and G. Italiano. Improved approximation for single-sink buy-at-bulk. *Algorithms and Computation*, pages 111–120, 2006.

[59] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

[60] S. Guha, A. Meyerson, and K. Munagala. A constant factor approximation for the single sink edge installation problems. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 383–388. ACM, 2001.

[61] A. Gupta, R. Krishnaswamy, and R. Ravi. Tree embeddings for two-edge-connected network design. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1521–1538. Society for Industrial and Applied Mathematics, 2010.

[62] A. Gupta, A. Kumar, et al. Approximation via cost sharing: Simpler and better approximation algorithms for network design. *Journal of the ACM (JACM)*, 54(3):11–es, 2007.

[63] A. Gupta, A. Kumar, M. Pal, and T. Roughgarden. Approximation via cost-sharing: a simple approximation algorithm for the multicommodity rent-or-buy problem. In *Proceedings of 44th Annual IEEE Symposium on Foundations of Computer Science*, pages 606–615. IEEE, 2003.

[64] A. Gupta, A. Kumar, and T. Roughgarden. Simpler and better approximation algorithms for network design. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 365–372. ACM, 2003.

[65] Nili Guttmann-Beck and Refael Hassin. Approximation algorithms for min-max tree partition. *J. Algorithms*, 24:266–286, August 1997.

[66] M.T. Hajiaghayi, G. Kortsarz, and M.R. Salavatipour. Approximating buy-at-bulk and shallow-light k-steiner tree. *Algorithmica*, 53(1):89–103, 2009.

[67] R. Hassin. Approximation schemes for the restricted shortest path problem. *Mathematics of Operations Research*, 17(1):36–42, 1992.

[68] R. Hassin and A. Levin. Minimum restricted diameter spanning trees. *Approximation Algorithms for Combinatorial Optimization*, pages 175–184, 2002.

[69] S.P. Hong, H. Lee, and B.H. Park. An efficient multicast routing algorithm for delay-sensitive applications with dynamic membership. In *INFOCOM'98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1433–1440. IEEE, 1998.

[70] F.K. Hwang and D.S. Richards. Steiner tree problems. *Networks*, 22(1):55–89, 1992.

[71] P. Jaillet and M.R. Wagner. Online vehicle routing problems: A survey. *The Vehicle Routing Problem: Latest Advances and New Challenges*, pages 221–237, 2008.

[72] K. Jain. A factor 2-approximation algorithm for the generalized steiner network problem. In *Proceedings. 39th Annual Symposium on Foundations of Computer Science*, pages 448–457. IEEE, 1998.

[73] K. Jain. A factor 2-approximation algorithm for the generalized steiner network problem. *Combinatorica*, 21(1):39–60, 2001.

[74] D.S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9(3):256–278, 1974.

[75] Raja Jothi and Balaji Raghavachari. Approximating the k-traveling repairman problem with repairtimes. *Journal of Discrete Algorithms*, 5:293–303, 2007.

[76] M.R. Khani and M.R. Salavatipour. Approximation algorithms for min-max tree cover and bounded tree cover problems. To appear in Approx 2011.

[77] M.R. Khani and M.R. Salavatipour. Improved approximations for buy-at-bulk and shallow-light k-steiner trees and $(k, 2)$-subgraph, 2011. Manuscript.

[78] S. Khot. Ruling out ptas for graph min-bisection, densest subgraph and bipartite clique. In *Foundations of Computer Science, 2004. Proceedings. 45th Annual IEEE Symposium on*, pages 136–145. IEEE, 2004.

[79] S. Khuller, B. Raghavachari, and N. Young. Balancing minimum spanning trees and shortest-path trees. *Algorithmica*, 14(4):305–321, 1995.

[80] P. Klein and R. Ravi. A nearly best-possible approximation algorithm for node-weighted steiner trees. *J. Algorithms*, 19(1):104–115, 1995.

[81] V.P. Kompella, J.C. Pasquale, and G.C. Polyzos. Multicast routing for multimedia communication. *IEEE/ACM Transactions on Networking (TON)*, 1(3):286–292, 1993.

[82] J. Könemann. *Approximation Algorithms for Minimum-Cost Low-Degree Subgraphs.* PhD thesis, Carnegie Mellon University, 2003.

[83] J. Könemann, A. Levin, and A. Sinha. Approximating the degree-bounded minimum diameter spanning tree problem. *Algorithmica*, 41(2):117–129, 2005.

[84] J. Könemann and R. Ravi. A matter of degree: improved approximation algorithms for degree-bounded minimum spanning trees. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 537–546. ACM, 2000.

[85] J. Konemann and R. Ravi. Primal-dual meets local search: approximating MST's with nonuniform degree bounds. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 389–395. ACM, 2003.

[86] G. Kortsarz and Z. Nutov. Approximating some network design problems with vertex costs. *Proc. APPROX-RANDOM*, pages 231–243, 2009.

[87] G. Kortsarz and D. Peleg. Approximation algorithms for minimum time broadcast. *Theory of Computing and Systems*, pages 67–78, 1992.

[88] G. Kortsarz and D. Peleg. Approximating the weight of shallow steiner trees. *Discrete Applied Mathematics*, 93(2-3):265–285, 1999.

[89] Guy Kortsarz and Zeev Nutov. Approximating minimum cost connectivity problems. In Erik D. Demaine, MohammadTaghi Hajiaghayi, and Dániel Marx, editors, *Parameterized complexity and approximation algorithms*, number 09511 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2010. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

[90] L. Kou, G. Markowsky, and L. Berman. A fast algorithm for steiner trees. *Acta informatica*, 15(2):141–145, 1981.

[91] P. Krysta and R. Solis-Oba. Approximation algorithms for bounded facility location problems. *Journal of combinatorial optimization*, 5(2):233–247, 2001.

[92] Piotr Krysta. Bicriteria network design via iterative rounding. In Lusheng Wang, editor, *Computing and Combinatorics*, volume 3595 of *Lecture Notes in Computer Science*, pages 179–187. Springer Berlin / Heidelberg, 2005.

[93] A. Kumar, A. Gupta, and T. Roughgarden. A constant-factor approximation algorithm for the multicommodity rent-or-buy problem. In *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*, pages 333–342. IEEE, 2002.

[94] L. Lau, S. Naor, M.R. Salavatipour, and M. Singh. Survivable network design with degree or order constraints. *SIAM J. on Computing (Special issue for selected papers of STOC 2007)*, 39(3):1062–1087, 2009.

[95] Chung-Lun Li, David Simchi-Levi, and Martin Desrochers. On the distance constrained vehicle routing problem. *Oper. Res.*, 40:790–799, July 1992.

[96] L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete mathematics*, 13(4):383–390, 1975.

[97] M.V. Marathe, R. Ravi, R. Sundaram, SS Ravi, D.J. Rosenkrantz, and H.B. Hunt III. Bicriteria network design problems. *Journal of Algorithms*, 28(1):142–171, 1998.

[98] A. Meyerson. Online algorithms for network design. In *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 275–280. ACM, 2004.

[99] A. Meyerson, K. Munagala, and S. Plotkin. Cost-distance: Two metric network design. *SIAM J. on Computing (Preliminary version in Proc. of IEEE FOCS 2000)*, pages 2648–1659, 2008.

[100] J.S.B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric tsp, k-MST, and related problems. *SIAM J. Comput.*, 28(4):1298–1309, 1999.

[101] T. Momke and O. Svensson. Approximationg graphic TSP by matchings. To appear in FOCS 2011.

[102] Marcin Mucha and Piotr Sankowski. Maximum matchings via gaussian elimination. In *IEEE Symposium on Foundations of Computer Science*, pages 248–255, 2004.

[103] Hiroshi Nagamochi. Approximating the minmax rooted-subtree cover problem. *IEICE Transactions on Fundamentals of Electronics*, E88-A:1335–1338, 2005.

[104] V. Nagarajan and R. Ravi. Poly-logarithmic approximation algorithms for directed vehicle routing problems. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 257–270, 2007.

[105] V. Nagarajan and R. Ravi. Approximation algorithms for distance constrained vehicle routing problems. *Networks*, 2008.

[106] M. Parsa, Q. Zhu, and JJ Garcia-Luna-Aceves. An iterative algorithm for delay-constrained minimum-cost multicasting. *IEEE/ACM Transactions on Networking (TON)*, 6(4):461–474, 1998.

[107] S. Rajagopalan and V. V. Vazirani. Logarithmic approximation of minimum weight k trees, 1995. Unpoblished manuscript.

[108] R. Ravi. Rapid rumor ramification: approximating the minimum broadcast time. In *Proceedings of 35th Annual Symposium on Foundations of Computer Science*, pages 202–213. IEEE, 1994.

[109] R. Ravi and M. Goemans. The constrained minimum spanning tree problem. *Algorithm Theory (Preliminary version in* Proc. of SWAT *1996)*, pages 66–75, 1996.

[110] R. Ravi, M.V. Marathe, SS Ravi, D.J. Rosenkrantz, and H.B. Hunt III. Many birds with one stone: Multi-objective approximation algorithms. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 438–447. ACM, 1993.

[111] R. Ravi, M.V. Marathe, SS Ravi, D.J. Rosenkrantz, and H.B. Hunt III. Approximation algorithms for degree-constrained minimum-cost network-design problems. *Algorithmica*, 31(1):58–78, 2001.

[112] R. Ravi and M. Singh. Delegate and conquer: An lp-based approximation algorithm for minimum degree MSTs. *Automata, Languages and Programming*, pages 169–180, 2006.

[113] R. Ravi, R. Sundaram, M.V. Marathe, D.J. Rosenkrants, and S.S. Ravi. Spanning trees short or small. *SIAM Journal on Discrete Mathematics*, 9(2):178–200, 1996.

[114] M.A. Safari and M. Salavatipour. A constant factor approximation for minimum λ-edge-connected k-subgraph with metric costs. *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pages 233–246, 2008.

[115] S. Sahni and T. Gonzalez. P-complete approximation problems. *Journal of the ACM (JACM)*, 23(3):555–565, 1976.

[116] F. S. Salman, J. Cheriyan, R. Ravi, and S. Subramanian. Approximating the single-sink link-installation problem in network design. *SIAM J. on Optimization (Preliminary version in* Proc. of ACM-SIAM SODA'97*)*, 11(3):595–610, 2000.

[117] F.S. Salman, J. Cheriyan, R. Ravi, and S. Subramanian. Buy-at-bulk network design: Approximating the single-sink edge installation problem. In *Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, pages 619–628. Society for Industrial and Applied Mathematics, 1997.

[118] A. Schrijver. *Combinatorial optimization: Polyhedra and Efficiency*. Springer-Verlag, Berlin, 2003.

[119] P.D. Seymour. Graph theory and related topics. *Graph Theory and Related Topics*, 1979.

[120] Mohit Singh and Lap Chi Lau. Approximating minimum bounded degree spanning trees to within one of optimal. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, STOC '07, pages 661–670, New York, NY, USA, 2007. ACM.

[121] C. Swamy and A. Kumar. Primal-dual algorithms for connected facility location problems. *Approximation Algorithms for Combinatorial Optimization*, pages 256–270, 2002.

[122] V.V. Vazirani. *Approximation algorithms*. Springer Verlag, 2001.

[123] A. Warburton. Approximation of Pareto optima in multiple-objective, shortest-path problems. *Operations Research*, 35(1):70–79, 1987.

[124] L. Wei and D. Estrin. The trade-offs of multicast trees and algorithms. In *Proceedings of the International Conference on Computer Communications and Networks (ICCCN)*, 1994.

[125] Zhou Xu and Qi Wen. Approximation hardness of min-max tree covers. *Operations Research Letters*, 38:169–173, 2010.

[126] Zhou Xu and Liang Xu. Approximation algorithms for min-max path cover problems with service handling time. In *Proceedings of the 20th International Symposium on Algorithms and Computation*, ISAAC '09, pages 383–392, Berlin, Heidelberg, 2009. Springer-Verlag.

[127] W. Zhao and P. Zhang. Approximation to the minimum rooted star cover problem. *Theory and Applications of Models of Computation*, pages 670–679, 2007.

[128] Q. Zhu, M. Parsa, and W.W.M. Dai. *An iterative approach for delay-bounded minimum Steiner tree construction*. Computer Research Laboratory,[University of California, Santa Cruz, 1994.