APPROXIMATION ALGORITHMS FOR SINGLE-SINK EDGE INSTALLATION PROBLEMS AND OTHER GRAPH PROBLEMS

by

RAJA JOTHI, B.E., M.S.

DISSERTATION

Presented to the Faculty of

The University of Texas at Dallas

in Partial Fulfillment

of the Requirements

for the Degree of

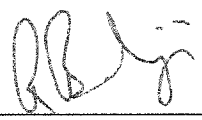
DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT DALLAS

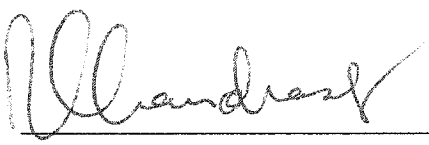
August 2004

APPROXIMATION ALGORITHMS FOR SINGLE-SINK EDGE INSTALLATION PROBLEMS AND OTHER GRAPH PROBLEMS

APPROVED BY SUPERVISORY COMMITTEE:

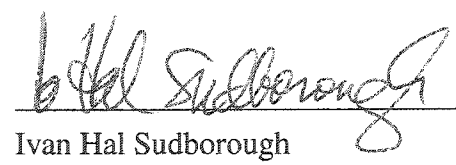


Balaji Raghavachari, Chair

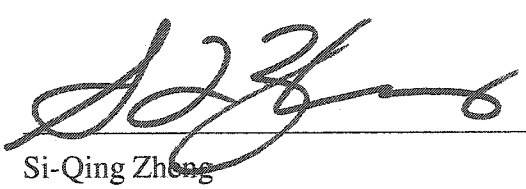


Ramaswamy Chandrasekaran

Ovidiu Daescu



Ivan Hal Sudborough



Si-Qing Zheng

To my parents M. Vijayalakshmi and G. Jothi,

my aunt G. Padmavathi,

and my grandmother M. Valliammal

ACKNOWLEDGMENTS

I am very thankful to Balaji Raghavachari for taking me under his wings when I was not sure of my interests and capabilities. His knowledge, guidance, and encouragement were instrumental in making me think the way I do today. I would like to thank R. Chandrasekaran for letting me knock on his door whenever I had questions in combinatorics, and Ovidiu Daescu for useful discussions on Computational Geometry and other topics. I learned a lot on how to write proofs from Hal Sudborough's Algorithms and Theory of Computation courses. Thanks to him and Si-Qing Zheng for serving on my thesis committee. I would also like to thank Kamil Sarac and Jason Jue for helping me get exposed to new areas of research.

I owe a great deal to many teachers who were responsible for the growth in my knowledge and maturity. Thanks to my roommate Pritesh for showing me what it takes be a researcher. Thanks also to my friends and colleagues at the University of Texas at Dallas for useful insights and interesting conversations.

My deepest gratitude is to my parents for their immense love, support, patience, and sacrifices. I would like to thank my sisters, Sujatha and Nithya, for putting up with me. Thanks also to my aunts and grandmother for their love, teachings, and sacrifices, and my cousins Chandrasekaran and Elangovan for their support during the times of need.

Finally, I would like to thank Sonia—my best discovery at UTDallas—for being there during the most important stretch of my life, and refining me into a person that I am today. She inspired me to reach levels that I would have never dreamt of reaching before.

I gratefully acknowledge funding support from NSF, UTDallas, IBM, and DIMACS.

APPROXIMATION ALGORITHMS FOR SINGLE-SINK EDGE INSTALLATION PROBLEMS AND OTHER GRAPH PROBLEMS

Publication No. _____

Raja Jothi, Ph.D.
The University of Texas at Dallas, 2004

Supervising Professor: Dr. Balaji Raghavachari

Many network design problems require designing networks that obey certain constraints. Minimum spanning tree, Steiner tree, vehicle routing, and facility location problems are examples. The goal is to design networks of minimal cost. In this thesis, we present approximation algorithms for minimum-cost *single-sink edge installation* problems and related graph problems.

Given an undirected graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, a cost function on the edges, root $r \in V$, capacity constraint k, and a set of demands $D \subseteq V$, with $w(v)$ denoting the flow that $v \in D$ wishes to route to r, the *Capacitated Minimum Steiner Tree* (CMStT) problem asks for a minimum cost Steiner tree, rooted at r, spanning D such that the sum of the vertex weights in each of the subtrees connected to r is at most k. When $D = V$, this problem is called the *Capacitated Minimum Spanning Tree* (CMST) problem. Informally, the problem is that of installing cables of capacity k along the edges of the given graph in order to facilitate simultaneous routing of flows from all the demands to the sink. In this thesis, we first present approximation algorithms for the CMST and the CMStT problems. Next, we present approximation results for the *Capacitated Minimum Spanning*

Network problem, a *survivable network* variant of the CMST problem, which requires that the flow can be routed to the sink even in case of single node/edge failure.

We then consider the *Single-Sink Buy-at-Bulk* problem, a generalization of the CMStT problem, in which we are given the option of using l different cables, instead of just one. Each cable has a given capacity and cost per unit length. It is not required that the final network be a tree. We present approximation algorithms for this problem when there is an additional restriction that the flow from a source cannot be bifurcated, and its variant in which the flow is splittable.

Finally, we present approximation results for other graph problems: the *k-Traveling Repairmen* problem, the *Bounded Latency* problem, and the *Bounded-Degree Minimum Spanning Tree* problem.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

1.1 Network design

A network is an interconnected system of objects. A network object may represent anything depending on the application, e.g., computers in a computer network, telephones in a telecommunication network, cities in a road network, oil-wells and refineries in a oil pipeline network. Designing or building a network involves interconnecting objects of interest under certain conditions. A network design is a blueprint for building a network [17].

Many network design problems require designing networks that obey certain constraints. Minimum spanning tree, Steiner tree, vehicle routing, and facility location problems are examples. Usually, the goal is to design networks of minimal cost. Many network design problems are modeled as optimization problems. In an optimization problem, the objective is to minimize or maximize a given objective function under certain constraints. It is these constraints that control the objective value that is being optimized. Unconstrained problems are easier to solve. For example, a loosely defined problem of connecting a given set of computers could potentially have numerous solutions if one were to ignore the cost of connecting them.

Networks are generally modeled as graphs. Graph theory is a branch of mathematics, which is considered by many as the language of networks. Graph theory is to network design is like calculus is to astrophysics. In the following section, we present the necessary fundamentals of graph theory related to our work.

1.2 Graph theory fundamentals

A graph G consists of a set of vertices V and a set of edges E. The vertices are often referred to as nodes or points, and the edges referred to as links or lines. A graph is denoted as a pair (V, E). Edges in a graph may be *directed* or *undirected*. A graph with only undirected edges is called an undirected graph, and a graph with only directed edges is called an directed graph. A graph that contains both directed as well as undirected edges are called *mixed* graphs. In this thesis, we will be dealing only with undirected graphs.

An edge $e \in E$ in a graph is usually associated with two vertices or end-points. It is possible that an edge is associated with just one vertex, i.e., both of its end-points are the same. Such edges are called *loops*. Most often, it is assumed that there is at most one edge between two given vertices. In some cases, there may be more than one edge between a given pair of vertices. Such edges are called *parallel* edges. A graph with no parallel edges and loops is called a *simple* graph. The degree $\delta(v)$ of a vertex v is the number of edges incident on v. Two nodes v_1 and v_2 in a graph are *adjacent* if there is an edge connecting v_1 and v_2. A graph $G' = (V', E')$ is a *subgraph* of graph $G = (V, E)$, if $V' \subseteq V$ and $E' \subseteq E$. Only edges with both end-points in V' are included in E'. A *connected* graph is a graph in which there exists at least one path between any two vertices. We call a graph to be c-connected if there exists at least c disjoint paths between any two vertices. In particular, we refer to a graph as c-vertex connected if there exists at least c vertex-disjoint paths between any two vertices, and c-edge connected if there exists at least c edge-disjoint paths between any two vertices.

A *simple path* P between two vertices v_1 and v_2 is a set of edges $\{e_1, e_2, \ldots, e_n\}$ such that e_i and e_{i+1} have a common end-point, and e_1 and e_n have end-points v_1 and v_2, respectively. Henceforth, we will be referring to simple paths as just paths. The length of the simple path, also known as the number of *hops*, between v_1 and v_2 is the number of edges in P. A *cycle* is a path from a vertex v_1 to itself, with at least one other intermediate vertex.

A graph is called *edge-weighted*, if a real number is associated with each of its edges. Depending on the application, these real numbers may represent the length or cost of an

edge. We use $|uv|$ to denote the cost/length of an edge connecting vertices u and v. In this thesis, we will assume that that the cost/length of an edge is non-negative. A graph is called *vertex-weighted*, if a real number is associated with each node in the graph.

The edges of a given graph is said to obey *triangle inequality*, if for any two given vertices u and v in the graph, $|uv|$ is less than or equal to the cost/length of any path between u and v. In this thesis, we will be considering only graphs whose edges obey *triangle inequality*. By "metric completion" of a given graph (whose edges obey triangle inequality), we refer to a complete graph. Throughout this paper, without loss of generality, we assume that the metric completion of the input graph is available. We also use the term "metric space" to refer the metric closure of the given graph.

1.2.1 Spanning Trees and Tours

Given a graph $G = (V, E)$, a spanning tree $T = (V, E')$ is a connected, simple, subgraph of G that has no cycles. A tree spanning n nodes has $n - 1$ edges. For a set of n nodes, there could be up to n^{n-2} spanning trees. A *minimum spanning tree* (MST) for a given graph is a spanning tree with minimum total edge weight. Note that the definition of an MST makes sense only for edge-weighted graphs. There are several algorithms for finding an MST of a given graph. Two popular algorithms are the Kruskal's algorithm [72] and the Prim's algorithm [83]. For further details on these algorithms, we refer the reader to [24].

For a given graph $G = (V, E)$, a tour is a closed walk (may go through the same edge more than once). In other words, it is a cycle spanning all the vertices. For an edge-weighted graph $G = (V, E)$, the *traveling salesman problem* (TSP) is to find a tour spanning V such that the cost/length of the tour is minimum.

1.3 Approximation algorithms

A polynomial-time algorithm for a given problem is an algorithm whose worst-case running time is $O(n^c)$, where n is the input size and c is a constant. Problems for which one can

write polynomial-time algorithms are called polynomial-time solvable. Sorting, selection, and counting are examples of problems that are polynomial-time solvable. The MST problem defined in the previous section is also polynomial-time solvable.

There are several naturally occurring problems in combinatorial optimization, such as the TSP for which there is no known polynomial-time algorithm. However, there are exponential time algorithms for these "difficult" or "hard" problems. Because of their super-polynomial running time, they can only solve small instances. In other words, the problem sizes that these algorithms can solve to optimality, in "reasonable" amount of time, are far from the sizes of real-life instances. Since such problems occur in real-life, there is a need for these problems to be solved in a *fast* and *efficient* way. A well-accepted approach has been to settle for *near-optimal* solutions that could be computed in polynomial time. A near-optimal solution is a feasible solution whose objective value is within some small *multiplicative* (or additive) factor of the optimal value.

An *α-approximation algorithm* for a combinatorial optimization problem is a polynomial time algorithm which, for any given instance of the problem, finds a feasible solution whose value is within a multiplicative factor α times than that of an optimal solution for that instance. We refer α to be the *approximation ratio* of the algorithm. The term approximation ratio is also referred to as the performance guarantee or the performance ratio. The value of α is greater than or equal to 1 for minimization problems, and less than or equal to 1 for maximization problems. The closer the value of α is to 1, the better is the approximation ratio. The value of α may be a function of n, where n is the size of the input.

1.4 Organization and overview of contributions

In the rest of this chapter, we present a brief overview of the problems considered in this thesis, and the results we have obtained for these problems. Problems considered in Chapters 2 to 6 fall under the category of *single-sink edge installation* problems. The common objective in these problems is to buy just enough cables and install them along the edges of given graph

so that the flow from all the source nodes in the graph can simultaneously be routed to the designated sink node. Our goal is to build a least cost network, but at the same time satisfying certain capacity and connectivity constraints. In Chapter 7, we consider the *traveling repairmen* problem, a variant of the TSP. Finally, in Chapter 8, we study the *Bounded-Degree Minimum Spanning Tree* problem, which asks for a minimum edge-weight spanning tree in which the degree of each vertex is at most a given degree bound δ. The results presented in this thesis appear in [53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63].

1.4.1 Capacitated Minimum Spanning Trees

In Chapter 2, we consider one of the extensively studied problems in telecommunications network design, the *Capacitated Minimum Spanning Tree* (CMST) problem. Given an undirected, vertex-weighted graph $G = (V, E)$ with non-negative costs on its edges, a root node $r \in V$, and a positive number k, the CMST problem asks for a minimum-cost spanning tree rooted at r, in which the sum of the vertex weights in every subtree connected to r is at most k. The CMST problem is NP-hard [35, 80]. The capacity constraint k must be at least as much as the largest vertex weight for the CMST problem to be feasible.

In telecommunications network design, the CMST problem is that of designing a low cost tree network, built by installing expensive (such as fiber-optic) cables along its edges. The cables have pre-specified capacities on the amount of demand they can handle, and can be bought at a certain cost per unit length. Every source node in the network has some demand that it has to transmit to the sink node. The objective is to construct a minimum-cost tree network for simultaneous routing of demands from the source nodes to the sink node.

The CMST problem has been well-studied in Computer Science and Operations Research for the past 40 years. Numerous heuristics and exact algorithms have been proposed (see Section 2.1 for more details). In this chapter, we restrict our discussion to approximation algorithms for the CMST problem. Gavish and Altinkemer [42] were the first ones to present an algorithm for the CMST problem with any approximation ratio. They subse-

quently presented an improved approximation algorithm [3] with ratio 4. We present a new 4-approximation algorithm. Our algorithm is simpler, and easier to analyze when compared to the algorithm in [3]. We also show how to obtain a 2-approximation ratio for graphs with unit vertex-weights and $k = 3$ or 4.

1.4.2 Capacitated Minimum Steiner Trees

The *Capacitated Minimum Steiner Tree* (CMStT) problem is a generalization of the CMST problem. The main difference is that not all vertices in the given graph are associated with weights, i.e., only a subset of vertices have positive weights, while the others have zero weights. Unlike the CMST problem, in which the final tree must span all the vertices, the final tree in the CMStT problem is a Steiner tree that needs to span only the vertices with positive weights. The Steiner tree may use zero weight nodes as internal nodes, if needed. Formally, the CMStT problem can be defined as follows. Given an undirected graph $G = (V, E)$ with non-negative costs on its edges, a root node $r \in V$, a set of demands $D \subseteq V$ with demand $v \in D$ wishing to route $w(v)$ units of flow (weight) to r, and a positive number k, the CMStT problem asks for a minimum-cost Steiner tree, rooted at r and spanning the vertices in $D \cup \{r\}$, in which the sum of the vertex weights in every subtree connected to r is at most k. When $D = V$, this problem is the *Capacitated Minimum Spanning Tree* (CMST) problem. The CMStT problem is therefore NP-hard.

The CMStT problem has not been studied before. But the CMST algorithms in [3] can be used to obtain feasible solutions with approximation ratios $2\rho_{ST} + 1$ and $2\rho_{ST} + 2$ for the uniform and non-uniform vertex-weighted graphs, respectively, where ρ_{ST} is the best achievable approximation ratio for the minimum Steiner tree problem. In Chapter 3, we present approximation algorithms for CMStT problem and several of their variants in network design. Our main results are the following.

- We give a $(\gamma\rho_{ST} + 2)$-approximation algorithm for the CMStT problem, where γ is the

inverse Steiner ratio[1].

- In particular, we obtain $(\gamma + 2)$-approximation ratio for the CMST problem, which is an improvement over the current best ratio of 4. For points in Euclidean and rectilinear planes, our result translates into ratios of 3.1548 and 3.5, respectively.

- For instances in the plane, under the L_p norm, where all vertices in D have uniform weights, we give a non-trivial $(\frac{7}{5}\rho_{ST} + \frac{3}{2})$-approximation algorithm for the CMStT problem. This translates into a ratio of 2.9 for the CMST problem with uniform vertex weights in the L_p metric plane. Our ratio of 2.9 solves the long-standing open problem of obtaining a ratio any better than three for this case.

1.4.3 Revisiting Esau-Williams' Algorithm: On the Design of Tree Networks

In Chapter 4, we present a fast and efficient heuristic for the CMST problem, based on some CMST properties and insights from our approximation algorithms for the CMStT problem in Chapter 3. Over the last 40 years, numerous heuristics, exact algorithms, and mathematical formulations have been proposed for the CMST problem. There are limitations with exponential worst-case running time exact algorithms and mathematical formulations as the size of the instances that can be solved to optimality, in *reasonable* amount of time, is far from the size of real-life instances. The most popular and efficient heuristic for the CMST problem is due to Esau and Williams (EW), presented in 1966, with a worst-case running time of $O(n^2 \log n)$. There are several other heuristics that are based on local search methods such as *Tabu search*, *simulated annealing*, and *genetic algorithms*. These heuristics start with an initial feasible solution, and improve the solution using local modifications of the tree during every iteration. The problem with these techniques is that the amount of improvement in each iteration could be so small that the number of iterations may be as large as the optimal value of the problem [93]. Local search heuristics by Amberg, Domschke and Voß [4], Sharaiha et

[1]The Steiner ratio is the maximum ratio of the costs of a minimum-cost Steiner tree versus the cost of a MST for the same instance.

al. [92], and Ahuja, Orlin and Sharma [1] are the current best heuristics in terms of the quality of the solutions produced, but their overall worst-case running time could be very large, potentially exponential. In a nut-shell, heuristics that outperform EW heuristic do so with an enormous increase in running time. Our heuristic comprehensively outperforms the EW heuristic without any increase in the worst-case running time. Experimental results on benchmark instances show that our heuristic obtains improvements of up to 17% when compared to EW heuristic.

1.4.4 Capacitated Minimum Spanning Networks

Single-sink edge installation problems with potential node/link failures has not been studied before. In Chapter 5, we introduce of concept of survivability into the CMST problem. A CMST can be viewed as a collection of local access tree networks, each with a total demand of at most k, connected to the root node. Most often, local access networks are prone to node/edge failures. To prevent such failures, the notion of survivable networks has been studied. We initiate the study of the *Capacitated Minimum Spanning Network* (CMSN) problem, which requires that the local access networks be resilient under an edge failure. We show that the CMSN problem is NP-hard, and present a 4-approximation algorithm. We also show how to obtain similar approximation results for a related problem, the 2-vertex-connected CMSN, which requires that the final network be resilient under a node failure.

1.4.5 Single-Sink Buy-at-Bulk Network Design

We can think of the CMStT problem as that of installing cables of capacity k, which can be bought at unit cost per unit length. A generalization of the CMStT problem is the *Single Sink Buy-at-Bulk* (SSBB) problem, also known as the single-sink edge installation problem, in which we are given the option of using K different cable types, instead of just one [89]. Each cable has a different capacity and the cost of the cables obey "economy of scale." In other words, it is cheaper to buy a cable of larger capacity than many cables (adding up to same capacity) of smaller capacity. Unlike the CMStT problem, the SSBB problem does not re-

quire that the final network to be a tree. But, it imposes an additional restriction (indivisibility constraint) that the flow from a node must follow a single path to the sink, i.e, the flow cannot be bifurcated. We refer to the version of the problem without the indivisibility constraint as the *divisible* SSBB (DSSBB) problem.

The SSBB problem has applications in the hierarchical design of telecommunication networks, in which the traffic from a source must follow a single path to the sink. The DSSBB problem has its own applications: a classic application would be that of routing oil from several oil wells to a major refinery [89]. After its introduction in 1996, the SSBB problem has received a lot of attention. Over the years, the approximation ratio for the SSBB problem in general metric spaces has been steadily improved as follows: $O(\log^2 n) \rightarrow O(\log n \log\log n) \rightarrow O(K) \rightarrow O(1) \rightarrow 216$, with n being the size of the problem and K being the number of available cable types.

In Chapter 6, we present a 145.6-approximation algorithm for the SSBB problem improving the previous best ratio of 216, due to Talwar [96]. For the DSSBB problem, we improve the previous best ratio of 72.8, due to Gupta et al. [49], to α_K, where α_K is less than 65.49 for all K. In particular, $\alpha_2 < 12.7, \alpha_3 < 18.2, \alpha_4 < 23.8, \alpha_5 < 29.3, \alpha_6 < 33.9$.

1.4.6 Minimum Latency Tours and the k-Traveling Repairmen Problem

In the TSP problem, the objective is to find a tour of minimum length visiting all the vertices. The TSP problem has applications in areas such as vehicle routing and scheduling. The TSP problem is only concerned about the welfare of the salesman (or a server, in the case of scheduling). It never considers the wait-time of the customers being visited (or the job being scheduled). In Chapter 7, we study a variant of the TSP problem, the *Traveling Repairmen* problem (also known as the *minimum latency problem*, the *delivery man* problem, and the *school bus-driver* problem). In the repairmen problem, the objective is not to minimize the total travel time of the repairmen, but to minimize the average wait-time of a customer. The repairmen problem is NP-hard.

Given an undirected, vertex and edge weighted graph $G = (V, E)$ and a source vertex $s \in V$, the k-traveling repairmen problem, also known as the minimum latency problem, asks for k tours, each starting at s and covering all the vertices (customers) such that the sum of the latencies experienced by the customers is minimum. The weight of a vertex represents the amount of time a repairman has to spend at that vertex, and the weight on an edge represents the time it takes to traverse it. *Latency* of a customer p is defined to be the time elapsed before visiting p for the first time. Previous literature on the repairmen problem has only considered the special case of the problem in which the vertex weights are all zero. We present the first constant factor approximation algorithm for this problem.

We also study the *Bounded Latency* problem, a complementary version of the k-traveling repairmen problem, in which we are given a latency bound L and are asked to find the minimum number of repairmen required to service all customers such that the latency of no customer is more than L. For this problem, we present a bicriteria approximation algorithm that finds a solution with at most $2/\rho$ times the number of repairmen required by an optimal solution, with the latency of no customer exceeding $(1 + \rho)L$, $\rho > 0$.

1.4.7 Bounded-Degree Minimum Spanning Trees

In Chapter 8, we consider the bounded-degree minimum spanning tree problem. Given n points in the Euclidean plane, the degree-δ minimum spanning tree problem asks for a spanning tree of minimum weight, in which the degree of each vertex is at most δ. The problem is NP-hard for $2 \leq \delta \leq 3$ [81], while the NP-hardness of the problem is open for $\delta = 4$. The problem is polynomial-time solvable when $\delta = 5$ [78], as there exists a MST, in which the degree of each vertex is at most 5. For the degree-4 MST problem, Chan [18] presented a 1.143-approximation algorithm. We show that, for any arbitrary collection of points in the plane, there always exists a degree-4 spanning tree of weight at most 1.1381, $(\sqrt{2}+2)/3$ to be exact, times than that an MST. In particular, we present an improved approximation analysis for Chan's degree-4 MST algorithm.

CHAPTER 2
CAPACITATED MINIMUM SPANNING TREES

2.1 Introduction

One of the extensively studied network design problem in telecommunications is the *Capacitated Minimum Spanning Tree* (CMST) problem. In graph theoretical terms, the CMST problem can formally be defined as follows.

> CMST: Consider a given undirected, vertex-weighted, graph $G = (V, E)$ with non-negative costs on its edges, root node $r \in V$, and capacity constraint k. The cost matrix on the edges is symmetric and satisfies the triangle inequality. The CMST problem asks for a minimum-cost spanning tree rooted at r, in which the sum of the vertex-weights in every subtree connected to r is at most k. The capacity constraint k must be at least as much as the largest vertex-weight for the CMST problem to be feasible.

The decision version of the CMST problem is NP-complete [35, 80]. The problem is NP-hard [35, 80] even for the case when vertices have unit weights and $k = 3$. The problem is polynomial-time solvable if all vertices have unit weights and $k = 2$ [35]. The problem can also be solved in polynomial time if vertices have 0,1 weights and $k = 1$, but remains NP-hard if vertices have 0,1 weights, $k = 2$ and all edge costs/lengths are 0 or 1 [35]. Even the geometric version of the problem, in which the edge costs are defined to be the Euclidean distance between the vertices they connect, remains NP-hard.

In telecommunications network design, the CMST problem is that of designing a minimum cost tree network by installing expensive (such as fiber-optic) cables along its edges.

The cables have pre-specified capacities on the amount of demand they can handle, and can be bought at a certain cost per unit length. Every source node in the network has some demand that it has to transmit to the sink node. The objective is to construct a minimum-cost tree network for simultaneous routing of demands from the source nodes to the sink node.

The CMST problem has been well studied in Computer Science and Operations Research for the past 40 years. Algorithms for finding a minimum spanning trees (MST) can be modified to find feasible solutions for the CMST problem, e.g., modified Kruskal's algorithm [72] and modified Prim's algorithm [74]. Several exact algorithms and mathematical formulations [19, 20, 26, 38, 39, 40, 45, 46, 47, 50, 66, 73] have been proposed over the years. The instance sizes that can be solved by these algorithms to optimality, in reasonable amount of time, is still far from the size of real-life instances.

Numerous heuristics for the CMST problem have been proposed. These heuristics can be classified into three categories: savings procedures, construction procedures and improvement procedures. Esau and Williams [27] gave a simple, but efficient, savings heuristic for the CMST problem. Some of the other heuristics that use savings procedure to construct the tree include [26, 41, 42, 65, 98]. Construction procedures [16, 20, 64, 68, 74, 75, 91, 94] start with an empty tree and add the best edge or node to grow the tree. The simplest of the construction procedures, called the *sweep* heuristic, was given by Sharma and El-Bardai [94]. Improvement procedures [26, 32, 64, 67, 73] start with an initial feasible solution, usually obtained by running Esau and Williams algorithm, and make improvements by adding and deleting edges as long as improvements are possible.

Out of all the CMST hheuristics, the Esau and Williams' heuristic [27], presented in 1966, is widely accepted to be the most popular and efficient. Its worst-case running time is $O(n^2 \log n)$. Almost all of the heuristics that have been proposed so far, use Esau and Williams' heuristic as a benchmark to compare their results. Improvement procedures by Amberg, Domschke and Voß [4], Sharaiha et al. [92], and Ahuja, Orlin and Sharma [1] are the current best heuristics in terms of the quality of the solutions produced, but their overall worst-

case running time could be very large, potentially exponential. These heuristics start with an initial feasible solution, and improve the initial solution by local re-alignments of nodes during every iteration. The problem with these techniques is that the amount of improvement in each iteration could be so small that the number of iterations could possibly be as large as the optimum solution itself [93]. Any other heuristic that outperforms Esau and Williams' heuristic does so with an enormous increase in running time. We will present a an $O(n^2 \log n)$ heuristic, based on Esau and Williams' heuristic, that comprehensively outperforms the Esau and Williams' heuristic in Chapter 4. We also introduced the problem of incrementally adding new nodes to an existing CMST without having to recompute the entire solution, and called it the *Dynamic Capacitated Minimum Spanning Tree* problem [58]. We propose three linear-time heuristics for this problem. For a detailed survey on CMST heuristics, we refer the reader to [4, 97].

Although most of the heuristics solve several well known instances close to optimum, they do not provide any approximation guarantee on the quality of the solutions obtained. Exact procedures are limited to solving smaller instances because of their exponential running time. Hence, arose a need to design algorithms that can provide approximation guarantees on the quality of the solutions obtained.

The first constant factor approximation algorithm for the CMST problem was presented in 1986 by Gavish and Altinkemer [42]. In [3], they presented a $(3 - \frac{2}{k})$-approximation algorithm for the special case of the CMST problem, in which the vertex-weights are all the same. They used this algorithm as a black-box to present a 4-approximation algorithm for the original CMST problem with arbitrary weights. Henceforth, we will be referring Altinkemer and Gavish by their initials AG. Algorithms for finding an MST with no constraints are well known. In fact, those algorithms can be modified to find a feasible CMST, albeit with no guaranteed approximation ratio. The celebrated clustering technique of Goemans and Williamson [44] can be used to find a feasible solution for the CMST problem with constant approximation guarantee, but the ratio will not be any better than that in [3]. The idea is to do

an exact partitioning of vertices into groups of weight exactly k, form a minimum spanning tree for each group, and connect each tree to the root vertex.

In this Chapter, we first give a brief overview of AG's 4-approximation algorithm. We then present a new 4-approximation algorithm for the CMST problem. Our algorithm is simpler and easier to analyze than AG's CMST algorithm [3]. Next, we show that there are instances for which the cost of the solution produced by our algorithm is about one third of that produced by AG's algorithm. Finally, we show how to obtain a 2-approximation ratio for a CMST instance with unit vertex-weights and $k = 3$ or 4.

Let $|uv|$ denote the cost of an edge connecting vertices u and v. For a given CMST instance, let OPT denote an optimal solution with C_{opt} being its cost, and let C_{mst} denote the cost of a MST. Let $w(v)$ denote the weight of vertex v.

2.2 Lower bounds

For a given CMST instance, C_{mst} is an obvious lower bound on the optimal cost. In the following lemma, we present another lower bound on the cost of an optimal solution, which we call the "spoke lower bound."

Lemma 2.2.1 $C_{opt} \geq \frac{1}{k} \sum_{v \in V} w(v)|rv|$.

Proof. Let t be the number of subtrees connected to r in OPT. Let l be one such subtree in OPT that is connected to r. Let S_l be the set of vertices in l. Let C_l be the sum of the cost of the edges incident on the vertices in l. By triangle inequality,

$$\begin{aligned} C_l &\geq \max_{v \in S_l}\{|rv|\} \\ &\geq \frac{\sum_{v \in S_l} w(v)|rv|}{\sum_{v \in S_l} w(v)} \\ &\geq \frac{\sum_{v \in S_l} w(v)|rv|}{k}. \end{aligned}$$

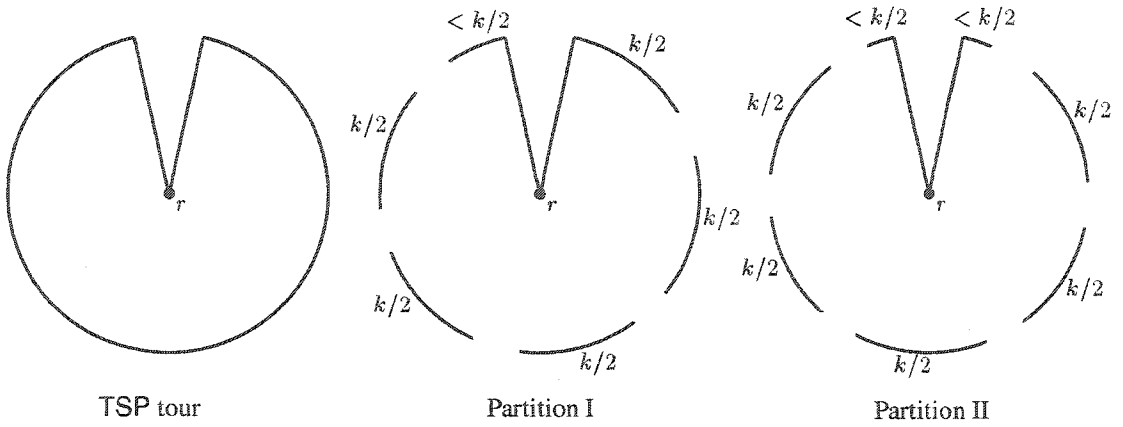


Figure 2.1. Two different partitionings of a TSP tour.

For t subtrees connected to r in OPT,

$$\begin{aligned} C_{opt} &= \sum_{l=1}^{t} C_l \\ &\geq \frac{\sum_{v \in V} w(v)|rv|}{k}. \end{aligned}$$

■

2.3 Overview of AG's algorithm

AG's algorithm converts a given CMST instance with arbitrary vertex-weights, and capacity k into a *new* instance with unit vertex-weights and capacity $k/2$, by replacing each vertex $v \in V$ of weight w_v with w_v vertices of unit weight. A traveling salesman tour (TSP) visiting all the vertices in the new instance is constructed. The tour is then partitioned into segments (paths) of weight $k/2$ (see Fig. 2.1). An optimal partitioning of the tour can be performed, instead of settling for an arbitrary partitioning, by choosing the best partitioning among all possible partitionings. If one chooses to do an optimal partitioning, there may be up to 2 segments that are of weight less than $k/2$ (see Fig. 2.1).

After partitioning the tour into segments, each path segment is connected to r. The solution for the unit weight instance with capacity $k/2$ is then converted into a feasible solution

for the given instance with capacity k. While AG's algorithm for the CMST problem with unit vertex-weights is straightforward, converting the solution for the unit vertex-weighted CMST instance with capacity $k/2$ into a feasible solution for the arbitrary vertex-weighted CMST instance with capacity k is quite involved. We refer the reader to [3] for more details.

The final cost of the solution comprises of two components: (i) the cost of the edges in the TSP, and (ii) the cost of the other edges (edges that connect the segments to r). Since a TSP can be constructed by doubling the edges of an MST, for any given instance, the cost of the TSP is at most $2C_{mst} \leq 2C_{opt}$, as one can construct a TSP whose cost is at most twice that of an MST. They show that the cost of connecting each segment to r is at most $2C_{opt}$. This translates into an overall cost of at most $4C_{opt}$.

2.4 A new 4-approximation algorithm

Our algorithms improve on the ideas of AG's algorithm. Unlike AG's algorithm, our algorithms can directly be applied to a given instance without having to do any conversions as in [3], thereby reducing the running time.

Algorithm PARTITION-TOUR (V^*, r, k)

1. Construct a TSP tour t on $V^* \cup \{r\}$.
2. Set `segmentWeight` = 0.
3. Starting from r, traverse the tour in clockwise direction.
4. Let $prev[v]$ and $next[v]$ be the vertices that are lying just before and after vertex v on the tour, respectively.
5. While not all vertices in t are traversed and `segmentWeight` $< k$, visit the next vertex v.
   (a) If `segmentWeight`$+w(v) < k$, increment `segmentWeight` by $w(v)$.

(b) Else if $w(v) \geq k/2$, remove v from t, and use shortcutting to connect $prev[v]$ to $next[v]$. (comment: v is a segment by itself)

(c) Else remove the edge connecting v to $prev[v]$ from t, and set `segmentWeight`$= w(v)$.

6. Return the segments.

Algorithm CMST-MAIN (V, r, k)

1. Construct an MST T spanning V.

2. Root T at the root vertex r.

3. Let $T_1, T_2, \ldots, T_m$ be the subtrees rooted at the children of r.

4. For every subtree T_i connected to r in T,

   (a) $S \leftarrow$ PARTITION-TOUR $(\{v | v \in T_i\}, r, k)$.

   (b) For every segment in S, if it is not already connected to r, connect r to the closest vertex in the segment.

It can be verified that the above algorithm outputs a feasible solution for a given CMST instance. In what follows, we show that the cost of the solution output by the above algorithm is at most four times the cost of an optimal solution.

Theorem 2.4.1 *Algorithm* CMST-MAIN *guarantees an approximation ratio of 4.*

Proof. The cost of the final solution comprises of two components: (i) the cost of the TSP tours for each subtree hanging off r, and (ii) the cost of connecting the individual segments, of weight at most k, to r. The cost of the TSP tours is at most twice the cost of the initial MST, since we can construct a tour spanning r and the vertices in a subtree rooted at r by just doubling the necessary MST edges and shortcutting the resulting Euler tour.

We now show that the cost of connecting the individual segments to r is at most twice the cost of OPT. As per the partitioning procedure, observe that all, but the last, segments in a tour are guaranteed to be of weight at least $k/2$. Moreover, notice that there is no need to connect the first and the last segment of a tour to r as they are already connected to r by the tour edges, whose cost has already been accounted. Let $s_1, s_2, \ldots, s_m$ be the segments for which the algorithm added edges to connect them to r. Recall that each of these edges connects r to the closest vertex in a segment. Let V' represents the set of all vertices in segments $s_1, s_2, \ldots, s_m$, and let v_i be the vertex in s_i that is connected to r. Let $W(s_i) = \sum_{j \in s_i} w(j)$. Then,

$$\begin{aligned}
|rv_i| &\leq \sum_{j \in s_i} \frac{w(j)|rj|}{W(s_i)} \\
&\leq \sum_{j \in s_i} \frac{w(j)|rj|}{k/2} \\
\sum_{i=1}^{m} |rv_i| &\leq \sum_{i=1}^{m} \sum_{j \in s_i} \frac{w(j)|rj|}{k/2} \\
&\leq \sum_{j \in V} \frac{w(j)|rj|}{k/2} \\
&\leq 2C^* \qquad \text{(by Lemma 2.2.1)}
\end{aligned}$$

Thus, we can conclude that the overall cost of the tree output by the algorithm is at most four times than that of an optimal solution. ∎

In practice, our algorithm may actually perform better than AG's algorithm. Let us consider an instance with unit weight vertices, with r placed at distance M from all the other vertices. Let the non-root vertices be clustered into k groups of $k-1$ vertices, with inter-distance between vertices in the same group being ϵ, and inter-distance between vertices in different groups being $2M$. It can be verified that the edges of the instance obey the triangle inequality. Figures 2.2 and 2.3 illustrate the execution of AG's algorithm and our algorithm, respectively, for the instance with $k = 8$. While the cost of the solution obtained by AG's algorithm is $3(k-1)M + (k-1)(k-2)\epsilon$, the cost of the solution obtained by our algorithm would only be $kM + k(k-2)\epsilon$. As $k \to \infty$ and $\epsilon \to 0$, our algorithm would return a solution

of cost about one third of that returned by AG's algorithm. Note that the solution obtained by our algorithm is in fact optimal.

2.4.1 Unit vertex-weights with $k = 3; 4$

The CMST problem is polynomial time solvable if all vertices have unit weights, and $k = 2$ [35]. We show that an optimal solution for $k = 2$ is a 2-approximation for $k = 3, 4$. For a given CMST instance, let opt_2, opt_3 and opt_4 be optimal costs for $k = 2$, $k = 3$, and $k = 4$, respectively.

Theorem 2.4.2 *Given an undirected graph $G = (V, E)$ with unit vertex-weights, $r \in V$ and $k = 3$ or 4, an optimal solution for the* CMST *problem with $k = 2$ is a 2-approximation for the* CMST *problem with $k = 3, 4$.*

Proof. We obtain the following inequalities from the fact that one could obtain a feasible solution for $k = 2$ by doubling the edges of an optimal solution for $k = 3, 4$.

$$opt_2 \leq 2 \times opt_3 \quad \text{and} \quad opt_2 \leq 2 \times opt_4.$$

If we were to use opt_2 as a feasible solution for $k = 4$, then

$$Ratio = \frac{opt_2}{opt_4} \leq \frac{opt_2}{opt_2/2} = 2.$$

Using the same argument, we can conclude that $opt_2/opt_3 \leq 2$. ∎

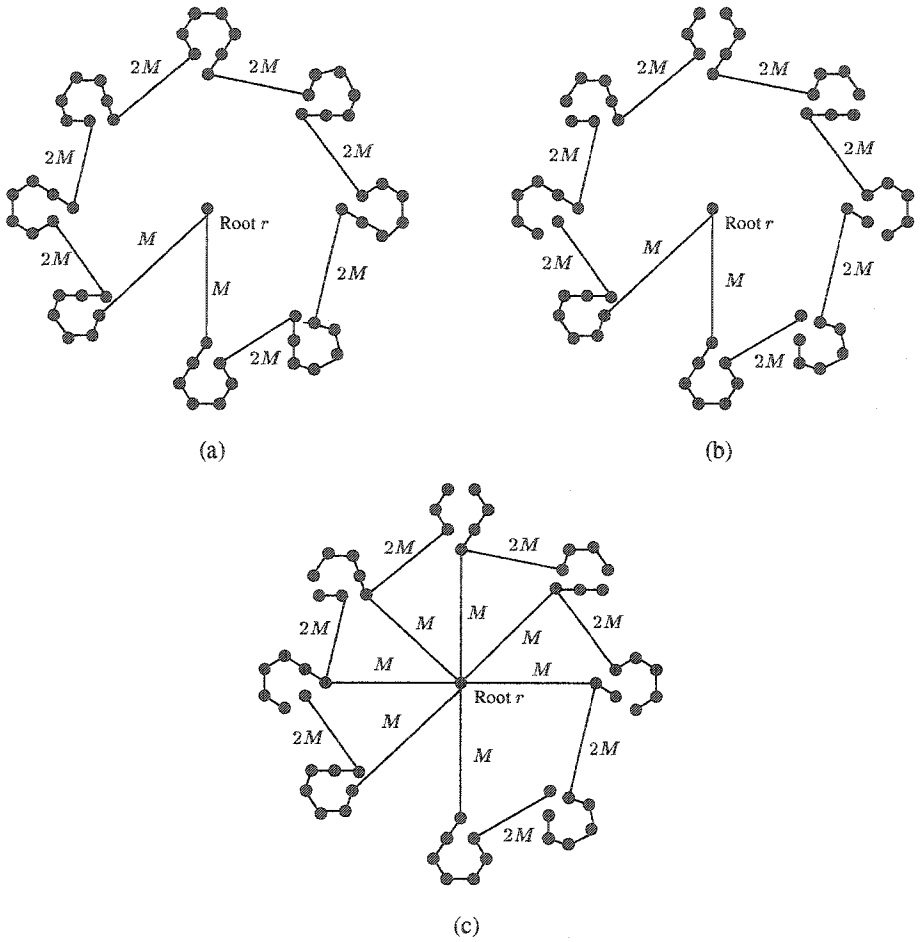


Figure 2.2. AG's algorithm on a sample instance with $k = 8$. (a) TSP tour spanning all the vertices. (b) Partitioning of the TSP tour. (c) Partitioned subtrees are connected to root r, with the cost of the final tree being $3(k-1)M + (k-1)(k-2)\epsilon$.

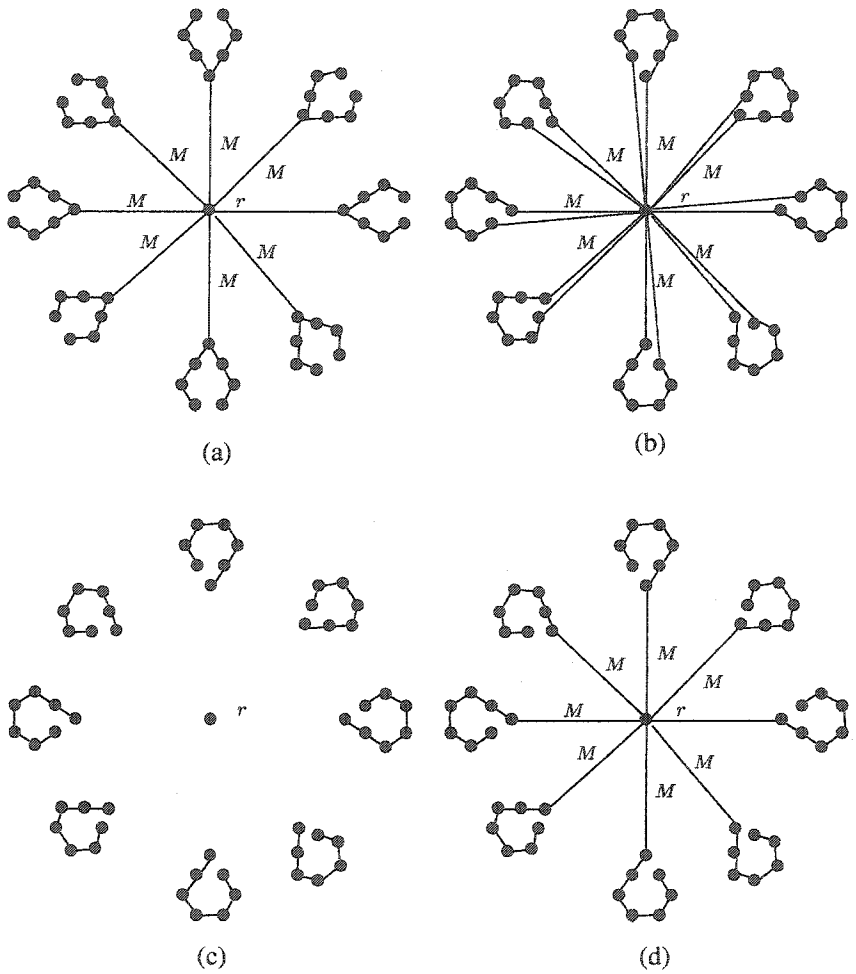


Figure 2.3. Our algorithm on the sample instance with $k = 8$. (a) MST spanning all the vertices. (b) TSP tours, each spanning r and the vertices in a subtree rooted hanging off r. (c) Partitioning of the TSP tours into segments. (d) Partitioned subtrees are connected to root r, with the cost of the final tree being $kM + k(k-2)\epsilon$.

CHAPTER 3
CAPACITATED MINIMUM STEINER TREES

3.1 Introduction

In this chapter, we consider the *Capacitated Minimum Steiner Tree* (CMStT) problem, a generalization of the CMST problem discussed in Chapter 2.

> CMStT: Given an undirected graph $G = (V, E)$ with non-negative costs on its edges, a root node $r \in V$, a set of demands $D \subseteq V$ with with demand $v \in D$ wishing to route $w(v)$ units of flow (weight) to r, and a positive number k, the *Capacitated minimum Steiner tree* (CMStT) problem asks for a minimum-cost Steiner tree, rooted at r, spanning the vertices in $D \cup \{r\}$, in which the sum of the vertex-weights in every subtree connected to r is at most k.

The capacity constraint k must be at least as much as the largest vertex-weight for the CMStT problem to be feasible. The CMStT problem is NP-hard, since the case with $k = \infty$ captures the minimum Steiner tree problem, which is NP-hard. When $D = V$, the CMStT problem is the well-known capacitated minimum spanning tree (CMST) problem.

The CMStT problem has not been studied before. However, the CMST algorithms in [3] can be used to obtain feasible solutions with approximation ratios $2\rho_{ST} + 1$ and $2\rho_{ST} + 2$ for uniform and non-uniform vertex-weighted graphs, respectively, where ρ_{ST} is the best achievable approximation ratio for the minimum Steiner tree problem. Algorithms for the *network loading* problem, due to Hassin, Ravi and Salman [51], can also be used obtain feasible solutions for the CMStT problem with the same ratios. In this chapter, we present approximation algorithms for CMStT problem and several of its variations in network design.

Since the CMST problem is a special case of the CMStT problem, our approximation ratios for the CMStT problem directly translates into improved approximation ratios for the CMST problem, thereby solving the long-standing open problem of obtaining better approximation ratios for the CMST problem. The main results are the following.

- We give a $(\gamma\rho_{ST} + 2)$-approximation algorithm for the CMStT problem, where γ is the *inverse Steiner ratio*[1]. Our ratio improves the current best ratio of $2\rho_{ST} + 2$ for this problem.

- In particular, we obtain $(\gamma+2)$-approximation ratio for the CMST problem, which is an improvement over the current best ratio of 4 for this problem. For points in Euclidean and Rectilinear planes, our result translates into ratios of 3.1548 and 3.5, respectively.

- For instances in the plane, under the L_p norm, with the vertices in D having uniform weights, we give a $(\frac{7}{5}\rho_{ST} + \frac{3}{2})$-approximation algorithm for the CMStT problem. This translates into a ratio of 2.9 for the CMST problem with uniform vertex-weights in the L_p metric plane. Our ratio of 2.9 solves the long standing open problem of obtaining a ratio any better than 3 for this case.

- For the *budgeted* CMST problem, in which the weights of the subtrees hanging off r could be up to αk instead of k ($\alpha \geq 1$), we obtain a ratio of $\gamma + \frac{2}{\alpha}$.

Of the above results, the 2.9-approximation result for the CMST problem is of most significance. This is due to the fact that obtaining a ratio any better than 3 for graphs defined in the Euclidean plane (with uniform vertex-weights) is not straightforward. There are several ways one can obtain a ratio of 3 for this problem ([3], modified algorithm of [51], our algorithm in Section 3.4.1). But the question was whether one can ever obtain a ratio smaller than $3 - \epsilon$ for this version of the CMST problem. We present an example (in Section 3.6),

[1]The Steiner ratio is the maximum ratio of the costs of the minimum-cost Steiner tree versus the minimum-cost spanning tree for the same instance.

which shows that it is not possible to obtain an approximation ratio any better than two using the current known lower bounds for the CMST problem. We introduce a novel concept of *X-trees* to overcome the difficulties in obtaining a ratio better than three.

Achieving ratios better than 3 and 4 for the uniform and non-uniform vertex-weighted CMST problems, respectively, has been an open problem for 15 years now. One major reason for the difficulty in finding better approximations is that there is no non-trivial lower bound for an optimal solution. There are instances for which the cost of an optimal solution can be as much as $\Omega(n/k)$ times than that of an MST. Inability to find better lower bounds has greatly impeded the process of finding better approximation ratios for this problem. Even though we were not able to completely eliminate the use of MST as a lower bound, we found ways to exploit its geometric structure, thereby achieving better performance ratios. Unlike the algorithms in [3], in which the MST lower bound contributes a factor of 2 to the final ratio, our algorithms reduces the use of the MST lower bound, thereby achieving better ratios.

3.2 Preliminaries

Length of an edge is also its cost. The terms points, nodes and vertices will be used interchangeably. For a given k, let OPT and APP denote optimal and approximate solutions, respectively, and let C_{opt} and C_{app} denote their respective costs. Let C_{mst} and C_{ST} denote the costs of an MST and an optimal Steiner tree, respectively.

In a rooted tree T, let T_v denote the subtree rooted at v. Let C_T denote the cost of tree T. Let $w(v)$ denote the weight of vertex v, and let $w(T_v)$ denote the sum of vertex-weights in the subtree rooted at v. For the CMStT problem, the weight of a vertex the is not in D is assumed to be 0. By weight of a subtree, we mean the sum of the vertex-weights in that subtree. We call as *spokes*, the edges incident on r of a CMStT. By *level* of a vertex, in a tree T rooted at r, we mean the number of tree edges on its path to r (also known as depth). Without loss of generality, we assume that the weights of vertices in $V \backslash D$ is zero. All our algorithms are for the CMStT problem—a generalization of the CMST problem.

3.3 Lower bounds

The "spoke lower bound" for the CMST problem given in Section 2.2 is a lower bound for the CMStT problem as well. Also, since any feasible solution to the CMStT problem is a Steiner tree, the cost of a minimum Steiner tree is a lower bound for the CMStT problem.

3.4 CMStT algorithms

We first construct a ρ_{ST}-approximate Steiner tree T spanning all the vertices in $D \cup \{r\}$, and then root T at the root vertex r. Next, we prune subtrees of weight at most k in a bottom-up fashion, and add edges to connect r to the closest node in each of the pruned subtrees. In simple terms, we cut T into subtrees of weight at most k and connect them to the root vertex.

It is safe to assume that nodes have integer weights. The assumption is not restrictive as any CMStT problem with rational weights can be converted to an equivalent problem with integer node weights. The optimal solution for the scaled problem is identical to that of the original problem [3].

Since our algorithm for the uniform vertex-weights case is quite complex, we first present the algorithm for the general case (non-uniform vertex-weights). Note that all our algorithms start with a ρ_{ST}-approximate Steiner tree of constant degree. Before we proceed to the algorithms, we present the following important lemma.

Lemma 3.4.1 *Consider a given graph $G = (V, E)$, a set of demands $D \subseteq V$, $r \in V$, and maximum capacity k. Let T_f be a feasible* CMStT *and let $T_1, T_2, \ldots, T_m$ be the subtrees hanging off r in T_f. Let $w(T_q)$ be the weight of a minimum weight subtree T_q hanging off r. For all i, if the cost of the edge connecting subtree T_i to r is minimum, then the cost C_{sp} of all the edges incident on r (spokes) in T_f is at most $k/w(T_q)$ times the cost of an optimal solution.*

Proof. Let Γ be the set of vertices in $T_1, \ldots, T_m$. For all i, let v_i be the vertex in T_i through which T_i is connected to r. Recall that edge rv_i is a spoke, and that it is a minimal cost edge

crossing the cut between r and T_i. Then,

$$\begin{aligned} |rv_i| &\leq \frac{\sum_{v \in T_i} w(v)|rv|}{\sum_{v \in T_i} w(v)} \\ &\leq \frac{\sum_{v \in T_i} w(v)|rv|}{w(T_q)}. \end{aligned}$$

The cost of the all the edges incident on r is given by

$$\begin{aligned} C_{sp} &= \sum_{i=1}^{m} |rv_i| \\ &\leq \frac{\sum_{v \in \Gamma} w(v)|rv|}{w(T_q)} \\ &= \frac{k}{w(T_q)} \frac{\sum_{v \in D} w(v)|rv|}{k} \\ &\leq \frac{k}{w(T_q)} C_{opt}. \quad \text{(by Lemma 2.2.1)} \end{aligned}$$

■

3.4.1 Non-uniform vertex-weights

The algorithm given below outputs a feasible CMStT for a given instance, whose edges obey triangle inequality. Note that during the course of the algorithm, we replace real vertices with *dummy* vertices of zero weight. These dummy vertices can be thought of as Steiner points. In the algorithm, we use c_i to denote the subtree rooted at child i of vertex v, and p_v to denote v's parent.

Algorithm CMStT-NONUNIFORM

Input: ρ_{ST}-approximate Steiner tree T rooted at r.

1. Choose a maximum level vertex $v \neq r$ such that $w(T_v) \geq k$. If there exists no such vertex then output the tree constructed thus far.

2. If $w(T_v) = k$, then replace the Steiner tree edges incident on the vertices in T_v with the edges of a minimal cost tree τ spanning only the vertices in $T_v \cap D$. Add a new edge connecting r to the closest vertex in τ.

3. Else if, for some i, $w(c_i) \geq k/2$, then replace the Steiner tree edges incident on the vertices in c_i with the edges of a minimal cost tree τ spanning only the vertices in $c_i \cap D$. Add a new edge connecting r to the closest vertex in τ.

4. Else if $\sum w(c_i) < k/2$ (i.e., $w(v) > k/2$), then replace v with a dummy vertex. In the final solution, add v and an edge connecting v to r.

5. Else collect a subset s of subtrees, each of which is rooted at one of v's children, such that $k/2 \leq w(s) \leq k$. Replace the Steiner tree edges incident on the vertices in s with the edges of a minimal cost tree τ spanning only the vertices in $s \cap D$. Add a new edge connecting r to the closest vertex in τ.

6. Go to step 1.

It can be verified that our algorithm outputs a feasible CMStT for a given k.

Theorem 3.4.1 *For a given* CMStT *instance, Algorithm* CMStT-NONUNIFORM *guarantees an approximation ratio of* $\gamma\rho_{ST} + 2$.

Proof. We show that the cost of the tree output by Algorithm CMStT-NONUNIFORM is at most $\gamma\rho_{ST} + 2$ times the cost of an optimal CMStT. The input to the algorithm is a ρ_{ST}-approximate Steiner tree T.

It can be verified that all the new edges added by the algorithm to the original tree T are either new spokes, or edges that interconnect vertices within the subtrees for which the new spokes were added. In what follows, we account for the cost of the new spokes added to T, followed by the cost of other edges in the final solution output by the algorithm.

A new spoke, incident on a subtree, is added to the original Steiner tree if and only if the weight of the subtree it connects is at least $k/2$. Notice that the algorithm outputs a tree with each subtree hanging off r being disjoint and the weight of every such subtree (for which a new spoke was added) is at least $k/2$. Let C_{sp} be the cost of the spokes that the algorithm

"adds" to the Steiner tree. Note that C_{sp} does not include the cost of the spokes that are already in the Steiner tree that was given as input to the algorithm. By Lemma 3.4.1,

$$C_{sp} \leq 2C_{opt}.$$

Now, we account for the cost of other edges in the final solution. These edges are either the Steiner tree edges or the edges that replaced the Steiner tree edges. We show that the total cost of all these edges together is at most γ times the cost of the initial Steiner tree. To prove this, it suffices to prove that the cost of the edges that replace the Steiner tree edges is at most γ times the cost of the Steiner tree edges that it replaces. For every subtree formed, notice that the algorithm replaced the edges of the Steiner tree spanning the vertices in that subtree by the edges of an MST spanning only the non-zero-weight vertices in that subtree. Since γ was defined to be the inverse Steiner ratio (maximum ratio of the cost of an MST versus the cost of an optimal Steiner tree), the cost of the MST spanning only the non-zero-weight vertices in a subtree is at most γ times the cost of an optimal Steiner tree spanning the non-zero-weight vertices in that subtree. Thus, we can conclude that the cost of the new edges is at most γ times the cost of the ρ_{ST}-approximate Steiner tree edges it replaces. The final cost of the tree output by the algorithm is given by

$$\begin{aligned} C_{app} &\leq C_{sp} + \gamma\rho_{ST}C_{ST} \\ &\leq 2C_{opt} + \gamma\rho_{ST}C_{opt} \\ &\leq (\gamma\rho_{ST} + 2)C_{opt}. \end{aligned}$$

∎

Corollary 3.4.1 *For the* CMStT *problem with uniform vertex-weights, Algorithm* CMStT-NonUniform *can be modified to guarantee an approximation ratio of* $\rho_{ST} + 2$.

Proof. Since we are dealing with uniform vertex-weights, without loss of generality, we can assume that they are of unit weight, and thus we can eliminate Step. 4 from Algorithm

CMStT-NONUNIFORM. Therefore no dummy vertices are introduced by the algorithm. Once a subtree t of size at least $k/2$ is found, instead of replacing the Steiner tree spanning the vertices in t with a MST spanning the non-zero-weight vertices in t, we can just use the edges in t, minus the edge that connects t to its parent, as they are. This eliminates the γ from the final ratio. ∎

Corollary 3.4.2 *For the* CMST *problem, Algorithm* CMStT-NONUNIFORM *guarantees a* $(\gamma + 2)$*-approximation ratio. In particular, for points in Euclidean and rectilinear planes, it guarantees a ratio of 3.1548 and 3.5, respectively.*

3.4.2 Uniform vertex-weights

Although our algorithm for uniform vertex-weights case is similar to Algorithm CMStT-NONUNIFORM at the top-level, contrary to expectations, there are some complicated issues that have to be handled in order to obtain an approximation ratio strictly smaller than $\rho_{ST} + 2$. From our analysis for the non-uniform vertex-weights case, we can see that the weight of the minimum weight subtree hanging off r plays a crucial role in the calculation of the approximation ratio. An obvious heuristic is to prune subtrees of weight as close as possible to k, so that the ratio drops considerably. We will soon see why pruning subtrees of weight significantly larger than $k/2$ is more difficult. The subtree given in Fig. 3.1 is a sample scenario where it is not quite possible to prune a subtree of weight, say, $3k/4$. To overcome the difficulty of pruning subtrees of size strictly greater than $k/2$, we introduce the concept of *X-trees*, which we define below. We call a subtree, T_v, rooted at vertex v as an *X-tree*, x, if all of the following properties are satisfied (follow Fig. 3.1).

- $k < w(T_v) < \frac{4}{3}k$.
- Weight of no subtree hanging off v is in the range $[\frac{2}{3}k, k]$.
- Sum of the weights of no two subtrees hanging off v is in the range $[\frac{2}{3}k, k]$.

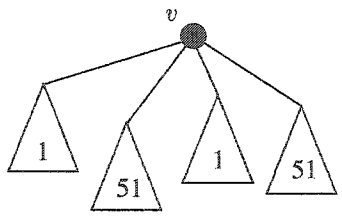


Figure 3.1. An X-tree with $k = 100$.

- Sum of the weights of no three subtrees hanging off v is in the range $[\frac{2}{3}k, k]$.

The following proposition follows from the definition of an X-tree.

Proposition 3.4.1 *Let v_1 be a maximum level vertex in an X-tree rooted at v such that T_{v_1} is also an X-tree (v_1 could be v itself). If there is no subtree (non-X-tree) of weight greater than k rooted at one of v_1's children, then there always exist two subtrees, t_α and t_β, hanging off v_1 such that $k < w(t_\alpha) + w(t_\beta) < \frac{4}{3}k$ and $\frac{1}{3}k < w(t_\alpha), w(t_\beta) < \frac{2}{3}k$.*

Since the vertices are of uniform weight, without loss of generality, we can assume that they are of unit weight, and scale k accordingly. We also assume that a ρ_{ST}-approximate Steiner tree is given as part of the input. Note that we are trying to solves instances in L_p metric plane. Even though, the maximum nodal degree in a Steiner tree on a plane is 3, we will continue as if it is 5. This is to ensure that our algorithm solves CMST instances on a plane, as the maximum degree of an MST on a L_p plane is 5 [78, 87]. Note that every vertex but the root in a tree with vertex degrees at most 5, has at most 4 children. The algorithm given below finds a feasible CMStT for instances defined on a L_p plane. In the algorithm, we use c_i to denote the subtree rooted at child i of vertex v, and x_j to denote the X-tree rooted at child j of vertex v.

Algorithm CMStT-UNIFORM

Input: ρ_{ST}-approximate Steiner tree T rooted at r, with vertex degrees at most 5.

1. Choose a maximum level vertex $v \neq r$ such that T_v is a non-X-tree with $w(T_v) \geq k$. If there exists no such vertex then go to step 11.

2. If $w(T_v) = k$, then add a new edge connecting r to the closest node in T_v. Remove edge vp_v from T.

3. Else if, for some i, $2k/3 \leq w(c_i) \leq k$, then add a new edge connecting r to the closest node in c_i. Remove the edge connecting v to c_i from T.

4. Else if, for some i and j $(i \neq j)$, $2k/3 \leq w(c_i) + w(c_j) \leq k$, then replace edges vc_i and vc_j by a minimal cost edge connecting c_i and c_j, merging the two subtrees into a single tree s. Add a new edge to connect r to the closest node in s.

5. Else if, for some i, j and z $(i \neq j \neq z)$, $2k/3 \leq w(c_i) + w(c_j) + w(c_z) \leq k$, then replace the Steiner tree edges incident on the vertices in c_i, c_j and c_z by a minimal cost tree s spanning all the vertices in c_i, c_j and c_z. Add a new edge to connect r to the closest node in s.

6. Else if, for some i, j and z $(i \neq j \neq z)$, $4k/3 \leq w(c_i) + w(c_j) + w(c_z) \leq 2k$, then do the following.

   Let E_i be the set of edges of T incident on vertices in c_i. We define E_j (E_z) with respect to c_j (c_z resp.) analogously. Without loss of generality, let E_j be the edge-set among E_i, E_j and E_z with least cost. Use DFS on c_j to partition the vertices in c_j into two sets g_1 and g_2 such that the total weight of vertices in $(c_i \cup g_1) \cap D$ is almost the same as the total weight of vertices in $(c_z \cup g_2) \cap D$. Remove all the edges incident on the vertices in subtrees c_i, c_j and c_z. Construct a minimal cost spanning tree s_1 comprising the vertices in $c_i \cup g_1$. Similarly, construct a minimal cost spanning tree s_2 comprising the vertices in $c_z \cup g_2$. Add new edges to connect r to the closest nodes in s_1 and s_2.

7. Else if, for some i and j $(i \neq j)$, $2k < w(x_i) + w(x_j) < 8k/3$, do the following.

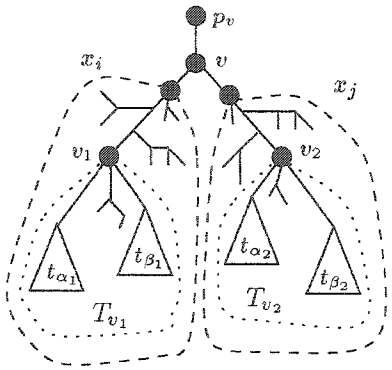


Figure 3.2. A sample scenario for Step 7

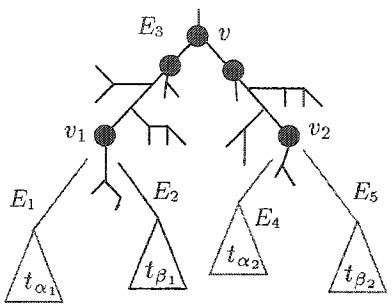


Figure 3.3. Edge sets

Let v_1 and v_2 be two maximum level vertices in X-trees x_i and x_j respectively, such that T_{v_1} and T_{v_2} are X-trees themselves (see Fig. 3.2). Recall, by Proposition 3.4.1, that there exist two subtrees t_{α_1} and t_{β_1} (t_{α_2} and t_{β_2}), hanging off v_1 (v_2 resp.) such that $k < w(t_{\alpha_1}) + w(t_{\beta_1}) < \frac{4}{3}k$ ($k < w(t_{\alpha_2}) + w(t_{\beta_2}) < \frac{4}{3}k$ resp.).

Let E_1 represent the set of edges incident on vertices in t_{α_1} (see Fig. 3.3). Let E_2 represent the set of edges incident on vertices in t_{β_1}. We define E_4 (E_5) with respect to t_{α_2} (t_{β_2} resp.) analogously. Let E_3 be the set of edges incident on vertices in x_i and x_j minus the edges in E_1, E_2, E_4 and E_5.

Let $G_1 = \{E_1, E_2\}$, $G_2 = \{E_3\}$, and $G_3 = \{E_4, E_5\}$ be three groups. Out of $\{E_1, E_2, E_3, E_4, E_5\}$, double two edge-sets of least cost such that they belong to differ-

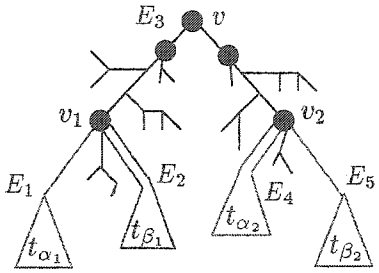


Figure 3.4. Step 7(a)

ent groups.

(a) If E_i and E_j were the two edges sets that were doubled, with E_i in G_1 and E_j in G_3, then form three minimal cost subtrees s_1, s_2 and s_3 spanning the vertices in x_i and x_j as follows. Without loss of generality, let E_2 and E_4 be the two low-cost edge-sets that were doubled (Fig. 3.4). Use shortcutting to form s_1 spanning all vertices in t_{α_1} and a subset of vertices in t_{β_1}, form s_3 spanning all vertices in t_{β_2} and a subset of vertices in t_{α_2}, and form s_2 with all the left-over vertices. Remove edge vp_v. Since $k < w(t_{\alpha_1}) + w(t_{\beta_1}) < 4k/3$, $k < w(t_{\alpha_2}) + w(t_{\beta_2}) < 4k/3$, and $2k \leq w(x_i) + w(x_j) \leq 8k/3$, we can form s_1, s_2 and s_3 of almost equal weight with $2k/3 \leq w(s_1), w(s_2), w(s_3) \leq k$.

(b) If E_i and E_j were the two edges sets that were doubled, with E_i in G_1 or G_3, and E_j in G_2, then form three minimal cost subtrees s_1, s_2 and s_3 spanning the vertices in x_i and x_j as follows. Without loss of generality, let E_2 and E_3 be the two low-cost edge-sets that were doubled (see Fig. 3.5). From t_{α_2} and t_{β_2} find a vertex w such that $|wr|$ is minimum. Without loss of generality, let t_{α_2} contain w. Use shortcutting to form s_3 spanning all the vertices in x_j minus the vertices in t_{β_2} (see Fig. 3.6). Note that $k/3 < w(s_3) < k$, as x_j and T_{v_2} are X-trees and $k/3 < w(t_{\alpha_2}), w(t_{\beta_2}) < 2k/3$. Also, since $k/3 < w(t_{\beta_2}) < 2k/3$ and $k < w(x_i) < 4k/3$, subtrees s_1 and s_2 together will be of weight at least $4k/3$ and at most $2k$ (see Fig. 3.6). Form subtrees s_1 and s_2, using the ideas in Step 6,

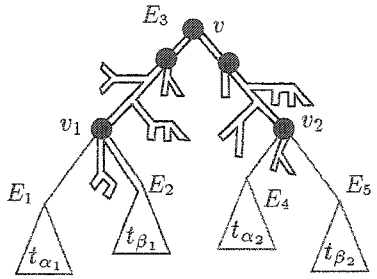


Figure 3.5. Step 7(b)

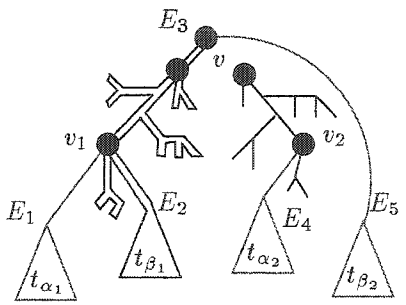


Figure 3.6. Subtree pruning

such that $2k/3 \leq w(s_1), w(s_2) \leq k$ and $4k/3 \leq w(s_2) + w(s_3) \leq 2k$.

(c) Add new edges to connect r to the closest nodes in s_1, s_2 and s_3.

8. Else if, for some i and j $(i \neq j)$, $4k/3 \leq w(x_i) + w(c_j) < 2k$, do the following.

   Let v_1 be a maximum level vertex in X-tree x_i such that T_{v_1} is an X-tree itself. Recall, by Proposition 3.4.1, that there exist two subtrees t_{α_1} and t_{β_1}, hanging off v_1 such that $k < w(t_{\alpha_1}) + w(t_{\beta_1}) < \frac{4}{3}k$.

   Let E_1 represent the set of edges incident on vertices in t_{α_1}. Let E_2 represent the set of edges incident on vertices in t_{β_1}. Let E_3 be the set of edges incident on vertices in x_i and c_j minus the edges in E_1 and E_2. Form subtrees s_1 and s_2 using the ideas in Step 6. Add new edges to connect r to the closest nodes in s_1 and s_2.

9. Else if, $4k/3 \leq w(T_v) \leq 2k$, do the following. Let v_1 be a maximum level vertex in

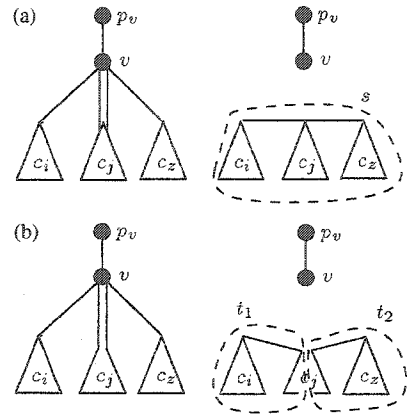


Figure 3.7. (a) Step 5 (b) Step 6

X-tree x_i such that T_{v_1} is an X-tree itself. Recall, by Proposition 3.4.1, that there exist two subtrees t_{α_1} and t_{β_1}, hanging off v_1 such that $k < w(t_{\alpha_1}) + w(t_{\beta_1}) < \frac{4}{3}k$.

Let E_1 represent the set of edges incident on vertices in t_{α_1}. Let E_2 represent the set of edges incident on vertices in t_{β_1}. Let E_3 be the set of edges incident on vertices in T_v minus the edges in E_1 and E_2. Form subtrees s_1 and s_2 using the ideas in Step 6. Add new edges to connect r to the closest nodes in s_1 and s_2.

10. Go to step 1.

11. While there is an X-tree, x, hanging off r, pick a maximum level vertex v_1 in x such that T_{v_1} is also an X-tree. Out of the two subtrees, t_α and t_β, hanging off v_1 (by Proposition 3.4.1), without loss of generality, let t_α be the subtree that is closer to r. Remove the edge connecting t_α to v_1, and add a new edge to connect r to the closest node in t_α.

Lemma 3.4.2 *Algorithm* CMStT-UNIFORM *considers all the possible cases for a given subtree* T_v.

Proof. Recall that we are working with a ρ_{ST}-approximate Steiner tree with maximum nodal degree at most 5. If the subtree T_v fits into one of Steps 2-6, then the algorithm will solve the

case using one of those steps. Hence for the rest of the proof, we can assume that subtree T_v does not fit into any of Steps 2-6. Since T_v is a subtree of weight greater than k and does not fit into any of Steps 2-6, there must be at least one X-tree hanging off v. As can be seen from the algorithm, Steps 7-9 capture all possible cases under this setting. ∎

Theorem 3.4.2 *For a given* CMStT *instance on a L_p plane, Algorithm* CMStT-UNIFORM *guarantees an approximation ratio of $\frac{7}{5}\rho_{ST} + \frac{3}{2}$.*

Proof. We show that the cost of the tree output by Algorithm CMStT-UNIFORM is at most $\frac{7}{5}\rho_{ST}+\frac{3}{2}$ times the cost of an optimal CMStT. The input to the algorithm is a ρ_{ST}-approximate Steiner tree T with maximum nodal degree at most 5.

The algorithm "adds" a new spoke to the tree whenever it prunes a subtree of weight at least $2k/3$. In some cases (Steps 6 and 11), the algorithm adds a spoke to pruned subtrees of weight less than $2k/3$. We continue our analysis as if all of the pruned subtrees have a weight at least $2k/3$. This supposition makes the analysis of spoke cost simpler. We will soon justify this supposition (in Cases 5 and 8) below.

The cost of the spokes that were added to the initial Steiner tree is given by

$$C_{sp} \leq \frac{3}{2} C_{opt}$$

by an argument analogous to that proving the cost of the spokes that the algorithm adds to the initial Steiner tree in Theorem 3.4.1. The above inequality follows immediately from the fact that a new spoke is added to the tree if and only if the subtree it connects to r is of weight at least $2k/3$.

Now, we account for the cost of other edges (in the final solution), except for the spokes added by the algorithm. We show that the cost of these edges is at most 7/5 times the cost of the initial Steiner tree given to the algorithm. To prove this, it suffices to show that the cost of the edges that replace the Steiner tree edges is at most 7/5 times the cost of the edges that

are replaced. In what follows, we show this by presenting a case-by-case analysis depending upon which step of the algorithm was executed.

Case 1. Steps 1, 2, 3 and 10 do not add any non-spoke edges. Whenever spokes are added in Steps 2 and 3, the weight of the subtrees is in the range $[2k/3, k]$.

Case 2. The minimum-cost edge connecting c_i and c_j in Step 4 is at most the sum of the two Steiner tree edges that connects c_i and c_j to v (by triangle inequality). Hence no additional cost is involved.

Case 3. In Step 5, the cost of the tree s spanning all the vertices in c_i, c_j and c_z is at most the cost of the tree obtained by doubling the minimum-cost edge out of vc_i, vc_j and vc_z (see Fig. 3.7(a)). Hence, we can conclude that the cost of the tree constructed in Step 5 is at most 4/3 times the cost of the Steiner tree edges it replaces.

Case 4. In Step 6, the total cost of the trees s_1 and s_2 spanning all the vertices in c_i, c_j and c_z is at most the total cost of the trees t_1 and t_2 obtained by doubling the minimum-cost edge-set out of the 3 edge-sets that are incident on the vertices in c_i, c_j and c_z, respectively (see Fig. 3.7(b)). Hence, we can conclude that the cost of the tree constructed in Step 6 is at most 4/3 times the cost of the Steiner tree edges it replaces.

Case 5. Step 7 forms three subtrees s_1, s_2 and s_3 from X-trees x_i and x_j. Since s_1, s_2 and s_3 can be formed by doubling two edge-sets of least cost (belonging to two different groups) out of the 5 possible edge-sets and shortcutting, we can conclude that the cost of the subtrees s_1, s_2 and s_3 constructed in Step 7 is at most 7/5 times the cost of the Steiner tree edges it replaces.

Accounting for the cost of the spokes added to the Steiner tree requires that each subtree pruned from the Steiner tree is of weight at least $2k/3$. We already proved that the cost of the spokes added to the Steiner tree is at most $3/2$ times the cost of an optimal solution. Without loss of generality, the requirement that each pruned subtree is of weight at least $2k/3$ can be interpreted as that of "charging" the spoke cost incident on a subtree to at least $2k/3$ vertices.

Notice that this interpretation is valid only if the spoke connecting the subtree to the root is of minimal cost (r is connected to the closest node in the subtree).

Step 7(a) of the algorithm constructs three subtrees s_1, s_2 and s_3, each containing at least $2k/3$ vertices. This ensures that there are at least $2k/3$ vertices to which each of these subtrees can charge their spoke cost. This is not the case with Step 7(b) of the algorithm. As can be seen, subtree s_3 could be of weight less than $2k/3$. Since s_2 contains at least $2k/3$ vertices and $w(s_2)+w(s_3) \geq 4k/3$, and w is a vertex in x_j such that $|wv|$ is minimum, we can always charge the spoke costs of s_2 and s_3 to at least $4k/3$ vertices. Hence, our initial assumption that every pruned subtree is of weight at least $2k/3$ does not affect the analysis since there are at least $2k/3$ vertices for every spoke to charge.

Case 6. Analysis for Steps 8 and 9 are similar to that for Step 6 (Case 4).

Case 8. Step 11 prunes one subtree off X-tree x. The cost of the spoke $|rw|$ to connect t_α to r can be charged to all the vertices in the X-tree x as per the following argument. After disconnecting t_α from the X-tree, we are left with a subtree of $w(x) - w(t_\alpha) < k$ vertices. We do not need a new spoke for the left-over subtree as it is already connected to r using the Steiner tree edge. Hence, even for this case, our initial assumption that every pruned subtree is of weight at least $2k/3$ does not affect the analysis since there are at least $\frac{2}{3}k$ vertices to charge for the spoke added.

In all of the above cases, the cost of the edges that replace the Steiner tree edges is at most 7/5 times the cost of the Steiner tree edges that the algorithm started with. Thus, the total cost of the tree output by the algorithm is

$$\begin{aligned} C_{app} &\leq \frac{7}{5}\rho_{ST}C_{ST} + \frac{3}{2}C_{opt} \\ &\leq \left(\frac{7}{5}\rho_{ST} + \frac{3}{2}\right)C_{opt}. \end{aligned}$$

■

Corollary 3.4.3 *For the* CMST *problem in L_p plane with uniform vertex-weights, Algorithm* CMStT-UNIFORM *guarantees a* 2.9*-approximation ratio.*

3.5 The budgeted CMST problem

In this section, we study the effect of using MST as a lower bound on the CMST problem. For this purpose, we try to compare our algorithms for the CMST problem with that of Altinkemer and Gavish [3]. The approximation ratios of [3] are dominated by the MST cost, while that is not the case with our algorithms.

We consider a relaxed version of the CMST problem, the *budgeted* CMST problem, in which a compromise can be made on the capacity constraint k. In this version, for a given k, the weights of the subtrees hanging off r could be up to αk, where $\alpha \geq 1$. This gives flexibility when the cost of final solution has to be below a certain budget B, $C_{mst} < B < \beta C_{opt}$, where β is the approximation ratio if the capacity constraint were k.

To solve this version, we can use our algorithm for non-uniform vertex-weights with αk as the capacity constraint instead of k. We obtain the following theorems for this version of the problem. The proof of the following theorem is similar to the one for Theorem 3.4.1.

Theorem 3.5.1 *Consider a given* CMST *instance with capacity k. Algorithm* CMStT-NONUNIFORM *finds a* CMST, *with the weight of every subtree hanging off r being at most αk, whose cost is at most $\gamma + \frac{2}{\alpha}$ times the cost of an optimal solution to the given problem.*

Theorem 3.5.2 *Consider a given* CMST *instance in L_p plane with uniform vertex-weights, and capacity k. Algorithm* CMStT-NONUNIFORM *finds a* CMST, *with the weight of every subtree hanging off r being at most αk, whose cost is at most $1 + \frac{2}{\alpha}$ times the cost of an optimal solution to the given problem.*

Proof. Without loss of generality, we can assume that the vertices are of unit weight and scale k accordingly. Since the vertices are of unit weight, Step 4 of the Algorithm CMST-NONUNIFORM will never be executed, and thus *dummy* vertices are never introduced. This directly means that the edges within every subtree hanging off r are MST edges. The rest of the proof follows the proof of Theorem 3.4.1. ∎

Table 3.1. Comparison of our ratios with that of [3] for the budgeted CMST problem.

	Uniform vertex-weights in L_p plane		Non-uniform vertex-weights in Euclidean plane	
α	Our ratio $1+\frac{2}{\alpha}$	AG's ratio [3] $2-\frac{2}{k}+\frac{1}{\alpha}$	Our ratio $\frac{2}{\sqrt{3}}+\frac{2}{\alpha}$	AG's ratio [3] $2+\frac{2}{\alpha}$
1	3	$3-\frac{2}{k}$	3.155	4
2	2	$2.5-\frac{2}{k}$	2.155	3
4	1.5	$2.25-\frac{2}{k}$	1.655	2.5
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
∞	1	2	1.155	2

Table 3.1 compares our ratios with that of [3] for various values of α (AG denotes Altinkemer and Gavish).

3.6 Remarks

Our ratios are not tight. We believe that there is room for improvement, at least for the CMST problem with uniform vertex-weights, for which we obtain a ratio of 2.9. We used two lower bounds for an optimal CMST: (i) the MST cost and (ii) the spoke lower bound (Lemma 2.2.1). Consider Fig. 3.8, which depicts $\psi^2 k$ points in a unit-spaced grid. MST cost of the points in the grid alone is $\psi^2 k - 1$. Let k be the distance between r and the closest node in the grid. For capacity constraint k, the cost of an optimal solution is $2\psi^2 k - \psi^2$, whereas the MST cost is $(\psi^2+1)k - 1$ and the spoke lower bound is $\psi^2 k$. This shows that with the

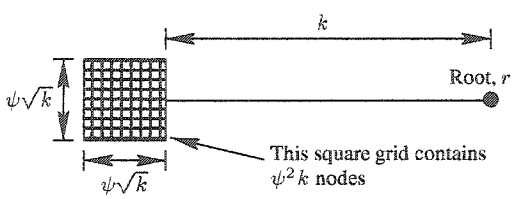


Figure 3.8. A tight example for 2-approximation ratio.

current lower bounds, one cannot get a ratio any better than 2. It should be interesting to see whether we can find a unified lower bound by combining the MST cost and the spoke cost in a some way, instead of just analyzing them separately. We do not see a reason why our of ratio of 2.9 cannot be improved to 2.

CHAPTER 4

A HEURISTIC FOR THE CAPACITATED MINIMUM SPANNING TREE PROBLEM

4.1 Introduction

Over the last four decades, numerous heuristics have been proposed to overcome the exponential time complexity of exact algorithms for the CMST problem. Almost all of the heuristics that find near-optimal solutions are based on local search techniques such as *Tabu search*, or *simulated annealing*. These heuristics start with an initial feasible solution, and improve the initial solution by local modifications of the tree in every iteration. The problem with these techniques is that the amount of improvement in each iteration could be so small that the number of iterations could possibly be as large as the optimal value [93].

Current best heuristics for the CMST problem, in terms of the quality of the solutions produced, are due to Amberg, Domschke and Voß [4], Sharaiha et al. [92], and Ahuja, Orlin and Sharma [1]. A major problem with these heuristics is that their worst-case running-times are not polynomial, i.e., they could be exponential [79, 93]. Because of their high running times, they may not be practical for solving large problem instances. The most popular and efficient heuristic for the CMST problem is due to Esau and Williams [27], with a worst-case running time of $O(n^2 \log n)$, where n is the total number of nodes. Even though there are several heuristics that claim to outperform Esau-Williams (EW) heuristic, none of them do so without substantial increase in running time.

In this chapter, we propose a new heuristic for the CMST problem, based on the EW heuristic. The worst-case running time of our heuristic is same as that of EW heuristic, i.e., $O(n^2 \log n)$. Our heuristic always outperforms or matches the EW heuristic in terms of

the quality of the solutions produced. Experimental results on benchmark instances indicate that our heuristic produces better results 67% of the time, when compared to EW heuristic. There are instances for which we obtain improvements of 17%. Our heuristic is the first to comprehensively outperform the EW heuristic with no increase in the worst-case running time.

4.2 Overview of the EW heuristic

Kruskal's algorithm [72] for constructing an MSTstarts with n subtrees, each containing just a node, and starts merging these subtrees until there is only one tree left. Merging of subtrees is based on the proximity of subtrees. In each step, two closest subtrees are merged by connected them using an edge. The final tree obtained in this manner is guaranteed to be an MST. A slightly modified version of the Kruskal's algorithm can be used to find a feasible CMST for a given instance. One of the modifications required in the Kruskal's algorithm is to connect a growing subtree to the root vertex once the sum of the weights of the vertices in the subtree reaches the capacity k. In addition, we should not to allow two subtrees whose sum of weights exceed k to merge, since it would result in a subtree of weight greater than k. If there exists no two subtrees that can be merged, then all the remaining subtrees are connected to the root using direct edges.

The EW heuristic, instead of using proximity to merge two different subtrees, introduces the notion of "savings" to merge any two subtrees. While Kruskal's algorithm merges two closest subtrees at any point in time, the EW heuristic computes the savings involved in merging any two subtrees, and connects the two subtrees that produces the maximum savings. Let `dist` be a $n \times n$ matrix, with `dist`(i, j) denoting the distance between nodes i and j. Kruskal's algorithm, in each iteration, chooses the smallest entry from `dist`, and connects the two nodes (merges the two subtrees in which the corresponding nodes are present) corresponding to that entry if no cycle is formed. EW heuristic, on the other hand, computes a

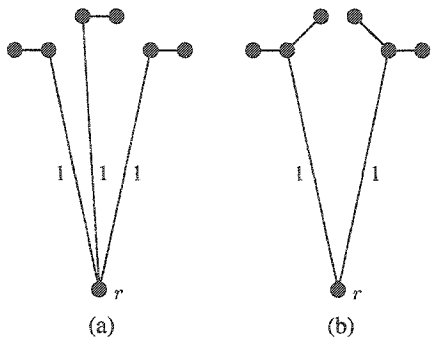


Figure 4.1. Problem instance with $k = 3$. *(a)* Solution obtained by EW heuristic with cost 3+. *(b)* Optimal solution with cost 2+.

$n \times n$ `savings` matrix using the following formula:

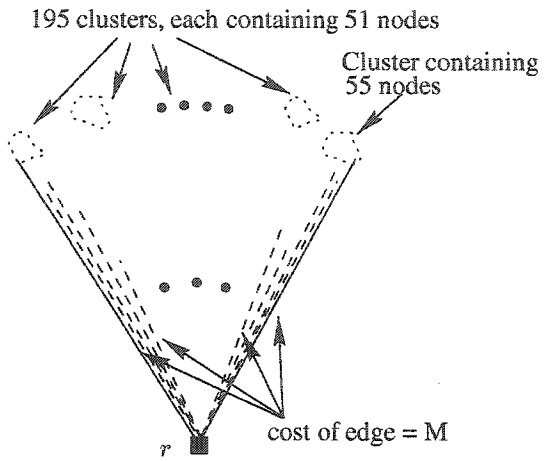
$$\texttt{savings}(i,j) = \texttt{dist}(i,j) - \texttt{dist}(i,r)$$

where r is the root vertex. A merge is attractive if the savings are negative. Subtrees i and j, which produce maximum savings (most negative value) and whose sum of vertex-weights is less than or equal to k, are merged and the `savings` matrix is recomputed for the next step.

4.2.1 Problems with Esau-Williams' heuristic

A major problem with EW heuristic is when all the subtrees formed thus far have weight just over $\frac{k}{2}$, making it impossible to merge any two of them. In this case, the EW heuristic will connect each subtree directly to the root. This could result in almost twice as many subtrees compared to an optimal solution. Fig. 4.1 depicts this pictorially for a sample scenario. While an optimal solution is of cost 2+, the solution produced by EW heuristic will be of cost 3+.

Consider Fig. 4.2 for the worst-case scenario, with 10000 unit weight nodes, a root node, and $k = 100$. Let the cost of the edge connecting the root node r to any other node be $M = 1,000,000$. Let the nodes be scattered in groups of 51 nodes with the cost of edges connecting any two nodes in the same group being equal to 1, and let the cost of edges connecting nodes from two different groups be $1 + \epsilon$, where $\epsilon = 1/M$. For this sample scenario, the EW heuristic will form subtrees of size 51 as shown in Fig. 4.3, while an optimal solution would



(a)

Figure 4.2. Problem instance with 10000 unit weight nodes and $k = 100$.

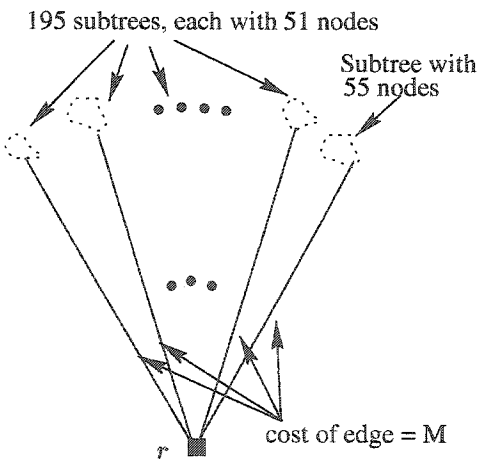


Figure 4.3. Solution obtained by EW heuristic with cost $\approx 196M$.

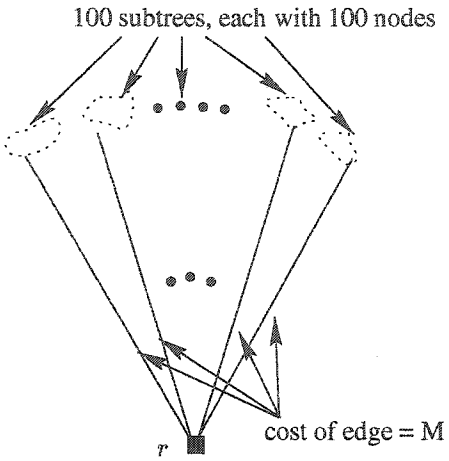


Figure 4.4. Optimal solution to problem instance in Fig. 4.2 with cost $\approx 100M$.

form subtrees of size exactly $k = 100$ connected to r as shown in Fig. 4.4. Thus, the cost of the solution obtained from EW heuristic will be approximately $M * 10000/51 \approx 196M$, while the cost of an optimal solution is roughly $100M$. Generalizing this example, one can construct examples for which the EW heuristic will produce solutions of cost almost twice as much as an optimal solution.

4.3 Our heuristic

4.3.1 Motivation

The main motivation for our heuristic arose from the fact that the EW heuristic could end up with a solution with twice the number of subtrees as compared to an optimal solution, which in turn could increase the cost of the obtained solution. Our initial idea was to reduce the number of almost-equal weight subtrees, especially subtrees of weight just over $k/2$, as this was the major cause of concern in the EW heuristic. To eliminate this problem, merging of subtrees should be done in a more controlled fashion. It could mean prioritizing the subtrees based on their weights so that, at any point during the execution of the heuristic, subtrees of greater weight have an influence in merging process. Giving a higher priority to a subtrees of greater weight will allow subtrees of greater weight to grow bigger and bigger (up to to a

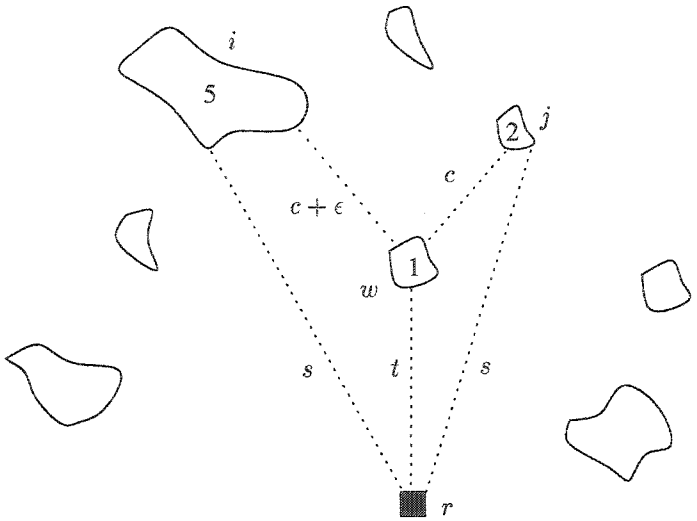


Figure 4.5. An illustration

weight of k), thereby reducing the possibility of almost-equal size subtrees with weight just over $k/2$.

We propose a new heuristic that modifies the EW heuristic and thus improve the performance. As we saw in Chapter 2, there are two kinds of costs involved in constructing a CMST. We have the cost of connecting the non-root nodes together which we call the *subtree cost*, and the cost of connecting the subtrees, of weight at most k, to the root using a "radial spoke" which we call as the *spoke cost*. Kruskal's algorithm and its variants only look at the subtree cost, while the EW heuristic tries to balance the subtree cost with the spoke cost.

We try to improve this balance achieved, by making the following observation. The EW heuristic makes a decision to add an edge (i, j) to connect two subtrees (one containing i and the other containing j) without considering the sizes of the two subtrees being merged. In reality, only TLB is independent of the subtree size. On the other hand, since the SLB is proportional to the number of nodes in the subtree, more weightage should be given to a subtree as it gets larger. In other words, when we add an edge to another subtree that takes us closer to the root, we save more when there are more nodes in this subtree than not.

The example in Fig. 4.5 illustrates this idea better. Let k be 10. Out of the several growing subtrees, let us focus on the following three subtrees: subtree i with weight 5, subtree j with weight 2, and subtree w with weight 1. Let the cost of the edge between w and j be c, and the cost of the edge between w and i be $c+\epsilon$, where $\epsilon << c$. Let the cost of the edge connecting i (or j) to root r be s, and the cost to connect w to r be $t < s$. According to the EW heuristic,

- $\texttt{savings}(i,w) = \texttt{dist}(i,w) - \texttt{dist}(i,r) = (c+\epsilon) - s$
- $\texttt{savings}(j,w) = \texttt{dist}(j,w) - \texttt{dist}(j,r) = c - s$

Since the potential savings from connecting j to w is better than that of connecting i to w, the EW heuristic will choose to connect subtrees j and w. In our heuristic, the option of connecting i to w is considered, depending on the value of a parameter κ (explained later in Section 4.3.2). Doing this could potentially prevent scenarios such as the one in Fig. 4.3. We suggest the following modification to the EW heuristic to improve its performance.

4.3.2 Modified savings equation

Let w_i denote the sum of the weights of the vertices in subtree i. We transformed the EW savings equation into the following weighted equation,

$$\texttt{savings}(i,j) = \Big(\texttt{dist}(i,j) - \texttt{dist}(i,r)\Big) \times w_i^{\kappa}$$

where κ is a user-defined constant between 0.0 and 1.0. Note that when $\kappa = 0.0$, the above equation is same the EW heuristic. The rest of the EW heuristic is left unchanged. Since computation of w_i is part of the EW heuristic, the time to compute the weighted equation is same as that of computing the EW savings equation.

Our experiments showed that it is not easy to fix *kappa*. In graphs with huge vertex weights and relatively small edge costs, fixing κ to some value could cause subtrees with more weight to completely dominate the savings value produced by the new savings equation. Similarly for graphs with huge edge costs and negligible vertex weights, fixing κ to any value

might not produce the desired result. Thus, no matter how $kappa$ is chosen, one can construct a graph for which the choice of κ is wrong.

Preliminary tests on some benchmark instances showed that best solutions were obtained most often for a fixed κ value in the range 0 to 0.25. Fig. 4.6 shows the performance plots for five benchmark instances ($tc40_*.txt, k = 1, \ldots, 5$) obtained from OR-library [12] with unit weight vertices and $k = 3$. Fig. 4.7 shows the performance plots for five benchmark instances ($cm50_*.txt, k = 1, \ldots, 5$) obtained from OR-library with non-unit vertex-weights given by $priz50.txt$ [12] and $k = 100$. After extensive testing on 105 bench mark instances (and numerous other randomly generated graphs), a clear pattern on κ value was observed. Table 4.1 shows the values of κ for which the best solutions were obtained. Every instance was tested with varying κ values. We chose κ in fixed increments of 0.05.

Despite the clear pattern on the value of κ for which the best solutions are obtained, we ran the heuristic for several different values of κ, since it increases the worst-case running time only by a constant factor a (number of different κ values). We chose a to be 21, with values of $\kappa = \{0.0, 0.05, 0.10, \ldots, 1.00\}$. Out of the 21 possible solutions obtained, our heuristic returns the best solution. Even though the running time of our heuristic is theoretically $O(n^2 \log n)$, in practice, for 21 values of κ, the running time would be 21 times than that of the EW heuristic, regardless of the value of n. If the practical running time is of paramount concern, one can always choose to run our heuristic for just 6 values of κ, say 0 to 0.25, which would make the running time to be 6 times than that of EW heuristic, but the quality of solution may be compromised. But, for any graph with arbitrary number of nodes, the worst-case running time of our heuristic is still $O(n^2 \log n)$.

Theorem 4.3.1 *Our heuristic always outperforms or matches the* EW *heuristic.*

Proof. Our heuristic with $\kappa = 0.0$ is just the EW heuristic. The fact that we run our heuristic for a different values of κ, one of which is 0.0, and output the best out of the a possible solutions completes the proof. ∎

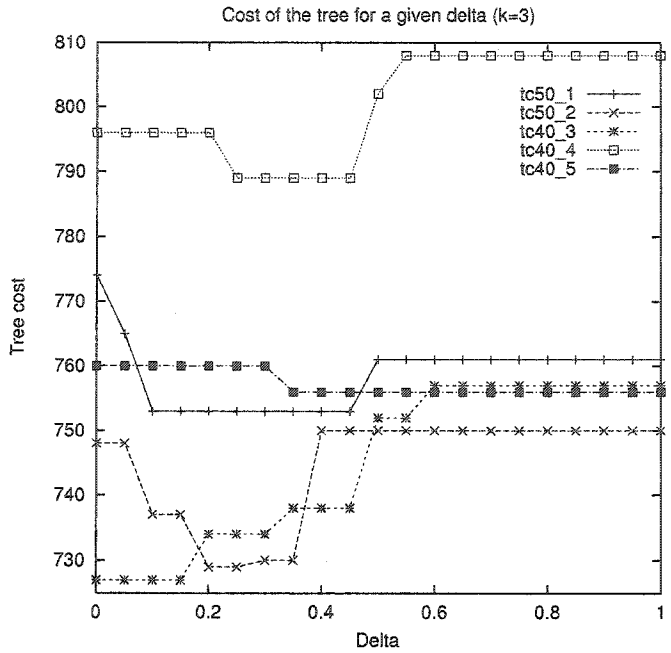


Figure 4.6. Five benchmark instances with unit vertex-weights.

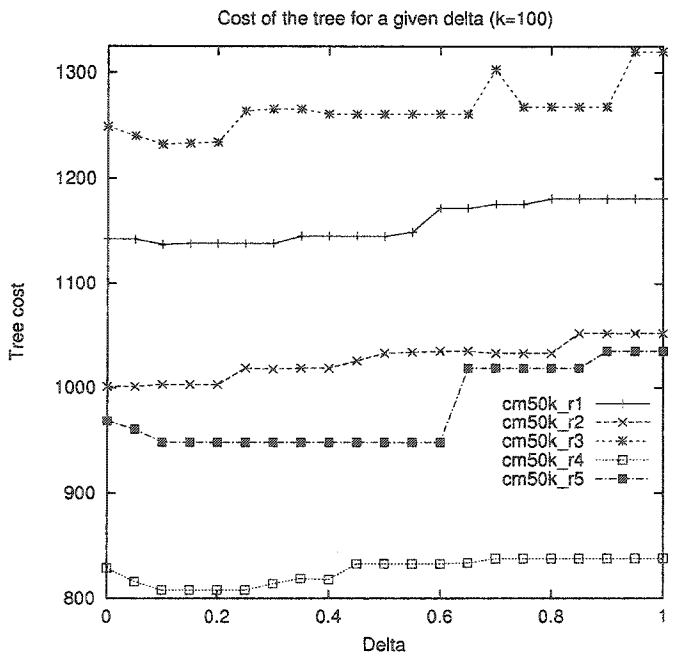


Figure 4.7. Five benchmark instances with non-unit vertex-weights.

Table 4.1. Number of best solutions obtained for different values of κ.

κ	Number of best solutions
0.00	48
0.05-0.25	72
0.30-0.50	19
0.55-0.75	9
0.80-1.00	6

4.4 Experimental results

For comparison purposes, we evaluated the performance of our heuristic on well-known benchmark instances from OR-library [12]. For a given instance, we use n to denote the number of nodes.

Our test set includes 105 benchmark instances. The test set was classified into three groups of problems. Our first two test groups are the tc and te groups (30 instances each) with unit vertex-weights. Edges in both tc and te problems satisfy triangle inequality. In tc problems, the root was placed in the center of the vertex scatter and in te problems, the root was placed in the corner of the vertex scatter. Both tc and te groups contain 10 problems each, 5 of size $n = 40$ ($tc40\_*$ and $te40\_*, * = \{1, \ldots, 5\}$) and 5 of size $n = 80$ ($tc80\_*$ and $te80\_*, * = \{1, \ldots, 5\}$). Every problem of size $n = 40$ was tested with values of $k = \{3, 5, 10\}$ and every problem of size $n = 80$ was tested with values of $k = \{5, 10, 20\}$.

Test results for tc problems (Table 4.2) indicate that our heuristic produces better solutions than the EW heuristic in 67% of the instances. On average, solutions obtained by our heuristic are off by 3.1% when compared to the known lower bounds for the respective problems, while solutions obtained by EW heuristic are off by 3.9%. As expected, test results for the tougher te problems (Table 4.3) were not as good as the ones obtained for tc problems. Our heuristic produced better solutions in only 30% of the te instances.

The third group of problems that we tested was the cm group (45 instances), with non-unit vertex-weights. Unlike tc and te problems, edges in cm problems do not satisfy triangle inequality. The cm group of problems consists of 15 problems, 5 of size $n = 50$

($cm50\_r*, * = \{1, \ldots, 5\}$), 5 of size $n = 100$ ($cm100\_r*, * = \{1, \ldots, 5\}$), and 5 of size $n = 200$ ($cm200\_r*, * = \{1, \ldots, 5\}$). Every cm problem was tested with values of $k = \{100, 200, 400\}$. Our test results for the cm problems (Table 4.4) indicate that our heuristic produces better solutions in 67% of the test instances. Our heuristic obtains improvements of up to 17% when compared to EW heuristic. On average, our heuristic obtains an improvement of 1.6% when compared to EW heuristic.

Table 4.2. Computational results for the tc problems with unit vertex-weights.

			Solutions			% Gap	
Problem	n	k	EW	JR	LB†	EW	JR
tc40_1	41	3	774	**753**	742*	4.31	1.48
tc40_2	41	3	748	**729**	717	4.32	1.67
tc40_3	41	3	727	727	716	1.54	1.54
tc40_4	41	3	796	**789**	775	2.71	1.81
tc40_5	41	3	760	**756**	741*	2.56	2.02
tc40_1	41	5	597	**595**	586*	1.88	1.54
tc40_2	41	5	588	**583**	578	1.73	0.87
tc40_3	41	5	607	607	577*	5.20	5.20
tc40_4	41	5	639	**623**	617*	3.57	0.97
tc40_5	41	5	615	615	600	2.50	2.50
tc40_1	41	10	506	506	498*	1.61	1.61
tc40_2	41	10	504	**502**	490	2.86	2.45
tc40_3	41	10	508	508	500*	1.60	1.60
tc40_4	41	10	530	530	512*	3.52	3.52
tc40_5	41	10	504	504	504*	0.00	0.00
tc80_1	81	5	1184	**1182**	1094	8.23	8.04
tc80_2	81	5	1153	1153	1090	5.78	5.78
tc80_3	81	5	1144	**1127**	1067	7.22	5.62
tc80_4	81	5	1146	**1136**	1070	7.10	6.17
tc80_5	81	5	1367	**1352**	1268	7.81	6.62
tc80_1	81	10	948	**933**	878	7.97	6.26
tc80_2	81	10	929	929	875	6.17	6.17
tc80_3	81	10	908	**904**	869	4.49	4.03
tc80_4	81	10	921	**914**	863	6.72	5.91
tc80_5	81	10	1025	1025	998	2.71	2.71
tc80_1	81	20	862	**842**	834*	3.36	0.96
tc80_2	81	20	834	834	820*	1.71	1.71
tc80_3	81	20	846	**836**	828*	2.17	0.97
tc80_4	81	20	830	830	820*	1.22	1.22
tc80_5	81	20	948	**936**	916*	3.49	2.18

* Optimum solution † Data obtained from [82]

Table 4.3. Computational results for the te problems with unit vertex-weights.

			Solutions			% Gap	
Problem	n	k	EW	JR	LB†	EW	JR
te40_1	41	3	1208	1208	1190	1.51	1.51
te40_2	41	3	1140	1140	1103	3.35	3.35
te40_3	41	3	1148	**1139**	1115*	2.96	2.15
te40_4	41	3	1153	1153	1132	1.86	1.86
te40_5	41	3	1139	**1124**	1104	3.17	1.81
te40_1	41	5	867	867	830	4.46	4.46
te40_2	41	5	822	822	792	3.79	3.79
te40_3	41	5	820	820	797	2.89	2.89
te40_4	41	5	870	**867**	814	6.88	6.51
te40_5	41	5	812	**805**	784	3.57	2.68
te40_1	41	10	639	639	596	7.21	7.21
te40_2	41	10	607	607	573	5.93	5.93
te40_3	41	10	587	587	568	3.35	3.35
te40_4	41	10	600	600	596	0.67	0.67
te40_5	41	10	593	593	572*	3.67	3.67
te80_1	81	5	2618	2618	2531	3.44	3.44
te80_2	81	5	2613	2613	2522	3.61	3.61
te80_3	81	5	2707	**2701**	2593	4.40	4.17
te80_4	81	5	2639	**2633**	2539	3.94	3.70
te80_5	81	5	2578	2578	2458	4.88	4.88
te80_1	81	10	1716	1716	1631	5.21	5.21
te80_2	81	10	1713	1713	1602	6.93	6.93
te80_3	81	10	1781	1781	1660	7.29	7.29
te80_4	81	10	1792	**1691**	1614	11.03	4.77
te80_5	81	10	1708	1708	1586	7.69	7.69
te80_1	81	20	1308	1308	1256	4.14	4.14
te80_1	81	20	1292	1292	1201	7.58	7.58
te80_1	81	20	1342	**1341**	1257	6.76	6.68
te80_1	81	20	1372	1372	1247	10.02	10.02
te80_1	81	20	1290	**1289**	1231	4.79	4.71

* Optimum solution † Data obtained from [82]

Table 4.4. Computational results for the *cm* problems with non-unit vertex-weights.

Problem	n	k	EW	JR	% Gap
cm50_r1	50	100	1142	**1137**	-0.44
cm50_r2	50	100	1001	1001	0.00
cm50_r3	50	100	1249	**1232**	-1.36
cm50_r4	50	100	829	**808**	-2.53
cm50_r5	50	100	968	**948**	-2.07
cm50_r1	50	200	717	**713**	-0.56
cm50_r2	50	200	658	658	0.00
cm50_r3	50	200	739	739	0.00
cm50_r4	50	200	571	**568**	-0.53
cm50_r5	50	200	638	638	0.00
cm50_r1	50	400	518	**514**	-0.77
cm50_r2	50	400	545	**538**	-1.28
cm50_r3	50	400	541	541	0.00
cm50_r4	50	400	491	**490**	-0.20
cm50_r5	50	400	513	**508**	-0.97
cm100_r1	100	100	700	**686**	-2.00
cm100_r2	100	100	766	**759**	-0.91
cm100_r3	100	100	749	**721**	-3.74
cm100_r4	100	100	562	**529**	-5.87
cm100_r5	100	100	562	**537**	-4.45
cm100_r1	100	200	315	**304**	-3.49
cm100_r2	100	200	335	**316**	-5.67
cm100_r3	100	200	302	**299**	-0.99
cm100_r4	100	200	278	278	0.00
cm100_r5	100	200	308	**254**	**-17.53**
cm100_r1	100	400	201	201	0.00
cm100_r2	100	400	192	**188**	-2.08
cm100_r3	100	400	183	183	0.00
cm100_r4	100	400	194	**193**	-0.52
cm100_r5	100	400	215	**205**	-4.65
cm200_r1	200	100	1332	1332	0.00
cm200_r2	200	100	1703	**1662**	-2.41
cm200_r3	200	100	1682	**1681**	-0.06
cm200_r4	200	100	1304	1304	0.00
cm200_r5	200	100	1306	**1305**	-0.08
cm200_r1	200	200	500	500	0.00
cm200_r2	200	200	621	621	0.00
cm200_r3	200	200	702	**693**	-1.28
cm200_r4	200	200	517	**504**	-2.51
cm200_r5	200	200	504	**501**	-0.60
cm200_r1	200	400	301	301	0.00
cm200_r2	200	400	329	329	0.00
cm200_r3	200	400	393	**392**	-0.25
cm200_r4	200	400	308	**304**	-1.30
cm200_r5	200	400	329	329	0.00

4.5 Remarks

We presented an efficient heuristic for the CMST problem. Our heuristic comprehensively outperforms the most popular and efficient EW heuristic, in terms of the quality of the solution obtained, with no increase in the worst-case time complexity. Our heuristic improves EW heuristic by introducing a weighted function in place of the original function, thereby reducing the number of subtrees connected to the root vertex. Even though there are other heuristic procedures that can produce better solutions, their running times are much larger. All heuristics that comprehensively outperform EW heuristic do so with an enormous increase in running time. Ours is the first $O(n^2 \log n)$ heuristic to comprehensively outperform the EW heuristic.

CHAPTER 5

SURVIVABLE NETWORK DESIGN: CAPACITATED MINIMUM SPANNING NETWORKS

5.1 Introduction

A CMST can be viewed as a collection of local access tree networks, each with a total demand of at most k, connected to the root node. Often, local access networks are prone to node/edge failures. To handle node/edge failures in local access networks, the notion of survivable networks has been studied. Survivable networks are resilient to node/edge failures. That is, a node/edge failure still allows communication between functioning sites. We call a network to be α-vertex-connected, if the failure of $\alpha-1$ vertices leaves the remaining network connected (α-edge-connectivity is defined with respect to edge failures analogously). Two-connectivity is a major feature in today's fast and reliable communication networks, since otherwise a single vertex/edge failure could cause intolerable losses. In this chapter, we consider the *Capacitated Minimum Spanning Network* (CMSN) problem, a variant of CMST. In the CMST problem, each of the local access networks be 2-edge-connected. In other words, the CMSN problem requires that the local access networks be resilient under an edge failure. The formal definition of the problem is given below.

> CMSN: Consider an undirected graph $G = (V, E)$, root $r \in V$, and capacity k. Each vertex $v \in V$ is associated with a positive number w_v representing the demand that v wishes to route to r, and each edge has a non-negative cost associated with it. The capacitated minimum spanning network problem asks for a minimum-cost spanning network such that the removal of r and its incident

edges breaks the network into a number of components (groups), each of which is 2-edge-connected with a total weight of at most k.

We also consider a variant of the CMSN problem, which we call the 2-vertex-connected CMSN problem (2VC-CMSN). The 2VC-CMSN problem asks for a minimum-cost partitioning of the set of vertices $V \backslash \{r\}$ into groups of weight at most k, with the vertices in each group along with r being 2-vertex-connected. A feasible solution to the 2VC-CMSN problem ensures that the network remains connected even after the removal (failure) of a non-root vertex.

Neither of the above problems have been studied before. But, CMST algorithms designed for graphs satisfying triangle inequality can be modified to produce feasible CMSN solutions. In particular, Altinkemer and Gavish's CMST algorithm [3] can be used to find a feasible solution that is within factor 6 of an optimal solution. The idea is to double the edges of the local access tree networks to construct tours. For the 2VC-CMSN problem, Altinkemer and Gavish's algorithm for the delivery problem [2] will guarantee a feasible solution that is within factor 4 of an optimal solution.

In this chapter, we show that the CMSN and 2VC-CMSN problems are NP-hard using a reduction from the minimum-cost 2-connected spanning subgraph problem. For the CMSN problem, we present a 4-approximation algorithm for graphs satisfying triangle inequality. For the 2VC-CMSN problem, we show that there is an algorithm with a performance ratio of 3.5 for graphs satisfying triangle inequality. We also show that the 2VC-CMSN problem is polynomial-time solvable if all vertices have unit weights and $k = 2$.

5.2 Preliminaries

Even though the input is a vertex-weighted graph, we can assume that the root vertex r has zero weight, since the cost of the CMSN does not dependent on it. Let $|uv|$ denote the cost of the edge between vertices u and v. Let OPT denote an optimal CMSN and let C_{opt} denote

its cost. Let T_v denote the subtree rooted at v in a given tree T rooted at r. Let w_v denote the weight of vertex v and let $w(T_v)$ denote the sum of the weights of vertices in T_v. We call as *spokes*, the edges incident on r in a CMSN. By *level* of a vertex, in a tree T rooted at vertex r, we mean the number of tree edges on its path to r. We can safely assume that all the vertices have integer weights. The assumption is not restrictive as any CMSN problem with rational vertex weights can be converted to an equivalent problem with integer vertex weights. The optimal solution for the scaled problem is identical to that of the original problem [3].

5.3 The capacitated minimum spanning network problem

5.3.1 Problem complexity

Given an edge weighted graph with the cost of an edge being its weight, the minimum-cost 2-edge-connected spanning subgraph (2ECSS) problem asks for a minimum-cost subgraph such that there exists at least two edge-disjoint paths between any two nodes. The 2-vertex-connected spanning subgraph (2VCSS) problem is defined analogously with respect to vertex-disjoint paths. In what follows, we show that the CMSN problem is NP-hard using a simple reduction from the 2ECSS problem.

Theorem 5.3.1 *The capacitated minimum spanning network problem is* NP*-hard.*

Proof. To prove that the CMSN problem is NP-hard, we show that the 2ECSS problem is polynomial-time reducible to the CMSN problem. Let $G = (V, E)$ be an instance of 2ECSS problem. The sum of costs of all the edges in G is given by $\Delta = \sum_{i,j \in V} |ij|$. Let $n = |V|$. Construct an instance $G' = (V', E')$ of CMSN problem, where $V' = V \cup \{r\}$ and $E' = E \cup \{(i, r) | i \in V\}$, and set the cost of each edge incident on r to be $\Delta + 1$. Also, set k to be n.

We now show that graph G has a 22ECSS of cost at most C, if and only if graph G' has a CMSN of cost at most $C + \Delta + 1$. Suppose that G has a 22ECSS S of cost at most C. We construct a feasible CMSN in G' using S by just adding an edge from r to an arbitrary

node $u \in V$. Since the extra edge has cost $\Delta + 1$, the cost of the resulting CMSN is at most $C + \Delta + 1$. Conversely, suppose that graph G' has a CMSN S^* of cost at most $C + \Delta + 1$. Observe that it has at most one edge incident to r since $C \leq \Delta$. Removing the edge incident on r from S^* will result in a feasible 22ECSS of G. Such a solution has cost at most C. ∎

Remark: Since 2ECSS problem is NP-hard even for graphs satisfying the triangle inequality, we conclude that the CMSN problem on graphs whose edges satisfy triangle inequality is NP-hard as well, because the transformed graph in Theorem 5.3.1 still obeys triangle inequality.

Theorem 5.3.2 *The 2-vertex-connected capacitated minimum spanning network problem is* NP*-hard.*

Proof. The proof is similar to that for Theorem 5.3.1 except that the reduction is from the Hamiltonian path problem. ∎

5.3.2 Lower bounds

For the CMSN problem, we use two different lower bounds on the cost of an optimal solution. The first is the cost of a MST, which is a lower bound on a CMSN because the MST is the least cost spanning network. The second lower bound is $\frac{1}{k}\sum_{v \in V} w_v|rv|$, which is presented as Lemma 2.2.1.

5.3.3 Algorithm and analysis

We now present our algorithm for constructing a CMSN of a given graph $G = (V, E)$ and root T at r. We first construct an MST, T, of G. Next, we gather nodes from T in groups such that the sum of vertex-weights in each group is in the range $[k/2, k]$. Finally, for each group of vertices, we construct a tour (cycle) spanning all vertices in that group, and connect the resulting tour to r. For easier analysis, in some case, we introduce *dummy* vertices with zero

weight, in place of real vertices, during the execution of the algorithm, which are removed from the final solution using shortcutting. The formal algorithm is given below.

Algorithm CAPMINSPANET

Input: MST T, rooted at r, and k.

While there exists a vertex $v \neq r$ such that $w(T_v) \geq k$ and level of v is maximum, do

If $w(T_v) = k$,

Remove all the edges incident on vertices in T_v.

Construct a minimum-cost tour t visiting all vertices in T_v. // #1

Add edge rw (spoke) such that $w \in t$ and $|rw| = \min_{u \in T_v}\{|ru|\}$.

Else if $\sum_{i \in \{v's\ children\}} w(T_{v_i}) < k/2$,

$w(v)$ must be greater than $k/2$ as $w(T_v) > k$.

Add edge rv (spoke), and set v as a *dummy* vertex.

Else

Initialize: $S = \emptyset$

Sort v's children, in non-decreasing order, based on the weight of the subtrees rooted at them.

Let $\{v_1, v_2, \ldots, v_p\}$ be the sorted list of v's children.

$S = S \cup \{x | x \in T_{v_p}\}$.

While $w(S) < \frac{k}{2}$, do

Choose an unprocessed child v_i of v.

$S = S \cup \{x | x \in T_{v_i}\}$.

Remove all the edges incident on vertices in S.

Construct a minimum-cost tour t visiting all vertices in S. // #2

Add edge rw (spoke) such that $w \in t$ and $|rw| = \min_{u \in S}\{|ru|\}$.

For every child v_j of r, which is not part of any tour

Remove all the edges in T_{v_j}.

Construct a minimum-cost tour t visiting all vertices in T_{v_j}. // #3

Remove *dummy* vertices from the final solution using shortcutting of the tours.

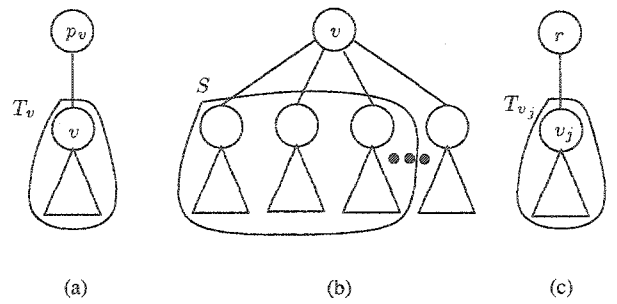


Figure 5.1. (a) Doubling the edges in T_v and shortcutting results in a tour. (b) Doubling the edges incident on the vertices in S and shortcutting results in a tour. (c) Doubling the edges in T_{v_j} and shortcutting results in a tour.

It can be verified that the above algorithm outputs a feasible CMSN for a given k.

Theorem 5.3.3 *Algorithm* CAPMINSPANET *is a 4-approximation algorithm for the* CMSN *problem.*

Proof. We prove the theorem by showing that, for any given instance, Algorithm CAPMINSPANET outputs a solution that has cost at most 4 times the cost of an optimal CMSN.

Fig. 5.1 depicts the 3 situations (#1, #2, #3) encountered in the algorithm, where minimum-cost tours are constructed. Since the edges of the graph obey triangle inequality, it can be seen that a tour—covering all the vertices in the concerned set—can be constructed by just doubling the necessary MST edges. This tour construction idea is possible as the vertices in the concerned set are together in MST. Remember that the *dummy* vertices, which serve only as place-holders, are introduced to ensure that the underlying MST edges are still available for doubling. The dummy vertices are removed from the final solution using shortcutting.

Except for those tours constructed in the last "for" loop of the algorithm, every tour constructed by the algorithm contains vertices whose weights add up to at least $k/2$. Thus, for every tour the algorithm adds a new spoke, it is guaranteed that the sum of the vertex-weights in that tour is at least $k/2$. Notice that every spoke added to T in this manner connects r to the closest vertex in the tour. Let $t_1, t_2, \ldots, t_m$ be the set of tours constructed by Algorithm

CAPMINSPANET for which a new spoke was added. Let Γ be the set of vertices in tours $t_1, \ldots, t_m$. Let t_i be one such tour. Let w_i be the vertex in t_i that is connected to r. Among the vertices in t_i, since w_i is the closest to r,

$$|rw_i| \leq \frac{\sum_{v \in t_i} w_v |rv|}{k/2}.$$

Thus, for m tours for which the new spokes were added,

$$\begin{aligned} C_{spokes} &= \sum_{i=1}^{m} |rw_i| \\ &\leq \frac{\sum_{v \in \Gamma} w_v |rv|}{k/2} \\ &\leq 2\frac{\sum_{v \in V} w_v |rv|}{k} \qquad \text{(since } \Gamma \subset V\text{)} \\ &\leq 2C_{opt} \qquad \text{(by Lemma 2.2.1).} \end{aligned}$$

Since the cost of the local access networks (tours) is at most twice the cost of the MST, and the cost of the new spokes that were added is at most twice the optimal CMSN, we conclude that Algorithm CAPMINSPANET outputs a solution of cost at most 4 times the cost of an optimal CMSN. ∎

5.4 The 2-vertex-connected capacitated minimum spanning network problem

For a given graph $G = (V, E)$, root $r \in V$, and a capacity k, the 2VC-CMSN problem asks for a minimum-cost partitioning of the set of vertices $V \backslash \{r\}$ into groups of weight at most k, with the vertices in each group along with r being 2-vertex-connected. Notice that a feasible solution to the 2VC-CMSN problem ensures that the network remains connected even after the removal (failure) of a non-root vertex. For the 2VC-CMSN problem, we show that Altinkemer and Gavish's algorithm for the delivery problem [2] finds a feasible solution that is within a factor 3.5 of an optimal solution.

5.4.1 Delivery problem

Given a set of customers (nodes), each having a positive demand, a depot (root node) s, and vehicles of capacity q, the delivery problem asks for routes for the vehicles such that the vehicles depart from s, serve a set of customers following their designated routes and return to s. The objective is to minimize the total length (cost) of the routes without violating the the capacity limit q and visiting each customer exactly once. Altinkemer and Gavish [2] presented an algorithm for this problem by partitioning a traveling salesman (TSP) tour. Their algorithm finds a feasible solution of cost at most $2 + \rho_{TSP}$ times the cost of an optimal solution, where ρ_{TSP} is the best achievable approximation ratio for TSP. Their idea is to construct a TSP tour visiting all the customers, partition the tour into paths with the sum of customer demands in each path being at most q, and connect the two ends of each path to s. Using the fact that an optimal TSP tour is a lower bound on an optimal solution for the delivery problem, they were able to show that the cost of the solution produced by their algorithm is at most $2 + \rho_{TSP}$ times the optimal cost (they show that the cost to connect the paths to s is at most 2 times the optimal cost).

Notice that the solution returned by the delivery algorithm for capacity $q = k$ with depot $s = r$ is a feasible solution for the 2VC-CMSN problem. Since an optimal TSP tour is not a lower bound for the 2VC-CMSN problem, their approximation ratio of $2 + \rho_{TSP}$ does not apply to the 2VC-CMSN problem. Rather, one can bound the cost of the approximate TSP tour to be at most twice the cost of an MST. Since MST cost is a lower bound for 2VC-CMSN, Altinkemer and Gavish's approach yields a ratio of four for the 2VC-CMSN problem.

5.4.2 Approximation analysis

Notice that an optimal 2VCSS cost is a lower bound for the 2VC-CMSN problem, because 2VC-CMSN problem is a generalization of the 2VCSS problem (2VCSS problem is the 2VC-CMSN problem with $k = \infty$). Suppose that Altinkemer and Gavish's algorithm for the

delivery problem uses Christofides' algorithm for TSP [23] to construct the initial tour. In this approach, we start with an MST T of the graph. A minimum-weight perfect matching of the odd-degree nodes of T in an arbitrary graph is then computed and added to T. The resulting Eulerian graph can then be converted into a TSP tour using shortcutting. Frederickson and JáJá [33] showed that the matching found by Christofides' algorithm is no more than half the cost of an optimal 2VCSS. Since MST cost is a lower bound for the 2VCSS problem, the cost of the Christofides' TSP tour is at most 1.5 times the cost of an optimal 2VCSS [33]. Since an optimal 2VCSS is a lower bound for the 2VC-CMSN problem, Altinkemer and Gavish's algorithm actually guarantees a ratio of 3.5 for the 2VC-CMSN problem.

Theorem 5.4.1 *Altinkemer and Gavish's algorithm for the delivery problem guarantees an approximation ratio of 3.5 for the* 2VC-CMSN *problem.*

5.4.3 Unit vertex-weights with $k = 2$

We now show that the special case when all vertices have unit weights and $k = 2$ is polynomial-time solvable using minimum-cost matching. Let $G = (V, E)$ be the given graph. Let $|uv|_G$ be the cost of the edge connecting vertices u and v in G. Let r be the root vertex in G. Construct a graph $G' = (V', E') = (V_1 \cup V_2, E_1 \cup E_2 \cup E_3)$ as follows:

- Set $V_1 = \{v' | v \in V\}$.
- Introduce an edge, into E_1, of cost $|xy|_G + |rx|_G + |ry|_G$ between vertices x' and y' in V_1, if $x' \neq y'$.
- Set $V_2 = \{v'' | v \in V\}$.
- Introduce an edge, into E_2, of cost zero between vertices x'' and y'' in V_2, if $x'' \neq y''$.
- for each $v \in V$, introduce an edge of cost $|rv|_G$, into E_3, between vertices v' and v'' in G' (v' and v'' correspond to the same vertex $v \in V$).

Lemma 5.4.1 *The cost of a minimum-cost perfect matching of the vertices in G' is equal to the optimal cost of a* 2VC-CMSN *in G.*

Proof. It suffices to prove the following:

1. Given a 2VC-CMSN of cost ω in G, there exists a perfect matching of cost ω in G'.

   If a local access network in the given 2VC-CMSN contains 2 vertices x and y, then match the corresponding x and y, x' and y', belonging to V_1 in G'. If a local access network in the given 2VC-CMSN contains just one vertex x, then match $x' \in V_1$ and $x'' \in V_2$. By doing this, all vertices belonging to V_1 in G' will be matched while there may be some vertices in V_2 that are not matched. Since all vertices in V_1 are matched, an even number of vertices in V_2 will be left unmatched, and they can be paired-up arbitrarily. Since the cost of matching $x \in V_2$ and $y \in V_2$ is zero, the final cost of the perfect matching would be the same as the cost of the 2VC-CMSN.

2. Given a matching of cost ω in G', there exists a 2VC-CMSN of cost ω in G.

   Let $n = |V|$. Thus, $|V'| = 2n$. Let (x, y) denote a matching between vertices x and y. If $x, y \in V_1$, connect the corresponding two vertices in G with r to form a cycle. If $x \in V_1$ and $y \in V_2$, connect the corresponding vertex in V (both x and y will correspond to the same vertex in V) to r.

■

Theorem 5.4.2 *The* 2VC-CMSN *problem with unit vertex-weights and* $k = 2$ *is polynomial-time solvable.*

CHAPTER 6
THE SINGLE-SINK BUY-AT-BULK NETWORK DESIGN PROBLEM

6.1 Introduction

Consider a given undirected graph $G = (V, E)$ with non-negative edge costs, a root node $r \in V$, and a set $D \subseteq V$ of *demands* with $w(v)$ representing the units of flow that demand $v \in D$ wishes to send to the root. We are also given K types of cables, each with a specified capacity and a cost per unit length. The cost per unit capacity per unit length of a high-capacity cable is typically less than that of a low-capacity cable, reflecting "economy of scale". In other words, it is cheaper to buy a cable of larger capacity than many cables (adding up to same capacity) of smaller capacity. The extensively studied *single-sink buy-at-bulk* (SSBB) problem, also known as the *single sink edge Installation* problem, asks for a low-cost installation of cables along the edges of G, such that the demands can simultaneously send their flows to sink/root r, under the restriction that the flow from a node must follow a single path to the sink (indivisibility constraint). We are allowed to install zero or more copies of a cable type on each edge. By *divisible* SSBB (DSSBB) problem, we refer to the version of the SSBB problem without the indivisibility constraint.

The SSBB problem has applications in the hierarchical design of telecommunication networks, in which the traffic from a source must follow a single path to the sink. The DSSBB problem has its own applications: e.g., routing oil from several oil wells to a major refinery [89].

The *buy-at-bulk* network design problem was introduced by Salman, Cheriyan, Ravi and Subramanian [89]. They showed that the problem is NP-hard, by showing a simple reduction from the Steiner tree problem or the knapsack problem. The problem remains NP-hard

even when only one cable type is available. They also presented a $O(\log n)$-approximation algorithm for the SSBB problem in Euclidean graphs. For problem instances in general metric spaces, Awerbuch and Azar [10] presented a $O(\log^2 n)$-approximation algorithm. Their algorithm works even for multi-root version of the problem. Bartal's tree embeddings [11] can be used to improve their ratio to $O(\log n \log\log n)$. Garg et al. [37] presented an $O(K)$-approximation algorithm based on LP-rounding. Guha, Meyerson and Munagala [48] presented the first constant-factor approximation algorithm; their ratio was estimated to be around 2000 by Talwar [96]. In the same paper, Talwar presented an LP-based rounding algorithm with an improved ratio of 216.

Recently, Gupta, Kumar and Roughgarden [49] presented an elegant 72.8-approximation algorithm for the SSBB problem. But unfortunately, their approach does not guarantee that the flow from a node follow a single path to the sink. In other words, their ratio of 72.8 holds for the DSSBB problem, but not for the SSBB problem. That leaves Talwar's ratio of 216 as the current best for the SSBB problem.

In this chapter, we design a 145.6-approximation algorithm for the SSBB problem, using ideas from Gupta, Kumar and Roughgarden [49], but guaranteeing the indivisibility constraint is not straightforward. We introduce a new "redistribution" procedure which is pivotal in guaranteeing that the flow from a source follows a single path to the sink. We also propose a modification to their DSSBB algorithm that reduces the ratio from 72.8 to α_K, where α_K is less than 65.49 for all K (the number of cable types). In particular, $\alpha_2 < 12.7, \alpha_3 < 18.2, \alpha_4 < 23.8, \alpha_5 < 29.3, \alpha_6 < 33.9$.

6.2 Preliminaries

Let $G = (V, E)$ be the input graph with $D \subseteq V$ being the set of demands. We use the term "demand" to denote a vertex or the flow out of it. Let c_e denote the length of edge e. We also use c_{xy} to denote the length of an edge connecting nodes x and y. We assume that the metric completion of the given graph is available. Let u_i and σ_i denote the capacity and cost per

unit length of cable type i. We define $\delta_i = \sigma_i/u_i$ to be the "incremental cost" of using cable type i. The value of δ_i can also be interpreted as the cost per unit capacity per unit length of cable type i. Let $\epsilon > 0$ be a constant, which we will chose later. Let us assume that each u_i and σ_i (and by definition δ_i) is a power of $1+\epsilon$, which can be enforced by rounding each capacity u_i down to the nearest power of $1+\epsilon$, and each σ_i up to the nearest power of $1+\epsilon$. This assumption is not without loss of generality, and can be accounted by losing a factor of $(1+\epsilon)^2$ in the approximation ratio. This idea of rounding is derived from [49], where they used powers of 2 (i.e., $\epsilon = 1$).

The following properties on the costs and capacities of cable types are known [48, 49]. Without loss of generality, assume that the cables are ordered such that $u_i < u_j$ and $\sigma_i < \sigma_j$ for all $i < j$. Note that if $u_i \leq u_j$ and $\sigma_i \geq \sigma_j$, then we can eliminate cable type i from consideration. We can also assume that $u_1 = \sigma_1 = 1$, since it can be obtained by appropriate scaling, though it may leave non-integer weights at vertices. For each $j < k$,

$$\frac{\sigma_k}{u_k} < \frac{\sigma_j}{u_j}. \tag{6.1}$$

Otherwise, cable type k can be replaced by u_k/u_j copies of cable type j without increase in cost. The fact that $\delta_j = \sigma_j/u_j$ is a power of $1+\epsilon$ implies that $\delta_{j+1} \leq \delta_j/(1+\epsilon)$ for all j, since $u_{j+1} \geq (1+\epsilon)u_j$. Let $g_k = \frac{\sigma_{k+1}}{\sigma_k}u_k$. By equation (6.1),

$$1 = u_1 < g_1 < u_2 < g_2 < \cdots < u_k < g_k = \infty.$$

Since σ_i is a power of $1+\epsilon$ for any i, and $\sigma_{j+1} > \sigma_j$, using equation (6.1) we get,

$$\frac{u_{j+1}}{u_j} > 1+\epsilon.$$

Let OPT denote an optimal solution with cost $C^* = \sum_j C^*(j)$, where $C^*(j)$ is the amount paid for cable type j in OPT. We state the following lemma and its proof from [49], as its understanding is crucial for understanding of our algorithms.

Lemma 6.2.1 (Redistribution Lemma [49]) *Let T be a tree rooted at r with each edge having capacity U. For each vertex $j \in T$, let $w(j) < U$ be the weight located at j with $\sum_j w(j)$ a multiple of U. Then there is an efficiently computable (random) flow on the tree that redistributes weights without violating edge capacities, so that each vertex receives a new weight $w'(j)$ that is either 0 or U. Moreover,*

$$\mathbf{Pr}\ [\ w'(j) = U\ ] = w(j)/U$$

Proof. Replace each edge in T with two oppositely directed arcs. Let Y be a value chosen uniformly at random from $(0, U]$. Take an Euler tour of the vertices in T, starting from r and visiting all the other vertices $\{j_1, j_2, \ldots, j_m\}$ in T. Let a counter Q be set to 0 initially. On visiting vertex j_k, we update $Q \leftarrow Q + w(j_k)$. Also, let Q_{old} and Q_{new} be the value of v just before and after visiting j_k, respectively. On visiting j_k, if $xU + Y \in (Q_{old}, Q_{new}]$ for some integer x, then "mark" j_k and ask that it send $Q_{new} - (xU + Y)$ weight to the next marked vertex lying clockwise on the tour. Otherwise, we ask that j_k send all its weight to the next marked vertex lying clockwise on the tour. This construction ensures that the maximum flow on an directed edge is at most U, and that the probability that a vertex j gets marked is $w(j)/U$, which is exactly the probability that j receives a weight of U.

Since we were working on a directed tour, the cost of this redistribution is at most twice the cost of the tree T, as an edge in T was replaced by two oppositely directed arcs. But, using simple flow-canceling argument, one can show that one copy of the edges in T is sufficient for such a redistribution. ∎

6.3 Algorithms and their analyses

We first show how to modify the algorithm of Gupta, Kumar and Roughgarden [49] to obtain an approximation ratio of α_K, where α_K is less than 65.49 for all K. Recall that their algorithm solves the DSSBB problem, and not the SSBB problem. We then present an approximation algorithm for the SSBB problem that achieves an approximation ratio of 145.6 in Section 6.3.2.

6.3.1 The *divisible* single-sink buy-at-bulk problem

The vertices of the graph $G = (V, E)$ may have non-integral weights as demands, because of the scaling done to make $u_1 = \sigma_1 = 1$. Since the flow is divisible, there is no loss of generality in assuming that $d_j \leq 1$, because a demand greater than 1 can be split into multiple demands by splitting a vertex into $\lceil d_j \rceil$ vertices. The algorithm is simpler with this assumption, and easily adapts to higher demands by adjusting the probabilities without actually splitting vertices.

Construct a ρ-approximate Steiner tree T_0, using cables with capacity $u_1 = 1$, spanning all the demands in D. Redistribute the demands using the construction in the proof of Lemma 6.2.1, with $U = 1$, and collect *integral* demands at some subset of vertices in D. The cost incurred to do this redistribution is just the cost of the Steiner tree [49], and since the optimal solution contains a candidate Steiner tree, we incur a cost of at most $\rho \times \sum_j C^*(j)/\sigma_j$. We can assume that the number of demands $|D|$ is a power of $1 + \epsilon$, as this can be achieved by placing *dummy* demands at the root vertex r.

The algorithm given below closely follows the incremental design of Gupta et al.'s algorithm [49] to build the network. The algorithm proceeds in stages using only cable types t and $t + 1$ in stage t, except for the last stage ($t = K$) in which only cables of type K are used.

At the beginning of the first stage, $D_1 = D$ with each demand $j \in D$ having weight $d_j = 1 = u_1$. In general, at the beginning of stage t, D_t is a set of $|D|/u_t$ vertices, each with demand u_t. During stage t, our algorithm (presented below) uses u_{t+1} as the "aggregation threshold" to combine several demands of weight u_t into a single demand of weight u_{t+1}. Unlike [49], where capacities are powers of 2 which ensures that u_{t+1} is an integral multiple of of u_t, in our algorithm u_{t+1} is not necessarily an integral multiple of u_t. As a result, during the aggregation process our algorithm may have to combine demands of weight u_t from, say, 1.33 vertices to obtain a demand of weight u_{t+1}. The cables required to perform such an aggregation are bought by the algorithm. The demand will reach the root at the end of the

algorithm. The final solution is then given by the union of all the paths used in the aggregation stages.

Given below are the steps performed at stage t of the algorithm. Its first three steps are exactly the same as in [49]. Whenever we mention a fraction of a vertex, we mean a fraction of the demand from that vertex.

D1. *Mark* each demand in D_t with probability $p_t = u_t/g_t$, and let D_t^* be the marked demands.

D2. Construct a ρ-approximate Steiner tree T_t on $F_t = D_t^* \cup \{r\}$. Install a cable of type $t+1$ on each edge of this tree.

D3. For each vertex $j \in D_t$, send its weight u_t to the nearest member of F_t using cables of type t. Let $w_t(i)$ be the weight collected at $i \in F_t$. All vertices that sent their demands to the same vertex i are said to belong to i's *family*, which we call as G_i.

D4. A vertex $i \in F_t$ collects the demands sent to it by all vertices in its family, G_i, divides it into groups of size u_{t+1}. Each member of G_i may partition its flow and contribute to at most two groups. Flow from a group g is sent back to a random vertex of g by buying a new cable of type $t+1$. If the whole vertex belongs to g, then the probability that that vertex receives back a weight of u_{t+1} is u_t/u_{t+1}. But if only a fractional part f of a vertex demand belongs to g, then the probability that that vertex receives back a weight of u_{t+1} is fu_t/u_{t+1}. Some *residual* demand may be left over at i which will be aggregated into demands of u_{t+1} using redistribution in the next step. Let the number of residual vertices at i be b_i.

D5. After rerouting the collected weight back from i to vertices in D_t in the above step for all $i \in F_t$, we aggregate the weights from residual vertices into groups of weight exactly u_{t+1} using Lemma 6.2.1 with $T = T_t, w_t(i) = b_i u_t$ and $U = u_{t+1}$. For every $i \in F_t$ that receives u_{t+1} weight as a result of this aggregation process, send the weight

back from i to one of i's b_i residual vertices, chosen uniformly at random, using newly bought cable of type $t+1$. If the whole vertex is a residual vertex, then the probability that that residual vertex receives back the weight of u_{t+1} is $1/b_i$. But if only a fractional part f of a vertex demand is residual, then the probability that that vertex receives the weight of u_{t+1} is f/b_i. In this scheme, a vertex j may receive back a weight of $2u_{t+1}$ in stage t as a result of it being in two groups, which can be viewed as duplicating the vertex.

At stage t, since the u_{t+1} demands from $i \in F_t$ for all i are returned back to a subset of vertices in D_t, $D_{t+1} \subseteq D_t$ for all t. When $t = K$, we set $p_K = 0$. Hence, in the Kth stage of the algorithm none of the demands are marked, and thus the weights of all vertices in D_K are sent directly to root r using cables of type K. We use the following lemmas to analyze our algorithm.

The lemma below appears as Lemma 4.2 in [49]. But its proof in [49] is not directly applicable to our algorithm, because the value of u_{t+1} in our algorithm is not necessarily an integral multiple of the value of u_t, for all t.

Lemma 6.3.1 *For every non-root vertex $j \in D$ and stage t*

$$\mathbf{Pr}[j \in D_t] = 1/u_t.$$

Proof. We prove the lemma by induction on t. The lemma is clearly true for the base case, $t = 1$, since $u_t = 1$. Suppose the lemma is true for stage t. We will show that it is true for stage $t+1$. In stage t, let $j \in D_t$ send its weight to $i \in F_t$. Vertex j must satisfy one of the following conditions: (i) j belongs to just one group, (ii) j belongs to two different groups, (iii) j is fully a residual vertex, and (iv) part of j belongs to a group while the rest of j is residual. Recall that a vertex can belong to at most 2 groups.

In case (i), the probability that j receives back the group weight of u_{t+1} is u_t/u_{t+1}. In case (ii), let a fraction f of j belong to group g_1 and the rest of j belong to group g_2. The

probability that j receives back g_1's weight of u_{t+1} is fu_t/u_{t+1}, while the probability that j receives back g_2's weight of u_{t+1} is $(1-f)u_t/u_{t+1}$. Overall, the probability that j receives back a weight of u_{t+1} is u_t/u_{t+1}. In case (iii), the probability that i is assigned the weight of u_{t+1} is $b_i u_t/u_{t+1}$, and the probability that j receives this weight from i is $1/b_i$, thus making the overall probability that j receives a weight of u_{t+1} to be u_t/u_{t+1}. By a similar argument, it is clear that the probability that j receives a weight of u_{t+1} in case (iv) is u_t/u_{t+1}. Thus, we conclude that

$$\begin{aligned}
\mathbf{Pr}\ [\, j \in D_{t+1}\,] &= \mathbf{Pr}\,[\, j \in D_{t+1} \mid j \in D_t\,] \cdot \mathbf{Pr}\,[\, j \in D_t\,] \\
&= (u_t/u_{t+1})(1/u_t) \\
&= 1/u_{t+1}.
\end{aligned}$$

∎

The following lemma, proved by Gupta, Kumar and Roughgarden [49], applies to our algorithm as well. The proof involves taking all cables of higher capacity used by an optimal solution, and then extending it using randomization to span F_t, and showing that this solution has low expected cost.

Lemma 6.3.2 ([49]) *Let T_t^* be the optimal Steiner tree on F_t, and $c(T_t^*) = \sum_{e \in T_t^*} c_e$. Then*

$$\mathbf{E}\,[\, c(T_t^*)\,] \leq \sum_{s>t} \frac{1}{\sigma_s} C^*(s) + \sum_{s \leq t} \frac{1}{\delta_s . g_t} C^*(s). \tag{6.2}$$

Lemma 6.3.3 *The expected cost incurred in stage t is at most $(2+\rho+\frac{2}{1+\epsilon})$ times $\sigma_{t+1}\mathbf{E}[c(T_t^*)]$, where T_t^* is the optimal Steiner tree on F_t.*

The proof of the above lemma is given as Lemma 4.4 in [49] with $\epsilon = 1$. Cost incurred during stage t is accounted for as follows: (i) the cost of the cables to construct the Steiner tree in Step D2 is at most $\rho\sigma_{t+1}c(T_t^*)$, (ii) the cost of the cables used in Step D3 is at most

$2\sigma_{t+1}c(T_t^*)$, and (iii) the cost incurred in Steps D4 and D5 to reroute the demands back to random vertices in D_t is at most

$$\left(\frac{\delta_{t+1}}{\delta_t}\right).2\sigma_{t+1}c(T_t^*) \leq \left(\frac{1}{1+\epsilon}\right).2\sigma_{t+1}c(T_t^*).$$

Theorem 6.3.1 *The approximation ratio α_K of our* DSSBB *algorithm is at most 65.49.*

Proof. Recall that by rounding the costs and capacities of cables to powers of $(1+\epsilon)$, we lost a factor of $(1+\epsilon)^2$ in the approximation ratio. We incurred a cost of

$$C_{T_0} \leq \rho \sum_j C^*(j)/\sigma_j$$

for the construction of Steiner tree T_0 to ensure integral demands at vertices. The total cost C_s incurred during the K stages of the algorithm can be obtained by substituting equation (6.2) in Lemma 6.3.3 and summing over all t, as shown below.

$$C_s \leq \left(2+\rho+\frac{2}{1+\epsilon}\right).\left(\sum_{t\geq 1}\frac{\delta_t}{\delta_1}C^*(1) + \sum_{s=2}^{K}\left(\sum_{t=1}^{s-1}\frac{\sigma_{t+1}}{\sigma_s} + \sum_{t\geq s}\frac{\delta_t}{\delta_s}\right)C^*(s)\right).$$

Let $\Delta = (1+\epsilon)^2$. The cost of the final solution is given by

$$\begin{aligned} C &= \Delta(C_{T_0} + C_s) \\ &< \Delta\left(2+\rho+\frac{2}{1+\epsilon}\right)\left(\left(\frac{1}{\sigma_1} + \sum_{t\geq 1}\frac{\delta_t}{\delta_1}\right)C^*(1) + \sum_{s=2}^{K}\left(\sum_{t=0}^{s-1}\frac{\sigma_{t+1}}{\sigma_s} + \sum_{t\geq s}\frac{\delta_t}{\delta_s}\right)C^*(s)\right). \end{aligned}$$

Since σ_t and δ_t are powers of $1+\epsilon$, the summations are upper bounded by $1+1/\epsilon$. This simplifies the above equation to

$$C < (1+\epsilon)^2\left(2+\rho+\frac{2}{1+\epsilon}\right) \times 2\left(1+\frac{1}{\epsilon}\right)\sum_s C^*(s), \tag{6.3}$$

which when optimized for ϵ gives a ratio of 65.4899 for $\epsilon \approx 0.585735$. Here we are using the current best approximation algorithm for finding a Steiner tree, which guarantees an approximation ratio of $\rho = 1 + \ln(3)/2$ [88]. ∎

Corollary 6.3.1 *For a fixed K, $\alpha_2 < 12.7, \alpha_3 < 18.2, \alpha_4 < 23.8, \alpha_5 < 29.3$, $\alpha_6 < 33.9$ and so on.*

Proof. The cost of the final solution $C = (1+\epsilon)^2(C_{T_0} + C_s)$ can be rewritten as

$$C \leq (1+\epsilon)^2 \left[\rho \sum_{s=1}^{K} \frac{1}{\sigma_s} C^*(s) + \left(2 + \rho + \frac{2}{1+\epsilon}\right) \cdot \left(\sum_{t \geq 1} \frac{\delta_t}{\delta_1} C^*(1) + \sum_{s=2}^{K} \left(\sum_{t=1}^{s-1} \frac{\sigma_{t+1}}{\sigma_s} + \sum_{t \geq s} \frac{\delta_t}{\delta_s} \right) C^*(s) \right) \right].$$

For a fixed K, there exists an $\epsilon > 0$ for which the corollary can be mathematically verified. ∎

6.3.2 The single-sink buy-at-bulk problem

During the preprocessing step, Gupta et al.'s algorithm [49] and our DSSBB algorithm in Section 6.3.1 use redistribution on T_0 to guarantee integral demands at the vertices. Later, vertices of integral demands are duplicated so that the demands at vertices are unit-weight. Because of this redistribution and duplication, there is no guarantee that the demand from a vertex in the input graph travels along a single path to the sink, because the demand at a vertex may have been split during the redistribution and/or duplication process. In our algorithm below, we make sure that demand at a vertex follows a single path to the sink. Like [49], we set $\epsilon = 1$, which makes u_i and σ_i (and by definition δ_i) powers of 2. This gives us the flexibility of generating u_{i+1} weighted nodes from integral number of u_i weighted nodes, thereby eliminating the need to split demands.

In what follows, we present a sequence of lemmas, which help in guaranteeing the indivisibility constraint. Recall that Lemma 6.2.1 redistributes the weights uniformly at random and the probability that a vertex receives a weight of U is proportional to its weight.

Lemma 6.3.4 *Either there exists at least one arc with zero flow in the directed tour t constructed in procedure of of Lemma 6.2.1, or there exists a redistribution (using Lemma 6.2.1), with zero flow on at least one arc of the directed tour, which produces the same assignment of weights.*

Proof. The proof is complete if the first part of the lemma were true. Suppose it were not true. Let t be the directed tour in the procedure of Lemma 6.2.1, which was used to redistribute the

weights. Let $m > 0$ be the smallest flow across a directed edge in t. Note that $m \leq U$. For each directed edge in t, subtract m from the flow on that edge. After this, we are guaranteed that at least one edge in t has a zero flow. Since this post-processing does not alter the distribution of weights, the proof is complete. ∎

Lemma 6.3.5 *There exists a redistribution using the procedure of Lemma 6.2.1 with $Y = U$, which produces the same assignment of weights as that with Y that is chosen uniformly at random from $(0, U]$.*

Proof. Let t be the directed tour in the proof of Lemma 6.2.1. As per Lemma 6.3.4, there exists at least one edge in t with zero flow. Let $e = (p, q)$ be an edge in t with zero flow. Without loss of generality, we can assume that $p \in D$. As per the procedure in the proof of Lemma 6.2.1, p must be one of the vertices that must have been marked. Since the flow on e is zero, it must be that Q_{new} at p is equal to $xU + Y$ for some integer x, which means that vertex g marked just after p must either have $(x+1)U + Y \in (Q_{old}$ at g , Q_{new} at $g]$ or $Y \in (Q_{old}$ at g, Q_{new} at $g]$. This means that Q_{new} at g is at least U greater than Q_{new} at p.

Recall from the proof of Lemma 6.2.1 that the vertices in t are visited starting from r. We now show that a procedure with $Y = U$ on t, visiting vertices starting from q (instead of r), produces the same assignment of weights as that with Y that is chosen uniformly at random from $(0, U]$. From the above discussion, since Q_{new} at g is at least U greater than Q_{new} at p, and the flow on e is zero, it can be seen that the procedure with $Y = U$ on t when we visit vertices starting at q produces the same outcome as what is desired, i.e., the set of vertices that were assigned a weight of U will be the same as that marked in the proof of Lemma 6.2.1. ∎

The following lemma differs from Lemma 6.2.1 in the following two aspects: (i) weights of vertices in T are powers of 2, and (ii) demand from a vertex is not split.

Lemma 6.3.6 *Let T be a tree with each edge having capacity U, a power of two. For all v in tree T, let $w(v)$ be a power of 2 with $w(v) < U$. Then there is an efficiently computable flow*

on T that redistributes the weights, respecting cable capacities and without splitting vertex-weights, so that each vertex receives a new weight $w'(j)$ that is either 0 or U. Moreover,

$$\mathbf{Pr}\ [\ w'(j) = U\ ] = w(j)/U$$

Proof. Using the argument in Lemma 6.3.5, we find a starting vertex from which we start visiting the vertices in the directed tour (obtained by replacing each edge in T with two oppositely directed arcs) in clockwise direction with $Y = U$. The value of Q is set to 0 initially. Increment Q by $w(j)$ on visiting vertex j. Let Q_{old} and Q_{new} be the value of Q just before and after visiting a vertex, respectively. Also, maintain set Z which is initially empty. Add v_j to Z on visiting vertex v_j. On visiting j, if for some integer x, $xU \in (Q_{old}, Q_{new}]$, then we do the following: (i) we find $W \subseteq Z$ such that $Q_{new} - xU = \sum_{i \in W} w(i)$, and (ii) ask the vertices in $Z \backslash W$ to send their weights to j and remove them from Z.

We now show how to find $W \subseteq Z$. Let g be the first vertex at which $Q_{new} \geq U$. The proof of Lemma 6.3.5 would have marked g and asked g to send $Q_{new} - U$ to the next marked vertex lying clockwise on the tour. We show that there exists a $W \subseteq Z$ whose removal from Z makes $Q_{new} - \sum_{i \in W} w(i) = U$. Equivalently, we show that there exists a set $M \subseteq Z$ with $\sum_{i \in M} w(i) = U$. Recall that no vertex in Z has a weight more than U. We repeat the following step: merge two vertices a and b of same weight w in Z into one vertex with weight $2w$. Since w is a power of 2, the weight of the new vertex remains a power of 2. Repeat this merging process until a vertex in Z has weight U. Note that we will never reach a stage where all weights are less than U and no merging is possible because $\sum_{k=0}^{i} 2^k < 2^{i+1}$. Once M is found, $W = Z \backslash M$. The vertices in W will be the sole contributors of the flow from g to the next vertex lying clockwise on the tour. This argument holds true for every vertex j at which $xU \in (Q_{old}, Q_{new}]$ for some integer x. Notice that the probability that a vertex $j \in T$ receives (gets assigned) a weight of U is $w(j)/T$, which is exactly what we needed, as per the statement of the lemma.

The proof will be complete once we show that the redistribution can be done on T rather than on the directed arcs of the Euler tour on T. Consider a leaf node $l \in T$ that is in the

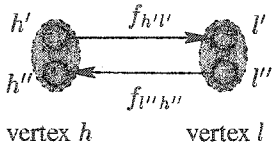


Figure 6.1. l is a leaf node with h being its parent in T.

Euler tour. Let $h \in T$ be the node that was visited just before and after l (h is l's parent in T, which is rooted at r). We use x' and x'' to represent vertex $x \in T$ in the directed Euler tour, with the tour entering x' and leaving x''. Let $f_{h'l'}$ and $f_{l''h''}$ be the flows on arcs from h' to l' and l'' to h'', respectively (see Fig. 6.1). During the redistribution process, if l had sent all its weight to some vertex— lying clockwise on the tour—that was assigned a weight of U, then ask h' to send the flow $f_{h'l'}$ directly to h'' instead of sending it through l. If l was assigned a weight of U in our redistribution process, then ask the vertices in W to reroute their flow bypassing l, i.e., make the flow coming into h' go directly to h'' instead of routing it through l. Remove l from T, and repeat this process for all leaf nodes in T. Note that whenever a leaf node is removed from T, the flow on the tree edge connecting that node to T is at most U. This process stops when there is just one node left in T. This completes the proof of Lemma 6.3.6. ∎

Let $G = (V, E)$ be the input graph with root $r \in V$, and let $D \subseteq V$ be the set of demands with d_j denoting the weight at j. Recall that the vertices in D may have non-integer weights because of the scaling we did to ensure $u_1 = \sigma_1 = 1$. Construct T_0, a ρ-approximate Steiner tree spanning D, using cables of capacity u_1. Use the procedure in the proof of Lemma 6.2.1 on T_0 with $U = u_1$, with $w(j)$ being the fractional part of $d_j \in D$, to collect demands at some subset of vertices in D such that the new weight $w'(j)$ of a vertex in D is either 0 or U. The cost of the redistribution will just be the cost of T_0. Since an optimal solution contains a candidate Steiner tree, the cost of T_0 is at most $\rho \sum_i C^*(i)/\sigma_i$.

As per the redistribution procedure, notice that (i) weight $w(j)$ of vertex $j \in T_0$ may

have been split and assigned to at most two different nodes in T_0, and (ii) the weight of U is collected at a vertex $j \in D$ if and only if $w(j) > 0$, as the probability of a vertex getting assigned a weight of U is $w(j)/U$ (by Lemma 6.2.1). Statement (i) is not consistent with our objective of routing the demands without having to split them across two nodes. To achieve our objective, we round the integral demands at D up to the nearest powers of 2, and solve the problem for these new (rounded) weights. Even though, this means that we may install twice the required cable capacities, thereby losing a factor of 2 in the approximation ratio, we will have enough cables installed to route the original demands without having to split them.

Now, replace vertex $v \in D$ of weight $w(v)$ by $w(v)$ unit-weight vertices. Let $\{v_1, .., v_{w(v)}\}$ be the set of unit-weight vertices that represent v. We call v to be the *origin* of v_i, $i = 1$ to $w(v)$. Our algorithm will ensure that the unit-weight demands having a common origin travel together—along a single path—towards the sink.

The algorithm given below proceeds in the same manner as that in [49]. At the beginning of stage 1, $D_1 = D$ with each demand $j \in D$ having weight $d_j = 1 = u_1$. In general, at the beginning of stage t, D_t is a set of $|D|/u_t$ vertices, each with demand u_t. During stage t, our algorithm (presented below) uses the value u_{t+1} as the "aggregation threshold" to combine several demands of weight u_t into a single demand of weight u_{t+1}. The cables required to perform such an aggregation are bought by the algorithm. The demand will reach the root at the end of the algorithm. The final solution is then given by the union of all the paths used in the aggregation stages. Given below are the steps performed at state t of the algorithm.

S1. *Mark* each demand v in D_t with probability $p_t = u_t/g_t$, and let D_t^* be the marked demands.

S2. Construct a ρ-approximate Steiner tree T_t on $F_t = D_t^* \cup \{r\}$. Install a cable of type $t+1$ on each edge of this tree.

S3. For each vertex $j \in D_t$, send its weight $w(j)$ to the nearest member of F_t using cables of type t. If two vertices have a common origin, ensure that both vertices send their

weight to the same $i \in F_t$, as this guarantees that vertices having a common origin travel together, thus satisfying the indivisibility constraint. Let S_v be the set of vertices, with common origin v, that sent their weights to $i \in F_t$. Let $w_t(i)$ be the weight collected at $i \in F_t$.

S4. For each $i \in F_t$, order the vertices that sent their weight of u_t to i in such a way that the vertices in S_v are ordered before the vertices in S_u, if $|S_v| \geq |S_u|$.

Divide the vertices in the ordered set into groups of u_{t+1}/u_t vertices, starting from the first vertex, leaving behind $b_i = (\frac{w_t(i)}{u_t} \bmod \frac{u_{t+1}}{u_t})$ *residual* vertices at the end. Send back the weight of u_{t+1} emanating from each group of u_{t+1}/u_t vertices back from i to a random member of that group, buying new cables of type $t+1$. Since u_t, u_{t+1} and $|S_k|$, for all k, are powers of two, our construction ensures the following: (i) set S_k, with $|S_k| \geq u_{t+1}/u_t$, is divided into p groups, where $p \geq 1$ is a power of 2, (ii) set S_k, with $|S_k| < u_{t+1}/u_t$ belongs to exactly one group. This implies that vertices with common origin travel together.

S5. For each $i \in F_t$, divide the b_i residual vertices into q_i sets $R_i^1, \ldots, R_i^{q_i}$, with each set containing vertices having common origin, and the weight $w(R_i^j)$ of a set R_i^j being the number of vertices it contains. Let $F_t' = \phi$ initially. For each $i \in F_t$, if $q_i \geq 1$, then add q_i copies of i into F_t', one for each set, with each copy carrying a weight of the sum of the vertex-weights in the set that it represents. Observe that the weights of the vertices in F_t' are powers of two. Also, note that T_t spans all the vertices in F_t', as the vertices in F_t' are mere copies of the vertices in F_t. Use the procedure of Lemma 6.3.6 on T_t spanning F_t' with $U = u_{t+1}$ to aggregate residual weights into groups of weight exactly u_{t+1} in a subset of vertices in F_t'. During the redistribution procedure, for every $i \in F_t$, ensure that its copies in F_t' are visited consecutively. This, along with the fact that $u_t \sum_{j=1}^{q_i} w(R_i^j) < U_{t+1}$ for every $i \in F_t$ ensures that at most one copy of i in F_t', representing $i \in F_t$, gets assigned a weight of u_{t+1}. Transform the redistribution among the vertices in F_t' into a redistribution among the vertices in F_t by assigning a

weight of u_{t+1} to vertex $i \in F_t$ if one of i's copy was assigned a weight of u_{t+1} in F'_t, and 0 otherwise. Notice that the probability that a vertex $i \in F_t$ is assigned a weight of u_{t+1} still depends on i's weight (residual weight, which is $b_i u_t$). For every $i \in F_t$ that receives a weight of u_{t+1}, choose a vertex $v \in b_i$ uniformly at random, and send the weight of u_{t+1} from i to v using cables of type $t+1$.

When $t = K$, we set the probability for non-root vertices $p_K = 0$, which implies that no vertex in stage $t = K$ is marked. The weights from all the vertices in D_K are directly routed to r using cables of capacity K. The approximation analysis for our SSBB algorithm is exactly the same as that for the Gupta et al.'s DSSBB algorithm [49]. All the lemmas used to prove Theorem 6.3.1 hold for our SSBB algorithm as well, but with $\epsilon = 1$. Recall that after the preprocessing step, we lose a factor of 2 from rounding up the weights of vertices to the nearest powers of 2. This means that our algorithm for the SSBB problem guarantees a ratio of twice that of Gupta et al.'s DSSBB algorithm. The cost C of our final solution is 2 times the cost in equation (6.3), and is given by

$$C < 2 \times 4 \times (2 + \rho + 1) \times \Big(2(1+1) \sum_s C^*(s)\Big).$$

Using the current best approximation ratio of $\rho = 1 + \ln(3)/2$ for finding a Steiner tree [88], we obtain a ratio of 145.6, which brings us to the following theorem.

Theorem 6.3.2 *Our algorithm for the* SSBB *problem guarantees an approximation ratio of 145.6.*

6.4 Remarks

We believe that finding an acceptable lower bound for the SSBB problem may help in obtaining better approximation ratios. Our approach of analyzing the ratio, adapted from [49], may not be the best way to analyze the final ratio after all. We do not see a reason why our ratio of 145.6 for the SSBB problem cannot be improved by a more careful analysis. Is it possible

that the ratio of our SSBB is actually 72.8 as claimed by Gupta et al.? Another interesting variant would be to impose an additional restriction that the final network be a tree, which would be a natural generalization of the CMStT problem.

CHAPTER 7

MINIMUM LATENCY TOURS AND THE k-TRAVELING REPAIRMEN PROBLEM

7.1 Introduction

7.1.1 The minimum latency problem

In the traveling salesman problem (TSP), the objective is to find a minimum length tour visiting all the vertices of a given graph. If the weights of edges are interpreted as travel times, the objective of TSP is to find a tour with minimum travel time. The TSP has direct applications in vehicle-routing and scheduling problems. The objective of TSP is only concerned about the welfare of the salesman (or server). It never considers into fact the waiting time of customers being visited or served. Modifying the objective in terms of the customer's perspective gives rise to the *Minimum Latency Problem* (MLP) or the *traveling repairmen* problem, whose formal definition is given below. In the literature, the traveling repairmen problem is also known as the *delivery man problem* [31, 76] or the *school bus-driver problem* [99]

> MLP: Given a set of n points, a symmetric distance matrix satisfying triangle inequality, and a starting point s, the MLP asks for a tour starting at s and covering all the points such that the sum of the latencies of the points is minimum. *Latency* of a point p is defined to be the length of the tour from the s to p.

The problem of minimizing the sum of the latencies is equivalent to the problem of minimizing the average latency, because the sum is the constant (number of points) times the average.

At first glance, even though the MLP looks like a simple variant of the TSP, a closer examination reveals that its difficulty is very different from the TSP. Unlike an optimal TSP tour, an optimal MLP tour may revisit a point an unbounded number of times even when the underlying graph has a bounded degree [13]. Consider an instance with points on a real line, and point p_j located at $(-3)^j$. For this instance, an optimal minimum latency tour starting at the origin will visit the points in order (p_j is the $j$the point visited), thereby crossing the origin $n-1$ times. A simple perturbation of this set of points will result in a new instance whose optimal minimum latency tour is not planar—a characteristic which distinguishes MLP from TSP.

The MLP is at least as hard as the TSP [13]. Consider an instance for which we wish to minimize the length of the TSP tour. Augment the instance by placing N additional points at distance *infinity*, for a large number N. An optimal MLP tour on this new instance will have to minimize the length of the TSP on the original set of points [13]. This demonstrates that the MLP is NP-hard even for points in the plane. As a result, it is NP-hard to approximate the MLP within any bounded ratio when the distance matrix does not satisfy the triangle inequality.

For a given graph G and a positive i, the i-MST problem asks for a minimum-cost tree spanning i vertices in G. The i-MST problem is NP-hard. A sequence of approximation results reduced the ratio from $O(\sqrt{i})$ [86] to $\log^2 i$, [14] and $\log i$ [84] to a constant factor [15]. The current best approximation ratio for the i-MST problem is $(2+\epsilon)$, due to Arora and Karakostas [9], an improvement over the previous best ratio of 3, due to Garg [36].

The first constant factor approximation for MLP was given by Blum et al. [13]. Goemans and Kleinberg [43] improved the ratio for MLP to 3.59α, where α is the best achievable approximation ratio for the i-MST problem. Archer, Levin and Williamson [5] presented faster algorithms for MLP with a slightly better approximation ratio of 7.18. Recently, Chaudhuri et al. [21] have reduced the ratio by a factor of 2, to 3.59. They build on Archer, Levin and Williamson's techniques with the key improvement being that they bound the cost of their

i-trees by the cost of a minimum cost path visiting i nodes, rather than twice the cost of a minimum cost tree spanning i nodes.

7.1.2 The k-traveling repairmen problem

A generalization of the MLP is the *k-Traveling Repairmen* (KTR) problem, in which the objective is to find k tours instead of just one. In graph theoretic terms, the KTR problem can formally be defined as follows.

> KTR: Given a finite metric on a set of vertices V and a source vertex $s \in V$, the k-traveling repairman (KTR) problem, a generalization of the metric traveling repairman problem, asks for k tours, each starting at s (depot) and covering all the vertices (customers) such that the sum of the latencies experienced by the customers is minimum. *Latency* of a customer p is defined to be the distance traveled (time elapsed) by a repairman before visiting p for the first time.

The KTR problem is NP-hard even for weighted trees [95]. For the KTR problem, Fakcharoenphol, Harrelson and Rao [28], presented a 8.497α-approximation algorithm. Their ratio was recently improved to $2(2+\alpha)$ by Chekuri and Kumar [22]. For a multidepot variant of the KTR problem, in which k repairmen start from k different starting locations, Chekuri and Kumar [22] presented a 6α-approximation algorithm. Recently, Chaudhuri et al. [21] have reduced the ratio to 6 for both the KTR problem and its multidepot variant.

7.2 The generalized k-traveling repairmen problem

Literature on the KTR problem shows that all the results thus far are based on the assumption that the repair-time of a customer is zero for latency calculations. In other words, it was assumed that a customer is served instantaneously once he/she is visited. In this section, we consider a generalization of the KTR problem, the *generalized* KTR (GKTR) problem,

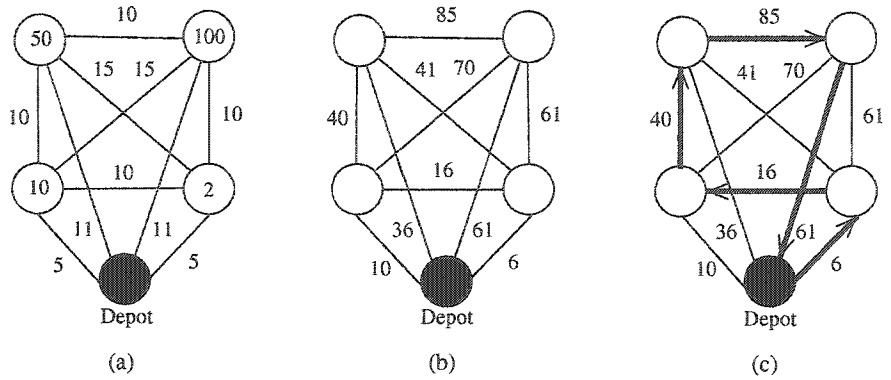


Figure 7.1. (a) Original graph G. (b) Transformed graph G^*. (c) Optimal tour for G^*.

in which the repair-time of a customer is not necessarily zero. The formal definition of the GKTR problem is given below.

> GKTR: Consider a given metric defined on a set of vertices, V, a source vertex $s \in V$ and a positive number k. Also given is a non-negative number for each vertex $v \in \{V - s\}$, denoting the repair-time at v. The GKTR problem asks for k tours, each starting at s, covering all the vertices such that the sum of the latencies of all the vertices is minimum.

The number associated with each vertex represents the amount of time a repairman has to spend at that vertex, and the weight on an edge represent the time it takes to traverse along that edge. The definition of "latency" for the GKTR problem is same as that of the KTR problem. The latency of a customer p does not include p's repair-time. By definition, latency is the amount of time a customer waits before being served.

It is easy to see that the GKTR problem resembles most real-life situations, one of which is that the repairmen have to spend some time at each customer's location, say, for the repair or installation of equipment. This applies even for deliverymen who spend some time delivering goods. Hence, it is natural to formulate the repairman problem with repair-times.

At first, even though it looks like that the GKTR problem can be reduced to the KTR problem in a straight forward manner, a careful analysis reveals that such a reduction may not be possible without a compromise in the approximation ratio. One idea would is to make the graph directed, by adding the repair-time of a vertex to the outgoing edges and making the vertex weights to be zero. Unfortunately, solution to the latency problem with asymmetric edge weights is not known.

A second idea is to to incorporate the repair-times associated with vertices into edge weights (where the weight of an edge represents the time to traverse that edge) as follows: for every edge $e = (i, j) \in G$, increase the weight (or distance) of e by $r_i/2 + r_j/2$, where r_i and r_j are the repair-times of i and j respectively. Fig. 7.1 depicts such a transformation for a sample instance with $k = 1$. The resultant graph G^* after such a transformation still obeys triangle inequality, which allows us to use any of the KTR algorithms, say, with approximation guarantee β. The solution obtained is a β-approximation for the modified graph G^*, but not a β-approximation for the original problem G. This is due to the reason that the lower bounds for the problems defined as G and G^* are different, as can be seen from the fact that the latency of a customer v in an optimal solution to G^* comprises half of v's repair-time, while this is not the case with an optimal solution to G. An optimal solution to G may be arbitrarily away from an optimal solution to G^*.

Another idea would be add the repair-times to the length of only those edges that does not have s as an endpoint, and use the KTR algorithm as it is. This does not guarantee a ratio of β either, as such a transformation violates the triangle inequality property, which is a requirement for using the KTR algorithm.

Let us consider the following instance to understand this better. Let there be a customer whose repair-time is arbitrarily large when compared to the repair-times of all other customers and edge weights in the graph. It is obvious that an optimal solution for such an instance will not include the repair-time of the last customer visited by the repairman. If we use the above described strategy to transform the given graph into a new graph, then the optimal solution

for the new graph will be much larger than the optimal solution to the original graph, since the latency of every vertex in such a solution will be more than half its repair-time. The above discussion demonstrates the difficulty involved in the reduction of the GKTR problem to the KTR problem. This shows that the algorithms presented for the KTR problem do not solve the GKTR problem with the same approximation guarantee.

We present algorithms that gets around the difficulties mentioned above. For fixed k, we present a $(\beta + 2)$-approximation algorithm for the GKTR problem, where β is the best achievable approximation ratio (currently 6) for the KTR problem. For arbitrary k, we obtain an approximation ratio of $\frac{3}{2}\beta + \frac{1}{2}$. When the repair-times of all the customers are the same, we present an approximation algorithm with a better ratio. Based on the current best β value, the ratio of our algorithm is 7.1604. Our ratios hold for the respective multidepot variants of the GKTR problem as well.

Our algorithms for the GKTR problem use the KTR algorithm as a black-box. The current best approximation ratio for the KTR problem is 6, due to Chaudhuri et al. [21]. We use $l(v)$ denote v's latency.

7.2.1 Algorithms and their analyses: Non-uniform repair-times

Let $G = (V, E)$ be the given graph. Let $M \subset V$ be the set of vertices with k largest repair-times. Let G' be the subgraph of G induced by $V \backslash M$. Construct a new graph G^* from G' such that for every edge $e' = (i, j) \in G'$, introduce an edge e^* connecting i and j in G^* with weight $|ij| + r_i/2 + r_j/2$. Make the repair-times of all the vertices in G^* to be zero. It can be easily seen that the edges in G^* obey the triangle inequality, and that G^* is a KTR instance. Let opt, opt' and opt^* denote the total latencies of all the customers in an optimum solution for G, G' and G^*, respectively. Let apx and apx' denote the total latencies of all the customers in our solution for G and G', respectively. Let APX' and APX^* denote the respective approximate solutions for G' and G^*. Before we proceed to the algorithm and its analysis, we present the following lemmas.

Lemma 7.2.1 $opt \geq opt'$.

Proof. Construct an approximate solution APX' for G' from an optimal solution of G by visiting only the vertices in $V \backslash M$ (using shortcutting). The fact that G' is a subgraph of G and that its edges obey triangle inequality proves that $opt \geq apx'$, which in turn proves the lemma. ∎

Lemma 7.2.2 *Let $V = \{x_1, \ldots, x_n\}$ be the set of vertices in G. Let r_i denote the repair-time of x_i and let R_k denote the sum of the k largest repair-times among the repair-times of all vertices. Let opt be the sum of the latencies of all vertices in an optimal solution using k repairmen. Then,*

$$opt \geq \left[\sum_{i=1}^{n}\Big(|sx_i| + r_i\Big)\right] - R_k.$$

Proof. The latency of every vertex in an optimal solution is at least $|sx_i|$. Also, the latency of any solution has to include all but the k largest repair-times. ∎

Lemma 7.2.3 $opt' = opt^* - \sum_{i \in V \backslash M} \frac{r_i}{2}$.

Proof. We prove the lemma by showing that

$$opt' \geq opt^* - \sum_{i \in V \backslash M} \frac{r_i}{2} \quad \text{and} \quad opt' \leq opt^* - \sum_{i \in V \backslash M} \frac{r_i}{2}.$$

- $opt' \geq opt^* - \sum_{i \in V \backslash M} \frac{r_i}{2}$. Let OPT' be an optimal solution to G'. Suppose $opt' < opt^* - \sum_{i \in V \backslash M} \frac{r_i}{2}$. Then, we can construct the same set of k tours for G^* as in OPT' such that the i^{th} tour in G^* visits the same set of vertices as visited by the i^{th} tour in OPT', and in the same order. The sum of the latencies of all the customers in such a solution for G^* will be $opt' + \sum_{i \in V \backslash M} \frac{r_i}{2}$, which contradicts the fact that opt^* is the optimal sum of latencies for G^*.

- $opt' \leq opt^* - \sum_{i \in V \setminus M} \frac{r_i}{2}$. Let OPT' be an optimal solution to G^*. Suppose $opt' > opt^* - \sum_{i \in V \setminus M} \frac{r_i}{2}$. Then, we can construct the same set of k tours for G' as in OPT^* such that the i^{th} tour in G' visits the same set of vertices as visited by the i^{th} tour in OPT^*, and in the same order. The sum of the latencies of all the customers in such a solution for G' will be $opt^* - \sum_{i \in V \setminus M} \frac{r_i}{2}$, which contradicts the fact that opt' is the optimal sum of latencies for G'.

■

We obtain an approximate solution APX' to G' as follows. We first obtain a β-approximate solution APX^* to G^*. Let $t_1, t_2, \ldots t_k$ be the set of k tours in APX^*. We then construct the same set of k tours for G' as in APX^* such that the i^{th} tour in G' visits the same set of vertices as visited by the i^{th} tour in APX^*, and in the same order. It can be seen that the sum of the latencies of all the customers in G' is

$$\begin{aligned} apx' &= \beta opt^* - \sum_{j \in V \setminus M} \frac{r_j}{2} \\ &= \beta\Big(opt' + \sum_{i \in V \setminus M} \frac{r_i}{2}\Big) - \sum_{i \in V \setminus M} \frac{r_i}{2}. \quad \text{(by Lemma 7.2.3)} \end{aligned} \tag{7.1}$$

Let $M = \{v_1, v_2, \ldots v_k\}$ be the set of vertices, with k largest repair-times in G. Extending the tour t_i in G' to include v_i as its last vertex, for all i, gives a feasible set of k tours for G, with apx denoting the sum of the latencies of all the customers in G. The latency of vertex v_i, $l(v_i)$, added to the i^{th} tour will be at most the sum of the latency of its predecessor vertex p_i (vertex visited by the i^{th} tour just before visiting v_i), its repair-time r_{p_i} and $|p_i v_i|$. Let p_i be the j^{th} vertex visited in the i^{th} tour and let $\{u_{(1,i)}, \ldots, u_{(j-1,i)}\}$ be the other vertices visited by the i^{th} tour before visiting p_i. Since $|sp_i| + |sv_i| \geq |p_i v_i|$, where s is the central depot, the latency of v_i is given by

$$l(v_i) \leq l(p_i) + r_{p_i} + |sp_i| + |sv_i|.$$

The sum of the latencies of all v_i's, where $i = 1 \ldots k$, is given by

$$\begin{aligned}\sum_{i=1}^{k} l(v_i) &\leq \sum_{i=1}^{k}\Big[l(p_i) + r_{p_i} + |sp_i| + |sv_i|\Big] \\ &\leq \sum_{i=1}^{k}\Big[|sp_i| + \sum_{g=1}^{j-1} 2|su_{(g,i)}| + \sum_{g=1}^{j-1} r_{u_{(g,i)}}\Big] + \sum_{i=1}^{k}\Big[r_{p_i} + |sp_i| + |sv_i|\Big] \quad (7.2)\end{aligned}$$

7.2.2 $(\frac{3}{2}\beta + \frac{1}{2})$-approximation for arbitrary k

We can rewrite (7.2) as follows:

$$\begin{aligned}\sum_{i=1}^{k} l(v_i) &\leq \sum_{i=1}^{k}\Big[|sp_i| + \sum_{g=1}^{j-1} |su_{(g,i)}|\Big] \\ &\quad + \sum_{i=1}^{k}\Big[\Big(\sum_{g=1}^{j-1} r_{u_{(g,i)}}\Big) + r_{p_i} + \Big(\sum_{g=1}^{j-1} |su_{(g,i)}|\Big) + |sp_i| + |sv_i|\Big] \\ &\leq \sum_{i=1}^{k}\Big[|sp_i| + \sum_{g=1}^{j-1} |su_{(g,i)}|\Big] + opt. \quad \text{(by Lemma 7.2.2)} \qquad (7.3)\end{aligned}$$

The sum of the latencies of all the customers in G is given by

$$apx = apx' + \sum_{i=1}^{k} l(v_i).$$

By substituting (7.1) and (7.3), we get

$$\begin{aligned}apx &\leq \beta opt' + opt + \frac{\beta-1}{2}\sum_{i\in V\setminus M} r_i + \sum_{i=1}^{k}\Big[|sp_i| + \sum_{g=1}^{j-1} |su_{(g,i)}|\Big] \\ &\leq (\beta+1)opt + \frac{\beta-1}{2}\sum_{i\in V\setminus M} r_i + \sum_{i=1}^{k}\Big[|sp_i| + \sum_{g=1}^{j-1} |su_{(g,i)}|\Big] \quad \text{(by Lemma 7.2.1)} \\ &= (\beta+1)opt + \frac{\beta-3}{2}\sum_{i\in V\setminus M} r_i + \sum_{i=1}^{k}\Big[|sp_i| + \sum_{g=1}^{j-1} |su_{(g,i)}|\Big] + \sum_{i\in V\setminus M} r_i \\ &\leq (\beta+1)opt + \frac{\beta-3}{2}\sum_{i\in V\setminus M} r_i + opt \quad \text{(by Lemma 7.2.2)} \\ &= (\beta+2)opt + \frac{\beta-3}{2}\sum_{i\in V\setminus M} r_i\end{aligned}$$

$$\begin{aligned} &\leq (\beta+2)opt + \frac{\beta-3}{2}opt \quad \text{(by Lemma 7.2.2)} \\ &= \left(\frac{3}{2}\beta + \frac{1}{2}\right)opt. \end{aligned}$$

Theorem 7.2.1 *For the* GKTR *problem, there exists a polynomial time algorithm whose approximation ratio is* $\frac{3}{2}\beta + \frac{1}{2}$*, where* β *is the best achievable approximation ratio for the* KTR *problem.*

7.2.3 $(\beta + 2)$-approximation for fixed k

Let Q be the set of vertices that are visited last in each of the k tours in an optimal solution to $G = (V, E)$. Since k is fixed, we can assume that we know Q, as it takes polynomial time in the order of $|V|^k$ to try out all possible values of Q. Knowing set Q strengthens Lemma 7.2.1 as shown in the following lemma.

Lemma 7.2.4

$$opt \geq opt' + \sum_{j \in V \setminus Q} r_j + \sum_{i \in Q} |sv_i|.$$

Proof. The lemma follows immediately from the following fact: every vertex $v \in Q$ that is visited last by a tour in an optimal solution to G must account for a latency of at least the sum of the repair-times of all the vertices that were visited by that tour prior to visiting v, and the cost of the path along that tour. ∎

For fixed k, we can rewrite (7.2) as follows:

$$\begin{aligned} \sum_{i=1}^{k} l(v_i) &= \sum_{i=1}^{k}\Big[\sum_{g=1}^{j} 2|su_g| + |sv_i| + \sum_{g=1}^{j} r_{u_g}\Big] \\ &= \sum_{i \in V \setminus Q} 2|si| + \sum_{i \in Q} |si| + \sum_{i \in V \setminus Q} r_i. \end{aligned} \tag{7.4}$$

As before, the sum of the latencies of all the customers in G is given by

$$apx = apx' + \sum_{i=1}^{k} l(v_i).$$

By substituting (7.1) and (7.4), we get

$$\begin{aligned} apx &= \beta\Big(opt' + \sum_{i\in V\setminus Q}\frac{r_i}{2}\Big) - \sum_{i\in V\setminus Q}\frac{r_i}{2} + \sum_{i\in V\setminus Q} 2|si| + \sum_{i\in Q}|si| + \sum_{i\in V\setminus Q} r_i \\ &= \beta opt' + \frac{\beta+1}{2}\sum_{i\in V\setminus Q} r_i + \sum_{i\in Q}|si| + \sum_{i\in V\setminus Q} 2|si| \\ &\leq \beta opt + \sum_{i\in V\setminus Q} 2|si| \quad \text{(by Lemma 7.2.4)} \\ &\leq \beta opt + 2opt \quad \text{(by Lemma 7.2.2).} \end{aligned}$$

Theorem 7.2.2 *For the* GKTR *problem with fixed k, there exists a polynomial time algorithm with $(\beta+2)$-approximation ratio, where β is the best achievable approximation ratio for the* KTR *problem.*

7.2.4 Multiple depot

Theorem 7.2.3 *For the* GKTR *problem, with k repairmen starting at k starting locations, there exists a polynomial time algorithm with (β_m+2)-approximation ratio for fixed k, and $(\frac{3}{2}\beta_m+\frac{1}{2})$-approximation ratio for arbitrary k, where β_m is the best achievable approximation ratio for the multidepot variant of the* KTR *problem.*

Proof. Since Lemmas 7.2.1, 7.2.2, 7.2.3 and 7.2.4 hold regardless of where the k repairmen start, the proof remains the same as that of Theorems 7.2.1 and 7.2.2. ∎

7.3 Algorithms and their analyses: Uniform repair-times

When the repair-times of all customers are the same, we show that the approximation guarantee can be improved considerably. To achieve a smaller approximation ratio, we present two algorithms that work at tandem.

Let $G = (V, E)$ be the given graph. Let r denote the repair-time of the customer i.e., $r_i = r$ for all $i \in V - \{s\}$ (repair-time of s is zero). Construct a new graph G^* from G such that for every edge $e = (i, j) \in G$, introduce an edge e^* connecting i and j in G^* with weight

$|ij| + r_i/2 + r_j/2 = |ij| + r$. Make the repair-times of all the vertices in G^* to be zero. It can be easily seen that the edges in G^* obey triangle inequality and that G^* is a KTR instance. Let *opt* and *opt** denote the total latencies of all the customers in an optimum solution for G and G^*, respectively. Let apx denote the total latency of all the customers in our solution for G. Let n denote the number of vertices in the graph.

Proposition 7.3.1 *Let C be a positive constant. Let x and y be positive variables such that $x + y = C$. Then $x(x-1) + y(y-1)$ is minimum when $x = y$.*

Lemma 7.3.1 *In any optimal solution, the contribution to the sum of latencies due to repair-times alone is at least $rn(\frac{n}{k} - 1)/2$.*

Proof. Now suppose that for a given n and k, there exists an optimal solution for which the contribution due to repair-times alone is opt_r. If, in such an optimal solution, there exists two repairmen who visit different number of customers, say y and z, then, we can always construct an alternate solution ALT from the optimal solution by making those two repairmen visit $\frac{y+z}{2}$ customers each. By Proposition 7.3.1, the contribution to the sum of latencies, in ALT, due to repair-times alone will be less than opt_r. We can continue to find a feasible solution in this manner, until all repairmen visit the same number of customers, which is $\frac{n}{k}$. That brings us to the following equation, which proves the lemma.

$$opt_r \geq rk\frac{\frac{n}{k}(\frac{n}{k} - 1)}{2} = \frac{rn(\frac{n}{k} - 1)}{2}.$$

■

7.3.1 Algorithm 1

This algorithm proceeds on a case-by-case basis, depending on how k compares to n. Let the customers be sorted in non-decreasing order with respect to their distances to the depot. Let $A = \{c_1, \ldots, c_n\}$ be the sorted set of n customers, i.e., $|sc_1| \leq |sc_2| \leq \cdots \leq |sc_n|$.

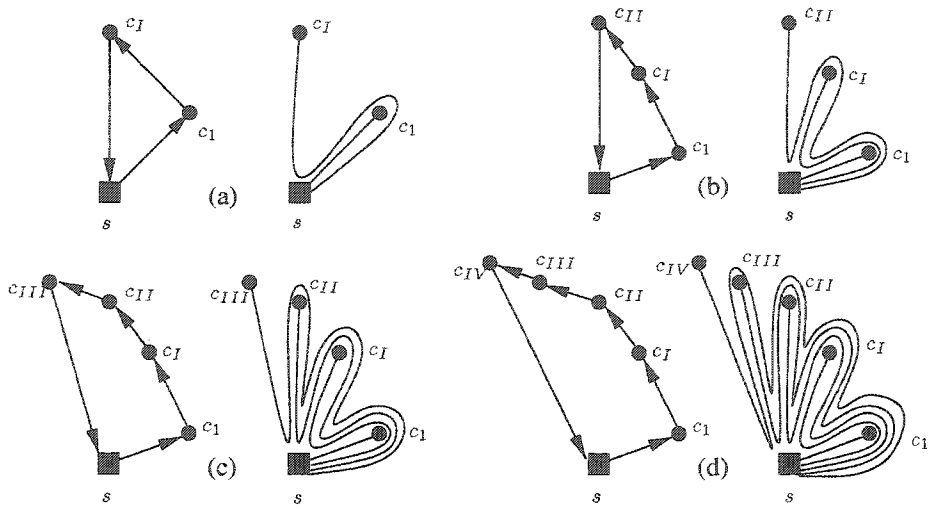


Figure 7.2. *(a)* $k \leq \frac{n}{2}$ *(b)* $\frac{n}{3} \leq k < \frac{n}{2}$ *(c)* $\frac{n}{4} \leq k < \frac{n}{3}$ *(d)* $\frac{n}{5} \leq k < \frac{n}{4}$

1. *Case* $k \geq n/2$. The i^{th} repairman visits customer c_i first, for all $i \leq k$. In addition, repairmen 1 to $n-k$ are assigned to visit one additional customer each, from the remaining pool of $n-k$ unassigned customers, as their second customer.

   Let t_1 be one of k such tours constructed in this manner. Let c_1, and maybe c_I, be the customers visited by tour t_1, in that order. The latency of c_1 would just be $|sc_1|$ and the latency of c_I would be at most $|sc_1| + r + |sc_1| + |sc_I|$. Fig. 7.2(a) depicts this pictorially. The sum of the latencies of customers c_1 and c_I is $|sc_1| + |sc_I| + r + 2|sc_1|$. The sum of the latencies of all the customers visited by k tours is then at most $\sum_{j=1}^{n} |sc_j| + (n-k)r + 2\sum_{j=1}^{n-k} |sc_j|$. By Lemma 7.2.2, the sum of the latencies is at most $opt + 2\sum_{j=1}^{n-k} |sc_j|$. Since A is sorted in non-decreasing order, the approximation ratio is less than or equal to $1 + 2(n-k)/n = 3 - 2\frac{k}{n}$. Since $k \geq n/2$, the ratio is at most 2.

2. *Case* $n/3 \leq k < n/2$. The i^{th} repairmen visits customer c_i first, for all $i \leq k$. Each repairmen picks one customer out of $\{c_{k+1}, \ldots, c_{2k}\}$ to be his next customer. In addition, repairmen 1 to $(n-2k)$ are assigned to visit one additional customer each, from the remaining pool of $n-2k$ unassigned customers, as their third customer.

Let t_1 be one of k such tours constructed in this manner. Let c_1, c_I, and maybe c_{II}, be the customers visited by tour t_1, in that order. The latency of c_1 would just be $|sc_1|$. The latencies of c_I and c_{II} would be at most $|sc_1| + r + |sc_1| + |sc_I|$ and $|sc_1| + r + |sc_1| + |sc_I| + r + |sc_I| + |sc_{II}|$, respectively (see Fig. 7.2(b)). The sum of the latencies of customers c_1, c_I and c_{II} is at most $|sc_1| + |sc_I| + |sc_{II}| + 4|sc_1| + 2|sc_I| + r + 2r$. The sum of the latencies of all the customers visited by k tours is given by

$$
\begin{aligned}
apx &\leq \sum_{j=1}^{n} |sc_j| + 2\sum_{j=1}^{k} |sc_j| + 2\sum_{j=1}^{n-2k} |sc_j| + 2\sum_{j=k+1}^{n-k} |sc_j| + kr + (n-2k)2r \\
&\leq \sum_{j=1}^{n} |sc_j| + \frac{2k}{n}\sum_{j=1}^{n} |sc_j| + \frac{2(n-2k)}{n}\sum_{j=1}^{n} |sc_j| + \frac{2(n-2k)}{n-k}\sum_{j=1}^{n} |sc_j| \\
&\quad +2(n-k)r - kr \\
&\leq \sum_{j=1}^{n} |sc_j| + \frac{2(n-k)}{n}\sum_{j=1}^{n} |sc_j| + \frac{2(n-2k)}{n-k}\sum_{j=1}^{n} |sc_j| + 2(n-k)r \\
&= 3\sum_{j=1}^{n} |sc_j| + 2(n-k)r - \frac{2k}{n}\sum_{j=1}^{n} |sc_j| + \frac{2(n-2k)}{n-k}\sum_{j=1}^{n} |sc_j| \\
&\leq 3opt - \frac{2k}{n}\sum_{j=1}^{n} |sc_j| + \frac{2(n-2k)}{n-k}\sum_{j=1}^{n} |sc_j| \quad \text{(by Lemma 7.2.2)} \\
&\leq 3opt - \frac{2k}{n}opt + \frac{2(n-2k)}{n-k}opt \quad \text{(by Lemma 7.2.2)} \\
&= \left[3 - \frac{2k}{n} + \frac{2(n-2k)}{n-k}\right]opt \\
&= \left[3 - 2x + \frac{2(1-2x)}{1-x}\right]opt
\end{aligned}
$$

where $x = k/n$. Since $n/3 \leq k < n/2$, apx is bounded by $\frac{10}{3}opt$.

3. *Case* $n/4 \leq k < n/3$. The i^{th} repairmen visits customer c_i first, for all $i \leq k$. Each repairmen picks one customer out of $\{c_{k+1}, \ldots, c_{2k}\}$ to be his second customer, and one customer out of $\{c_{2k+1}, \ldots, c_{3k}\}$ to be his third customer. In addition, repairmen 1 to $(n - 3k)$ are assigned to visit one additional customer each, from the remaining pool of $n - 3k$ unassigned customers, as their fourth customer (see Fig. 7.2(c)). The analysis proceeds in the same manner as in case 2 and the sum of the latencies of all

the customers visited by k tours is given by

$$apx \leq \left[3 - 2x + \frac{2(1-2x)}{1-x} + \frac{2(1-3x)}{1-2x}\right] opt$$

where $x = k/n$. Since $n/4 < k \leq n/3$, apx is bounded by $\frac{29}{6} opt$.

4. *Case* $n/5 \leq k < n/4$. The i^{th} repairmen visits customer c_i first, for all $i \leq k$. Each repairmen picks one customer out of $\{c_{k+1}, \ldots, c_{2k}\}$ to be his second customer, one customer out of $\{c_{2k+1}, \ldots, c_{3k}\}$ to be his third customer, and one customer out of $\{c_{3k+1}, \ldots, c_{4k}\}$ to be his fourth customer. In addition, repairmen 1 to $(n - 4k)$ are assigned to visit one additional customer each, from the remaining pool of $n - 4k$ unassigned customers, as their fifth customer (Fig. 7.2(d)). The analysis proceeds in the same manner as in case 2 and the sum of the latencies of all the customers visited by k tours is given by

$$apx \leq \left[3 - 2x + \frac{2(1-2x)}{1-x} + \frac{2(1-3x)}{1-2x} + \frac{2(1-4x)}{1-3x}\right] opt$$

where $x = k/n$. Since $n/5 < k \leq n/4$, apx is bounded by $\frac{193}{30} opt$.

5. *Case* $n/6 \leq k < n/5$. Each repairman visits 5-6 customers in non-increasing order of distances, as in previous cases. The analysis proceeds in the same manner as in case 2 and the sum of the latencies of all the customers visited by k tours is given by

$$apx \leq \left[3 - 2x + \frac{2(1-2x)}{1-x} + \frac{2(1-3x)}{1-2x} + \frac{2(1-4x)}{1-3x} + frac2(1-5x)1-4x\right] opt$$

where $x = k/n$. Since $n/6 < k \leq n/5$, apx is bounded by $8.10025opt$. If $k \geq 0.188364n$, then apx is bounded by $7.1604opt$.

7.3.2 Algorithm 2

Just like in non-uniform repair-time case, we find a β-approximate solution APX^* to G^*. Let $t_1, t_2, \ldots t_k$ be the set of k tours in it. Construct the same set of k tours in G as in APX^* such

that the i^{th} tour in G visits the same set of vertices as visited by the i^{th} tour in APX^*, and in the same order. It can be seen that the sum of the latencies of all the customers in G is

$$\begin{aligned} apx &= \beta opt^* - \sum_{i=1}^{n} \frac{r_i}{2} \\ &= \beta opt^* - \frac{nr}{2}. \end{aligned} \tag{7.5}$$

By Lemma 7.2.3,

$$\begin{aligned} opt &= opt^* - \sum_{i=1}^{n} \frac{r_i}{2}. \\ &= opt^* - \frac{nr}{2}. \end{aligned}$$

Substituting for opt^* in (7.5), we get

$$apx = \beta opt + \Big(\frac{\beta - 1}{2}\Big) nr.$$

The approximation ratio of Algorithm 2 can be calculated from the above equation as follows.

$$\begin{aligned} \frac{apx}{opt} &= \frac{\beta opt + \left(\frac{\beta-1}{2}\right) nr}{opt} \\ &\leq \beta + \frac{(\frac{\beta-1}{2})nr}{\frac{rn(\frac{n}{k}-1)}{2}} \quad \text{(by Lemma 7.3.1)} \\ &\leq \beta + \frac{\beta - 1}{\frac{n}{k} - 1}. \end{aligned}$$

Substituting $\beta = 6$, it can be easily verified that the ratio is at most 7.1604 when $k \leq 0.188364n$. Since the the ratio is bounded by 7.1604 for values of $k \geq 0.188364n$ (Algorithm 1), we get the following theorem.

Theorem 7.3.1 *For the* GKTR *problem with uniform repair-times, there exists a polynomial time algorithm with 7.1604-approximation ratio. The ratio is valid even if the k repairmen start at k different starting locations.*

7.4 The Bounded-Latency Problem

The bounded-latency problem (BLP) is a complementary version of the KTR problem, in which we are given a latency bound L and are asked to find the minimum number of repairmen required to service all the customers such that the latency of no customer is more than L. Unlike the GKTR problem, in which the sum of the latencies are minimized, in BLP the objective is to minimize the number of repairmen with the constraint that the latency of no customer exceeds L. The formal definition of the BLP is given below.

> BLP: Given a metric defined on a set of vertices, V, a source vertex $s \in V$ and a positive number L. The BLP asks for minimum number of tours, each starting at s, covering all the vertices such that the latency of no customer is more than L.

The bounded-latency problem is common in real-life as most service providers work only during the day, generally a 8-hour work day. Under these circumstances, the service provider naturally wants to provide service to all its outstanding customers within the work day, by using the least number of repairmen. For the BLP, we present a bicriteria approximation algorithm that finds a solution with at most $2/\rho$ times the number of repairmen required by an optimal solution, with the latency of no customer exceeding $(1+\rho)L$ for any $\rho > 0$.

Proposition 7.4.1 *Length of any optimal set of tours for the* BLP *is at least the length of an* MST.

Proof. Removing the edge connecting the last customer in a tour to the depot, for each tour, gives a spanning tree, the length of which is at least the length of an MST. ∎

Given below is an algorithm which groups the customers, and assigns one repairman to each group. Let $\rho > 0$.

1. Construct a tour for the given set of vertices (depot and customers) using the best available approximation algorithm for TSP.

2. Remove the depot from the tour.

3. Set `lengthTraveled` = 0;

4. Starting from some vertex, traverse the tour.

5. While not all edges in the tour are traversed, traverse the next edge e on the tour.

   (a) If `lengthTraveled` + `length`(e) $\leq \rho L$, increment `lengthTraveled` by `length`(e).

   (b) Else remove e from the tour, and set `lengthTraveled` = 0.

At the end of the above algorithm, we will be left with segments, each of which has length at most ρL (see Fig. 7.3). For each segment, introduce two edges to connect its endpoints (vertices) to the depot. Since our tour is of length at most twice than that of an MST, by Proposition 7.4.1, our solution will require at most $2/\rho$ repairmen to traverse the $2/\rho$ tours. Assuming that there exists a feasible solution for a given instance, the length of an edge connecting any vertex to the depot is at most L. Hence, regardless of which direction each tour in our solution is traversed, each customer will have a latency of at most $(1+\rho)L$.

Theorem 7.4.1 *For the bounded-latency problem, in which we are a given a latency bound L, there exists a bicriteria approximation algorithm that finds a solution with at most $2/\rho$ times the number of repairmen required by an optimal solution, with the latency of no customer exceeding $(1+\rho)L$ for any $\rho > 0$.*

7.5 Remarks

We presented approximation algorithms for the generalized version of the KTR problem, in which the time spent by a repairman at a customer's location is considered to be non-zero. For the case when the repair-times are different for each customer, our algorithm guarantees a ratio of $2-\beta$ for fixed k, $3\beta/2+1/2$ for arbitrary k. Here β is the best achievable

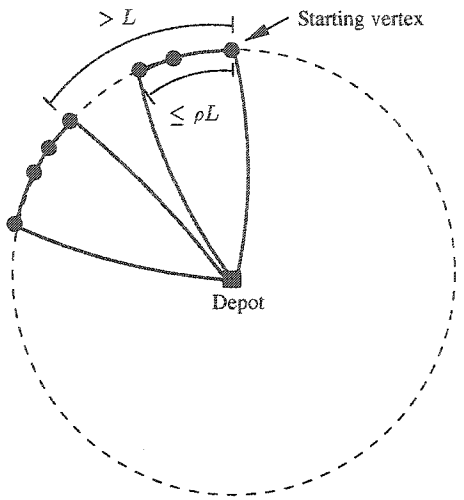


Figure 7.3. Cutting the tour into segments of size at most ρL.

approximation ratio for the original KTR problem with zero repair-times. For the case when the repair-times are all the same, we presented a 7.1604-approximation algorithm. We also introduced a complementary version of the KTR problem, the BLP, in which we are given a latency bound L and are asked to find minimum number of tours such that no customer experiences a latency more than L. For this problem, we presented a bicriteria approximation algorithm.

It should be interesting to see whether one can design algorithms that do not use the KTR algorithm as a black-box, and obtain better ratios.

CHAPTER 8
DEGREE-BOUNDED MINIMUM SPANNING TREES

8.1 Introduction

In this chapter, we consider the degree-δ MST problem, whose formal definition is given below.

> Degree-δ minimum spanning tree: Given n points in the Euclidean plane, with the weight of an egde defined to be its length, the degree-δ minimum spanning tree problem asks for a minimum weight spanning tree in which the degree of each vertex is at most δ.

The degree-δ minimum spanning tree problem is a generalization of the Hamiltonian path problem, which is NP-hard [35, 52]. Papadimitriou and Vazirani [81] showed that the Euclidean version of the problem in $\Re^2$ is NP-hard for $\delta = 3$ and conjectured that it remains NP-hard for $\delta = 4$ as well. The problem is polynomial-time solvable when $\delta = 5$, because the always exists an MST with $\delta = 5$ [78]. In this chapter, we show that, for any arbitrary collection of points in the plane, there always exists a degree-4 spanning tree of weight at most 1.1381, $(\sqrt{2}+2)/3$ to be exact, times the weight of a minimum spanning tree (MST). In particular, we present an improved approximation analysis for Chan's degree-4 MST algorithm [18].

8.1.1 Previous work

Christofides [23] gave a 1.5-approximation algorithm for the traveling salesman problem (TSP) for general metrics. Arora [6] and Mitchell [77] presented polynomial-time approxi-

mation schemes (PTAS) that finds Hamiltonian cycles (TSP) in Euclidean metric, for fixed dimensions. Unfortunately, neither Arora's nor Mitchell's algorithm for degree-2 trees extend to find degree-3 or degree-4 trees. As of now, there is no PTAS for finding degree-3 or degree-4 spanning trees [7]. Recently, Arora and Chang [8] have devised a QPTAS for the Euclidean degree-δ spanning tree problem in $\Re^d$. A QPTAS is an approximation scheme whose running time is quasi-polynomial, e.g., $n^{O(\log^c n)}$, where c is constant that may depend on the dimension d.

For any collection of points in the plane, Khuller, Raghavachari, and Young [69] showed that there exist degree-3 and degree-4 spanning trees whose weights are at most 1.5 and 1.25 times the weight of an MST, respectively. They also showed that for any $d \geq 3$, an arbitrary collection of points in $\Re^d$ contains a degree-3 spanning tree whose weight is at most 5/3 times the weight of an MST. The ratio for degree-4 spanning trees was improved to 1.175 by Jothi and Raghavachari [55]. In an independent and parallel work, Chan [18] improved the ratio for degree-4 spanning trees to 1.143. He also improved the ratio for degree-3 spanning trees to 1.402, for points in the plane, using an elegant recursive algorithm. For degree-3 spanning trees in higher dimensions, Chan showed that Khuller et al.'s [69] constant of 5/3 can be reduced to 1.633 by a more careful analysis.

8.1.2 Our contributions

The approximation results presented in this chapter were discovered independent of [8, 18]. We present an improved approximation analysis for Chan's degree-4 MST algorithm [18] thereby showing that, for any arbitrary collection of points in the plane, there always exists a degree-4 spanning tree of weight at most 1.1381, $(\sqrt{2}+2)/3$ to be exact, times the weight of a minimum spanning tree (MST). Our improvement assumes significance considering the amount of effort required to obtain it. The difficulties in improving Chan's ratio was overcome by using a more careful charging scheme complemented by a new savings analysis. Our new ideas were crucial in improving the ratio, without which improvement of the ratio might not

have been possible. In addition, we show our ratio is tight and cannot be improved unless a more global approach is considered, instead of just the local changes employed by the earlier papers. This is due to the fact there exists placement of points in the plane for which our ratio of $(\sqrt{2}+2)/3$ cannot be improved with just local changes.

We first show that the angle enclosed between any two sides of a triangle can be used to bound the weight on the third side in a precise manner. Of course, the third side can be expressed exactly using trigonometry, but this formulation is unsuitable due to its non-linear nature. Our method provides a linear approximation which is more easily analyzed. We then show that two MST edges that intersect at a point at an acute angle force edge-weight constraints on each other. The latter result plays an important role in the improvement of the ratio. If an MST is given as part of the input, Chan's algorithm runs in $O(n)$ time, and in $O(n \log n)$ time otherwise.

8.1.3 Related work

Papadimitriou and Vazirani [81] showed that all MSTs with integer coordinates as vertices have maximum degree of at most 5. Monma and Suri [78] showed that for any arbitrary collection of points in the plane, there exists an MST of degree 5. Fürer and Raghavachari [34] gave a polynomial time algorithm that finds a spanning tree or a Steiner tree of a given subset of vertices in a graph with degree at most one more than the degree of an optimal tree. Fischer [30] extended their results to weighted graphs that finds an MST whose degree is within a constant multiplicative factor plus an additive $O(\log n)$ of the optimal degree. Könemann and Ravi [70] presented a simple and elegant $O(1)$-approximation algorithm for the degree-δ MST which outputs a tree whose degree is $O(\delta + \log n)$. In a recent work [71], they presented a new algorithm with same results, but this time for the case when the bounds on degree of each vertex are individually specified.

Ravi et al. [85] presented bicriteria approximation algorithms for finding bounded-degree Steiner trees of low weight. Salowe [90], and Das and Heffernan [25] presented algorithms

for computing bounded-degree graph spanners. Robin and Salowe [87] studied bounds on the maximum degree of MSTs under various metrics. Fekete et al. [29] presented an approximation algorithm (for graphs satisfying triangle inequality) for finding a low-weight spanning tree in which bounds on the degrees of vertices are individually specified.

8.2 Preliminaries

Let $S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ be a given set of points in the plane. Let G be the complete graph induced by S with the weight of an edge between any two points defined to be the Euclidean distance between them. The terms points, nodes and vertices will be used interchangeably. Let $|uv|$ denote the Euclidean distance between the points u and v. Let $\angle ABC$ denote the angle formed at B between the line segments AB and BC. Let T_{min} be an MST of G rooted at one of its leaf nodes. Let $w(T_{min})$ denote its weight. Chan's algorithm starts with T_{min} as the initial tree T and decreases the degree of high degree nodes by local changes around it. Let x be a child of node v in T. Node x is defined to be a *biological* child of v if x is a child of v in T_{min}, else it is a *foster* child.

Lemma 8.2.1 ([18]) *If* $a_1, \ldots, a_m \geq 0$, *then*

$$\min\{a_1 x_1, \ldots, a_m x_m\} \leq \frac{1}{m}\text{H.M.}\{a_1, \ldots, a_m\}(x_1 + \cdots + x_m),$$

where H.M. *denotes the Harmonic Mean.*

8.3 Points in space

We first note some interesting geometric properties, including that of MSTs in $\Re^d$.

Lemma 8.3.1 *Let* AB *and* BC *be two edges incident on point* B. *Let* $x = |AB|, y = |AC|, z = |BC|$ *and* $\theta_1 = \angle ABC < 60°$. *Let* $z \geq y \geq x$. *Then, for a fixed* θ_1, $z - y$ *is minimum when* $x = y$.

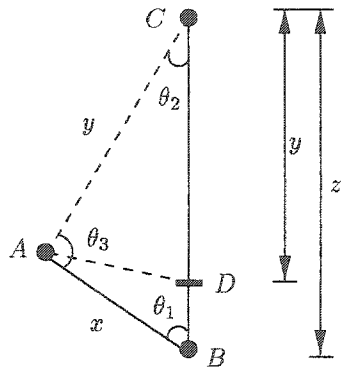


Figure 8.1. Geometric property

Proof. Since $z \geq y \geq x$, $\theta_2 = \angle ACB \leq \theta_1$. Let D be a point on BC such that $|CD| = y$ (see Fig. 8.1). Let $\theta_3 = \angle BAC$. Then $\angle ADB = 90° + \theta_2/2$ and $\angle BAD = 90° - \theta_1 - \theta_2/2$. Using simple trigonometry,

$$\frac{z-y}{\sin(90° - \theta_1 - \frac{\theta_2}{2})} = \frac{x}{\sin(90° + \frac{\theta_2}{2})}$$

It could be easily verified that for any fixed θ_1, $z - y$ is minimum when $\theta_1 = \theta_2$ ($x = y$). ∎

8.3.1 Strengthened triangle inequality

A standard method for decreasing the degree of a node B in a given tree T is by deleting an edge BC incident on B and adding the edge AC, where A is another neighbor of B, with $|AB| \leq |BC|$. The increase in the weight of the tree after such an interchange is $|AC| - |BC|$. In the following lemma, we prove an upper bound on this increased weight in terms of the angle enclosed between $|AB|$ and $|BC|$.

Lemma 8.3.2 ([18, 55]) *Let AB and BC be two edges incident on point B. Let $|AB| \leq |BC|$ and let $\theta = \angle ABC$. Then*

$$|AC| \leq F(\theta)|AB| + |BC|$$

where $F(\theta) = \sqrt{2(1 - \cos\theta)} - 1 = 2\sin(\theta/2) - 1$.

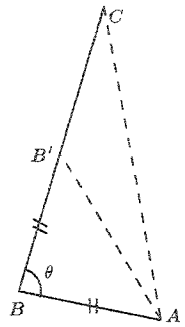


Figure 8.2. Strengthened triangle inequality

Proof. Let B' be a point on BC such that $|BB'| = |AB|$ (see Fig. 8.2). Using trigonometry,

$$\begin{aligned} |AB'| &= \sqrt{|AB|^2 + |BB'|^2 - 2|AB| \cdot |BB'| \cdot \cos\theta} \\ &= \sqrt{2(1-\cos\theta)}\,|AB| \end{aligned}$$

We use the above equation along with the triangle inequality:

$$\begin{aligned} |AC| &\leq |AB'| + |B'C| \\ &= |AB'| + |BC| - |BB'| \\ &= |AB'| + |BC| - |AB| \\ &= \left(\sqrt{2(1-\cos\theta)} - 1\right)|AB| + |BC| \end{aligned}$$

■

Our lemma provides a better and precise bound for the increase in the weight of the tree than just the triangle inequality. It can be verified that $|AC| \leq F(\theta)|AB| + |BC| \leq |AB| + |BC|$.

8.3.2 Bounds on edge weights of an MST

We now prove that MST edges that intersect at a node, at an acute angle, force edge-weight constraints on each other. The following lemma and its corollary are crucial in achieving the required bounds for Chan's algorithm.

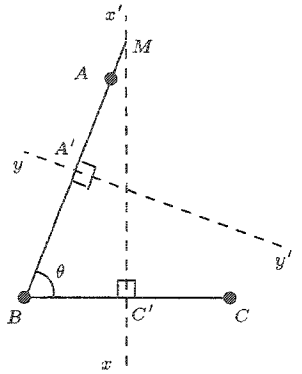


Figure 8.3. Bounds on edge weights of an MST

Lemma 8.3.3 *Let AB and BC be two edges that intersect at point B in an* MST *of set of points in $\Re^d$. Let $\theta = \angle ABC$. If $\theta < 90°$ then,*

$$2|BC|\cos\theta \leq |AB| \leq \frac{|BC|}{2\cos\theta}$$

Proof. Since AB and BC are MST edges, $|AB|, |BC| \leq |AC|$. Let A' be the midpoint of AB and let C' be the midpoint of BC (see Fig. 8.3). Let xx' and yy' be perpendicular bisectors of BC and AB respectively, passing through C' and A'. Since $\theta < 90°$, extending the line segment AB away from B will intersect xx', say at M. Observe that point A cannot lie on the right-hand side of line xx', since in that case $|AB| > |AC|$ which contradicts the fact that edge AB was chosen over edge AC to be an MST edge. Therefore $|AB| \leq |BM| = |BC|/2\cos\theta$. Also, by symmetry $|BC| \leq |AB|/2\cos\theta$. Combining these inequalities yields the lemma. ∎

Corollary 8.3.1 *Let AB and BC be two edges that intersect at point B and let AB be an* MST *edge and BC be a non-*MST *edge. Let $\theta = \angle ABC$. If $\theta < 90°$ then,*

$$|BC| \geq 2|AB|\cos\theta$$

8.4 Charging scheme

An easy way of reducing a node's degree is by removing one of the node's incident edges and introducing a new edge to connect the disconnected node to one of its siblings. The question

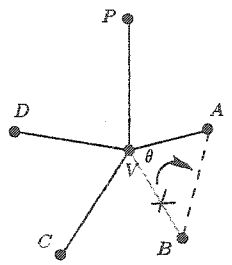


Figure 8.4. Reducing the degree of node V by transferring one child to another.

of which edge to remove, and to which child the disconnected node is to be connected arises as there are several different ways in which a node's degree could be decreased.

Lemma 8.4.1 *Let V be a degree-5 node in an* MST *T of a set of points in $\Re^2$. Let P be its parent and A, B, C, and D be its children. Let the degree of V be decreased from 5 to 4 by replacing BV by AB, where $|AV| \leq |BV|$ (as shown in Fig. 8.4). Let $\angle AVB = \theta$. Let k of the children of V be at a distance of $|AV|$ or more from V. Then the increase in the weight of the tree is at most*

$$\frac{F(\theta)}{k}\Big(|AV| + |BV| + |CV| + |DV|\Big)$$

Proof. The increase in weight of replacing BV by AB is $|AB| - |BV|$. By Lemma 8.3.2, we know that $|AB| - |BV| \leq F(\theta)|AV|$. Since $|AV| \leq |BV|$ and V has four children, we see that $2 \leq k \leq 4$. Since V has at least k children at distance $|AV|$ or more, $|AV| \leq (|AV| + |BV| + |CV| + |DV|)/k$. Combining this with the previous equation proves the lemma. ∎

Therefore, the increase in weight can be "charged" to the k edges from V to its children, and the charge on each of these edges is at most $F(\theta)/k$.

8.5 Degree-4 spanning trees

We first give a brief overview of Chan's degree-4 spanning tree algorithm [18] before proceeding to its approximation analysis.

8.5.1 Overview of Chan's algorithm

Chan's algorithm recursively transforms the rooted tree T into a new degree-4 spanning tree with the inductive hypothesis that the root v of tree T has degree 3 in the new tree. It should be noted here the Khuller et al.'s [69] approach is very similar except that their induction hypothesis was stronger allowing root v to have degree of only 2 in the new tree.

Let $\tau = 1.143$. Let T and T' be two subtrees, of an original MST, rooted at v and v', respectively. Let $T \nwarrow T'$ be a tree obtained by making v' a child of T. Chan's algorithm recursively transforms $T \nwarrow T'$ to a new tree such that v has degree at most 3 in the new tree and the new tree has weight at most $|vv'| + \tau(w(T) + w(T'))$. His algorithm basically chooses a convenient permutation $v_1, \ldots, v_k$ of the k children of v in T together with v' (with $T_1, \ldots, T_k$ being their corresponding subtrees) for transformation. Given below is Chan's algorithm, with its pictorial illustration in Fig. 8.5.

- If $k \leq 2$, then transform $T_1, \ldots, T_{k+1}$ recursively and keep the edges $vv_1, \ldots, vv_{k+1}$.
- If $k = 3$, then transform $T_1 \nwarrow T_2, T_3$ and T_4 recursively and keep the edges vv_1, vv_3, vv_4. The extra weight due to the transformation is $|v_1v_2| - |vv_2|$ and the weight of the new tree is at most $|vv_1| + |v_1v_2| + |vv_3| + |vv_4| + \tau \sum_{i=1}^{4} w(T_i)$.
- If $k = 4$, then transform $T_1 \nwarrow T_2, T_3 \nwarrow T_4$ and T_5 and keep the edges vv_1, vv_3, vv_5. The extra weight due to the transformation is $|v_1v_2| - |vv_2| + |v_3v_4| - |vv_4|$ and the weight of the new tree is at most $|vv_1| + |v_1v_2| + |vv_3| + |v_3v_4| + |vv_5| + \tau \sum_{i=1}^{5} w(T_i)$.

Chan shows that there always exists a permutation with extra weight at most $(\tau - 1) \sum_{v_i \neq v'} |vv_i|$ for $k = 3$ and $k = 4$. The following are the ratios he was able to obtain for each case of his algorithm.

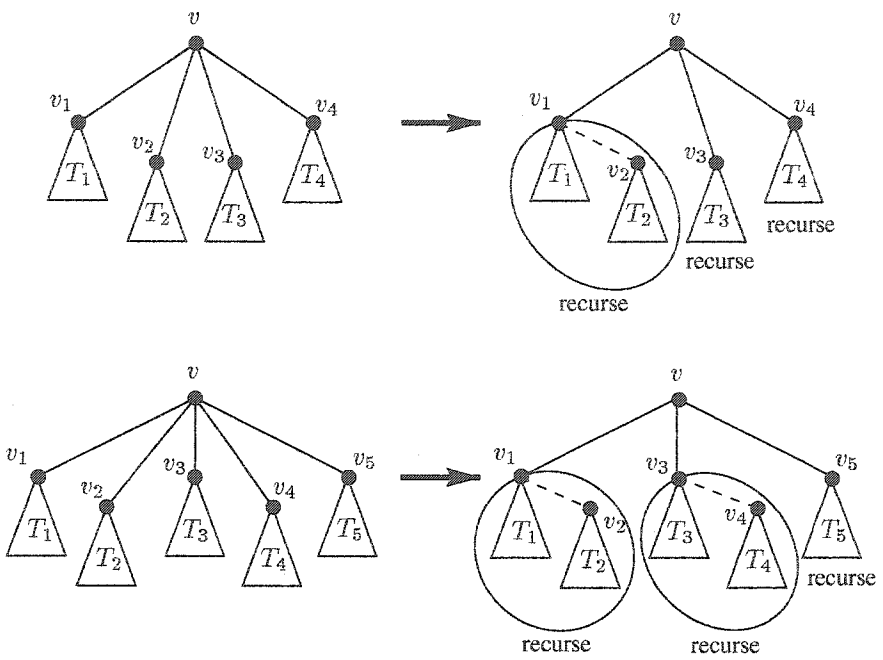


Figure 8.5. Chan's algorithm for degree-4 spanning trees

- **Case** $k = 3$. Approximation ratio $\leq (\sqrt{2}+2)/3 < 1.1381$.
- **Case** $k = 4$. Approximation ratio < 1.143

For more details and analysis of his algorithm, we refer the reader to [18]. His algorithm runs in $O(n)$ time, if an MST is given as input, and in $O(n \log n)$ time otherwise.

8.5.2 Improved approximation analysis

Let v be the vertex under consideration whose degree has to be reduced. Let v have k biological children and at most 1 foster child. When $k \leq 3$, Chan showed that the ratio is bounded by $(\sqrt{2}+2)/3 < 1.1381$. We were able to improve Chan's ratio of 1.143 by tackling the case, $k = 4$, for which his analysis is tight. As per his induction hypothesis, v has a total of at most 5 children (4 biological and 1 foster). In essence, our objective is to reduce the degree of v from 5 to 3 (degree induced on v by its parent is excluded, but counts in the final solution which makes v's degree to be 4). The algorithm reduces v's degree from 5 to 3 by performing

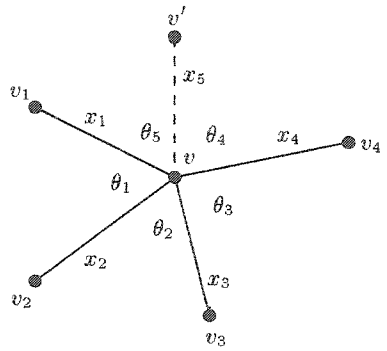


Figure 8.6. Notation for $k = 4$ analysis.

local changes around v.

To understand our analysis in a nutshell, consider Fig. 8.6 with v being the node whose degree we wish to reduce from 5 to 3, nodes v_1, v_2, v_3, v_4 being v's biological children, and v' being v's foster child. Suppose $\angle v_1vv' = \theta_5 \leq 60°$ (this is possible as vv' is a non-MST edge). Say, Chan's algorithm considers a transformation which involves replacing edges vv' with v_1v' and, say, vv_4 with v_3v_4. While Chan's analysis would directly charge the extra weight involved in such a transformation to the MST edges involved, our analysis proceeds by calculating the potential savings due to the replacement of edge vv' by v_1v' (notice that $\theta_5 \leq 60°$ and $vv' \geq vv_1$ as vv_1 was chosen over v_1v' to be the MST edge) and use it to absorb part of the extra charge incurred due to the other replacement ($vv_4 \rightarrow v_3v_4$).

Given below is our analysis for the case $k = 4$. To make the description easier, we introduce a function called "Reduce".

Reduce(v, x, y): Let vx and vy be two edges incident on point v. Reduce(v, x, y) replaces the edge $\max\{vx, vy\}$ by xy. In simple terms, v's degree is reduced by 1, by donating one of $\{x, y\}$.

Let v_1, v_2, v_3, v_4 be the biological children of v in T and let v' be the foster child of v. Let v and its children be placed as shown in Fig. 8.6. Let $|vv_1| = x_1, |vv_2| = x_2, |vv_3| = x_3, |vv_4| = x_4, |vv'| = x_5, \theta_1 = \angle v_1vv_2, \theta_2 = \angle v_2vv_3, \theta_3 = \angle v_3vv_4, \theta_4 = \angle v_4vv'$ and

$\theta_5 = \angle v'vv_1$. Since vv_1, vv_2, vv_3 and vv_4 are MST edges, $\theta_1, \theta_2, \theta_3, \theta_4 + \theta_5 \geq 60°$. Also, $\max\{\theta_1, \theta_2, \theta_3, \theta_4 + \theta_5\} \geq 120°$ considering the fact that one other MST edge, connecting v to its parent exists (not shown in figure). We consider three cases (the missing one is symmetric).

Case 1: $\theta_4 \leq 60°$ and $\theta_5 \leq 60°$. Handled the same way as in Chan's algorithm. For details, we refer the reader to [18]. Extra weight involved is bounded by 0.1331.

Case 2: $\theta_4 \geq 60°$ and $\theta_5 \leq 60°$. Since $\theta_5 \leq 60°$, obviously $x_1 \leq x_5$ (otherwise $|v'v_1| < |vv_1|$, which contradicts the fact that vv_1 was chosen over $v'v_1$ to be an MST edge).

Case 2.1: $\theta_1 \geq 120°$ or $\theta_4 + \theta_5 \geq 120°$.

Call Reduce(v, v_1, v'). Since $\theta_5 \leq 60°$, no extra weight is incurred due to the call. By Lemma 8.3.2, we have permutations with extra weight bounded by

$$F(\theta_2)\min\{x_2, x_3\}, F(\theta_3)\min\{x_3, x_4\}.$$

Thus, the minimum extra weight is at most the smaller of the following values:

$$F(\theta_2)x_2, \min\{F(\theta_2), F(\theta_3)\}x_3, F(\theta_3)x_4.$$

By Lemma 8.2.1, the minimum of these quantities is at most

$$\frac{1}{3}\text{H.M.}\{F(\theta_2), \min\{F(\theta_2), F(\theta_3)\}, F(\theta_3)\}(x_2 + x_3 + x_4).$$

Since $\theta_2 + \theta_3 \leq 180°$, the above coefficient is bounded by $\frac{1}{3}F(90°) = (\sqrt{2} + 2)/3 < 0.1381$.

Case 2.2: $\theta_2 \geq 120°$ (Case $\theta_3 \geq 120°$ is symmetric).

Case 2.2.1: x_3 or x_4 is the smallest among $\{x_1, x_2, x_3, x_4\}$.

- If $\theta_3 \leq 101.8°$, then call Reduce(v, v_1, v'). Since $\theta_5 \leq 60°$, no extra weight is incurred due to the call. Call Reduce(v, v_3, v_4). By Lemma 8.4.1, extra weight $F(\theta_3)\min\{x_3, x_4\}$ is charged to $\{vv_1, vv_2, vv_3, vv_4\}$ and is bounded by $0.1381(x_1 + x_2 + x_3 + x_4)$.

- Else if $\max\{x_1, x_2, x_4\} \neq x_4$, then choose θ_1 and θ_4. Note that $\theta_1 + \theta_4 + \theta_5 \leq 138.2°$. Call Reduce$(v, v_1, v_2)$ and Reduce(v, v_4, v'). By Lemma 8.4.1, if $\theta_4 \leq 69.36°$, the extra weights $F(\theta_1)\min\{x_1, x_2\}$ and $F(\theta_4)\min\{x_4, x_5\}$ are charged to $\{vv_1, vv_2\}$ and $\{vv_4\}$ respectively, else extra weights $F(\theta_1)\min\{x_1, x_2\}$ and $F(\theta_4)\min\{x_4 m, x_5\}$ are charged to $\min\{vv_1, vv_2\}$ and $\{\max\{vv_1, vv_2\}, vv_4\}$ respectively.
- Else $(\max\{x_1, x_2, x_4\} = x_4)$ if $\theta_4 \leq 69.36°$, then call Reduce(v, v_1, v_2) and Reduce(v, v_4, v_5). Since, $\theta_1 + \theta_4 + \theta_5 \leq 138.2°$ and $\theta_1, \theta_4 \geq 60°$, by Lemma 8.4.1, extra weights of at most $F(78.2°)\min\{x_1, x_2\}$ and $F(69.36°)\min\{x_4, x_5\}$ are charged to $\{vv_1, vv_2\}$ and $\{vv_4\}$, respectively, and is bounded by $0.1381(x_1 + x_2 + x_4)$.
- Else $\theta_5 \leq 8.84°$. Hence $\theta_1 + \theta_5 \leq 68.84°$ and $\theta_4 + \theta_5 \leq 78.2°$. Call Reduce$(v, v_2, v')$ and Reduce(v, v_1, v_4). By Lemma 8.4.1, extra weights $F(\theta_1+\theta_5)\min\{x_2, x_5\}$ and $F(\theta_4+\theta_5)\min\{x_1, x_4\}$ are charged to $\{vv_2\}$ and $\{vv_1, vv_4\}$, respectively, and is bounded by $0.1381(x_1 + x_2 + x_4)$.

Case 2.2.2: x_3 or x_4 is the second smallest among $\{x_1, x_2, x_3, x_4\}$.

- If $\theta_3 \leq 90°$, then call Reduce(v, v_1, v'). Since $\theta_5 \leq 60°$, no extra weight is incurred due to the call. Call Reduce(v, v_3, v_4). By Lemma 8.4.1, extra weight $F(\theta_3)\min\{x_3, x_4\}$ is charged to $\{vv_3, vv_4\}$ and the longest of $\{vv_1, vv_2\}$, and is bounded by $0.1381(x_1 + x_2 + x_3 + x_4)$.
- Else $\theta_1 + \theta_4 + \theta_5 \leq 150°$ and hence $\theta_5 \leq 30°$.
  - If $x_1 = \min\{x_1, x_2\}$, w.l.o.g. let $x_2 \leq x_4$. Since $\min\{\theta_1, \theta_4 + \theta_5\} \leq \frac{240°-\theta_3}{2}°$, by Lemma 8.3.3, $x_1 \geq 2x_2\cos(\frac{240°-\theta_3}{2})$. Call Reduce$(v, v_1, v')$. Since $\theta_5 \leq 30°$, no extra weight is incurred due to the call. Also, since vv_1 is an MST edge, $x_5 > x_1$ and thus, by Corollary 8.3.1, $x_5 \geq 2x_1\cos\theta_5$. By Lemma 8.3.1, $|vv'| - |v_1v'|$ results in savings of at least $(2\cos\theta_5 - 1)x_1$. Let T_{before} be the subtree induced by nodes

v, v_1, v_2, v_3, v_4 and v' and let T_{after} be the subtree induced by nodes v, v_1, v_2, v_3 and v_4. Clearly, as per our argument above, the weight of T_{after} is $(2\cos\theta_5 - 1)x_1$ less than that of T_{before}. Since our goal is to bound the extra weight, incurred during local transformations, to within 0.1381 times the MST weight, as per our charging policy, every MST edge e can be charged an extra weight of $0.1381e$. The savings obtained, due to the transformation from T_{before} to T_{after}, is equivalent to having atleast $\frac{2\cos 30° - 1}{0.1381}$ extra vv_1 edges, each of which can be charged $0.1381x_1$. In other words, it is as if we have at least an additional $(\frac{2\cos 30° - 1}{0.1381})vv_1$ to charge. Call Reduce(v, v_3, v_4). By Lemma 8.4.1, extra weight $F(\theta_3)\min\{x_3, x_4\}$ is charged to $\{vv_1, vv_2, vv_3, vv_4\}$ and $(\frac{2\cos 30° - 1}{0.1381})vv_1$, and is given by

$$\frac{F(\theta_3)(x_1 + x_2 + x_3 + x_4 + \frac{2\cos 30° - 1}{0.1381}x_1)}{3 + 2\cos(\frac{240° - \theta_3}{2})\left(1 + \frac{2\cos 30° - 1}{0.1381}\right)}$$

which is bounded by $0.079(x_1 + x_2 + x_3 + x_4 + \frac{2\cos 30° - 1}{0.1381}x_1)$.

- Else $(x_1 \neq \min\{x_1, x_2\})$ the analysis proceeds in the same way as done in the previous step, except that the extra weight $F(\theta_3)\min\{x_3, x_4\}$ is charged to $\{vv_1, vv_3, vv_4\}$ and $(\frac{2\cos 30° - 1}{0.1381})vv_1$, and is given by

$$\frac{F(\theta_3)(x_1 + x_2 + x_3 + x_4 + \frac{2\cos 30° - 1}{0.1381}x_1)}{3 + \frac{1}{0.1381}(2\cos 30° - 1)}$$

which is bounded by $0.048(x_1 + x_2 + x_3 + x_4 + \frac{2\cos 30° - 1}{0.1381}x_1)$.

Case 2.2.3: $x_3, x_4 \geq x_1, x_2$.

- If $\theta_3 \leq 79.29°$, Call Reduce(v, v_1, v'). Since $\theta_5 \leq 60°$, no extra weight is incurred due to the call. Call Reduce(v, v_3, v_4). By Lemma 8.4.1, extra weight $F(\theta_3)\min\{x_3, x_4\}$ is charged to $\{vv_3\}$ and $\{vv_4\}$, and is bounded by $0.1381(x_3 + x_4)$.
- Else if $\theta_4 \leq 69.36°$ and $\theta_1 \leq 90°$, then call functions Reduce(v, v_4, v_5) and Reduce(v, v_1, v_2). By Lemma 8.4.1, extra weights $F(\theta_4)\min\{x_4, x_5\}$

and $F(\theta_1)\min\{x_1, x_2\}$ are charged to vv_4 and $\{vv_1, vv_2, vv_3\}$, respectively, and is bounded by $0.1381(x_2 + x_2 + x_3 + x_4)$.

- Else if $\theta_4 \leq 69.36°$ and $\theta_1 > 90°$, then $\theta_5 \leq 10.71°$ and $60° \leq \theta_4 + \theta_5 \leq 70.91°$. Since $\theta_2 + \theta_4 + \theta_5 = 360° - \theta_1 - \theta_3 \leq 190.71°$, by Lemma 8.3.3, $x_1 \geq 2x_4 \cos(190.71° - \theta_2)$. Call Reduce$(v, v_1, v')$. Since $\theta_5 \leq 10.71°$, no extra weight is incurred due to the call. Also, since vv_1 is an MST edge, $x_5 > x_1$ and thus, by Corollary 8.3.1, $x_5 \geq 2x_1 \cos\theta_5$. By Lemma 8.3.1, $|vv'| - |v_1v'|$ results in savings of at least $(2\cos\theta_5 - 1)x_1$. In other words, it is as if we have at least an additional $(\frac{2\cos 10.71° - 1}{0.1381})vv_1$ to charge. Call Reduce(v, v_2, v_3). By Lemma 8.4.1, the extra weight $F(\theta_2)\min\{x_2, x_3\}$ is charged to $\{vv_1, vv_2, vv_3, vv_4\}$ and $(\frac{2\cos 10.71° - 1}{0.1381})vv_1$, and is given by

$$\frac{F(\theta_2)(x_1 + x_2 + x_3 + x_4 + \frac{2\cos 10.71° - 1}{0.1381}x_1)}{3 + 2\cos(190.71° - \theta_2)\left(1 + \frac{2\cos 10.71° - 1}{0.1381}\right)}$$

which is bounded by $0.089(x_1 + x_2 + x_3 + x_4 + \frac{2\cos 10.71° - 1}{0.1381}x_1)$.

- Else $(\theta_4 > 69.36°)$ $\theta_5 \leq 31.35°$.
  - If $\theta_5 \leq 11°$, then since $\theta_1 + \theta_4 + \theta_5 = 360° - \theta_3 - \theta_2 \leq 280.71° - \theta_2$ and $x_2 \leq x_4$, by Lemma 8.3.3, $x_1 \geq 2x_2 \cos(\frac{280.71° - \theta_2}{2})$. Call Reduce$(v, v_1, v')$. Since $\theta_5 \leq 11°$, no extra weight is incurred due to the call. Also, since vv_1 is an MST edge, $x_5 > x_1$ and thus, by Corollary 8.3.1, $x_5 \geq 2x_1 \cos\theta_5$. By Lemma 8.3.1, $|vv'| - |v_1v'|$ results in savings of at least $(2\cos\theta_5 - 1)x_1$. In other words, it is as if we have at least an additional $(\frac{2\cos 11° - 1}{0.1381})vv_1$ to charge. Call Reduce(v, v_2, v_3). By Lemma 8.4.1, extra weight $F(\theta_2)\min\{x_2, x_3\}$ is charged to $\{vv_1, vv_2, vv_3, vv_4\}$ and $(\frac{2\cos 11° - 1}{0.1381})vv_1$, and is given by

$$\frac{F(\theta_2)(x_1 + x_2 + x_3 + x_4 + \frac{2\cos 11° - 1}{0.1381}x_1)}{3 + 2\cos(\frac{280.71° - \theta_2}{2})\left(1 + \frac{2\cos 11° - 1}{0.1381}\right)}$$

which is bounded by $0.13(x_1 + x_2 + x_3 + x_4 + \frac{2\cos 11° - 1}{0.1381}x_1)$.

- Else if $11° < \theta_5 \leq 25°$, then since $\theta_1 = 360° - \theta_2 - \theta_3 - \theta_4 - \theta_5 \leq 200.35 - \theta_2$, by Lemma 8.3.3, $x_1 \geq 2x_2 \cos(200.35° - \theta_2)$. Call Reduce$(v, v_1, v')$. Since $\theta_5 \leq 25°$, no extra weight is incurred due to the call. Also, since vv_1 is an MST edge, $x_5 > x_1$ and thus, by Corollary 8.3.1, $x_5 \geq 2x_1 \cos\theta_5$. By Lemma 8.3.1, $|vv|' - |v_1v'|$ results in savings of at least $(2\cos\theta_5 - 1)x_1$. In other words, it is as if we have at least an additional $(\frac{2\cos 25° - 1}{0.1381})vv_1$ to charge. Call Reduce(v, v_2, v_3). By Lemma 8.4.1, extra weight $F(\theta_2)\min\{x_2, x_3\}$ is charged to vv_1, vv_2, vv_3, vv_4 and $(\frac{2\cos 25° - 1}{0.1381})vv_1$, and is given by

$$\frac{F(\theta_2)(x_1 + x_2 + x_3 + x_4 + \frac{2\cos 25° - 1}{0.1381}x_1)}{3 + 2\cos(200.35° - \theta_2)\left(1 + \frac{2\cos 25° - 1}{0.1381}\right)}$$

which is bounded by $0.138(x_1 + x_2 + x_3 + x_4 + \frac{2\cos 25° - 1}{0.1381}x_1)$.

- Else $(25° < \theta_5 \leq 31.35°)$, since $\theta_1 = 360° - \theta_2 - \theta_3 - \theta_4 - \theta_5 \leq 186.35 - \theta_2$, by Lemma 8.3.3, $x_1 \geq 2x_2 \cos(186.35° - \theta_2)$. Call Reduce$(v, v_1, v')$. Since $\theta_5 \leq 31.25°$, no extra weight is incurred due to the call. Also, since vv_1 is an MST edge, $x_5 > x_1$ and thus $x_5 \geq 2x_1 \cos\theta_5$. By Lemma 8.3.1, $|vv'| - |v_1v'|$ results in savings of at least $(2\cos\theta_5 - 1)x_1$. In other words, it is as if we have at least an additional $(\frac{2\cos 31.35° - 1}{0.1381})vv_1$ to charge. Call Reduce(v, v_2, v_3). By Lemma 8.4.1, extra weight $F(\theta_2)\min\{x_2, x_3\}$ is charged to $\{vv_1, vv_2, vv_3, vv_4\}$ and $(\frac{2\cos 31.35° - 1}{0.1381})vv_1$, and is given by

$$\frac{F(\theta_2)(x_1 + x_2 + x_3 + x_4 + \frac{2\cos 31.35° - 1}{0.1381}x_1)}{3 + 2\cos(186.35° - \theta_2)\left(1 + \frac{2\cos 31.35° - 1}{0.1381}\right)}$$

which is bounded by $0.1(x_1 + x_2 + x_3 + x_4 + \frac{2\cos 31.35° - 1}{0.1381}x_1)$.

Case 3: $\theta_4 \geq 60°$ and $\theta_5 \geq 60°$.

Case 3.1: $\theta_5 \leq 69.36°$ (Case $\theta_4 \leq 69.36°$ is symmetric).

Call Reduce(v, v_1, v'). By Lemma 8.4.1, extra weight $F(\theta_5)\min\{x_1, x_5\}$ is charged to vv_1 and is bounded by $0.1381x_1$. By Lemma 8.3.2, we have permutations with extra weight bounded by

$$F(\theta_2)\min\{x_2, x_3\}, F(\theta_3)\min\{x_3, x_4\}.$$

By Lemma 8.2.1, the minimum of these quantities is at most

$$\frac{1}{3}\text{H.M.}\{F(\theta_2), \min\{F(\theta_2), F(\theta_3)\}, F(\theta_3)\}(x_2 + x_3 + x_4).$$

Since $\theta_2 + \theta_3 \leq 180°$, the above coefficient is bounded by $\frac{1}{3}f(90°) = (\sqrt{2} + 2)/3 < 0.1381$.

Case 3.2: $\theta_4 + \theta_5 > 138.72°$. W.l.o.g., let $x_1 \leq x_4$.

Case 3.2.1: $\theta_5 \leq 79.29°$.

- If $\theta_2 \leq 79.29°$, then call Reduce(v, v_1, v_5) and Reduce(v, v_2, v_3). By Lemma 8.4.1, extra weights $F(\theta_5)x_1$ and $F(\theta_2)\min\{x_2, x_3\}$ are charged to $\{vv_1, vv_4\}$ and $\{vv_2, vv_3\}$, respectively, and is bounded by $0.1381(x_1 + x_2 + x_3 + x_4)$.
- Else $\theta_1 + \theta_3 \leq 141.99°$.
  - If $\theta_1, \theta_3 \leq 79.29°$, then call Reduce$(v, v_1, v_2)$ and Reduce(v, v_3, v_4). Extra weights $F(\theta_1)\min\{x_1, x_2\}$ and $F(\theta_3)\min\{x_3, x_4\}$ are charged to $\{vv_1, vv_2\}$ and $\{vv_3, vv_4\}$, respectively (by Lemma 8.4.1), and is bounded by $0.1381(x_1 + x_2 + x_3 + x_4)$.
  - Else $\theta_1 + \theta_3 \geq 139.29°$ and $\theta_2 \leq 81.99°$. W.l.o.g., let $\theta_3 > 79.29°$. Then $\theta_1 = 360° - \theta_2 - \theta_3 - (\theta_4 + \theta_4) \leq 360° - 79.29° - 79.29° - (138.72°) = 62.7°$ and $\theta_2 + \theta_3 = 360° - \theta_1 + (\theta_4 + \theta_5) \leq 360° - 60° - (138.72°) = 161.28°$. Call Reduce$(v, v_1, v_2)$ and Reduce(v, v_3, v_4). By Lemma 8.4.1, extra weight $F(\theta_1)\min\{x_1, x_2\}$ is charged to $\{vv_1\}$ and is bounded by $0.05x_1$ and extra weight $F(\theta_3)\min\{x_3, x_4\}$ is charged

to $\{vv_2, vv_3, vv_4\}$ and is given by

$$\frac{F(\theta_3)(x_2 + x_3 + x_4)}{2 + 2\cos(161.28° - \theta_3)}$$

which is bounded by $0.132(x_2 + x_3 + x_4)$.

Case 3.2.2: $\theta_5 > 79.29°$.

- If $\theta_1, \theta_3 \leq 79.29°$, then call Reduce$(v, v_1, v_2)$ and Reduce(v, v_3, v_4). By Lemma 8.4.1, extra weights $F(\theta_1)\min\{x_1, x_2\}$ and $F(\theta_3)\min\{x_3, x_4\}$ are charged to $\{vv_1, vv_2\}$ and $\{vv_3, vv_4\}$, respectively, and is bounded by $0.1381(x_1 + x_2 + x_3 + x_4)$.
- Else $\theta_1 + \theta_3 > 139.29°$. W.l.o.g., let $\theta_3 > 79.29°$. Then $\theta_1 = 360° - (\theta_2 + \theta_3) + \theta_4 + \theta_5 = 360° - (139.29°) - 69.36° - 79.29° \leq 72.06°$.
  - If $69.36° < \theta_1 \leq 71.06°$ ($\theta_2 + \theta_3 \leq 141.99°$), then call Reduce$(v, v_1, v_2)$ and Reduce(v, v_3, v_4). Extra weight $F(\theta_1)\min\{x_1, x_2\}$ is charged to $\{vv_1\}$ and a third of $\{vv_2\}$ (by Lemma 8.4.1) and is given by

    $$\frac{F(\theta_1)(x_1 + \frac{1}{3}x_2)}{4/3}$$

    which is bounded by $0.135(x_1 + \frac{1}{3}x_2)$. Again by Lemma 8.4.1, extra weight $F(\theta_3)\min\{x_3, x_4\}$ is charged to $\{vv_3, vv_4\}$ and two thirds of $\{vv_2\}$ and is given by

    $$\frac{F(\theta_3)(\frac{2}{3}x_1 + x_3 + x_4)}{2 + 2\cos(141.99° - \theta_3) \times \frac{2}{3}}$$

    which is bounded by $0.12(\frac{2}{3}x_2 + x_3 + x_4)$.
  - Else if $61.35° < \theta_1 \leq 69.36°$ ($\theta_2 + \theta_3 \leq 150°$), then call Reduce$(v, v_1, v_2)$ and Reduce(v, v_3, v_4). By Lemma 8.4.1, extra weight $F(\theta_1)\min\{x_1, x_2\}$ is charged to $\{vv_1\}$ and is bounded by $1.1381x_1$ and extra weight $F(\theta_3)\min\{x_3, x_4\}$ is charged to $\{vv_2, vv_2, vv_3\}$ and is given by

    $$\frac{F(\theta_3)(x_2 + x_3 + x_4)}{2 + 2\cos(150° - \theta_3)}$$

    which is bounded by $0.1381(x_1 + x_2 + x_3)$.

- Else ($\theta_1 \leq 61.35°$) $\theta_2 + \theta_3 \leq 151.35°$. Call Reduce$(v, v_1, v_2)$ and Reduce(v, v_3, v_4). By Lemma 8.4.1, extra weight $F(\theta_1)\min\{x_1, x_2\}$ is charged to one fifth of $\{vv_1\}$ and is given by

$$\frac{F(\theta_1)(\frac{1}{5}x_1)}{1/5}$$

which is bounded by $0.102(\frac{1}{5}x_1)$. Again by Lemma 8.4.1, extra weight $F(\theta_3)\min\{x_3, x_4\}$ is charged $\{vv_2, vv_3, vv_4\}$ and four fifths of $\{vv_1\}$, and is given by

$$\frac{F(\theta_3)(\frac{4}{5}x_1 + x_2 + x_3 + x_4)}{2 + 2\cos(151.35° - \theta_3) + 2\cos 61.35°(2\cos(151.35° - \theta_3)) \times \frac{4}{5}}$$

which is bounded by $0.12(\frac{4}{5}x_1 + x_2 + x_3 + x_4)$

For convenience, we used 0.1381 instead of $(\sqrt{2} - 1)/3$, wherever it was used. The following theorem follows from the fact that the ratio is bounded by $(\sqrt{2} + 2)/3 < 1.1381$ in all cases.

Theorem 8.5.1 *For any arbitrary collection of points in the Euclidean plane, there always exists a degree-4 spanning tree of weight at most* $(\sqrt{2} + 2)/3$ *times the weight of an* MST.

8.6 Remarks

By presenting an improved approximation analysis for Chan's degree-4 MST algorithm, we showed that, for any arbitrary collection of points, there always exists a degree-4 spanning tree of weight at most $(\sqrt{2} + 2)/3 < 1.1381$ times the weight of an MST. Our ratio for degree-4 spanning trees cannot be improved unless a more global approach is considered, instead of just the local changes that we considered, as there exists placement of points for the case $k = 3$ (see Fig. 8.7), such that doing local changes alone does not reduce the ratio. There exists degree-4 and degree-3 trees (regular pentagon and square with an extra point at the center) whose weights are at most $(2\sin 36° + 4)/5$ and $(\sqrt{2} + 3)/4$ times the weight of an MST [29], respectively. It should be interesting to see whether better approximation algorithms can be developed to achieve ratios anywhere close to these lower bounds.

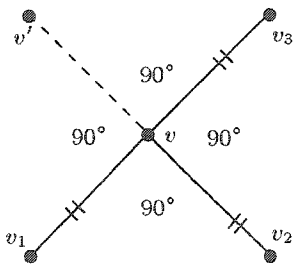


Figure 8.7. Tight case.

CHAPTER 9
CONCLUSIONS

9.1 Summary

In this thesis, we presented approximation algorithms for several NP-hard graph problems. The problems we studied have applications in telecommunications and network design, scheduling, and vehicle-routing.

Chapters 2 to 6, we considered single-sink edge installation problems. These problems have a common objective of installing cables along the edges of a given graph, such that the flow from all the source nodes can be simultaneously routed to the designated sink node(s). The goal is to minimize the cost of the network being built, and at the same time meeting the given capacity and connectivity constraints. Given below is the list of problems we studied, and the results we obtained.

- We presented a simple 4-approximation algorithm for extensively studied CMST problem. We also showed how to obtain a ratio of 2 for instances with unit-weight vertices and capacity $k = 3$ or 4.

- We studied the CMStT problem, a generalization of the CMST problem, in which only some nodes have non-zero demands. For this problem, we presented an approximation algorithm, which guarantees a ratio of $\gamma\rho_{ST} + 2$. We also obtain a ratio of $\gamma + 2$ for the CMST problem. For instances in the plane, under the L_p norm, with uniform vertex-weights, we presented a $(\frac{7}{5}\rho_{ST} + \frac{3}{2})$-approximation algorithm for the CMStT problem. This result translates into a ratio a 2.9 for the CMST problem under the same setting. Our ratio of 2.9 solves the long-standing open problem of obtaining a ratio any better

than 3 for this case. We also presented approximation results for several variants of the CMST problem.

- Based on the theoretical insights obtained from our approximation algorithms for the CMStT problem, we proposed a modification of the Esau and Williams' CMST heuristic [27]. Experimental evaluation of the modified heuristic on benchmark instances indicates that the proposed modification helps obtain better solutions, with improvements of up to 17%.

- We presented approximation results for the "survivable network" variant of the CMST problem, the CMSN problem, which requires that the final network satisfies certain connectivity constraints.

- We presented a 145.6-approximation algorithm for the SSBB problem, a generalization of the CMStT problem, in which we are are given K cable types, each having a capacity and cost per unit length. We also presented approximation results for the DSSBB problem, a variant of the SSBB problem, which does not require that the flow from a source follow a single path to the sink.

- We presented approximation results for the GKTR problem, a generalization of the KTR problem. Unlike the KTR problem, the repairtime at a customer's location in the GKTR problem may be non-zero. We also studied a special case in which all the customers having the same repairtime. For a related BLP problem, we presented a bicriteria approximation algorithm.

- We presented an improved approximation analysis for Chan's degree-4 MST algorithm [18] for points in the plane. Our approximation analysis shows that there exists a degree-4 MST whose cost is within 1.1381 times the cost of an MST.

9.2 Open questions

- For the CMST problem, the lower bounds that we used are weak and one can easily construct an instance whose optimal CMST cost is at least $\Omega(n/k)$ times the cost of an MST. Similarly one can construct an instance whose optimal CMSN cost is at least $\Omega(k)$ times the cost of the spoke lower bound (Lemma 2.2.1). This raises a question whether one could better lower bounds, which may help design better algorithms.

- An interesting open problem is to approximate CMSTs for general graphs (whose edges do not satisfy triangle inequality). We do not know at this time how to obtain a non-trivial approximation ratio for general graphs.

- With the currently available lower bounds for the CMST problem, one cannot hope to obtain a ratio any better than 2. But, our ratios for the CMST problem are around 3. Is there a way to bridge this gap by either finding better lower bounds or designing better algorithms?

- Unlike the CMST problem, for which the approximation ratio for uniform vertex-weighted graphs is smaller than that for non-uniform vertex-weighted graphs, our CMSN algorithm guarantees an approximation ratio of 4 for all graphs. Are there better approximations for the CMSN problem in graphs with uniform vertex-weighs? Also, are there better approximations for geometric graphs?

- Are there better lower bounds for the SSBB problem that could help obtain better approximation ratios? Is our analysis for our SSBB algorithm tight?

- What happens if the final network is required to be a tree in the SSBB problem? Can we still guarantee the same ratio with this restriction?

- Is there a way to solve the GKTR problem using the KTR algorithm so that the approximation ratio for the GKTR problem remains the same as that for the KTR problem?

- Is the degree-4 MST problem NP-hard?

- There exists a degree-4 tree whose cost is $(\sqrt{2}+3)/4$ times than that of an MST. Is there a way to reduce our ratio of 1.1381 to anywhere close to $(\sqrt{2}+3)/4$?

BIBLIOGRAPHY

[1] R.K. Ahuja, J.B. Orlin, and D. Sharma. A composite neighborhood search algorithm for the capacitated minimum spanning tree problem. *Oper. Res. Letters*, 31:85–94, 2003.

[2] K. Altinkemer and B. Gavish. Heuristics for unequal weight delivery problems with a fixed error guarantee. *Oper. Res. Letters*, 6:149–158, 1987.

[3] K. Altinkemer and B. Gavish. Heuristics with constant error guarantees for the design of tree networks. *Management Science*, 34:331–341, 1988.

[4] A. Amberg, W. Domschke, and S. Voß. Capacitated minimum spanning trees: Algorithms using intelligent search. *Combinatorial Optimization: Theory and Practice*, 1:9–39, 1996.

[5] A. Archer, A. Levin, and D.P. Williamson. Faster approximation algorithms for the minimum latency problem. In *Proc. 14th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 88–96, 2003.

[6] S. Arora. Polynomial-time approximation schemes for Euclidean TSP and other geometric problems. *JACM*, 45:753–782, 1998.

[7] S. Arora. Approximation schemes for NP-hard geometric optimization problems: A survey. *Math. Programming*, 97:43–69, 2003.

[8] S. Arora and K.L. Chang. Approximation schemes for degree-restricted MST and Red-Blue separation problem. In *Proc. 30th Intl. Colloquium on Automata, Languages and Programming (ICALP)*, pages 176–188, 2003.

[9] S. Arora and G. Karakostas. A $2 + \epsilon$ approximation for the k-MST problem. In *Proc. 11th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 754–759, 2000.

[10] B. Awerbuch and Y. Azar. Buy-at-bulk network design. In *Proc. 38th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 542–547, 1997.

[11] Y. Bartal. *Competitive analysis of distributed on-line problems-distributed paging*. PhD thesis, Tel-Aviv University, Israel, 1994.

[12] J.E. Beasley. OR-library. `http://www.ms.ic.ac.uk/info.html`.

[13] A. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan, and M. Sudan. The minimum latency problem. In *Proc. ACM 26th Symp. on Theory of Computing (STOC)*, pages 163–171, 1994.

[14] A. Blum, P. Chalasani, and S. Vempala. A constant-factor approximation for the k-MST problem in the plane. In *Proc. 27th ACM Symp. on Theory of Computing (STOC)*, pages 294–302, 1995.

[15] A. Blum, R. Ravi, and S. Vempala. A constant-factor approximation algorithm for the k-MST problem. In *Proc. 28th ACM Symp. on Theory of Computing (STOC)*, pages 442–448, 1996.

[16] R.R. Boorstyn and H. Frank. Large-scale network topological optimization. *IEEE Trans. on Communications*, 25:29–47, 1977.

[17] R.S. Cahn. *Wide area network design: Concepts and tools for optimization.* Morgan Kaufmann, 1998.

[18] T. Chan. Euclidean bounded-degree spanning tree ratios. In *Proc. 19th ACM Symp. of Computational Geometry (SOCG)*, pages 11–19, 2003.

[19] K.M. Chandy and T. Lo. The capacitated minimum spanning tree. *Networks*, 3:173–181, 1973.

[20] K.M. Chandy and R.A. Russell. The design of multipoint linkages in a teleprocessing tree network. *IEEE Trans. on Communications*, 21:1062–1066, 1972.

[21] K. Chaudhuri, B. Godfrey, S. Rao, and K. Talwar. Paths, tours, and minimum latency tours. In *Proc. 44th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 36–45, 2003.

[22] C. Chekuri and A. Kumar. A note on the k-traveling repairmen problem, 2003. Manuscript.

[23] N. Christofides. Worst case analysis of a new heuristic for the traveling salesman problem. Technical report, GSIA, Carnegie-Mellon University, 1976.

[24] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, McGraw-Hill, New York, 2nd edition, 2002.

[25] G. Das and P.J. Heffernan. Constructing degree-3 spanners with other sparseness properties. *Intl. J. of Foundations of Computer Science*, 7:121–136, 1996.

[26] D. Elias and M.J. Ferguson. Topological design of multipoint teleprocessing networks. *IEEE Trans. on Communications*, 22:1753–1762, 1974.

[27] L.R. Esau and K.C. Williams. On teleprocessing system design. *IBM Systems J.*, 5:142–147, 1966.

[28] J. Fakcharoenphol, C. Harrelson, and S. Rao. The k-traveling repairman problem. In *Proc. 14th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 655–664, 2003.

[29] S.P. Fekete, S. Khuller, M. Klemmstein, B. Raghavachari, and N. Young. A network-flow technique for finding low-weight bounded-degree spanning trees. *J. Algorithms*, 24:310–324, 1997.

[30] T. Fischer. Optimizing the degree of minimum weight spanning trees. Technical report, Department of Computer Science, Cornell University, April 1993.

[31] M. Fischetti, G. Laporte, and S. Martello. The delivery man problem and cumulative matroids. *Oper. Res.*, 41:1065–1064, 1993.

[32] H. Frank, I.T. Frisch, R. Van Slyke, and W.S. Chou. Optimal design of centralized computer design network. *Networks*, 1:43–57, 1971.

[33] G.N. Frederickson and J. JáJá. On the relationship between the biconnectivity augmentation and traveling salesman problems. *Theoretical Computer Science*, 19(2):189–201, 1982.

[34] M. Fürer and B. Raghavachari. Approximating the minimum-degree Steiner tree to within one of optimal. *J. Algorithms*, 17:409–423, 1994.

[35] M.R. Garey and D.S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. W.H. Freeman, San Francisco, 1979.

[36] N. Garg. A 3-approximation for the minimum tree spanning k vertices. In *Proc. 37th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 302–309, 1996.

[37] N. Garg, R. Khandekar, G. Konjevod, R. Ravi, F.S. Salman, and A. Sinha. On the integrality gap of a natural formulation of the single-sink buy-at-bulk network design problem. In *Proc. 8th Intl. Conf. on Integer Programming and Combinatorial Optimization (IPCO)*, pages 170–184, 2001.

[38] B. Gavish. Topological design of centralized computer networks—formulations and algorithms. *Networks*, 12:355–377, 1982.

[39] B. Gavish. Formulations and algorithms for the capacitated minimal directed tree problem. *J. ACM*, 30:118–132, 1983.

[40] B. Gavish. Augmented Lagrangean based algorithms for centralized network design. *IEEE Trans. on Communications*, 33:1247–1257, 1985.

[41] B. Gavish. Topological design of telecommunication networks—local access design methods. *Annals of Oper. Res.*, 33:17–71, 1991.

[42] B. Gavish and K. Altinkemer. Parallel savings heuristics for the topological design of local access tree networks. In *Proc. IEEE INFOCOM*, pages 130–139, 1986.

[43] M. Goemans and J. Kleinberg. An improved approximation ratio for the minimum latency problem. In *Proc. 7th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 152–158, 1996.

[44] M. Goemans and D. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Computing*, 24:296–317, 1995.

[45] L. Gouveia. A $2n$ constraint formulation for the capacitated minimal spanning tree problem. *Oper. Res.*, 43:1995, 130-141.

[46] L. Gouveia. A comparison of directed formulations for the capacitated minimal spanning tree problem. *Telecommunication Systems*, 1:51–76, 1993.

[47] L. Gouveia and J. Paixão. Dynamic programming based heuristics for the topological design of local access networks. *Annals of Oper. Res.*, 33:305–327, 1991.

[48] S. Guha, A. Meyerson, and K. Mungala. A constant factor approximation for the single sink edge installation problems. In *Proc. 33rd ACM Symp. on Theory of Computing (STOC)*, pages 383–399, 2001.

[49] A. Gupta, A. Kumar, and T. Roughgarden. Simpler and better approximation algorithms for network design. In *Proc. 35th ACM Symp. on Theory of Computing (STOC)*, pages 365–372, 2003.

[50] L.A. Hall. Experience with a cutting plane approach for the capacitated spanning tree problem. *INFORMS J. on Computing*, 8:219–234, 1996.

[51] R. Hassin, R. Ravi, and F.S. Salman. Approximation algorithms for capacitated network design problems. In *3rd Intl. Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, pages 167–176, 2000.

[52] A. Itai, C.H. Papadimitriou, and J.L. Szwarcfiter. Hamilton paths in grid graphs. *SIAM J. Computing*, 11:676–686, 1982.

[53] R. Jothi and B. Raghavachari. Survivable network design: the capacitated minimum spanning network problem. *Information Processing Letters*. in press.

[54] R. Jothi and B. Raghavachari. Design of local access networks. In *Proc. 15th Intl Conf. on Parallel and Distributed Computing and Systems (PDCS)*, pages 883–888, 2003.

[55] R. Jothi and B. Raghavachari. Low-degree minimum spanning trees. Technical report, Department of Computer Science, University of Texas at Dallas, 2003.

[56] R. Jothi and B. Raghavachari. Approximation algorithms for the capacitated minimum spanning tree problem and its variants in network design. In *Proc. 31st Intl. Colloquium on Automata, Languages and Programming (ICALP)*, 2004.

[57] R. Jothi and B. Raghavachari. Degree-bounded minimum spanning trees. In *Proc. 16th Canadian Conf. on Computational Geometry (CCCG)*, 2004.

[58] R. Jothi and B. Raghavachari. Dynamic capacitated minimum spanning trees. In *Proc. 3rd Intl. Conf. on Networking (ICN)*, 2004.

[59] R. Jothi and B. Raghavachari. Improved approximation algorithms for the single-sink buy-at-bulk network design problems. In *Proc. 9th Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 336–348, 2004.

[60] R. Jothi and B. Raghavachari. Minimum latency tours and the k-traveling repairmen problem. In *Proc. 6th Latin American Theoretical INformatics (LATIN)*, pages 423–433, 2004.

[61] R. Jothi and B. Raghavachari. A note on Altinkemer-Gavish's algorithm on the design of tree networks. In *Proc. 7th INFORMS Telecommunications Conf.*, pages 78–80, 2004.

[62] R. Jothi and B. Raghavachari. Revisiting Esau-Williams' algorithm: On the design of local access networks. In *Proc. 7th INFORMS Telecommunications Conf.*, pages 104–107, 2004.

[63] R. Jothi and B. Raghavachari. Survivable network design: the capacitated minimum spanning network problem. In *Proc. 7th INFORMS Telecommunications Conf.*, pages 50–52, 2004.

[64] M. Karnaugh. A new class of algorithms for multipoint network optimization. *IEEE Trans. on Communications*, 24:500–505, 1976.

[65] A. Kershenbaum. Computing capacitated minimal spanning trees efficiently. *Networks*, 4:299–310, 1974.

[66] A. Kershenbaum and R. Boorstyn. Centralized teleprocessing network design. *Networks*, 13:279–293, 1983.

[67] A. Kershenbaum, R.R. Boorstyn, and R. Oppenheim. Second-order greedy algorithms for centralized teleprocessing network design. *IEEE Trans. on Communications*, 28:1835–1838, 1980.

[68] A. Kershenbaum and W. Chou. A unified algorithm for designing multidrop teleprocessing networks. *IEEE Trans. on Communications*, 22:1762–1772, 1974.

[69] S. Khuller, B. Raghavachari, and N. Young. Low-degree spanning trees of small weight. *SIAM J. Computing*, 25:355–368, 1996.

[70] J. Könemann and R. Ravi. A matter of degree: Improved approximation algorithms for degree-bounded minimum spanning trees. In *Proc. 32nd ACM Symp. on Theory of Computing (STOC)*, pages 537–546, 2000.

[71] J. Könemann and R. Ravi. Primal-dual algorithms come of age: Approximating MST's with nonuniform degree bounds. In *Proc. 35th ACM Symp. on Theory of Computing (STOC)*, pages 389–395, 2003.

[72] J.B. Kruskal Jr. On the shortest spanning subtree of a graph and the traveling salesman problem. In *Proc. American Math. Soc.*, volume 7, pages 48–50, 1956.

[73] K. Malik and G. Yu. A branch and bound algorithm for the capacitated minimum spanning tree problem. *Networks*, 23:525–532, 1993.

[74] J. Martin. *Design of real-time computer systems*. Prentice Hall, Englewood Cliffs, 1967.

[75] P.V. McGregor and D. Shen. Network design: An algorithm for the access facility location problem. *IEEE Trans. on Communications*, 25:61–73, 1977.

[76] E. Minieka. The delivery man problem on a tree network. *Annals of Oper. Res.*, 18:261–266, 1989.

[77] J.S.B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: Part II—a simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. *SIAM J. Computing*, 28:1298–1309, 1999.

[78] C. Monma and S. Suri. Transitions in geometric minimum spanning trees. *Discrete Comput. Geom.*, 8:265–293, 1992.

[79] J. Orlin. Personal communication, 2003.

[80] C.H. Papadimitriou. The complexity of the capacitated tree problem. *Networks*, 8:217–230, 1978.

[81] C.H. Papadimitriou and U.V. Vazirani. On two geometric problems related to the traveling salaesman problem. *J. Algorithms*, 5:231–246, 1984.

[82] R. Patterson, H. Pirkul, and E. Rolland. A memory adaptive reasoning technique for solving the capacitated minimum spanning tree problem. *J. Heuristics*, 5:159–180, July 1999.

[83] R.C. Prim. Shortest connection networks and some generalizations. *Bell Systems Tech. Journal*, 36:1389–1401, 1957.

[84] S. Rajagopalan and V.V. Vazirani. Logarithmic approximation of minimum weight k-trees, 1995. unpublished manuscript.

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

[86] R. Ravi, R. R. Sundaram, M.V. Marathe, D.J. Rosenkrantz, and S.S. Ravi. Spanning trees short or small. In *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 546–555, 1994.

[87] G. Robins and J.S. Salowe. Low-degree minimum spanning trees. *Discrete Comput. Geom.*, 14:151–166, 1995.

[88] G. Robins and A. Zelikovsky. Improved Steiner tree approximation in graphs. In *Proc. 11th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 770–779, 2000.

[89] F.S. Salman, J. Cheriyan, R. Ravi, and S. Subramanian. Approximating the single-sink link-installation problem in network design. *SIAM J. Optimization*, 11:595–610, 2000.

[90] J.S. Salowe. Euclidean spanner graphs with degree four. *Disc. Appl. Math.*, 54:55–56, 1994.

[91] G.M. Schneider and M.N. Zastrow. An algorithm for the design of multilevel concentrator networks. *Computer Networks*, 6:563–581, 1982.

[92] Y.M. Sharaiha, M. Gendreau, G. Laporte, and I.H. Osman. A tabu search algorithm for the capacitated shortest spanning tree problem. *Networks*, 29:161–171, 1997.

[93] D. Sharma. Personal communication, 2003.

[94] R.L. Sharma and M.T. El-Bardai. Suboptimal communications network synthesis. In *Proc. Intl. Conf. on Communications (ICC)*, volume 19, pages 11–16, 1970.

[95] R. Sitters. The minimum latency problem is NP-hard for weighted trees. In *Proc. 9th Conf. on Integer Programming and Combinatorial Optimization (IPCO)*, pages 230–239, 2002.

[96] T. Talwar. Single-sink buy-at-bulk LP has constant integrality gap. In *Proc. 9th Conf. on Integer Programming and Combinatorial Optimization (IPCO)*, pages 475–486, 2002.

[97] S. Voß. Capacitated minimum spanning trees. In C.A. Floudas and P.M. Pardalos, editors, *Encyclopedia of Optimization*, volume 1, pages 225–235. Kluwer, Boston, 2001.

[98] V.K.M. Whitney. *A study of optimal file assignment and communication network configuration in remote access computer message processing and communications systems.* PhD thesis, Univ. of Michigan, Ann Arbor, 1970.

[99] T. Will. *Extremal results and algorithms for degree sequences of graphs.* PhD thesis, University of Illinois at Urbana-Champaign, 1993.

VITA

Raja Jothi, born in Tamilnadu, INDIA, on May 26, 1977, is the son of G. Jothi and M. Vijayalakshmi. After completing his work at the Seventh Day Adventist Higher Secondary School, Chennai, INDIA, in 1994, he entered the University of Madras, and graduated in 1988 with a Bachelor of Engineering degree in Computer Science and Engineering. In August 1999, Raja started his graduate studies at the Clemson University, South Carolina, and transferred to the University of Texas at Dallas in January 2000, from where he received his Master of Science degree in Computer Science in December 2000, and expects to receive his Ph.D. in Computer Science in August 2004.