Universidade Estadual de Campinas
Instituto de Computação

Hugo Kooki Kasuya Rosado

An Approximation Algorithm for the q-Metric Node-Weighted Steiner Tree Problem

Um Algoritmo de Aproximação para o Problema da Árvore de Steiner q-Métrico com Peso nos Vértices

CAMPINAS
2017

Hugo Kooki Kasuya Rosado

An Approximation Algorithm for the q-Metric Node-Weighted Steiner Tree Problem

Um Algoritmo de Aproximação para o Problema da Árvore de Steiner q-Métrico com Peso nos Vértices

Dissertação apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Thesis presented to the Institute of Computing of the University of Campinas in partial fulfillment of the requirements for the degree of Master in Computer Science.

Supervisor/Orientador: Prof. Dr. Lehilton Lelis Chaves Pedrosa
Co-supervisor/Coorientador: Prof. Dr. Flávio Keidi Miyazawa

Este exemplar corresponde à versão final da Dissertação defendida por Hugo Kooki Kasuya Rosado e orientada pelo Prof. Dr. Lehilton Lelis Chaves Pedrosa.

CAMPINAS
2017

Agência(s) de fomento e nº(s) de processo(s): CNPq, 162305/2015-0

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

R71a

Rosado, Hugo Kooki Kasuya, 1989-
An approximation algorithm for the q-metric node-weighted Steiner tree problem / Hugo Kooki Kasuya Rosado. – Campinas, SP : [s.n.], 2017.

Orientador: Lehilton Lelis Chaves Pedrosa.
Coorientador: Flávio Keidi Miyazawa.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Steiner, Árvore de. 2. Algoritmos de aproximação. 3. Otimização combinatória. I. Pedrosa, Lehilton Lelis Chaves, 1985-. II. Miyazawa, Flávio Keidi, 1970-. III. Universidade Estadual de Campinas. Instituto de Computação. IV. Título.

Informações para Biblioteca Digital

Título em outro idioma: Um algoritmo de aproximação para o problema da árvore de Steiner q-métrico com peso nos vértices
Palavras-chave em inglês:
Steiner tree
Approximation algorithms
Combinatorial optimization
Área de concentração: Ciência da Computação
Titulação: Mestre em Ciência da Computação
Banca examinadora:
Lehilton Lelis Chaves Pedrosa [Orientador]
Vinícius Fernandes dos Santos
Rafael Crivellari Saliba Schouery
Data de defesa: 02-10-2017
Programa de Pós-Graduação: Ciência da Computação

Universidade Estadual de Campinas
Instituto de Computação

Hugo Kooki Kasuya Rosado

An Approximation Algorithm for the q-Metric Node-Weighted Steiner Tree Problem

Um Algoritmo de Aproximação para o Problema da Árvore de Steiner q-Métrico com Peso nos Vértices

Banca Examinadora:

- Prof. Dr. Lehilton Lelis Chaves Pedrosa
  Instituto de Computação - UNICAMP
- Prof. Dr. Vinícius Fernandes dos Santos
  Departamento de Ciência da Computação - UMFG
- Prof. Dr. Rafael Crivellari Saliba Schouery
  Instituto de Computação - UNICAMP

A ata da defesa com as respectivas assinaturas dos membros da banca encontra-se no processo de vida acadêmica do aluno.

Campinas, 02 de outubro de 2017

Resumo

O Problema da Árvore de Steiner de Custo Mínimo é um problema de otimização clássico em que, dado um grafo e subconjunto de vértices, chamados terminais, procura-se obter um subgrafo conexo com todos os terminais de forma que a soma dos pesos das arestas seja mínima. Consideramos a variante em que o grafo é completo e a função de peso sobre as arestas é uma métrica e, além disso, existe uma função de pesos não negativos sobre os vértices. O objetivo é encontrar uma árvore que contém todos os terminais e que minimiza o peso total de vértices e arestas. Nesta dissertação, observamos que o problema geral admite uma 2-aproximação. Para o caso particular em que o peso de um vértice é no máximo q vezes o peso da aresta mais leve, para uma constante q, obtemos um algoritmo aleatorizado baseado em arredondamento de PL com fator de aproximação 1,62 e um algoritmo guloso com fator de aproximação 1,55.

Abstract

The Minimum Cost Steiner Tree is a classical optimization problem where, given a graph and a subset of vertices called terminals, one is asked to find a connected subgraph spanning the set of terminals and whose edge weight is minimum. We consider the variant where the graph is complete and the edge weight function is a metric and there is a non-negative weight function on the vertices. The objective is to find a tree spanning the terminals such that the sum of edge and vertex weights is minimum. In this thesis, we observe that the general problem admits a 2-approximation. For the special case where the weight of a vertex is at most q times the weight of lightest edge, for a constant q, we obtain a randomized LP-rounding algorithm with approximation factor 1.62 and a greedy algorithm with approximation factor 1.55.

Contents

Chapter 1

Introduction

1.1 Motivation and Thesis Overview

Suppose we own a telecommunications company and we are given the task to run cables and deploy routers throughout the city in order to connect a given set of buildings into a single network. The simplest of the solutions might be connecting cables from a building to the next one, saving the money spent with the deployment of costly routers and relying purely on the cables. If every building is aligned as in a straight line, this would turn into a really good solution (and perhaps even the cheapest one!). Unfortunately such a scenario is very unlikely as building are usually scattered all across the city. In this situation, deploying routers to connect smaller sets of buildings and interconnecting the routers might be a good idea. However, the cost of installing too many machines might overweight the budget reserved for the task. Actually, it is very hard to tell precisely which solution minimizes the total expense, and tackling problems like this is the main goal of combinatorial optimization.

Broadly speaking, combinatorial optimization is a topic whose studied problems may be described as in the following: given an instance of a problem, find the "best" solution among all the feasible solutions. This topic is widely studied by researchers from Applied Mathematics and Theoretical Computing Science. The simplest strategy is brute-force: enumerating all solutions and selecting the best one. Unfortunately, such exhaustive search (brute force) is unpractical, as for many problems, one can show mathematically that, with current computer power, the time necessary to solve some of these problems might be longer than the planet's lifespan, even for relatively small instances. It turns out that the telecommunication problem described is one of the many variants of the famous *Steiner Tree Problem*, named after Jakob Steiner, a Swiss mathematician who contributed to the field of Geometry.

Deciding whether an optimal solution of an instance of the Steiner Tree problem costs at most a value k is NP-hard. This means that we do not expect to find efficient algorithms for this decision problem, and, furthermore, we do not expect that there is an efficient algorithm to find an optimal solution for the same instance. For problems in the NP-hard class, it is usual to look after alternative approaches. This includes the use of heuristics to help speed up the running time and to improve solutions, or even the use artificial intelligence. In this thesis, we focus on *approximation algorithms*. These algorithms can efficiently find feasible solutions of optimization problems, but concedes not finding the best solution. Instead, these

algorithms produce solutions whose costs are within a mathematically provable margin of error. This margin is formalized as the *approximation factor* of the approximation algorithm.

The most common version of a Steiner problem consists of a set of objects and connections between them. One is asked to interconnect a special subset of objects, called *terminals*, into a single structure we refer as the *Steiner tree*. The objective is to minimize the cost of used connections between terminal and, possibly, nonterminals. This problem is called the *Steiner Tree Problem* (STP), thoroughly studied in the last century [20]. Yet, only in the last two decades significant improvements have been derived on the scope of approximation algorithms, where it has been shown that one can compute efficiently a solution whose cost is within a small constant factor from the optimum [3, 5, 6]. The importance of STP comes from its many practical applications in network design, optimization of VLSI circuitry and logistics. Moreover, solving STP is also a subproblem to many other complex and important tasks.

In the practical telecommunication scenario we described earlier, the routers are not free. This variation of the problem is known as the *Node-Weighted Steiner Tree Problem* (NSTP) [12]. Unfortunately, we do not expect to find an approximation algorithm with constant factor for it, because achieving this would imply the existence of efficient algorithms for other NP-hard problems. Despite this fact, it is natural to assume that the connection costs satisfy certain properties. For example, in practical situations, it is natural to assume that the cost to connect two objects is proportional to the distance, and thus the connection cost satisfies the triangle inequality. We call this version the *Metric Node-Weighted Steiner Tree Problem* (MNSTP). In this thesis, we show that a 2-approximation for this problem can be readily obtained. Moreover, we study the special case which we call the *q-Metric Node-Weighted Steiner Tree Problem* (q-MNSTP), where the ratio between the cost of a router and the cost of a connection is bounded by the constant q. For this variant, we show that the factor can be further improved, and describe an LP-rounding and a greedy algorithm, with approximation factors 1.62 and 1.55, respectively.

In Chapter 2, we give the preliminary definitions, and concepts used through the thesis, as well as some basic results. In this chapter, we formally define approximation algorithms, linear programming, we introduce some motivation and concepts on probabilistic theory and graph theory. Also, we define the Steiner Tree Problem and considered variants, and finish describing a few related works in the field. In Chapter 3, we present a simple approximation for MNSTP, and show that to obtain an approximation for q-MNSTP, one can consider only solutions in a restricted form; this result is important, as it allows the use of techniques from the classical STP, thus leading to the approximation algorithms. In Chapter 4, we present an LP-rounding algorithm, and show that it obtains a solution with expected cost at most 1.62 times the optimal. Finally, in Chapter 5, we present a greedy approximation algorithm, and show that it has approximation factor of 1.55.

Chapter 2

Preliminaries and Related Works

This chapter contains a summary of definitions and basic concepts used throughout the thesis. We define approximation algorithms, and introduce basic notions of graph theory, probabilistic theory and linear programming. Some of the more detailed and specific concepts are given as we prove our results in the following chapters.

2.1 Probabilistic Theory

A randomized algorithm is an algorithm that makes choices depending on an auxiliary input sequence of random bits. The study of such algorithms is a well established area, and there are many techniques of analysis [17]. This section gives only an introductory grasp of probability that will be used in the analysis of the probabilistic algorithm given in Chapter 4.

A *probabilistic space* is a tuple with three components: a *sample space* Ω, which contains every possible outcome of a random process; a family $\mathscr{F}$ of subsets of Ω, called *events*, which represents the allowed outcomes; and a *probability function* $Pr : \mathscr{F} \to [0,1]$ such that $Pr(\Omega) = 1$ and $Pr(\bigcup_{i\geq 1} E_i) = \sum_{i\geq 1} Pr(E_i)$ for any set of pairwise disjoint events $\{E_1, E_2, \ldots\}$.

For example, consider the random process of rolling a fair six-sided die. We define the sample space Ω with every number in the die, which is $\Omega = \{1,2,3,4,5,6\}$. The set of events is $\mathscr{F} = \{E \subseteq \Omega\}$, and, since the die is fair, the probability of getting any number is $1/6$, and thus $Pr(\{i\}) = 1/6$, for $1 \leq i \leq 6$. Also, for $1 \leq i < j \leq 6$, events $\{i\}$ and $\{j\}$ are disjoint, and thus $Pr(\{i,j\}) = Pr(\{i\}) + Pr(\{j\}) = 2/6$, and so on.

If one roll a four-sided die twice, and sum the drawn numbers, the probability space Ω for this situation is the set of pairs of numbers. The sum of these numbers is now a random variable X that relates each outcome of Ω to a number. One may also consider the probability that variable X assumes a desired number. Formally, a *random variable* is a function $X : \Omega \to \mathbb{R}$. The probability of the event, in which X assumes value n is denoted by $Pr(X = n)$, the probability that $X \leq n$ as $Pr(X \leq n)$ and so on. In the example, the probabilistic space is defined as follows. Let d_i represent the number given by the i-th die roll, then $\Omega = \{1,2,3,4\}^2$, $\mathscr{F} = \{E : E \subseteq \Omega\}$, and the random variable is $X((d_1,d_2)) = d_1 + d_2$. For example, $Pr(X = 3) = 1/8$, because there are exactly two possibilities out of sixteen outcome possibilities in which the sum gives the value 3, namely $(1,2)$ and $(2,1)$; also $Pr(X = 9) = 0$, because no outcome sums up to 9; $Pr(X \leq 4) = Pr(X = 2 \cup X = 3 \cup X = 4) = Pr(X = 2) + Pr(X = 3) + Pr(X = 4) = (1+2+3)/16 = 3/8$,

because the events of which $X = i$ and $X = j$ are disjoint when $i \neq j$.

To model the situation that an event E is influenced by another event F, one might measure the probability of event E happening, given that F occurred previously. If $Pr(F)$ is not zero, then the *conditional probability* of E, given F, is as $Pr(E \mid F) = Pr(E \cap F)/Pr(F)$. Intuitively, this measures the probability of the event $E \cap F$ within the set of events defined by F.

A useful property of a random variable is its *expectation*. Informally, the expectation $\mathbb{E}[X]$ of a random variable X is the weighted average of the value it assumes. If the image of X is countable, then the expected value is defined as $\mathbb{E}[X] := \sum_i i \cdot Pr(X = i)$. For example, the expected value of the number obtained by rolling a single six-sided die is

$$\begin{aligned}\mathbb{E}[X] &= 1 \cdot Pr(X = 1) + 2 \cdot Pr(X = 2) + \cdots + 6 \cdot Pr(X = 6) \\ &= 1 \cdot \frac{1}{6} + 2 \cdot \frac{1}{6} + 3 \cdot \frac{1}{6} + 4 \cdot \frac{1}{6} + 5 \cdot \frac{1}{6} + 6 \cdot \frac{1}{6} \\ &= (1+2+3+4+5+6)\frac{1}{6} \\ &= 3.5.\end{aligned}$$

Notice that the expectation $\mathbb{E}[X]$ need not be a value in the image of X. Consider now the situation where one rolls a four-sided die twice and sum the obtained numbers. The expected value of the sum in the experiment is

$$\begin{aligned}\mathbb{E}[X] &= 2 \cdot Pr(X = 2) + 2 \cdot Pr(X = 3) + \cdots + 8 \cdot Pr(X = 8) \\ &= 2 \cdot \frac{1}{16} + 3 \cdot \frac{2}{16} + 4 \cdot \frac{3}{16} + 5 \cdot \frac{4}{16} + 6 \cdot \frac{3}{16} + 7 \cdot \frac{2}{16} + 8 \cdot \frac{1}{16} \\ &= 5.\end{aligned}$$

A useful property of the expectation is its linearity. Let X and Y be random variables, and c be a constant. It is easy to verify that $\mathbb{E}[X + cY] = \mathbb{E}[X] + c \cdot \mathbb{E}[Y]$. This equality aids in the computation of more complex random variables. For example, the expected value of the sum of numbers given by rolling a four-sided die twice is easily calculated by considering the expectation of each roll individually as $X = X_1 + X_2$, where X_i is the value obtained in the i-th time.

$$\mathbb{E}[X] = \mathbb{E}[X_1 + X_2] = \mathbb{E}[X_1] + \mathbb{E}[X_2] = \sum_{i=1}^{4} i \cdot \frac{1}{4} + \sum_{i=1}^{4} i \cdot \frac{1}{4} = 2 \cdot \frac{1}{4} \sum_{i=1}^{4} i = 5.$$

Similarly to conditional probability, one can define the notion of *conditional expectation* as the expectation of a random value X given that an event F has occurred, $\mathbb{E}[X \mid F]$. Formally, $\mathbb{E}[X \mid F] = \sum_x x \cdot Pr(X = x \mid F)$, where the summation is over all x in the image of X. For example, the expectation of the sum of numbers given by rolling a four-sided die twice, given that the first drawn number is a 2, is calculated as

$$\mathbb{E}[X \mid X_1 = 2] = \sum_{i=2}^{8} i \cdot Pr(X = i \mid X_1 = 2) = \sum_{i=3}^{6} i \cdot \frac{1}{4} = 4.5.$$

Notice that the second expression is simplified by observing that the sum cannot be $2, 7$ nor 8,

i.e., $Pr(X = j \mid X_1 = 2) = 0$ for $j = 2, 7, 8$.

It is also possible to partition the random space for a random variable X by considering every possible outcome of another random variable Y. This gives rise to an important identity on the expectation of X that is given by the following lemma.

Lemma 1. *For any random variables X and Y,*

$$\mathbb{E}[X] = \sum_y Pr(Y = y) \cdot \mathbb{E}[X \mid Y = y],$$

where the sum is over all values in the image of Y.

Proof. Let X_x denote the event where $X = x$ and Y_y the event where $Y = y$. Because the events $X_x \cap Y_y$, $y \in \mathbb{R}$, are disjoint and cover the entire sample space such that $X = j$, it follows that $Pr(X_x) = \sum_y Pr(X_x \cap Y_y)$.

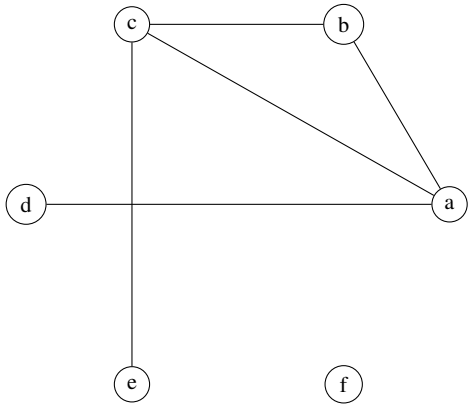
$$\begin{aligned}
\sum_y Pr(Y = y) \cdot \mathbb{E}[X \mid Y = y] &= \sum_y Pr(Y_y) \cdot \mathbb{E}[X \mid Y_y] \\
&= \sum_y Pr(Y_y) \cdot \sum_x x \cdot Pr(X_x \mid Y_y) \\
&= \sum_x x \cdot \sum_y Pr(X_x \mid Y_y) \cdot Pr(Y_y) \\
&= \sum_x x \cdot \sum_y Pr(X_x \cap Y_y) \\
&= \sum_x x \cdot Pr(X_x) \\
&= \sum_x x \cdot Pr(X = x) \\
&= \mathbb{E}[X].
\end{aligned}$$

□

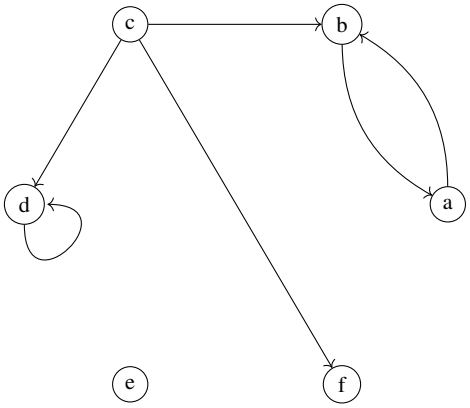
2.2 Graph Theory

A *graph* is an abstract structure useful to model relations between objects. In this thesis a (simple) *graph* G is denoted as an ordered pair (V, E), where the set of objects is denoted by V and its elements are called *vertices*, and we represent the relation between two vertices by a set E of 2-element subsets of V, i.e., $E \subseteq \binom{V}{2}$, and its elements are called *edges* [4]. If $e = \{u, v\}$ is an edge, then we say that vertices u, v are *incident* to e, and that e is *incident* to u, and that u and v are *adjacent*. For a graph G, we denote the set of vertices and edges of a graph G by $V(G)$ and $E(G)$ whenever needed to avoid ambiguity. A *directed graph* G is an ordered pair (V, A), where V is the set of vertices, and A is a set of ordered pairs of vertices of G called *arcs*. If $a = (u, v)$ is an arc, then we say that u is the *tail* of a and v is the *head* of a. Figure 2.1 illustrates a graph and a directed graph.

A *multigraph* is a generalization of a (simple) graph in which we allow *parallel* edges and *loops*. Two edges are said to be parallel if they are incident to the same vertices. An edge is a loop if it has only one incident vertex. Formally a multigraph $M = (V, E)$ is an ordered pair of a set of vertices V and a multiset E, where each element of E is associated to a subset of V with either one or two elements each.



(a) A graph $G = (V,E)$ with vertex set $V = \{a,b,c,d,e,f\}$ and edge set $E = \{\{a,b\},\{a,c\},\{a,d\},\{b,c\},\{c,e\}\}$.



(b) A directed graph G with vertex set $V = \{a,b,c,d,e,f\}$ and arc set $A = \{(a,b),(b,a),(c,b),(c,d),(c,f),(d,d)\}$.

Figure 2.1: Example of a graph and one of its subgraphs.

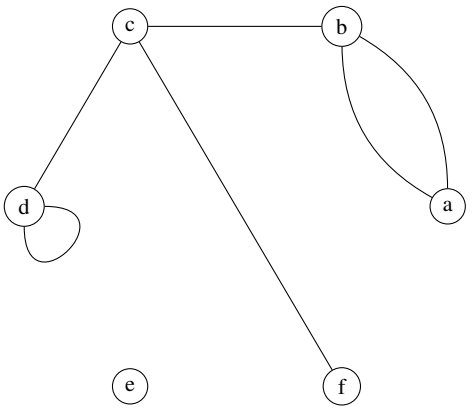
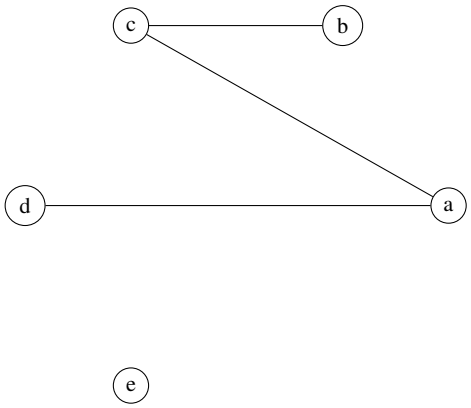


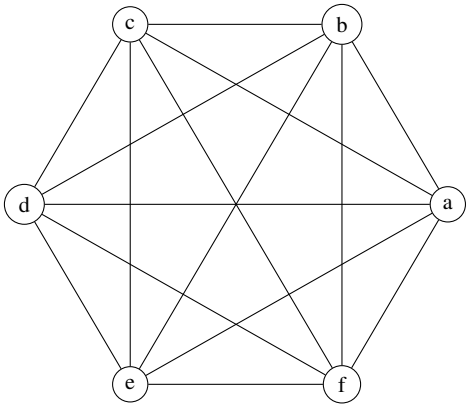
Figure 2.2: Example of a multigraph $M = (V,E)$, where $V = \{a,b,c,d,e,f\}$ and $E = \{\{a,b\},\{a,b\},\{b,c\},\{c,d\},\{c,f\},\{d\}\}$, which resembles the directed graph from Figure 2.1b.

We say that a graph H is a *subgraph* of G, denoting by $H \subseteq G$, if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$, i.e., the vertices of H is a subset of vertices of G and the edges of H is a subset of the edges of G. Given a graph G and a subset of its vertices $R \subseteq V$, we say a subgraph H *spans* R if $R \subseteq V(H)$. We say that a graph G is *complete* if its edge set $E(G)$ contains every possible edge in $\binom{V}{2}$. The *degree* of a vertex $v \in V$, denoted $d(v)$ is the number of edges incident to v. Figure 2.3 gives examples of a subgraph and a complete graph.

In a simple graph $G = (V,E)$, a *walk* $W = (v_0,\{v_0,v_1\},v_1,\{v_1,v_2\},...,\{v_{m-1},v_m\},v_m)$, of *length* $|W| = m$, is an alternating sequence of vertices and edges which starts and ends with vertices and we also call W a *closed walk* if $v_0 = v_m$. A *trail* T is a walk in which all the edges in the sequence are distinct and we say that T is a *circuit* if $v_0 = v_m$. A *path* P is a trail in which all the vertices are distinct, except possibly the first and the last vertices, and when they are not, we say that P is a *cycle*. We say that a graph is *connected* if G contains a path that connects every pair of $u,v \in V$, and we say that a maximal connected subgraph of G is a *connected component*. In this thesis, we say that a graph is Eulerian if every vertex in the graph has even degree and is connected. It is well-known that if a graph G is Eulerian, then there is a circuit which contains every edge of G (known as *Eulerian circuit*) [4]. We refer to Figure 2.4



(a) A representation of a graph H, subgraph of the graph G from Figure 2.1a, where $V = \{a,b,c,d,e\}$ and $E = \{\{a,c\},\{a,d\},\{b,c\}\}$. Note that $d(a) = d(c) = 2, d(b) = d(d) = 1$, and $d(e) = 0$.



(b) A representation of a complete graph H', where $V = \{a,b,c,d,e,f\}$ and $E = \binom{V}{2}$. Note that both the graph H and the graph G from Figure 2.1a are subgraphs of H'. The degree of every vertex is $|V|-1$.

Figure 2.3: Examples of a subgraph and complete graph.

as examples for some of these concepts.

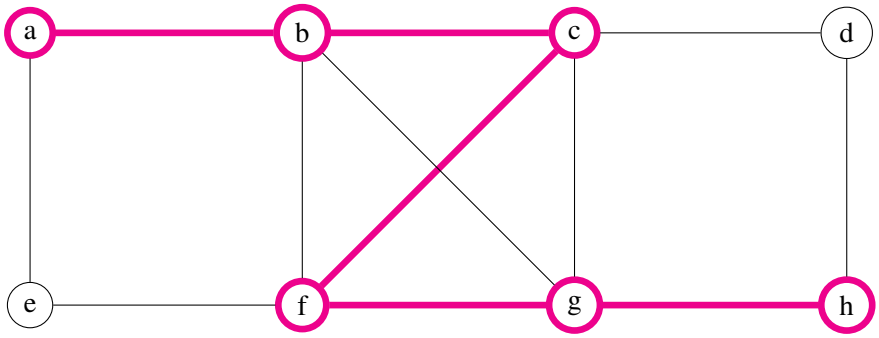
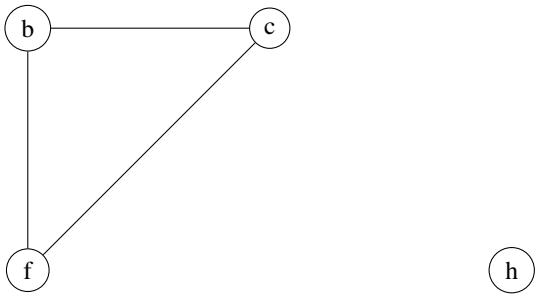


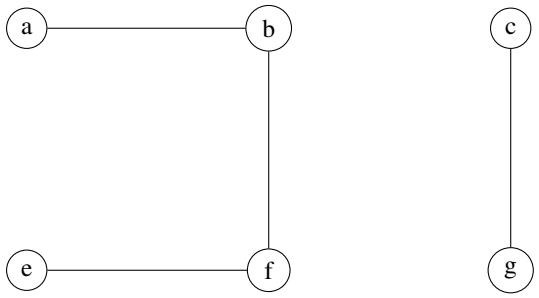
Figure 2.4: An example of an Eulerian graph G. In this graph, the sequence of vertices and edges $(a,\{a,b\},b,\{b,f\},f,\{f,e\},e,\{f,e\},f)$ is an example of a walk with length 4; the sequence $(a,\{a,b\},b,\{b,c\},c,\{c,f\},f,\{f,g\},g,\{g,h\},h)$ is an example of path (drawn in thick magenta) with length 5, and the sequence $(a,\{a,b\},b,\{b,c\},c,\{c,d\},d,\{d,h\},h,\{h,g\},g,\{g,b\},b,\{b,f\},f,\{f,c\},c,\{c,g\},g,\{g,f\},f,\{f,e\},e,\{e,a\},a)$ is an example of an Eulerian circuit with length 12.

Let $G = (V,E)$ be a graph and $U \subseteq V$. The *vertex-induced subgraph* $G[U]$ is the subgraph of G with vertex set U, and whose edge set contains every edge of $E(G)$ which has incident vertices in U. Now let $F \subseteq E$, then the *edge-induced subgraph* $G[F]$ is the subgraph of G which has edge set F and every vertex incident to some edge of F, and no more. We will refer to these subgraphs as *induced subgraphs*. Figure 2.5 exemplifies the two definitions. If $V' \subseteq V(G)$ and $E' \subseteq E$, then we also adopt the notation that $G \backslash V' = G[V \backslash V']$ and $G \backslash E' = G[E \backslash E']$.

A *tree* T is a graph which is connected, and has no cycles on it. Equivalently, a tree can be defined as a graph which is connected, and has exactly $|V|-1$ edges or as a graph which has



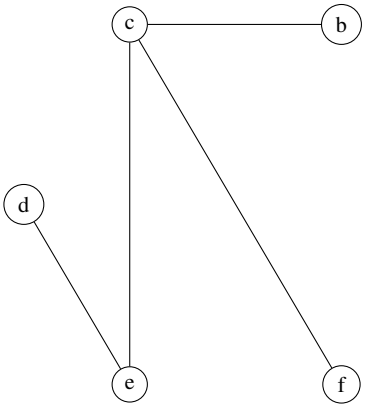
(a) Induced subgraph $G[U]$, where $U = \{b,c,f,h\}$.



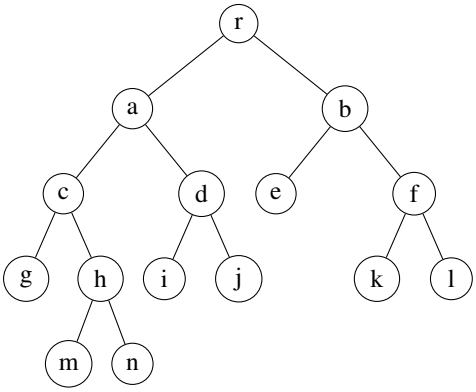
(b) Induced subgraph $G[F]$, where $F = \{\{a,b\},\{b,f\},\{c,g\},\{f,e\}\}$.

Figure 2.5: Examples induced subgraphs of the graph G from Figure 2.4. In both cases we have a graph with two connected components.

exactly $|V|-1$ edges and has no cycles [4]. A subgraph of a tree, that is also a tree, is called a *subtree*. A vertex of a tree with degree one is a *leaf* and all the others are *internal vertices*. A *rooted tree* is a pair (T,r), where T is a tree and $r \in V(T)$ is the *root*. If the root r is fixed, then we might say just tree T. We say that T has *height* h if the largest path starting at the root has length h. Also, we say that a vertex v is in the i-level, denoting by $level(v) = i$, if the length of the unique path from r to v has length $i-1$ (the unicity of the path comes from the fact that T is a tree). A vertex $u \in V$, in the i-th level is said to be a *descendant* of a vertex $v \in V$ (and that v is an *ancestor* of u), if there is a path $(v = v_0, \{v, v_1\}, v_1, ..., \{v_{l-1}, v_l\}, v_l = u)$ from v to u such that $level(v_i) < level(v_{i+1})$, $0 \leq i < l$. For practicality we also say that the immediate descendant of a vertex is its *child* and its immediate ancestor is its *father*. We say a tree T is a *binary tree* if the degree of every vertex is at most 3. Also, a binary tree is *regular* if every non-root internal vertex has degree 3. Figure 2.6 gives some example of trees.



(a) A representation of a tree T which spans $\{b,c,d,e,f\}$ and is a subgraph of the complete graph H' from Figure 2.3b. If we choose b,d or f to be the root, then T has height 3, but if c or e is chosen, then T has height 2.



(b) A representation of a regular binary tree T with root r and height 4. In this example, every vertex besides r is a descendant of r and, in special, a,b are its children, a,c are ancestors of h,m,n, and b is the father of e,f.

Figure 2.6: Example graphs which are trees.

Let G and H be graphs. We denote the union of graphs as $G \cup H = (V(G) \cup V(H), E(G) \cup$

$V(H)$). We define the addition of a vertex as $G+v = (V(G) \cup \{v\}, E(G))$ and subtraction as $G-v = G[V \setminus \{v\}]$. Similarly, if $u,v \in V(G)$, we denote addition and subtraction of edges as $G+\{u,v\} = (V(G), E(G) \cup \{u,v\})$ and $G-\{u,v\} = (V(G), E(G) \setminus \{u,v\})$.

In this thesis, we also need the *contraction* operation. The idea is to contract a connected subgraph into one of its vertex maintaining edges that connected vertices in the subgraph to other vertices outside it. Consider a graph $G = (V,E)$. Then the contraction of a connected subgraph H of G into a vertex $s \in V(H)$ results in a new graph $G' = (V',E')$, where $V' = \{s\} \cup (V(G) \setminus V(H))$, and E' is the set of edges of $E(G)$ that were not incident to any vertex of $V(H)$, plus a new edge $\{s,x\}$ for each vertex $x \in V(G) \setminus V(H)$ such that there exists an edge incident to x and to a vertex of $V(H)$ (we add only one edge $\{s,x\}$ for each x). If the graph G has an associated edge weight function $w_e : E(G) \to \mathbb{Q}_{>0}$, then in the case where multiple edges connect to the same outer vertex, we keep the lightest edge after each contraction operation, i.e., the new edge weight function for G' is defined as $w'_e : E' \to \mathbb{Q}_{>0}$ is given by

$$w'_e(\{a,b\}) = \begin{cases} \min_{v \in V(H)} \{w_e(\{v,b\})\} & \text{if } s = a, \\ w_e(\{a,b\}) & \text{otherwise.} \end{cases}$$

One can think that the new edges inherited the weight of the lightest edge among the edges that generated it. Also, note that we can trivially associate every new added edge to an edge that was deleted with the same weight in a one-to-one fashion. Even if we perform successive contraction operations, we can still map each edge to an edge of the original graph with the same weight.

2.3 Approximation Algorithms

Consider a minimization problem P. The decision version of P is the problem where, given an instance I of P and a number k, one wants to decide whether there exists a solution for I with value at most k. If the decision version of P is NP-hard, then one cannot expect expect to find an exact algorithms with polynomial-time complexity for it, unless P = NP. One could design an algorithm that obtains an optimal solution, but whose worse case can run in superpolynomial-time. One alternative is to find a polynomial-time algorithm that does not find an exact solution, but whose value is within a given ratio. The algorithms of the latter are known as *approximation algorithms* [20].

Let I be an instance for a minimization problem. Given an algorithm A, with polynomial-time complexity, denote by $A(I)$ the value of the solution computed by A when it is given instance I, by $\text{Opt}(I)$ the value of an optimal solution for I. An algorithm has an *approximation factor* of α if $A(I)/\text{Opt}(I) \leq \alpha$ for every instance I. In this case, A is an *α-approximation*. If A is a randomized algorithm, and $\mathbb{E}[A(I)]/\text{Opt}(I) \leq \alpha$, where $\mathbb{E}[A(I)]$ is the expected value of the solution derived by the algorithm A over all random choices taken by A, the A is a *randomized α-approximation*. As an example, consider the following problem:

Problem 2. *(Minimum Vertex Cover Problem (*MVCP*)) Given a simple graph $G = (V,E)$, the objective is to find the smallest subset $C \subseteq V$ such that every $e \in E$ has at least one of its incident vertices in C.*

Now consider the following approximation algorithm for MVCP:

VCALG:
1) $C = \emptyset$.
2) While there is an edge $e = \{u, v\}$ such that neither $u \notin S$, nor $v \notin S$:
   a) Add u and v in S.
3) Return C.

Lemma 3. VCALG *is a* 2*-approximation algorithm for the* MVCP.

Proof. It is easy to verify that the VCALG gives a feasible solution C and that it runs in time $O(|E|)$. Consider a maximum set of pairwise disjoint edges M, and notice that any feasible solution must have at least one of the incident vertices from each edge in M because of the disjointness, thus $|\text{Opt}| \geq |M|$. Because the edges considered by the algorithm are also pairwise disjoint, the number of these edges is at most $|M|$, thus the algorithm includes at most $2 \cdot |M|$ vertices in the solution C. Therefore

$$|C| \leq 2 \cdot |M| \leq 2 \cdot \text{Opt}.$$ □

Many sophisticated strategies for the development of approximation algorithms have been studied in the last decades. Some of the techniques include: *rouding of solutions via linear programming*, *duality of linear programming and the primal-dual method*, *probabilistic algorithms and derandomization*, *Lagrangian relaxation*, *iterative randomized rounding* (which we use in Chapter 4), and many others.

2.4 The q-Metric Node-Weighted Steiner Tree Problem

The *Minimum Spanning Tree* problem is a classical optimization problem and there are many polynomial-time algorithms, e.g. Prim's algorithm [8]. We define the problem in the following.

Problem 4. *(Minimum Spanning Tree) Given a simple graph $G = (V, E)$ and an edge weight function $w_e : E \to \mathbb{Q}_{>0}$, we are asked to find a tree T which spans V and has minimum total edge weight $w_e(T) := \sum_{e \in E(T)} w_e(e)$. See Figure 2.7.*

In this thesis, we denote one fixed minimum spanning tree of a graph G by $\text{MST}(G)$. A very useful lemma is the following:

Lemma 5. *(Cycle property) Let G be a simple graph and $w_e : E \to \mathbb{Q}_{>0}$. If C is a cycle of G, and there exists an edge e of C such that $w_e(e) > w_e(f)$ for any other edge f of C, then e does not belong to the any minimum spanning tree of G.*

Proof. Assume, for a contradiction, that there exists a minimum spanning tree T that contains e. By removing e from T, one obtains two distinct trees T_1 and T_2. Because T spans $V(G)$, $T_1 \cup T_2 \cup C$ is connected. But since C is a cycle, there is at least one edge f, different from e, that connects vertices of T_1 and T_2. Thus $T' = (T_1 \cup T_2) + f$ is a tree whose weight is lesser than that of T. This is a contradiction, since T is minimum. □

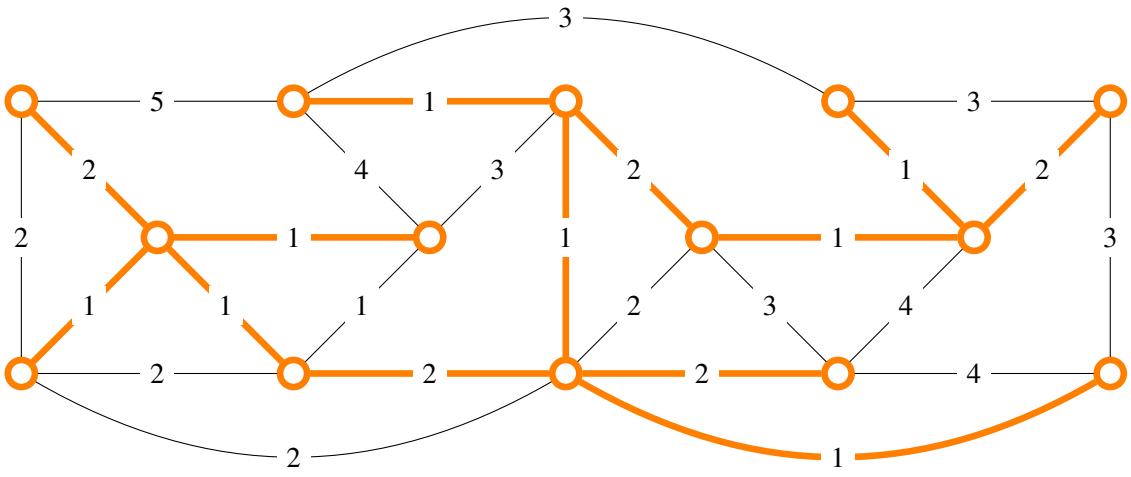


Figure 2.7: An example of a graph G with edge weights. The tree $\mathrm{MST}(G)$ is colored in thick orange and has total weight $w_e(\mathrm{MST}(G)) = 16$.

A slightly different variant, which considers the possibility that certain vertices are not spanned, is the *Steiner Tree Problem* (STP). Contrary to MST, even finding a solution for STP with weight smaller than $\frac{96}{95}$ times the weight of an optimal solution NP-hard [7]. STP is relevant in many areas of science and we describe it as follows.

Problem 6. *(Steiner Tree Problem (*STP*)) Given a simple graph $G = (V,E)$, a subset $R \subseteq V$ of vertices called terminals (the vertices in $V \setminus R$ are referred to as* Steiner vertices*) and an edge weight function $w_e : E \to \mathbb{Q}_{>0}$, a* Steiner tree *is a tree, in G, that spans R. The problem's objective is to find a Steiner tree S with minimum edge weight $w_e(S) := \sum_{e \in E(S)} w_e(e)$.*

Figure 2.8 gives an example of a Steiner tree.

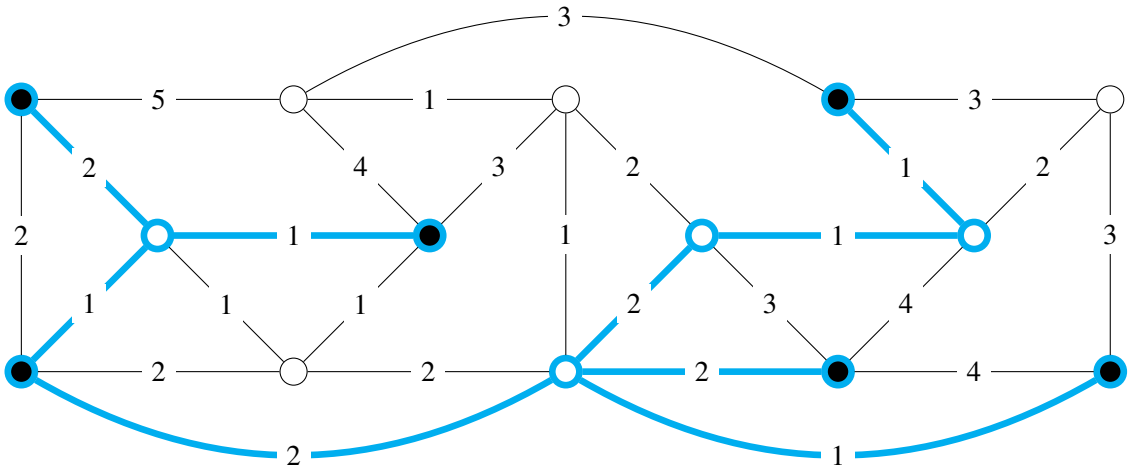


Figure 2.8: An instance of STP with the corresponding edge weights. Black vertices are terminals and white ones are Steiner vertices. An optimal Steiner tree S is given in thick cyan and has edge weight of $w_e(S) = 13$.

One important variant of the STP is the restriction to *metric spaces*. Given a complete graph $G = (V,E)$ with edge weight function $w_e : E \to \mathbb{Q}_{>0}$, we say that w_e is a metric if

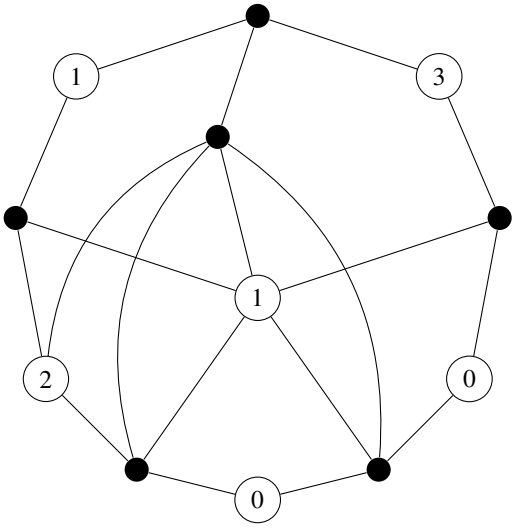
$w_e(\{u,v\}) \leq w_e(\{u,w\}) + w_e(\{w,v\})$ for every $u,v,w \in V$. A complete graph G associated to a metric function is called *metric*. The particular case of STP restricted to metric graphs is called the *Metric Steiner Tree Problem* (MSTP).

It is also well-known that there is an approximation-preserving polynomial-time from the general STP to the MSTP, i.e., if there is an α-approximation algorithm for the MSTP then there is also an α-approximation algorithm for the STP [20].

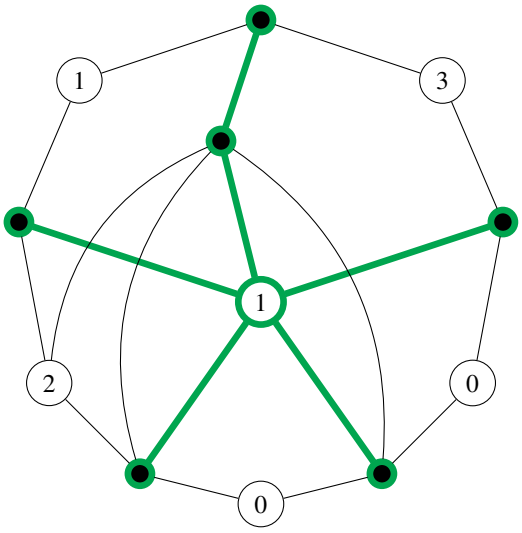
We now define a variant of STP in which, in addition to the edge weight, there is a cost associated with each vertex.

Problem 7. *(Metric Node-Weighted Steiner Tree Problem (*MNSTP*)) Given a simple complete graph $G = (V,E)$, a subset $R \subseteq V$ of vertices called terminals, a metric edge weight function $w_e : E \to \mathbb{Q}_{>0}$, and a vertex weight function $w_v : V \to \mathbb{Q}_{\geq 0}$, we are asked to find a tree T, in G, spanning R with minimum total weight $w(T) := w_e(T) + w_v(T)$, where $w_e(T) := \sum_{e \in E(T)} w_e(e)$ and $w_v(T) := \sum_{v \in V(T)} w_v(v)$.*

If no restriction is given upon the edge weights, then the problem is called *Node-Weighted Steiner Tree Problem* (NSTP), which does not accept an approximation algorithm with constant approximation factor unless P = NP [10, 12]. This gives relevance for the consideration of metric graphs in the hypothesis. Also, since terminals are always included in a solution, we may assume without loss of generality that $w_v(v) = 0$ for each $v \in R$. Figure 2.9 gives an example of an instance for this problem and a corresponding optimal solution.



(a) A representation of a complete graph G where edges are drawn between two vertices whenever its weight is 1 and an edge is not drawn if its weight is 2. The weight of the vertices are explicitly written.



(b) An example of optimal solution S, denoted in thick green, to G. Note that $w_v(S) = 1$ and $w_e(S) = 6$, thus its total weight $w(S) = 7$.

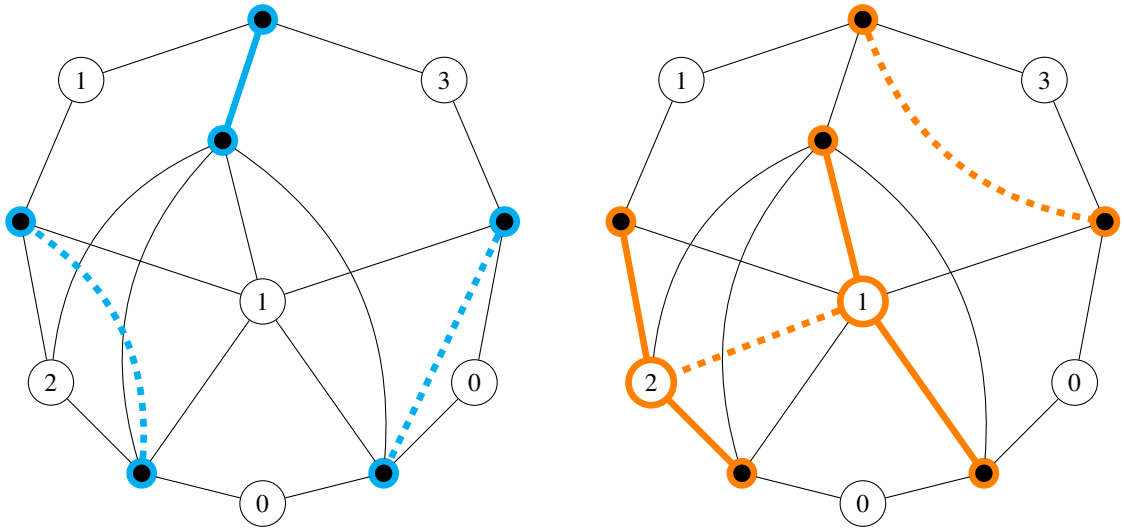
Figure 2.9: An example of an instance of MNSTP and its optimal solution. Terminals are drawn as black circles and Steiner vertices are in white.

We also consider a particular case when the vertex weight function is restricted in such a way that the cost of a vertex is bounded by a constant q times the weight of the lightest edge. We call this variant q-Metric Node-Weighted Steiner Tree Problem.

Problem 8. *(q-Metric Node-Weighted Steiner Tree Problem (q-*MNSTP*)) Given a complete graph $G = (V,E)$, a subset $R \subseteq V$ of vertices called terminals, a metric edge weight function $w_e : E \to \mathbb{Q}_{>0}$, and a vertex weight function $w_v : V \to \mathbb{Q}_{\geq 0}$ such that $w_v(v) \leq q \cdot \min_{e \in E} w_e(e)$, we are asked to find a tree T, in G, spanning R with minimum total weight $w(T) := w_e(T) + w_v(T)$, where $w_e(T) := \sum_{e \in E(T)} w_e(e)$ and $w_v(T) := \sum_{v \in V(T)} w_v(v)$.*

In Figure 2.9, the example is an instance of q-MNSTP for $q = 3$. The restrictions considered in the q-MNSTP generalizes the STP, thus there cannot be an approximation algorithm with approximation factor smaller than $\frac{96}{95}$ for the q-MNSTP, unless P = NP [7].

The best known approximation algorithms for STP use the notion of *k-restricted Steiner trees* and *full components* (or just *component*, for short). A k-full component is a Steiner tree on k terminals and whose leaves are terminals and internal nodes are Steiner vertices. For $k \geq 2$, we denote the set of full components with at most k terminals and at most $k-2$ Steiner vertices by $\mathscr{C}_k$ (note that $\mathscr{C}_k \subseteq \mathscr{C}_{k+1}$). Figure 2.10 shows some full components of the sets $\mathscr{C}_2$ and $\mathscr{C}_4$. The total weight of a full component K is defined as the sum of the weight of its edges and vertices, i.e., $w(K) = \sum_{e \in E(K)} w_e(e) + \sum_{v \in V(K)} w_v(v)$.



(a) Representation of some components of $\mathscr{C}_2$, in thick cyan.

(b) Representation of some components of $\mathscr{C}_4$, in thick orange.

Figure 2.10: Examples of full components of $\mathscr{C}_k$ when $k \in \{2,4\}$ for the instance from Figure 2.9a. Dashed lines are used to explicitly identify the edges with weight 2 in the components.

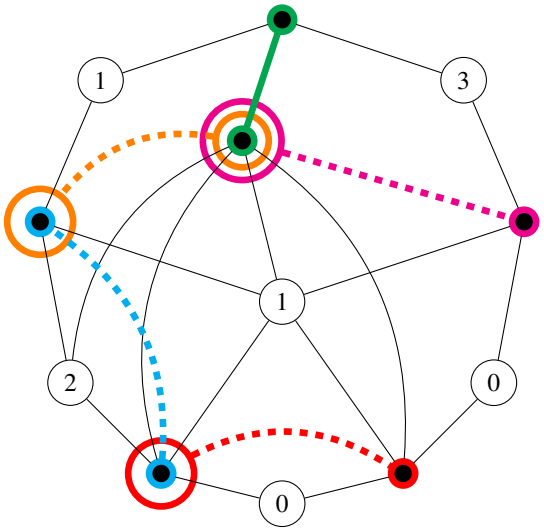
Let G be a metric graph and $r \in R$ be one of its terminals, then a subset $S_k \subseteq \mathscr{C}_k$ is said to be a *k-restricted Steiner tree* of G if for every $U \subseteq R \setminus \{r\}, U \neq \emptyset$, there is a component $K \in S_k$ with terminals $u, v \in V(K) \cap R$, such that $u \in U$ and $v \notin U$. We define two variants, which correspond to MSTP and q-MNSTP.

Problem 9. *(k-restricted* MSTP*) Given an instance of* MSTP*, the goal is find a k-restricted Steiner tree S_k of minimum total weight $w(S_k) := \sum_{K \in S_k} w(K)$.*

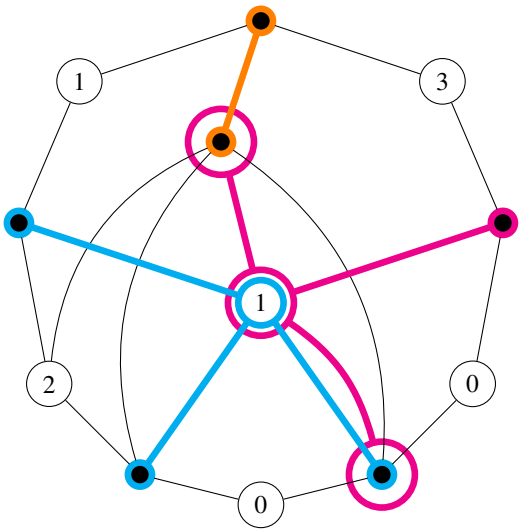
Borchers and Du [5] proved that the cost of a minimum Steiner tree whose full components are restricted to $\mathscr{C}_k$ is not much larger than the cost of an optimal Steiner tree as k grows. This will be further discussed in Chapter 3.

Problem 10. *(k-restricted q-*MNSTP*) Given an instance of q-*MNSTP *in which* $w_v(v) = 0$ *for all terminals* $v \in R$*, we are asked to find a k-restricted Steiner tree* S_k *of minimum total weight* $w(S_k) := \sum_{K \in S_k} w(K)$.

Notice that, in the k-restricted variant, a solution may repeat edges and vertices and it may contain only a subset of components of the non k-restricted problem. As is the case for k-restricted MSTP, for constant q, the cost of an optimal solution of k-restricted q-MNSTP is not much larger than the cost of an optimal solution of q-MNSTP for the same instance. Indeed, we show this fact in the Chapter 3, by extending the ideas of Borchers and Du [5]. In the example given in Figure 2.11 optimal solutions for the 2-restricted and 3-restricted q-MNSTP are given for the same instance of the graph given in Figure 2.9. Note that the optimal solution given in Figure 2.9b coincides with an optimal 5-restricted solution.



(a) An optimal 2-restricted solution S of weight $w(S) = w_e(S) = 10$. A 2-restricted solution never contains Steiner vertices.



(b) An optimal 3-restricted solution S with $w_e(S) = 7$, $w_v(S) = 2$, and $w(S) = 9$. Note that an Steiner vertex of weight 1 is used twice in the solution as well as one of its incident edges.

Figure 2.11: Examples of k-restricted optimal solutions to the instance from Figure 2.9a. Dashed lines are used to explicitly identify the edges with weight 2 and the chosen components are thick-color coordinated.

2.5 Linear Programming

Linear programming (LP) is the problem of optimizing a real linear function, called *objective function*, subject to linear inequality constraints. We give a simple example below with two variables and four constraints.

$$\begin{array}{lll} \text{maximize} & 5x_1 + 6x_2 & \\ \text{subject to} & x_1 + 3x_2 & \leq 7; \\ & -x_1 + 3x_2 & \leq 1; \\ & 2x_1 - x_2 & \leq 4; \\ & x_1, x_2 & \geq 0. \end{array} \qquad \text{(Primal)}$$

A solution for the problem with n variables is an n-dimensional vector. If the vector satisfies every constraint it is called a *feasible solution*, and if there is a feasible solution that optimizes the objective function, then it is called an *optimal solution*. For example, in the problem above, $x_1 = 0$ and $x_2 = 0$ is a feasible solution, but not optimal, and $x_1 = 2.6$ and $x_2 = 1.2$, which gives a value of 20.2 for the objective function, is an optimal solution. Linear programming can be solved by polynomial-time algorithms [8].

Theorem 11. *(Grotschel et at. 1981 [11]) Given an unfeasible solution x of an* LP*, if a violated constraint can be found in polynomial-time, then an optimum solution for the* LP *can also be found in polynomial-time.*

The variant of the problem in which variables are restricted to integer values is called *integer linear* programming (ILP) In the following we show that we can express STP using integer linear programming, and, therefore, ILP is NP-hard.

Let P be an ILP. A natural way to obtain bounds of the cost of an optimal solution may be done by relaxing the variables from P by allowing fractional values. This new LP, P_r, is called the *relaxation* of P. Another way to obtain lower bounds is by computing the *dual program* P' of the *primal program* P. If the primal program is of maximization, then the dual problem of minimization, and if the primal is of minimization, then the dual program is of maximization [20]. Below is an example of the relaxed dual program of the LP example given before.

$$\begin{array}{lll} \text{minimize} & 7y_1 + y_2 + 4y_3 & \\ \text{subject to} & y_1 - y_2 + 2y_2 & \geq 5; \\ & 3y_1 + 3y_2 - y_3 & \geq 6; \\ & y_1, y_2, y_3 & \geq 0. \end{array} \qquad \text{(Dual)}$$

The LP-duality has interesting properties. For example, if x and y are an optimal solutions for the primal and dual programs, then the objective function of both yields the same value. Indeed, in the examples given, an optimal solution to the Dual LP is $y_1 = 0$, $y_2 = 3.4$, and $y_3 = 4.2$, which obtains a value of 20.2 in the objective function, the same obtained by an optimal solution in the Primal LP.

For a directed graph G, and $U \subseteq V(G)$, denote by $\delta^+(U)$ the set of arcs with the tail in U and head in $V \backslash U$. Let us exchange every edge of an instance of STP with two directed arcs (u,v) and (v,u) with of same weight each, and let us fix a terminal vertex as a root r. The ILP below models an instance of STP. This problem is known as the bidirectional cut formulation for STP.

$$\begin{array}{lll} \text{minimize} & \sum_{a\in A} w_e(a)x_a & \\ \text{subject to} & \sum_{a\in\delta^+(U)} x_a \geq 1, & \forall U \subseteq V\setminus\{r\}, U\cap R\neq\emptyset; \\ & x_a \in \{0,1\}, & \forall a \in A. \end{array} \tag{BC}$$

Each variable x_a corresponds to an arcs a, so that a is selected if $x_a = 1$, and is not selected otherwise. Note that the objective function aims to minimize the total weight of selected edge set of a solution. The first set of constraints, together with the objective function, guarantees that the selected edges induce a feasible solution of STP. Conversely, every Steiner tree induces a vector x that is a feasible for BC.

Recall that the particular case of STP when $R = V$ is the Minimum Spanning Tree. The relaxation of BC gives important insight on the structure of the original problem. For the BC LP, we have the following result.

Theorem 12. *(Edmonds 1967 [9]) Given an instance of* STP *with $R = V$, then the value of an integral optimal solution for* BC *equals the value of an optimal solution of the relaxation.*

Although the LP of BC has exponential size in regards to the number of vertices, the corresponding separation problem can be solved in polynomial-time, and thus there is an optimal solution can be obtained in polynomial-time (in the size of STP instance). The proof for this claim is very similar to the one given for Lemma 20 in Chapter 4.

2.6 Related Works

STP is a classical NP-hard problem. Indeed, even fiding a solution that costs at most $\frac{96}{95}$ times the optimum is NP-hard [7]. Therefore, we do not expect to design an approximation algorithm with factor smaller than $\frac{96}{95}$. It is also well known that STP can be reduced to a metric instance preserving any approximation factor [20]. Because a minimum cost tree spanning only the terminals costs at most twice as the cheapest one, and finding such tree can be done in polynomial-time (e.g. Prim's algorithm [8]), we conclude that there exists an approximation algorithm with factor 2 for STP [20]. In the variant where the STP instance considered is geometric, i.e., the set of terminals is a finite subset of $\mathbb{R}^d$, Steiner vertices can be selected as any point of $\mathbb{R}^d$, for constant d, and the metric is defined by the Euclidean norm, there exists a $(1+\varepsilon)$-approximation, where ε is an arbitrarily small positive constant [2].

It was a long standing open problem to obtain an approximation with factor smaller than 2 for STP. A sequence of improved approximation algorithms appeared after Borchers and Du, in 1997, showed that, to obtain a good approximation for MSTP (thus also for STP), it was sufficient to focus on the k-restricted MSTP [5]. For a constant k, we can compute in polynomial-time the whole set $\mathscr{C}_k$, and, while the best solution for the restricted problem might be more expensive than an optimal solution for MSTP, Borchers and Du found that this difference could only rise by a very small factor dependent only on k, which decreases as k grows. In the literature, many approximation algorithms improved the running time, or improved the approximation factor for the STP and they are mostly based on the idea of full components [6, 19, 22]. Zelikovsky gave an $\frac{11}{6}$-approximation which greedly contracts cheap full components with 3

terminals [22]. Later, an 1.55-approximation algorithm, given by Robins and Zelikovsky, iteratively chooses full components that maximizes the improvement of a partial solution but commits to the full component just partially, this allows possibly "conflicting" full components to be accepted later [19]. Currently, the best known approximation algorithm for STP was given by Byrka et al. and has approximation factor of 1.39 [6]. This last algorithm is an LP based probabilistic algorithm and uses a novel technique named as *iterative randomized rounding*.

An also considered variant is the *Prize-Collecting Steiner Tree Problem* (PCSTP). In this problem, there is no distinction between terminals and Steiner vertices, but, besides a non-negative cost associated to each edge, a non-negative penalty cost $\pi(v)$ has to be paid if vertex v is not included the solution. Any tree is a feasible solution, but an optimal one has to balance the cost of included edges and the penalty of neglected vertices. As in the case of STP, it was a long stading open question to obtain an approximation strictly better than 2. The current best approximation for this problem runs Byrka et al.'s algorithm as a black box and has an approximation factor of 1.97. This factor goes down to $\frac{17}{9}$ if STP could be solved optimally [1]. Although PCSTP is similar to MNSTP, the restriction that penalty costs must be non-negative prevents a direct reduction between the two problems.

Many variants of the STP consider specific kind of trees as feasible solutions. For example, the *Partial-Terminal Steiner Tree Problem* (PTSTP) is the particular case of MSTP where every vertex of a distinguished subset of the terminals $R' \subseteq R$ must be a leaf in the solution, which admits an algorithm with approximation factor of 2.13 [13]. This approximation algorithm runs the algorithm from Byrka et al. as black box. Another particular case is the "inverse" of PTSTP, the *Selected-Internal Steiner Tree Problem* (SISTP), where every terminal from a given subset R' of R may not be a leaf in the solution. The best approximation for this problem is 2.39 (when considering the additional constraint that $|R \backslash R'| \geq 2$) [15] and the restricted case where $R' = R$, known as the *Internal Steiner Tree Problem* (ISTP) has an approximation of 3.78 [14]. Both PTSTP and SISTP are equivalent to STP when $R' = \emptyset$.

An important special case of MSTP is the case where the edges of considered graphs have weight either 1, or 2, and is known as the *Steiner Tree Problem with Metric* 1 *and* 2 (STP(1,2)). The best known approximation is 1.25 [3], and uses a greedy strategy with a non-trivial analysis based on the *distributed potential of budget*. Moreover, there are improved approximation factors for special cases of ISTP(1,2), PTSTP(1,2), and TSTP(1,2) with approximation factors of $\frac{9}{7}$, $\frac{5}{3}$, and 1.42, respectively [14, 16, 21].

A problem that is closely related to MNSTP is the Node-Weighted Steiner Tree (NSTP). Differently from the MNSTP, this problem has no restrictions on the set of edges nor on the edge weight function. Assuming P $\neq$ NP, NSTP is strictly more general than MNSTP, since there is no approximation-preserving reduction from NSTP to MNSTP, unless there is a constant approximation the Set Cover problem, which is NP-hard to approximate within $(1-\delta) \cdot \ln(|U|)$, for any $\delta > 0$ where U is the set of elements [10]. In fact, the best currently known approximation has a factor of $1.35 \cdot \ln(|R|)$, where R is the terminal set [12]. This inapproximability result for NSTP shows that assuming that the graph is metric has significant impact on the complexity of MNSTP. However, to our knowledge, there are no approximation algorithms in the literature for this specific variant.

Chapter 3

Results on MNSTP and on q-MNSTP

In this chapter we give an approximation algorithm for q-MNSTP and then we prove an important relation between q-MNSTP and its k-restricted variant. Throughout the thesis we denote by Opt, Opt^q and Opt_k^q optimal solutions of MNSTP, q-MNSTP and k-restricted q-MNSTP, respectively. Because every full components composed by exactly two terminals and a single edge connecting them are always included in $\mathscr{C}_k$, for any $k \geq 2$, we abuse notation considering spanning trees on $G[R]$ also as a valid k-restricted Steiner tree.

3.1 An approximation for MNSTP

Consider the ratio $\alpha := w_v(\text{Opt})/w(\text{Opt})$ and, similarly, let $\alpha_k := w_v(\text{Opt}_k^q)/w(\text{Opt}_k^q)$.

Theorem 13. *If T is a minimum cost tree spanning only the terminals in $G[R]$, then $w_e(T) \leq 2(1-\alpha)w(\text{Opt})$. Also, $w_e(T) \leq 2(1-\alpha_k)w(\text{Opt}_k^q)$, for every constant k.*

Proof. In a metric graph G, given edges $\{u,w\}$ and $\{w,v\}$, we refer to "shortcut" a vertex w as the procedure of replacing edges $\{u,w\}$ and $\{w,v\}$ by $\{u,v\}$ in a subgraph H of G.

Consider an optimal solution Opt for MNSTP, and let T be a minimum spanning tree in $G[R]$. Recall that $w_v(T) = 0$, then, by duplicating the edges of Opt, we obtain a connected multigraph M where each vertex has an even degree (an Eulerian multigraph) and whose edges weight exactly $2w_e(\text{Opt})$. Consider an Eulerian circuit C on M, and shortcut each Steiner vertex, obtaining a cycle C' on R. Clearly $w_e(T) \leq w_e(C') \leq w_e(M) = 2w_e(\text{Opt})$.

Now let $\alpha := w_v(\text{Opt})/w(\text{Opt})$. We get

$$w_e(T) \leq 2w_e(\text{Opt}) = 2(w(\text{Opt}) - w_v(\text{Opt})) = 2(1-\alpha)w(\text{Opt}).$$

Since T is also a feasible solution to the k-restricted q-MNSTP, we also have that

$$w_e(T) \leq 2(1-\alpha_k)w(\text{Opt}_k^q),$$

where $\alpha_k := w_v(\text{Opt}_k^q)/w(\text{Opt}_k^q)$. □

Corollary 14. *There is a polynomial-time algorithm for* MNSTP *with approximation factor* 2.

Proof. Theorem 13 proves that the minimum spanning tree in $G[R]$ has weight cost of at most 2 times the cost of an optimal solution, thus any algorithm that computes a minimum spanning tree on a graph is a 2-approximation algorithm for q-MNSTP. Example of such algorithms are the Prim's algorithm and Kruskal's algorithm [8]. □

While straightforward, this theorem gives insights on an optimal solution. In particular, for any optimal solution, the total vertex weight cannot exceed the total edge weight, i.e., $0 \leq \alpha \leq \frac{1}{2}$.

Corollary 15. *For any optimal solution* Opt *for an instance of* MNSTP *and any optimal solution* Opt_k^q *for an instance of the k-restricted* q-MNSTP, $w_v(\text{Opt}) \leq w_e(\text{Opt})$ *and* $w_v(\text{Opt}_k^q) \leq w_e(\text{Opt}_k^q)$.

Proof. Because a minimum spanning tree T on R is a feasible solution for MNSTP, by Theorem 13, we have

$$w(\text{Opt}) \leq w_e(T) \overset{Thm\ 13}{\leq} 2(1-\alpha)w(\text{Opt}) = 2(w(\text{Opt}) - w_v(\text{Opt})) = 2w_e(\text{Opt}),$$

which implies that $w_v(\text{Opt}) \leq w_e(\text{Opt})$. Similarly, we can prove the result to Opt_k^q. □

3.2 Reduction of q-MNSTP to k-restricted q-MNSTP

Borchers and Du [5] showed that the optimal value for k-restricted MSTP is not much larger than the optimal value for MSTP as k increases. To prove a similar result for q-MNSTP, we need to prepare a few lemmas beforehand.

Lemma 16. *(Borchers and Du 1997 [5]) For any regular binary tree B, there exists a one-to-one function f from internal vertices to leaves, such that:*

(A) for any internal vertex u, $f(u)$ is a descendant of u;

(B) the set of paths $p(u)$ from u to $f(u)$ are edge disjoint; and

(C) there is a leaf v such that the path from the root to v is edge disjoint from every other path $p(u)$.

Proof. We prove by induction on the height of the tree. If the height is 0, all three statements are trivially true.

Let B be a regular binary tree of height $h \geq 1$ rooted at d. Since B has two smaller regular binary subtrees B_1 and B_2, each rooted at one of the children of d, of height strictly lesser than h, by induction hypothesis there is a function f_1 and f_2 on the internal vertices of B_1 and B_2, respectively, that satisfy statements (A) and (B) and leaves v_1 and v_2 that satisfy statement (C) on each of the subtrees. Define a function f on the internal vertices of B as

$$f(u) = \begin{cases} f_1(u), & \text{if } u \in B_1, \\ f_2(u), & \text{if } u \in B_2, \\ v_1, & \text{if } u \text{ is the root of } B. \end{cases}$$

Since f satisfies (A) and (B) and v_2 satisfies (C), we concluded the proof. □

Consider k-restricted q-MNSTP for some constant $k \geq 2$, and let $r, s \in \mathbb{Z}_{\geq 0}$, with $0 \leq s < 2^r$, such that $k = 2^r + s$, and let T be a regular binary tree. We define the following labeling process using labels from the set $\{1, 2, ..., r2^r + s\}$. Starting at the first level, the level of the root, we label the root with the set $\{1, 2, ..., 2^r\}$. Then we label the two vertices on the second level with the sets $\{2^r + 1, 2^r + 2, ..., 2^r + 2^r\}$. Up to the r-th level we label every vertex on the i-th level, $1 \leq i \leq r$ with the set $\{(i-1)2^r + 1, (i-1)2^r + 2, ..., (i-1)2^r + 2^r\}$.

To label the remaining vertices, we follow up with an inductive labeling: assuming that the first i levels have already been labeled, $i \geq r$, and that the disjointness property is held up to the i-th level, we label the vertices in the $(i+1)$-th level by doing:

(R1) Let v be a vertex at level $i + 1 - r$ with label set $S_v = \{l_1, l_2, ..., l_{2^r}\}$. Label the j-th descendant of v on level $i + 1$ with the set $S_j = \{l_j, l_{j+1}, ..., l_{j+2^r-s-1}\}$, where each subscript is reduced by (mod 2^r) to maintain the subscripts between 1 and 2^r.

(R2) For a vertex at level $i + 1$, add to its label set the s labels that are not in the label sets of any of its immediate r ancestors.

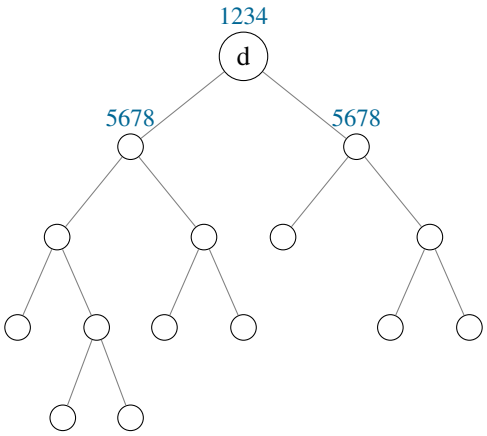
Figure 3.1 exemplifies this labeling rules. This labeling has some interesting properties, from which the main one is the following lemma.

Lemma 17. *(Borchers and Du 1997 [5]) Let T be a regular binary tree, then each vertex v receives exactly 2^r distinct labels by the labeleing process and the label sets of any $r - 1$ immediate ancestors of v, and the label set of v, are disjoint.*

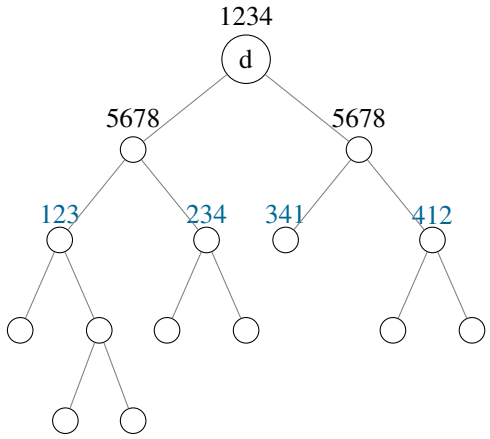
Proof. It is clear that the disjointness property is preserved up to the r-th level. For the inductive part of the process, note that before we apply Rule (R2), since a vertex v, at level $i + 1 - r$, has at most 2^r descendants on level $i + 1$, we have that at most $S_1, S_2, ..., S_{2^r}$ label sets are created by Rule (R1) on that level, where each label set S_j has exactly $2^r - s$ elements and each label from the set S_v appears in at most $2^r - s$ of these sets (at this level). By our inductive hypothesis, the disjointness property is valid up to the i-th level, so the r immediate ancestors of a vertex at level $i + 1$ are labeled by r disjoint sets of size 2^r, thus there are exactly s unused numbers from the set $\{1, 2, ..., r2^r + s\}$, which are then added by Rule (R2). As the labels added by Rule (R2) are also different from the ones added by Rule (R1), each vertex at level $i + 1$ are given exactly 2^r labels by the two rules. □

Next, we finally extend the result by Borchers and Du [5], which is the core result to obtain the approximation factors from Chapter 4 and 5. The idea is to, given a fixed optimal solution $\text{Opt}^q(I)$ of an instance I of q-MNSTP, transform $\text{Opt}^q(I)$ into a regular binary tree T and apply the inductive labeling process, which induces $r2^r + s$ distinct choices of k-restricted Steiner trees, one tree per label. We then show that at least one of these trees does not weight much more than $\text{Opt}^q(I)$, which also implies that an optimal k-restricted Steiner tree $\text{Opt}_k^q(I)$ also does not weight much more than $\text{Opt}^q(I)$. Define the (k,q)-*Steiner ratio* as $\rho_k^q := sup_I \left\{ \frac{w(\text{Opt}_k^q(I))}{w(\text{Opt}^q(I))} \right\}$ for every instance I of q-MNSTP, then we have the following.

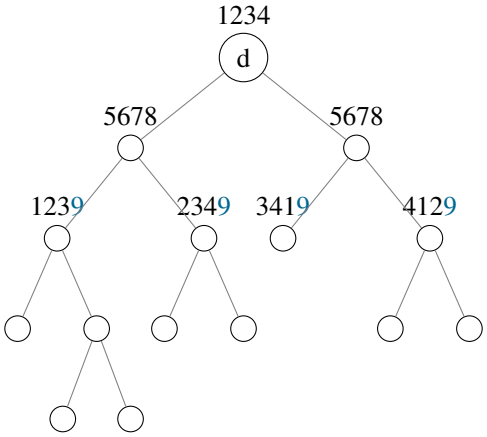
Theorem 18. $\rho_k^q \leq 1 + \delta(k,q)$, *where* $\delta(k,q) := (q+1)/\lfloor log_2 k \rfloor$.



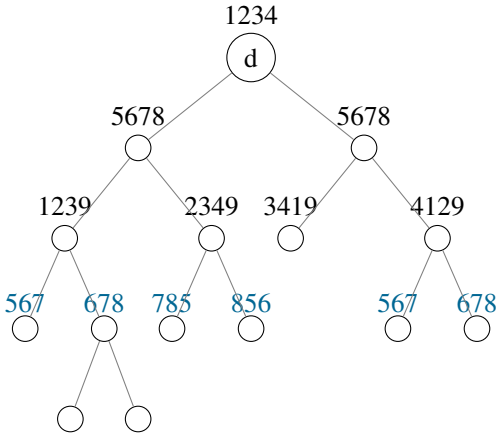
(a) We first label every vertex in the i-th level, $1 \leq i \leq r$, with the label set $\{(i-1)2^r+1, (i-1)2^r+2, ..., (i-1)2^r+2^r\}$.



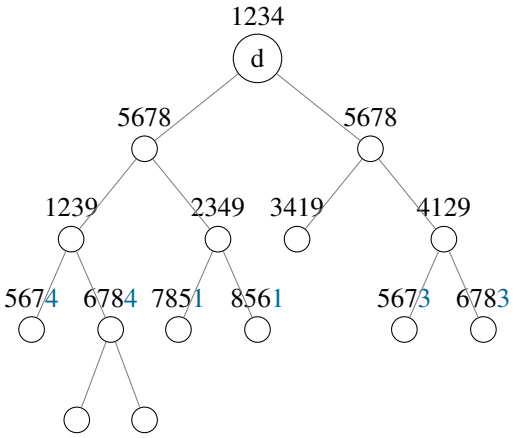
(b) Then we apply Rule (R1) on the third level.



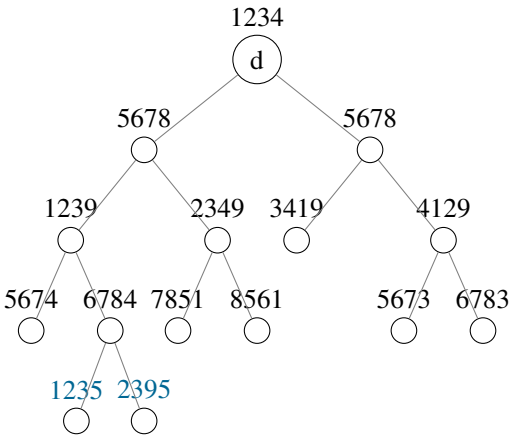
(c) Then we apply Rule (R2) on the third level.



(d) Then we apply Rule (R1) on the forth level.



(e) Then we apply Rule (R2) on the forth level.



(f) Then we apply Rules (R1) and (R2) on the fifth level.

Figure 3.1: Example of the labeling of a regular binary tree when $k = 5 = 2^2 + 1$, so $r = 2$ and $s = 1$, where d is the root, as described in the proof of Lemma 17 on the regular binary tree from Figure 2.6b.

Proof. Recall that $k = 2^r + s$, $r, s \in \mathbb{Z}_{\geq 0}$, with $0 \leq s < 2^r$. We prove that, for any $k \geq 2$ and any instance I of q-MNSTP, the following holds through induction on $|R|$:

$$\frac{w(\text{Opt}_k(I))}{w(\text{Opt}(I))} \leq 1 + \frac{2^r(q+1)}{r2^r + s}.$$

If $k \geq |R|$ the inequality holds trivially, since for every maximal subgraph tree T' of $\text{Opt}^q(I)$, which leaves coincides with terminals, we have that $T' \in \mathscr{C}_k$. Thus $\text{Opt}^q(I) = \text{Opt}^q_k(I)$.

If $k < |R|$, fix an optimal solution $\text{Opt}^q(I)$. If $\text{Opt}^q(I)$ has a terminal vertex v with degree $d(v) \geq 1$, we split $\text{Opt}^q(I)$ on v into two Steiner trees, each with fewer terminals than $|R|$. Denote the terminal set on each of these trees by R_1 and R_2 and denote by $opt(R_i)$ and $opt_k(R_i)$ the total weight of an optimal solution and the minimum weight k-restricted Steiner tree respectively. Then $w(\text{Opt}^q(I)) = opt(R_1) + opt(R_2)$ and $w(\text{Opt}^q_k(I)) \leq opt_k(R_1) + opt_k(R_2)$. Then, under the hypothesis that the claim is valid when the number of terminals is smaller than $|R|$, it follows that:

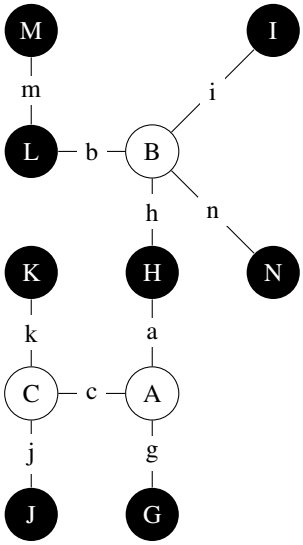
$$\frac{w(\text{Opt}^q_k)}{w(\text{Opt}^q)} \leq \frac{opt_k(R_1) + opt_k(R_2)}{opt(R_1) + opt(R_2)} \leq max\left\{\frac{opt_k(R_1)}{opt(R_1)}, \frac{opt_k(R_2)}{opt(R_2)}\right\} \overset{I.H.}{\leq} 1 + \frac{2^r(q+1)}{r2^r + s}.$$

Thus we may consider only the case which every terminal vertex of Opt^q is a leaf.

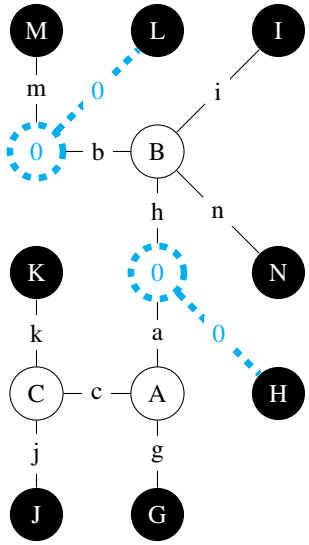
We turn Opt^q into a regular binary tree T by adding weighted copies of vertices and dummy edges of weight zero such that the root is a dummy vertex of weight zero: for every non-leaf terminal v, append a dummy vertex v' of weight zero to v with a zero weight dummy edge and replace v with v' in the set of terminals; for any non-terminal vertex u with degree $d(u) \geq 4$, replace u, with a path $(u_1, \{u_1, u_2\}, u_2, \{u_2, u_3\}, u_3 ..., \{u_{d-1}, u_{d-2}\}, u_{d(u)-2})$ where each u_i is a weighted copy of u but every edge $\{u_i, u_{i+1}\}$ in this path has weight zero, then take u_1 and $u_{d(u)-2}$ to inherit two edges of u each, and every other vertex in the path to inherit one edge of u; finally shortcut any vertex with degree 2 and split an arbitrary edge with a dummy vertex of weight zero to become the root (one of the edges incident to the root inherits the weight of the split edge and the other weights zero). Figure 3.2 shows an example of this process. Without loss of generality, we assume that the weight of the lightest edge from the original graph weights exactly 1. This simplify the calculations for it implies that the weight of any vertex in the graph cannot exceed q.

Note that now T has a nice structure: T is a regular binary tree, each vertex of T is a terminal if it is a leaf, and it is a Steiner vertex otherwise. Lemma 17 states that we can label the vertices of T, using $\{1, 2, ..., r2^r + s\}$ labels, in such a way that each vertex v receives exactly 2^r distinct labels and the label sets of any $r-1$ immediate ancestors of v, and the label set of v, are disjoint. We then proceed to construct $r2^r + s$ k-restricted Steiner trees where each label l generates a k-restricted Steiner tree T_l.

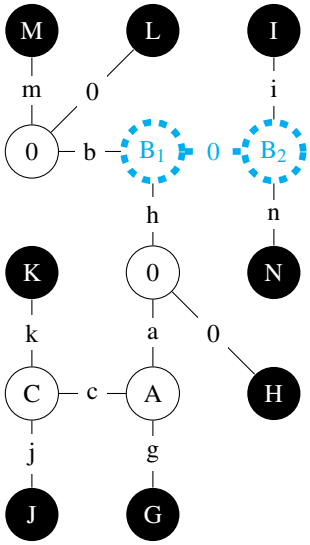
First, select the root of T and each Steiner vertex v labeled with l. Each such vertex become the root of a new subtree that will correspond to a full component in T_l, and thus they are called *component roots*. Each component root v is connected to every descendant u that is also labeled with l, and whose path from v to u in T has no other vertex label with l. These vertices u are called the *intermediate leaves*. Moreover, v is also connected to a leaf u if the path from v to u in T contains no vertex labeled l. The set of paths for each component root v induce a partial



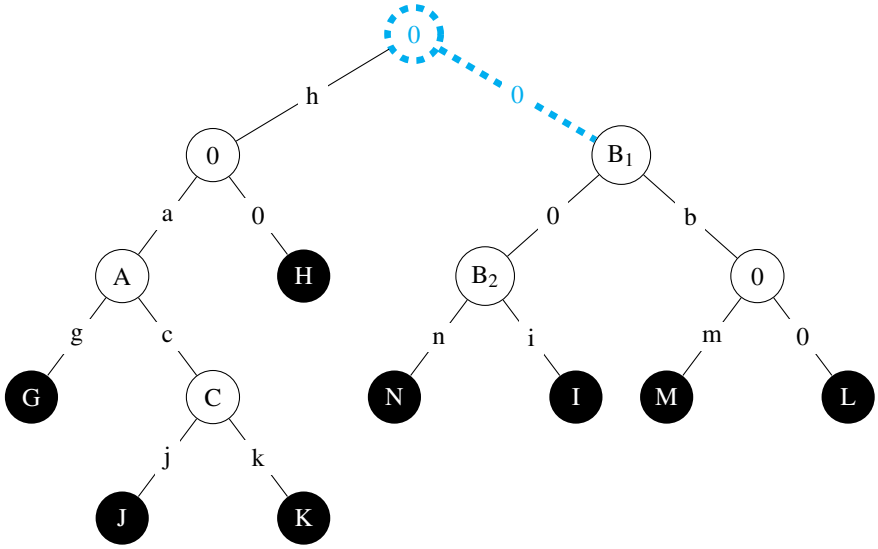
(a) Steiner tree S.



(b) Terminals that are internal vertices become leaves.



(c) Vertices with degree greater than 3 are replaced by paths.



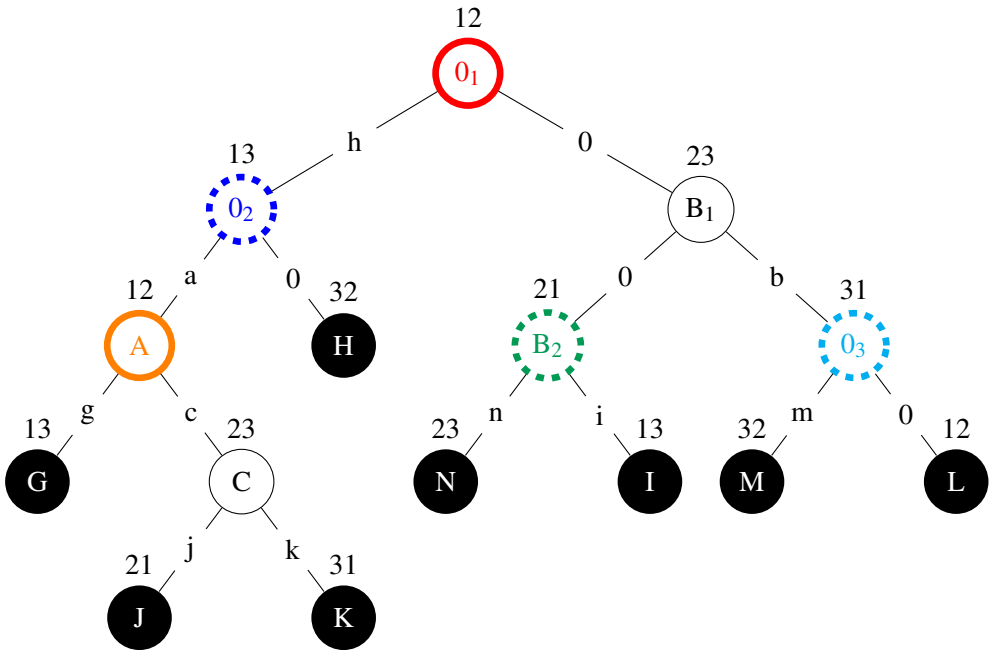
(d) Splitting an arbitrary edge with a root vertex of weight 0 yields a regular binary tree. Note that the regular binary tree has the same edge weight of the original Steiner tree but the vertex weight increased as now there are two vertices with the weight of B.

Figure 3.2: An example of transformation of a Steiner tree S into a regular binary tree T. Steiner vertices are white and terminals black, and the weight of each edge is indicated on the edge.

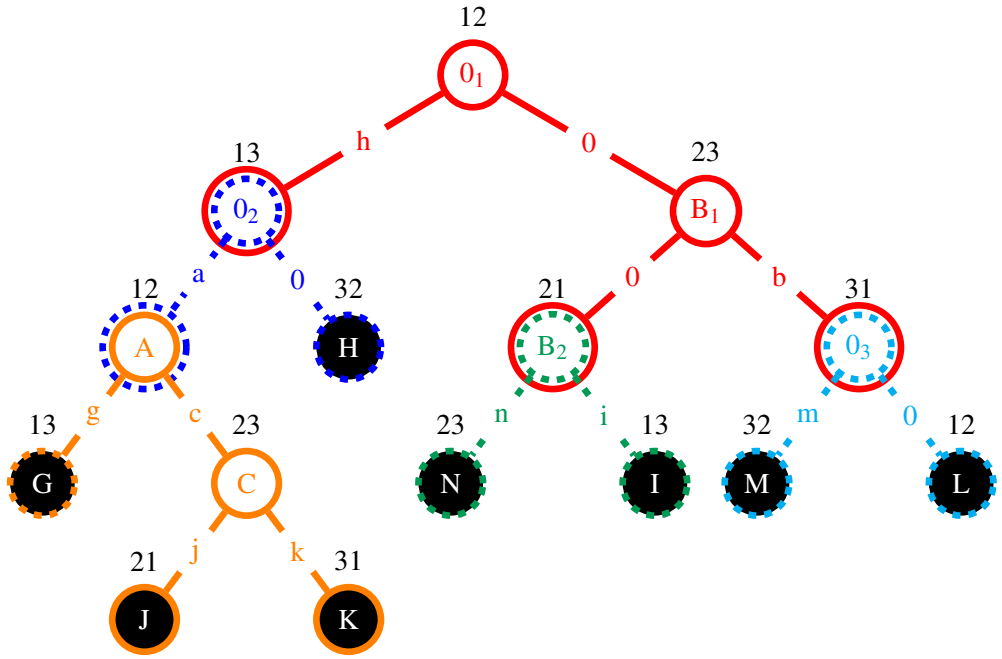
subtree. Notice that intermediate leaves are Steiner vertices.

To complete the description of each subtree, and obtain a full component, we connect each intermediate leaf, that is a Steiner vertex, to a distinct terminal. Let f be a function as described in Lemma 16. Connect each intermediate leaf u to $f(u)$ using the path $p(u)$ in T. The union of all subtrees constructed so far form an intermediary tree T_l'. Figures 3.3a, 3.3b, and 3.3c illustrate graphically the construction of T_l'.

Now we prove that T_l' spans the terminal set R and that each subtree has at most k terminals. First we prove that each tree T_l' spans the terminal set. Let v be a component root and D_v the set of descendants of v in T. We show by induction on the height of v that $T_l'[D_v]$ spans the



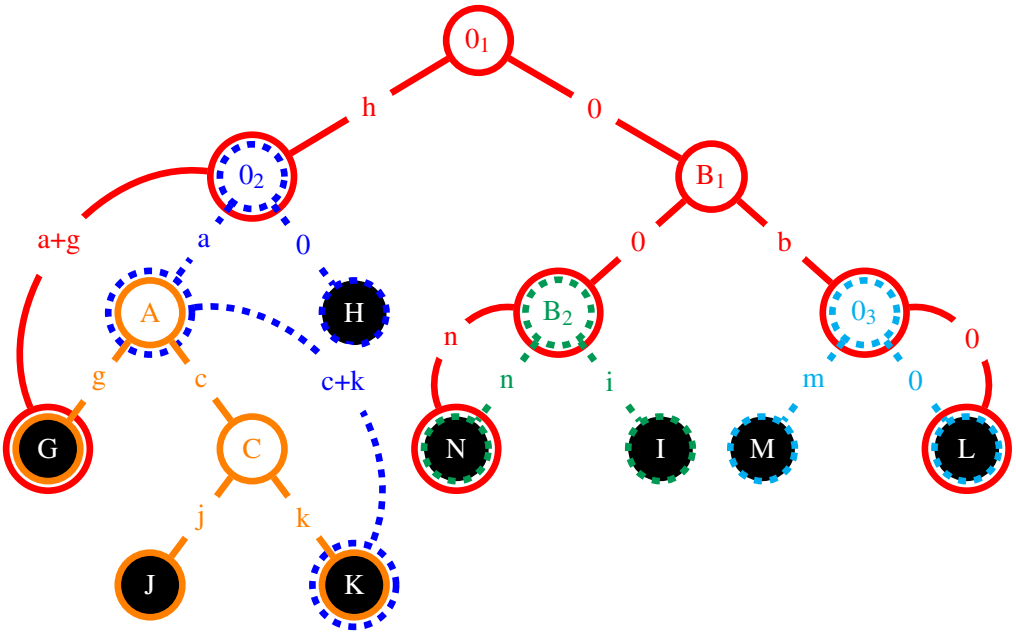
(a) We first label the vertices as described in the proof of Lemma 17. Each internal vertex (and the root) with label 1 becomes a component root. In this example we have five component roots and each is drawn with a different color and pattern.



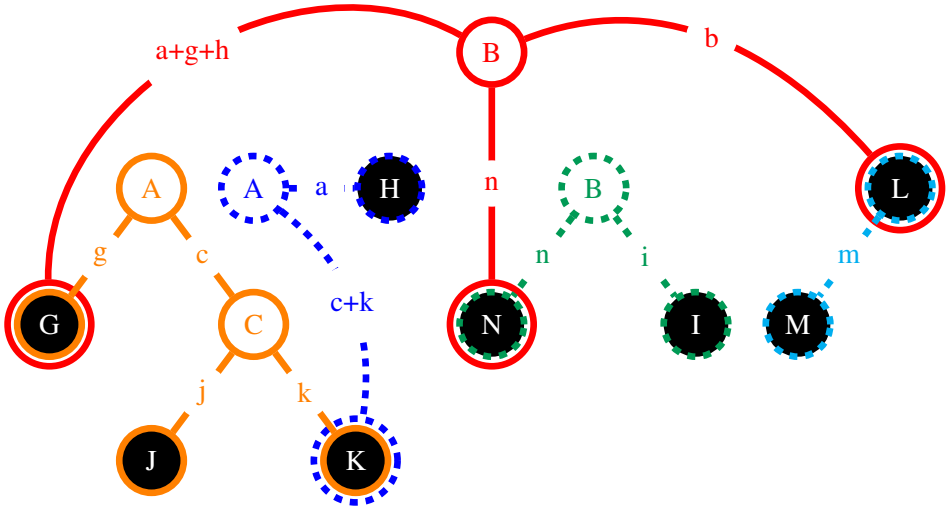
(b) Then each component root extends to each first occurrence of a descendant with label 1 (intermediate leaves) or a leaf (terminal) descendant.

terminals in $T[D_v]$. If the height is 0, then the claim is trivially true, so suppose the height is at least 1. Let C_v be the subtree corresponding to v and consider the subtrees rooted at the intermediate leaves $u_1, u_2, \ldots, u_n$ of C_r. By induction, $T_l'[D_{u_i}]$ spans every terminal in $T[D_{u_i}]$. Since C_v connects each of these subtrees, we conclude $T_l'[D_v]$ spans $T[D_v]$. This completes the induction. By making v the root of T, we conclude that T_l' spans R.

To show that each subtree has at most k terminals, consider a component in T_l' that has



(c) Each intermediate leaf u extends, through paths on T, to $f(u)$ where f has the properties of Lemma 16 and, in this example, $f(0_2) = \mathrm{G}, f(\mathrm{A}) = \mathrm{K}, f(\mathrm{B}_2) = \mathrm{N}, f(0_3) = \mathrm{L}$. This completes the tree T_1'. Each subtree of T_l' is drawn with a distinct color and pattern.



(d) The k-restricted Steiner tree T_1 obtained by removing copied vertices within each subtree and short-cutting vertices of degree 2 from T_1'. Here, the weight of the edges are an upper bound to the actual edge weights. Each component is drawn with a different color and pattern.

Figure 3.3: An example of a the tree T_1 generated by a the label 1, where $k = 3 = 2^1 + 1$, i.e., $r = 1$ and $s = 1$, on the regular binary tree from Figure 3.2d. Do note that the tree T_1 weights more than T, specifically, $0 \leq w_e(T_1) - w_e(T) \leq a + g + c + k + n$ and $w_v(T_1) - w_v(T) = w_v(A) + w_v(B)$.

a component root v. Without loss of generality, we do not consider subtrees which reach a terminal (leaf of T), as their size may only be smaller. We consider two cases for the subtrees separately.

In the first case, suppose v is not labeled by l. This implies that v is the root of T and thus

$l \geq 2^r + 1$. By our labeling process, the labels $2^r + 1, 2^r + 2, ..., r2^r$ all appear in the first $r-1$ levels below the root and by Rule (R2) every remaining s labels are then placed on the r-th level. Because T is a regular binary tree, we conclude that the subtree with v as component root has at most $2^r \leq k$ intermediate leaves, and thus it has at most $2^r \leq k$ terminals.

In the second case, consider that v is labeled by l. Then by the Rule (R1) of the labeling process, we know that exactly s vertices, r levels below v, are not labeled by l, and that all the other remaining $2^r - s$ descendants of v, at the same level, are indeed labeled with l. Consider, then, a vertex w that is among these vertices that is not labeled by l. Since l labels v, by the disjointness property, no vertex between w and v is in the unique path P_{vw} between the two is labeled by l. Because $length(P_{uv}) = r + 1$ and v is the only vertex labeled by l among all the vertices in this path, by Rule (R2) of the labeling process, the children of w are, for sure, labeled with l. So the subtree with component root v has exactly $2s$ descendants intermediate leaves at $r+1$ levels below and v and $2^r - s$ intermediate leaves r levels below, adding up to $2^r + s = k$ intermediate leaves. Thus this subtree spans exactly k terminals.

We now describe how the final k-restricted Steiner tree T_l is created. Each subtree of T'_l has its own set of vertices and edges, and will determine a full component. First, replace each path $p(u)$, from an intermediate leaf u to leaf $f(u)$, by an edge between u and $f(u)$; then, contract each set of vertices connected by zero-weight edges into a new vertex whose weight is the weight of the heaviest vertex (notice that the vertices in such a set are either copies of the same vertex, or dummy vertices of weight 0). By applying these two operations, we ensure that each component incurs the cost of only a single copy of each vertex in the component. Finally, we shortcut any vertex with degree of 2. Because each full component is a tree where leaves coincide with terminals and because we shortcut vertices of degree 2 in the end, the number of Steiner nodes, i.e., the number of internal leaves, does not exceed $k-2$ (see Figure 3.3d).

The cost of each k-restricted Steiner tree T_l is upper bounded by the total cost of the original tree $\text{Opt}^q(I)$ plus the edges and vertices added in the process. The additional edges are due the paths $p(u)$ from each intermediate leaf u to $f(u)$. The additional vertices correspond to copies of vertices that appear in different full components of T_l (see Figure 3.3c). For a fixed label l, let L_l denote the total length of paths $p(u)$ from intermediate leaves u to the tree leaves. Also, let V_l denote total weight of intermediate leaves of T'_l. Notice that vertices that are repeated in different components of T_l must correspond to intermediate leaves.

Now, we consider the full set of labels. Since each internal vertex v in T is an intermediate leaf in exactly 2^r of the trees T'_l, each path $p(u)$ is counted exactly 2^r times in the sum $L_1 + L_2 + ... + L_{r2^r+s}$. Also, because the set of paths $p(u)$ are disjoint, this sum will be at most 2^r times the total edge weight of T. Formally,

$$\sum_{i=1}^{r2^r+s} L_i \leq 2^r w_e(T).$$

Recall that any regular binary tree with $|R|$ leaves has at most $|R|-1$ inner vertices, and any tree spanning the terminal set R has at least $|R|-1$ edges. Also, recall that each edge has weight

at least 1. We obtain

$$\sum_{i=1}^{r2^r+s} V_i = 2^r w_v(T_l') \leq 2^r \max_{v \in V(\text{Opt}^q)} \{w_v(v)\}(|R|-1) \leq 2^r q(|R|-1) \leq 2^r q \cdot w_e(T).$$

And since $w_e(T) = w_e(\text{Opt}^q(I))$:

$$\sum_{i=1}^{r2^r+s} (L_i + V_i) \leq 2^r(q+1)w_e(T) = 2^r(q+1)w_e(\text{Opt}^q(I)).$$

Thus, there is a $d \in \{1,2,...,r2^r+s\}$ such that

$$(r2^r+s)(L_d+V_d) \leq 2^r(q+1)w_e(\text{Opt}^q(I)) \leq 2^r(q+1)w(\text{Opt}^q(I)).$$

Then the total cost of the k-restricted Steiner tree T_d is upper bounded by

$$w(T_d) \leq L_d + V_d + w(\text{Opt}^q(I)) \leq \left(1 + \frac{2^r(q+1)}{r2^r+s}\right) w(\text{Opt}^q(I)),$$

which implies that

$$\frac{w(\text{Opt}_k^q(I))}{w(\text{Opt}^q(I))} \leq \frac{w(T_d)}{w(\text{Opt}^q(I))} \leq 1 + \frac{2^r(q+1)}{r2^r+s} \leq 1 + \delta(k,q),$$

where $\delta(k,q) := \frac{q+1}{\lfloor log_2 k \rfloor}$. Since $\rho_k^q := sup_I\{w(\text{Opt}_k^q(I))/w(\text{Opt}^q(I))\}$ we have proved the desired result. □

From the perspective of approximation algorithms this result implies that one can reduce the q-MNSTP to its k-restricted variant preserving the approximation, i.e., there is an $(\alpha + \delta(q,k))$-approximation for q-MNSTP if there exists an α-approximation for the k-restricted q-MNSTP.

Chapter 4

A Probabilistic 1.62-Approximation to q-MNSTP

In this chapter we show how one could adapt the iterative randomized rounding technique introduced by Byrka et al. [6] to the k-restricted q-MNSTP. At each iteration, the algorithm solves a relaxed LP, with respect to the current instance, and contracts a full component chosen randomly over a distribution determined by LP solution. When there are no more components to be chosen, the algorithm returns the graph induced by the edges of full components that were contracted throughout its execution. The expected cost of the returned solution is the sum of expected costs of each sampled full component, and the cost of a full component corresponds to the sum of vertex and edge weights. Recall that in the k-restricted Steiner tree problems multiple full components may contain a common vertex, and thus the total cost of a solution accounts the cost of a vertex multiple times, once for each full component that contains it.

4.1 Linear Programming Modeling

In this section we introduce two LP models. The first LP models q-MNSTP and it is a variation of a very well-studied LP known as *bidirect cut relaxation* (BCR) [6]. Given an instance of q-MNSTP, let $G = (V,E)$ be the instance graph and fix an arbitrary terminal $r \in R$ as the root. Next, we build a directed graph $G' = (V(G),A)$ by taking the vertex set of G and replacing every edge $\{u,v\}$ of G by two arcs (u,v) and (v,u) with weight $w_e(\{u,v\})$. For a given subset $U \subseteq V$, we define $\delta^+(U) = \{(u,v) \in A : u \in U, v \notin U\}$ to be the set of arcs leaving U. The LP is

$$\begin{array}{lll} \text{minimize} & \sum_{a\in A} w_e(a)x_a + \sum_{v\in V} w_v(v)y_v & \\ \text{subject to} & \sum_{a\in\delta^+(U)} x_a \geq 1, & \forall U \subseteq V\setminus\{r\}, U\cap R \neq \emptyset; \\ & y_v \geq x_a, & \forall a \in A, a = (u,v); \\ & y_u \geq x_a, & \forall a \in A, a = (u,v); \\ & x_a \geq 0, & \forall a \in A; \\ & y_v \geq 0, & \forall v \in V. \end{array} \tag{BCR}$$

We can think of the value x_a as the capacity of an installed arc a and y_v as the capacity of an installed vertex v. Thus, the LP may be seen as computing the minimum-cost capacities that

support a flow of 1 from each terminal to the root. It is clear that when the vertex weight of every vertex of the input graph is zero, BCR reduces to a model that solves STP. In q-MNSTP, because terminals have zero weight, i.e., $w_v(v) = 0$ for $v \in R$, whenever $R = V$, Theorem 12 (that states the this LP is integral when all vertices are terminals) also applies.

Corollary 19. *Given an instance of* q-MNSTP*, if* $R = V$*, the polyhedron of* BCR *is integral.*

The algorithm we present starts with the set of full components $\mathscr{C}_k$ and creates a set $\mathscr{D}_k$ of directed trees (*directed components*) by making a copy D of every $C \in \mathscr{C}_k$ for each choice of terminal v in C, and directing all edges of D towards v; in such a case, v is identified by $sink(D)$ and the set of the other terminals in $V(D)$ is denoted by $source(D)$, and we also say that C is the non-directed full component of D. Note that, if k is constant, this takes polynomial-time. Let $\delta_k^+(U)$ denote the set of directed components with $sink(D_j)$ not in U and at least one terminal from $source(D_j)$ in U, then, we solve the LP below, for an arbitrarily chosen terminal root r, where x_i is the indicating variable of component D_i. The algorithm runs iteratively, by choosing at each iteration a component D_i with probability $x_i / \sum_{D_j \in \mathscr{D}_k} x_j$.

$$\begin{array}{lll} \text{minimize} & \sum_{D_j \in \mathscr{D}_k} w(C_j) x_j & \\ \text{subject to} & \sum_{D_j \in \delta_k^+(U)} x_j \geq 1 & \forall U \subseteq R \setminus \{r\}, U \neq \emptyset, \\ & x_j \geq 0 & \forall D_j \in \mathscr{D}_k, \end{array} \qquad (k\text{-DCR})$$

Lemma 20. *(Byrka et al. 2013 [6])* k-DCR *can be solved in polynomial-time if* k *is constant.*

Proof. By Theorem 11, it suffices to show that, for any unfeasible solution, we can find a constraint that is violated, in polynomial-time.

An *s-r cut* is a subset $U \subseteq V'$ such that $s \in U$ and $r \notin U$. Construct a directed graph $G' = (V', A)$, with vertex set $V' = R \cup \{v_j : j = 1, ..., |\mathscr{D}_k|\}$, and, for every directed component D_j, insert an arc (u, v_j) for each $u \in source(D_j)$, and an arc $a_j = (v_j, sink(D_j))$. Setting the capacity of each arcs a_j to $c(a_j) = x_j$ and $c(a) = \infty$ for all the other arcs, we have that for any terminal $s \in R \setminus \{r\}$, there is an s-r cut $U \subseteq V'$ of minimum capacity $\sum_{a \in \delta^+(U)} c(a)$, consisting only of arcs a_j (for every $u \in U \cap R$, it suffices that every $(u, v_j) \in A$ is also in U, see Figure 4.0). In particular, given a non-empty subset $R' \subseteq R \setminus \{r\}$, there is a cut U such that

$$\sum_{a \in \delta^+(U)} c(a) = \sum_{a_j : D_j \in \delta_k^+(R')} c(a_j) = \sum_{D_j \in \delta_k^+(R')} x_j,$$

i.e., the capacity of a minimum-cut from G' identifies a constraint with the minimum value with respect to a subset of vertices containing s. Since finding an s-r minimum-cut in G' can be done in polynomial-time, e.g., using Edmonds-Karp algorithm [8], we can find a violated constraint if there is any by solving at most $|R| - 1$ maximum flow instances, once for each choice of source $s \in R$. □

4.2 The Probabilistic Algorithm

The 1.62-approximation probabilistic algorithm is described in the following. The algorithm receives as input the original instant I^1 and parameter k.

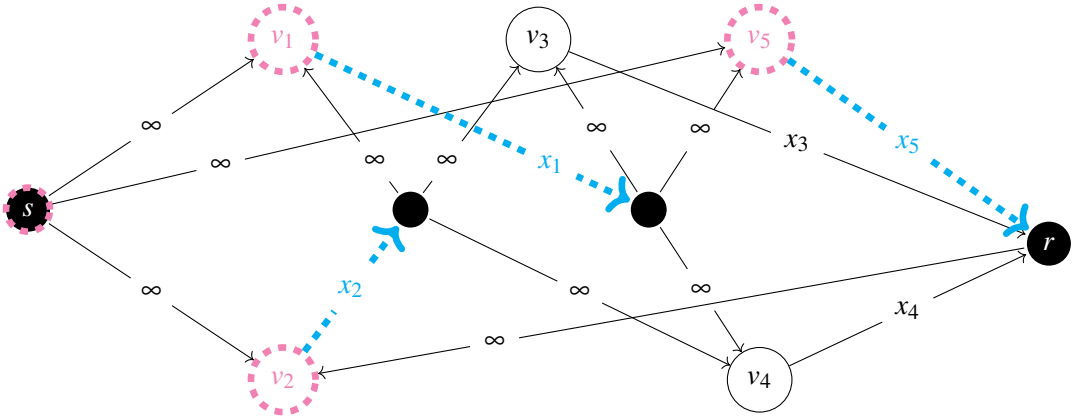


Figure 4.0: An example of a s-r cut $U = \{s, v_1, v_2, v_5\}$ in a directed graph G' with capacity $x_1 + x_2 + x_4$ on the set of edges in $\delta^+(U)$. In this graph we search whether there is a cut which prevents s from sending a flow of 1 to r.

ITERATIVEROUNDING$_k(I^1, k)$:

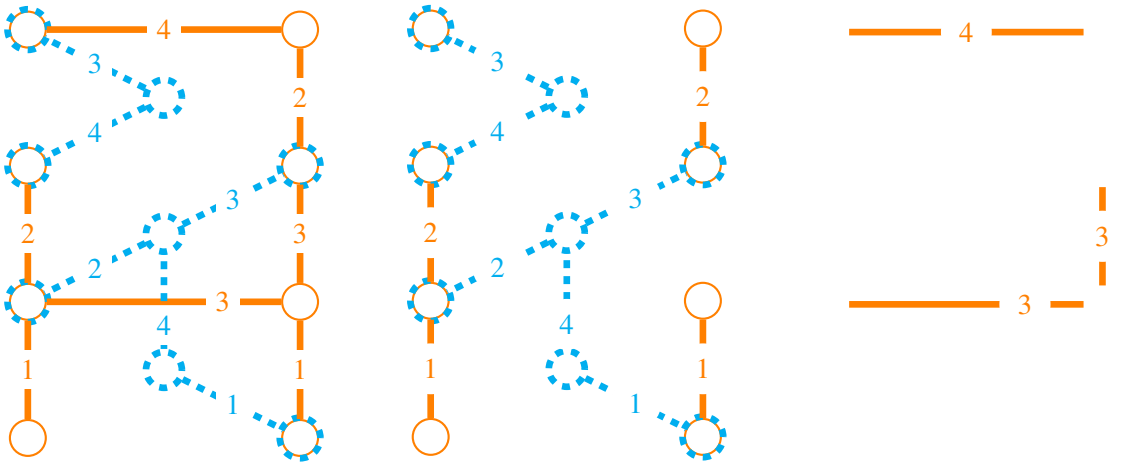
1) For each $t = 1, 2, \ldots$:
   a) Compute set $\mathscr{D}_k^t$ for I^t.
   b) Solve k-DCR with respect to I^t, and obtain solution x^t.
   c) Choose D_i independently with probability $x_i^t / \sum_{D_j \in \mathscr{D}_k^t} x_j^t$, and let C^t be the non-directed full component of D_i.
   d) Contract C^t into a single terminal of C^t, and let I^{t+1} be the new instance.
   e) If I^{t+1} has only a single terminal, return $G[\bigcup_{j=1}^t f_j(E(C^j))]$.

In the algorithm, for each sampled full component C^j, function f_j maps each edge e of the full component C^j to the corresponding edge in the original instance I^1.

In the algorithm, the value of $\sum_{D_j^t \in \mathscr{D}_k^t} x_j^t$ varies depending on the iteration. To bound the expected cost of returned solution, we will analyze an equivalent algorithm in which, for each iteration t, a dummy component D_r, consisting solely of terminal r, is added to $\mathscr{D}_k^t$, and constraint $x_r = M - \sum_{D_j \in \mathscr{D}_k^t} x_j$ is added to k-DCR, for some fixed $M \geq |\mathscr{D}_k|$. The expected cost obtained with the alternative algorithm is the same, as sampling component D_r does not change the current instance and adds zero cost to the solution. Therefore, assume that $M = \sum_{D_j \in \mathscr{D}_k^t} x_j^t$ in each iteration $t \geq 1$.

Let T be a subgraph of G that is also a tree, and H be a subgraph of G. We denote by $T[\![H]\!]$ a minimum-cost subgraph of $T \cup H$ which contains H and connects the vertices of T. The set of edges of T not in $T[\![H]\!]$ are called the *bridges* of T with respect to H and is denoted by $Br_T(H)$. See Figure 4.1. If H is a tree, then a bridge is a heavy edge of a cycle created by extending H with edges of T.

If one starts with a terminal spanning tree T and contracts some terminals connected by H, then some edges of T might become unnecessary. This set corresponds to the bridges of T. The core idea in the analysis of the algorithm is to maintain a tree T that provides an upper bound on the cost of an optimal solution of current instance I^t. Whenever the algorithm samples a component, this component is included in the solution, and thus the corresponding bridges



(a) A tree $T \subseteq G$ in orange and a subgraph $H \subseteq G$ in dashed cyan.

(b) The resulting subgraph $T[\![H]\!]$.

(c) The edges of $Br_T(H)$.

Figure 4.1: An example of $T[\![H]\!] \subseteq G$. Each edge has its weight written on it respectively.

become unnecessary. Therefore, the upper bound on the value of an optimal solution of next instance I^{t+1} decreases by at least the average weight of bridges. The next lemma formalizes this.

Lemma 21. *(Byrka et al. 2013 [6]) [Bridge Lemma] Let T be a terminal spanning tree, C_i be the non-directed full component of D_i, and x be a solution to k-DCR. If D_i is drawn with probability $\frac{x_i}{M}$, then $\mathbb{E}[w_e(Br_T(C_i))] \geq \frac{w_e(T)}{M}$.*

Proof. First, we define a *bridge tree* from a full component D_j. Consider a (directed) full component D_j and the corresponding non-directed full component C_j. Create a subgraph C'_j from T by removing the set of bridges $Br_T(C_j)$, i.e., take $V(C'_j) := V(C_j)$ and $E(C'_j) := E(T) \setminus Br_T(C_j)$. From the definition of the bridge set, one may show that C'_j is such that: each of its connected components has exactly one terminal of C_j; and each bridge connects exactly two connected components. Let $R(C)$ denote the terminals of the full component C. The *bridge tree* Y_j of D_j is the tree whose vertex set is $R(C_j)$ and which contains edge $\{u,v\}$ if, and only if, there is a bridge b between the connected components of u and v in C'_j. The weight of edge an edge $\{u,v\}$ is $w'_e(u,v) := w_e(b)$. Finally, we direct every edge of Y_j towards $sink(D_j)$ (see Figure 4.2).

Now, for every component D_j, construct the bridge tree Y_j as described above. From the definition of the bridge set, we know $w'_e(Y_j) = w_e(Br_T(C_j))$. Define a function $y : R \times R \to \mathbb{Q}_{\geq 0}$ by setting $y(u,v) := \sum_{Y_j \ni (u,v)} x_j$, i.e., $y(u,v)$ is the sum of x_j for every bridge tree that contains (u,v).

We interpret the values of $y(u,v)$ as the flow capacity of an edge (u,v). The following properties of the flow network are obtained. First, each bridge tree Y_j supports a flow x_j from each of its sources in $source(D_j)$ to its sink $sink(D_j)$. As a consequence, from the constraints of k-DCR, each cut that separates a non-root terminal to the root has capacity at least one, and thus function y supports one unit flow from each terminal to the root. Therefore, y induces a feasible solution to problem BCR whose underlying graph is the complete graph on

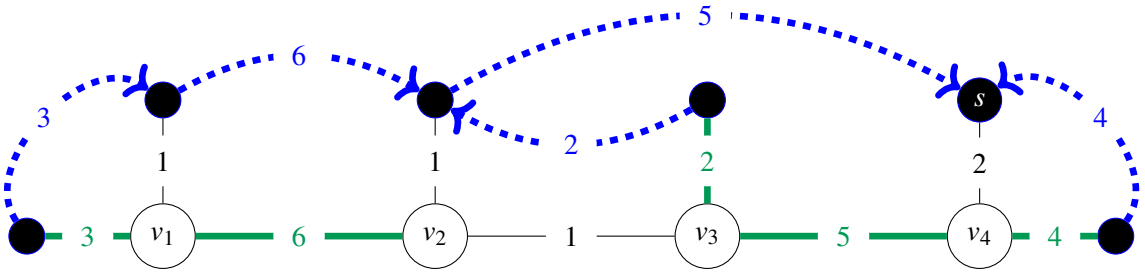


Figure 4.2: An example of a full component C_j of a directed component D_j with $sink(D_j) = s$. $Br_{E_0(C_j)}(C_j)$ is drawn in thick green and the bridge tree Y_j, of D_j, is drawn in dashed blue. The respective edge weight is explicitly written on the edges and arcs.

R, and whose weight function is w'_e. Using Corollary 19, we obtain an optimal integral solution that corresponds to a terminal spanning tree F. Since y is feasible to BCR, we know that $w'_e(F) \leq \sum_{\{u,v\}\in R\times R} w'_e(u,v)y(u,v)$.

Notice that, from the definition of the bridge set, $w'_e(u,v)$ is the weight of the heaviest edge of the unique cycle in $T + \{u,v\}$. It follows from the cycle property (Lemma 5) that $w'_e(F) \geq w_e(T)$. This results in

$$\sum_{D_j\in\mathscr{D}} x_j w_e(Br_T(C_j)) = \sum_{D_j\in\mathscr{D}} x_j w'_e(Y_j) = \sum_{\{u,v\}\in R\times R} w'_e(u,v)y(u,v) \geq w'_e(F) \geq w_e(T).$$

Dividing by M yields

$$\frac{w_e(T)}{M} \leq \sum_{D_j\in\mathscr{D}} \frac{x_j}{M} w_e(Br_T(C_j)) = \mathbb{E}[w_e(Br_T(C_i))]. \qquad \square$$

In the following, $t \geq 1$ denotes the iteration number of the algorithm: I^t is the residual instance of the problem in the beginning of the iteration; T^t is a minimum cost tree spanning the remaining terminals in I^t; C^t is full component drawn at the iteration; $\text{Opt}^{q,t}_{k,f}$ denotes an optimal fractional solution for I^t; and $\text{Opt}^{q,t}_k$ denotes an optimal integral solution for I^t (in particular, $\text{Opt}^{q,1}_k = \text{Opt}^q_k$).

Lemma 22 below gives an upper bound on the expected weight of T^t. Lemma 24 is analogous, but bounds the expected weight of $\text{Opt}^{q,t}_k$.

Lemma 22. *(Byrka et al. 2013 [6])* $\mathbb{E}[w_e(T^t)] \leq \left(1 - \frac{1}{M}\right)^{t-1} w_e(T^1)$.

Proof. Since after iteration t, full component C^t is included in the solution, the residual instance will contain one terminal that corresponds to the contraction of C^t. Thus, the decrease in the weight of T^{t-1} when compared to T^t is at least $w_e(Br_{T^{t-1}}(C^t))$. Let $\mathscr{F}^t$ be the set of all possible sequences of sampled components up to the t-th iteration, and let $F \in \mathscr{F}^t$ denote the event where one such sequence is sampled. We compute:

$$\begin{aligned}\mathbb{E}[w_e(T^t)] &\overset{Lem\ 1}{=} \sum_{F\in\mathscr{F}^{t-1}} Pr(F)\cdot\mathbb{E}[w_e(T^t) \mid F] \\ &\leq \sum_{F\in\mathscr{F}^{t-1}} Pr(F)\cdot\mathbb{E}[w_e(T^{t-1}) - w_e(Br_{T^{t-1}}(C^t)) \mid F]\end{aligned}$$

$$
\begin{aligned}
&= \sum_{F \in \mathscr{F}^{t-1}} Pr(F) \cdot \mathbb{E}\left[w_e(T^{t-1}) - \frac{\sum_{D_j \in \mathscr{D}_k^t} x_j^t}{M} \cdot w_e(Br_{T^{t-1}}(C_j)) \;\middle|\; F \right] \\
&\overset{Lem\ 21}{\leq} \sum_{F \in \mathscr{F}^{t-1}} Pr(F) \cdot \mathbb{E}\left[\left(1 - \frac{1}{M}\right) w_e(T^{t-1}) \;\middle|\; F \right] \\
&\overset{Lem\ 21}{\leq} \left(1 - \frac{1}{M}\right) w_e(T^{t-1}) \cdot \sum_{F \in \mathscr{F}^{t-1}} Pr(F) \cdot \mathbb{E}[1 \mid F] \\
&\overset{Lem\ 1}{=} \left(1 - \frac{1}{M}\right) w_e(T^{t-1}),
\end{aligned}
$$

where C_j denotes the non-directed full component of D_j.

By induction we get the desired result

$$\mathbb{E}[w_e(T^t)] \leq \left(1 - \frac{1}{M}\right)^{t-1} w_e(T^1). \qquad \square$$

Corollary 23. *Let* $\alpha := w_v(\mathrm{Opt}^q)/w(\mathrm{Opt}^q)$*, then*

$$\mathbb{E}[w(\mathrm{Opt}^{q,t})] \leq \left(1 - \frac{1}{M}\right)^{t-1} 2(1-\alpha) w(\mathrm{Opt}^q)$$

Proof. This follows as an optimal solution cannot be worse than a minimum spanning tree, thus $w(\mathrm{Opt}^{q,t}) \leq w_e(T^t)$ and, from Theorem 13, we have $w_e(T^1) \leq 2(1-\alpha)w(\mathrm{Opt}^q)$. $\square$

Lemma 24. $\mathbb{E}\left[w(\mathrm{Opt}^{q,t})\right] \leq \left(1 - \frac{1}{4M}\right)^{t-1} w(\mathrm{Opt}^q)$.

Proof. First, turn $\mathrm{Opt}^{q,t-1}$ into a regular binary tree S, by adding dummy edges and dummy vertices of zero weight, such that the leaves of S coincides with terminals. Now, for each inner vertex v, select the lightest edge between the edges to its two children. It follows that the set of selected edges form a set B whose weights is at most $\frac{w_e(S)}{2} = \frac{w_e(\mathrm{Opt}^{q,t-1})}{2}$. Notice that the set B connects every inner vertex of S to exactly one terminal leaf, and each maximal path in $S[B]$ corresponds to a connected component B'. Consider the terminal spanning tree Y that emerges from S when we contract each connected component B' of $S[B]$ into the unique terminal in B'. Let $f : E(Y) \to E(S)$ be the function which maps each edge e of Y to the corresponding edge of S before the contraction operation, and whose weight e has inherited.

Let G^t and R^t be the graph and the terminal set of instance I^t, respectively. Also, let $\alpha^t := w_v(\mathrm{Opt}^{q,t})/w(\mathrm{Opt}^{q,t})$. Consider a set of edges $E' \subseteq E(Y)$. If $G^t[(E(Y) \backslash E') \cup E(C^t)]$ connects R^t, then $G^t[(E(S) \backslash f(E')) \cup E(C^t)]$ also connects R^t. This implies that

$$w_e(Br_{\mathrm{Opt}^{q,t-1}}(C^t)) = w_e(Br_S(C^t)) \geq w_e(Br_Y(C^t)).$$

Let $\mathscr{F}^t$ denote the set of all possible sequence of sampled components up to the t-th iteration. We get

$$
\begin{aligned}
\mathbb{E}\left[w_e(Br_{\mathrm{Opt}^{q,t}}(C^t)) \mid F \right] &= \mathbb{E}\left[w_e(Br_{\mathrm{Opt}^{q,t}}(C^t)) \mid F \right] \\
&\geq \mathbb{E}\left[w_e(Br_Y(C^t)) \mid F \right]
\end{aligned}
$$

$$
\begin{aligned}
&= \mathbb{E}\left[\frac{w_e(\mathrm{Opt}^{q,t-1}) - w_e(B)}{M} \;\middle|\; F\right] \\
&\geq \mathbb{E}\left[\frac{w_e(\mathrm{Opt}^{q,t-1})}{2M} \;\middle|\; F\right] \\
&= \frac{w_e(\mathrm{Opt}^{q,t-1})}{2M}\mathbb{E}[1 \mid F] \\
&\overset{Cor\ 15, Lem\ 1}{\geq} \frac{1}{4M} w(\mathrm{Opt}^{q,t-1}).
\end{aligned}
$$

It then follows:

$$
\begin{aligned}
\mathbb{E}\left[w(\mathrm{Opt}^{q,t})\right] &\overset{Lem\ 1}{=} \sum_{F \in \mathscr{F}^{t-1}} Pr(F) \cdot \mathbb{E}\left[w(\mathrm{Opt}^{q,t}) \mid F\right] \\
&\leq \sum_{F \in \mathscr{F}^{t-1}} Pr(F) \cdot \mathbb{E}\left[w(\mathrm{Opt}^{q,t-1}) - w_e(Br_{\mathrm{Opt}^{q,t}}(C^t)) \;\middle|\; F\right] \\
&= \sum_{F \in \mathscr{F}^{t-1}} Pr(F) \cdot \left(w(\mathrm{Opt}^{q,t-1}) - \mathbb{E}\left[w_e(Br_{\mathrm{Opt}^{q,t}}(C^t)) \;\middle|\; F\right]\right) \\
&\leq \sum_{F \in \mathscr{F}^{t-1}} Pr(F) \cdot \left(1 - \frac{1}{4M}\right) w(\mathrm{Opt}^{q,t-1}) \\
&= \left(1 - \frac{1}{4M}\right) w(\mathrm{Opt}^{q,t-1}) \cdot \sum_{F \in \mathscr{F}^{t-1}} Pr(F) \\
&= \left(1 - \frac{1}{4M}\right) w(\mathrm{Opt}^{q,t-1}).
\end{aligned}
$$

Thus, by induction:

$$
\mathbb{E}\left[w(\mathrm{Opt}^{q,t})\right] \leq \left(1 - \frac{1}{4M}\right)^{t-1} w(\mathrm{Opt}^{q}). \qquad \square
$$

Recall that Opt^q denotes an optimal solution to an instance I of q-MNSTP and that $\alpha := w_v(\mathrm{Opt}^q)/w(\mathrm{Opt}^q)$. We prove that the algorithm's expected approximation factor converges to 1.61889 as k grows.

Theorem 25. *For any $\delta > 0$, there is a constant k such that* IterativeRounding$_k$ *runs in polynomial time and has an expected approximation factor at most* $4 - \frac{3}{\sqrt[3]{2-2\alpha}} + \delta < 1.62$.

Proof. Firstly notice that after each iteration the number of terminals is reduced in at least one, because each component that may be sampled and contracted has at least two terminals, so the loop is executed at most $|V|$ times. Also, because the set $\mathscr{D}_k$ can be constructed in $O(|V|^k)$, each the contraction of a sampled component and each instance of k-DCR can be solved in polynomial time (Lemma 20), it follows that the total running time of IterativeRounding$_k$ is polynomial.

The approximation factor is obtained by direct calculation, but is technical. Without loss of generality, assume that the value $M \cdot \ln\left([2-2\alpha]^{4/3}\right)$ is integer and at least 1. Also, let S denote the solution given by the algorithm and let k be a constant such that $k > 4^{(1+q)/\delta}$. Therefore, using Theorem 18, we obtain that $w(\mathrm{Opt}_k^{q,t}) \leq (1 + \frac{\delta}{2}) w(\mathrm{Opt}^{q,t})$.

Now we bound the approximation factor:

$$\begin{aligned}
\mathbb{E}\left[\frac{w(S)}{w(\mathrm{Opt}^q)}\right] &= \mathbb{E}\left[\frac{\sum_{t\geq 1} w(C^t)}{w(\mathrm{Opt}^q)}\right] \\
&= \sum_{t\geq 1} \mathbb{E}\left[\frac{\sum_{D_j\in\mathscr{D}_k^t} x_j^t}{M\cdot w(\mathrm{Opt}^q)} w(C^t)\right] \\
&= \sum_{t\geq 1} \frac{1}{M}\mathbb{E}\left[\frac{w(\mathrm{Opt}_{k,f}^{q,t})}{w(\mathrm{Opt}^q)}\right] \\
&\leq \frac{1}{M}\sum_{t\geq 1} \mathbb{E}\left[\frac{w(\mathrm{Opt}_{k}^{q,t})}{w(\mathrm{Opt}^q)}\right] \\
&\overset{Thm\ 18}{\leq} \frac{1+\delta/2}{M}\sum_{t\geq 1} \mathbb{E}\left[\frac{w(\mathrm{Opt}^{q,t})}{w(\mathrm{Opt}^q)}\right] \\
&\overset{Cor23,Lem\ 24}{\leq} \frac{1+\delta/2}{M}\sum_{t\geq 1} \min\left\{1, 2(1-\alpha)\left(1-\frac{1}{M}\right)^{t-1}, \left(1-\frac{1}{4M}\right)^{t-1}\right\}.
\end{aligned}$$

The above summation is minimized by considering the $\left(1-\frac{1}{4M}\right)^{t-1}$ bound for the first $\mu = M\cdot\ln\left([2-2\alpha]^{4/3}\right)$ iterations and then considering the $\left(1-\frac{1}{M}\right)^{t-1}2(1-\alpha)$ bound for all iterations thereafter. Thus

$$\begin{aligned}
\mathbb{E}\left[\frac{w(S)}{w(\mathrm{Opt}^q)}\right] &\leq \frac{1+\delta/2}{M}\sum_{t\geq 1} \min\left\{1, 2(1-\alpha)\left(1-\frac{1}{M}\right)^{t-1}, \left(1-\frac{1}{4M}\right)^{t-1}\right\} \\
&\leq \frac{1+\delta/2}{M}\left(\sum_{t=1}^{\mu}\left(1-\frac{1}{4M}\right)^{t-1} + \sum_{t>\mu} 2(1-\alpha)\left(1-\frac{1}{M}\right)^{t-1}\right) \\
&\leq \left(1+\frac{\delta}{2}\right)\left(4-4\left(1-\frac{1}{4M}\right)^{\mu} + 2(1-\alpha)\left(1-\frac{1}{M}\right)^{\mu}\right),
\end{aligned}$$

where we used the formula $\sum_{j=1}^{n} x^{j-1} = \frac{1-x^n}{1-x}$ on the last inequality. Because the function $(1-\alpha)(1-1/M)^{M\ln\left([2-2\alpha]^{4/3}\right)} - 2(1-1/(4M))^{M\ln\left([2-2\alpha]^{4/3}\right)}$ is monotonically increasing for $M\cdot\ln\left([2-2\alpha]^{4/3}\right) > 1$ when $\alpha\in[0,\frac{1}{2}]$, we take the upper bound given when M tends to infinity, which, by the the identity $\lim_{x\to\infty}\left(1+\frac{1}{x}\right)^x = e$, yields

$$\begin{aligned}
\mathbb{E}\left[\frac{w(S)}{w(\mathrm{Opt}^q)}\right] &\leq \left(1+\frac{\delta}{2}\right)\left(4-4\left(1-\frac{1}{4M}\right)^{\mu} + 2(1-\alpha)\left(1-\frac{1}{M}\right)^{\mu}\right) \\
&\leq \left(1+\frac{\delta}{2}\right)\left(4-4e^{-\frac{\mu}{4M}} + 2(1-\alpha)e^{-\frac{\mu}{M}}\right) \\
&= \left(1+\frac{\delta}{2}\right)\left(4-4e^{-\frac{1}{4}\ln\left([2-2\alpha]^{4/3}\right)} + 2(1-\alpha)e^{-\ln\left([2-2\alpha]^{4/3}\right)}\right) \\
&= \left(1+\frac{\delta}{2}\right)\left(4-\frac{4}{(2-2\alpha)^{1/3}} + \frac{2(1-\alpha)}{(2-2\alpha)^{4/3}}\right) \\
&= \left(1+\frac{\delta}{2}\right)\left(4-\frac{3}{\sqrt[3]{2-2\alpha}}\right)
\end{aligned}$$

$$< 4 - \frac{3}{\sqrt[3]{2-2\alpha}} + \delta. \qquad \square$$

Chapter 5

A Greedy 1.55-Approximation to q-MNSTP

The next algorithm is based on a greedy strategy to select full components. It maintains a feasible solution S, which begins as a minimum terminal spanning tree. The solution S is improved at each iteration by adding a full component C and removing redundant edges. The greedy criterion is to select the full component which maximizes the ratio between the weight of redundant edges in $S \cup C$ (that correspond to the so called bridge edges) and the total weight of a certain part of C.

This algorithm is based on the algorithm by Robins and Zelikovsky [19]. The main idea is that, when the algorithm selects a new full component, only part of the component is accounted in the cost of final solution. Such part is called the commit of the component. This allows edges of selected components, but which are not "committed", to be discarded in future iterations. This happens because uncommitted edges may form cycles as new components are selected.

5.1 Preliminaries

Before presenting the algorithm, we give some important concepts. We highlight that, also in this chapter, each full component has its own copy of edges and Steiner vertices. Let G be the graph of an instance of q-MNSTP, H be a subgraph of G, and T be a terminal spanning tree. Recall that $T[\![H]\!]$ denotes the minimum-cost subgraph of $T \cup H$ which contains H and connects the vertices of T, then the *gain* of H with respect to T is defined as $gain_T(H) := w_e(T) - w(T[\![H]\!])$ and expresses how much a solution T improves if H replaces some edges of T. Notice that the weight of $T[\![H]\!]$ is not smaller than $w(\mathrm{MST}(T \cup H))$, thus $gain_T(H) \leq w_e(T) - w(\mathrm{MST}(T \cup H))$. Also, if H is a Steiner tree, then $w(T[\![H]\!])$ is $w(H)$, because $T[\![H]\!]$ must contain every edge of H.

Let H be a subgraph, then we denote by $E_0(H)$ the complete graph on the set of terminals of H, and whose edge and vertex weights are all zero. The following observation follows from the definition of gain and of the bridge set.

Lemma 26. *Let T be a terminal only spanning tree, then $gain_T(H) = w_e(T) - w_e(\mathrm{MST}(T \cup E_0(H))) - w(H) = w_e(Br_T(H)) - w(H)$.*

Now, denote by E_0 an arbitrary set of zero weight edges between terminals and let C be a full component, then we have the following simple result.

Lemma 27. *Let T be a terminal only spanning tree, then $gain_{\mathrm{MST}(T\cup E_0)}(C) \leq gain_T(C)$.*

This fact follows because a full component C cannot improve $\mathrm{MST}(T \cup E_0)$ better than it can improve T. Although simple, this leads to the following general property.

Lemma 28. *(Robins and Zelikovsky 2005 [19]) Let T be a terminal spanning tree and $C_1, C_2, \ldots, C_n$ be a set of full components. Then $gain_T\left(\bigcup_{i=1}^{n} C_i\right) \leq \sum_{i=1}^{n} gain_T(C_i)$.*

Proof. The result follows from the fact that bridges may be repeated on the right-hand side of the equation, whilst no bridge is repeated in the left-hand side. Note that, since each full component has its own copy of edges and Steiner vertices, the weight of $\bigcup_{i=1}^{n} C_i$ is $\sum_{i=1}^{n} w(C_i)$.

We calculate:

$$\begin{aligned}
\sum_{i=1}^{n} gain_T(C_i) &\overset{Lem\ 27}{\geq} \sum_{i=1}^{n} gain_{\mathrm{MST}(T\cup E_0(C_1)\cup E_0(C_2)\cup\cdots\cup E_0(C_{i-1}))}(C_i) \\
&\overset{Lem\ 26}{=} \sum_{i=1}^{n} [w_e(\mathrm{MST}(T \cup E_0(C_1) \cup \cdots \cup E_0(C_{i-1}))) \\
&\qquad - w_e(\mathrm{MST}(T \cup E_0(C_1) \cup \cdots \cup E_0(C_i))) - w(C_i)] \\
&= w_e(T) - w_e(\mathrm{MST}(T \cup E_0(C_1) \cup \cdots \cup E_0(C_n))) - \sum_{i=1}^{n} w(C_n) \\
&\overset{Lem\ 27}{=} gain_T(C_1 \cup \cdots \cup C_n)
\end{aligned}$$

where the second equality is obtained by canceling the terms. □

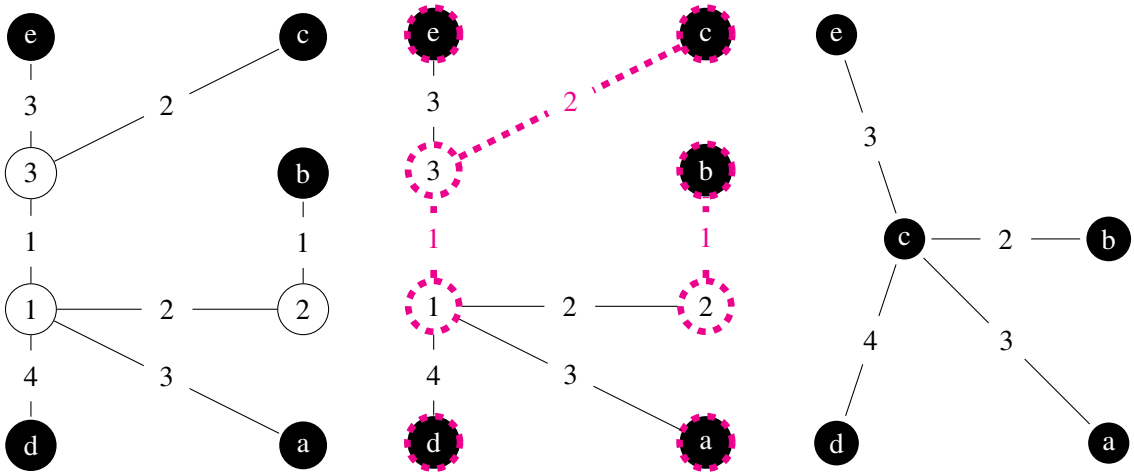
Another concept we introduce is the *commit* of a component. The commit of a component C is a lightest forest, denoted by $Commit(C)$, that spans every vertex of $V(C)$ such that each connected component has exactly one terminal vertex, and we denote by $com(C)$ the sum of the weights of vertices and edges of $Commit(C)$. We also define the weight of the commit of a union of full components as the sum of the weights of their individual commits. It is easy to compute the commit of a full component C, for that we just need to compute a minimum spanning tree of $C \cup E_0(C)$.

Lemma 29. *(Robins and Zelikovsky 2005 [19]) For any full component C, $Commit(C) = \mathrm{MST}(C \cup E_0(C)) - E_0(C)$.*

Proof. Consider the forest $F = \mathrm{MST}(C \cup E_0(C)) - E_0(C)$. Notice that F connects every Steiner vertex of C to terminals of C and has total weight equals to $w(\mathrm{MST}(C \cup E_0(C)))$. Since every edge of $E_0(C)$ has weight zero and $Commit(C) \cup E_0(C)$ spans every vertex of C, it follows that $w(Commit(C) \cup E_0(C)) \leq w(\mathrm{MST}(C \cup E_0(C)))$ and $Commit(C) = \mathrm{MST}(C \cup E_0(C)) - E_0(C)$, because $E(E_0(C)) \cap E(C) = \emptyset$. □

We also define the *commit-contracted component* $¢(C)$ of a full component C as the tree obtained after contracting each connected component of the commit of C into its terminal. Thus,

the commit-contracted component $¢(C)$ is the set of edges between terminals that correspond to edges originally connecting different connected components of $Commit(C)$. The weight of $¢(C)$ is thus $w(¢(C)) = w(C) - com(C)$. Similarly, if H is a k-restricted Steiner tree, then $¢(H)$ is the terminal spanning tree in which the commit of every full component is contracted (see Figure 5.1).



(a) A full component C with edge and vertex weights.

(b) The (forest) $Commit(C)$ of C drawn in dashed magenta where $com(C) = 11$.

(c) The resulting tree $¢(C)$ after contracting the connected components of the commit of C.

Figure 5.1: An example of $Commit(C)$ and $¢(C)$ of a full component C. White circles denotes Steiner vertices and black ones are terminals.

5.2 The Greedy Algorithm

We now present the algorithm and its analysis. The algorithm is the following:

GREEDYALG$_k(I,k)$:

1) Construct set of full components $\mathscr{C}_k$.
2) $T_0 \leftarrow MST(G[R])$, $H \leftarrow G[R]$.
3) For each $t = 1,2,\ldots$:
   a) Find $C_t \in \mathscr{C}_k$ with maximum $r := gain_{T_{t-1}}(C_t)/com(C_t)$.
   b) If $r \leq 0$, stop and return $S \leftarrow MST(H)$.
   c) $T_t \leftarrow MST(T_{t-1} \cup ¢(C_t))$, $H \leftarrow H \cup C_t$.

Let $C_1,\ldots,C_{last}$ be the components selected by the algorithm during its execution, where *last* is the number of iterations. At each iteration t, a new full component C_t is selected. Only the commit of C_t will actually be included in the solution, and the union of the commits of the full components selected so far form a set of connected components, each with exactly one terminal. These connected components are connected by the current terminal spanning tree. After full component C_t is selected, the commit-contracted component of C_t is joined to the terminal spanning tree from the previous iteration, and an improved terminal spanning tree T_t

is built. One can think of this process as if at each iteration the algorithm is only paying for the commit of each selected full component. Thus, the objective is to maximize the gain of a full component per unit of the commit cost.

At each iteration t, $MST(H)$ is a feasible solution for the original problem. This solution may be equivalently obtained by joining T_t with the commits of full components selected until the t-th iteration. Because the cost of such solution can only decrease, we can upper bound the cost of returned solution S by the sum $w_e(T_t) + com(C_1) + com(C_2) + \cdots + com(C_t)$, for any $t \geq 1$.

To bound T_t we use the following lemma.

Lemma 30. *(Robins and Zelikovsky 2005 [19]) Let H be a Steiner tree, if $gain_{¢(H)}(C) \leq 0$ for every $C \in \mathscr{C}_k$, then*

$$w(¢(H)) = w(H) - com(H) \leq w(\mathrm{Opt}_k^q).$$

Proof. Recall that because Opt_k^q is an optimal k-restricted Steiner tree. For a terminal spanning tree T, we have $T[\![\mathrm{Opt}_k^q]\!] = \mathrm{Opt}_k^q$, because Opt_k^q already contains every terminal, and thus extending it with edges of T can only make it heavier. Now, let $C_1, \ldots, C_p$ be the full components of Opt_k^q, then

$$\begin{aligned} w_e(¢(H)) - w(\mathrm{Opt}_k^q) &= w_e(¢(H)) - w(¢(H)[\![\mathrm{Opt}_k^q]\!]) \\ &= gain_{¢(H)}(\mathrm{Opt}_k^q) \\ &= gain_{¢(H)}(C_1 \cup C_2 \cup \cdots C_p) \\ &\overset{Lem\ 28}{\leq} gain_{¢(H)}(C_1) + \cdots + gain_{¢(H)}(C_p) \\ &\leq 0. \end{aligned}$$

Then the claim follows because $w(¢(H)) = w(H) - com(H)$. □

The algorithm stops when no full component can improve the current terminal spanning tree T_{last}. Since $w_e(T_{last}) = w(¢(H))$ by the construction of T_{last}, this lemma implies that $w_e(T_{last})$ is upper bounded by $w(\mathrm{Opt}_k^q)$.

Corollary 31. $w_e(T_{last}) \leq w(\mathrm{Opt}_k^q)$.

Next auxiliary lemmas will be used to bound the commit weight of selected full components.

Lemma 32. *(Robins and Zelikovsky 2005 [19]) Let $i \leq last$, then $gain_{T_{i-1}}(C_i) = w_e(T_{i-1}) - w(\mathrm{MST}(T_{i-1} \cup C_i))$.*

Proof. Recall that $gain_{T_{i-1}}(C_i) = w_e(T_{i-1}) - w(T_{i-1}[\![C_i]\!])$, so it suffices to show $w(T_{i-1}[\![C_i]\!]) = w(\mathrm{MST}(T_{i-1} \cup C_i))$.

Let W be the set of edges which are in both C_i and $MST(T_{i-1} \cup C_i)$, i.e., consider the set $W := E(C_i) \cap E(MST(T_{i-1} \cup C_i))$. Also, let $F := (V(C_i), W)$ be the graph on the vertices of C_i with edge set W. If $F = C_i$, then $w(T_{i-1}[\![C_i]\!]) = w(\mathrm{MST}(T_{i-1} \cup C_i))$, and we are done. So assume there is an edge e of C_i which is not in F. It follows that F contains at least two distinct connected components A' and B', each containing a terminal. Let $A, B \in \mathscr{C}_k$ be two full

component such that $A \subseteq A'$ and $B \subseteq B'$. These sets exist as either A', B' are full components, or one can remove non-terminal leaves.

Notice that $T_{i-1}[\![F]\!]$ and $\mathrm{MST}(T_{i-1} \cup C_i)$ contain the same set of vertices. Since $F \subseteq \mathrm{MST}(T_{i-1} \cup C_i)$, we have $w(T_{i-1}[\![F]\!]) = w(\mathrm{MST}(T_{i-1} \cup C_i))$. Suppose, for the sake of contradiction, that $w(\mathrm{MST}(T_{i-1} \cup C_i)) < w(T_{i-1}[\![C_i]\!])$. We obtain

$$\begin{aligned} gain_{T_{i-1}}(C_i) &= w_e(T_{i-1}) - w(T_{i-1}[\![C_i]\!]) \\ &< w_e(T_{i-1}) - w(\mathrm{MST}(T_{i-1} \cup C_i))) \\ &= w_e(T_{i-1}) - w(T_{i-1}[\![F]\!]) \\ &\leq w_e(T_{i-1}) - w(T_{i-1}[\![A \cup B]\!]) \\ &= gain_{T_{i-1}}(A \cup B) \\ &\overset{Lem\ 28}{\leq} gain_{T_{i-1}}(A) + gain_{T_{i-1}}(B). \end{aligned}$$

Without loss of generality, if $com(A)$ is zero, i.e., A just a terminal, then $gain(B) = gain(C_i)$ but has lighter commit, which contradicts the choice of C_t by the algorithm, since A and B are also full components. Otherwise, assume $com(A), com(B) > 0$. Moreover, note that $com(C_i) \geq com(A) + com(B)$. It follows

$$\frac{gain_{T_{i-1}}(C_i)}{com(C_i)} < \frac{gain_{T_{i-1}}(A) + gain_{T_{i-1}}(B)}{com(A) + com(B)} \leq \max\left\{ \frac{gain_{T_{i-1}}(A)}{com(A)}, \frac{gain_{T_{i-1}}(B)}{com(B)} \right\},$$

which, again, contradicts the choice of C_i. □

The *true gain* of a graph H, with respect to a Steiner tree T, is defined as $tg_T(H) := gain_T(H) + com(H)$. Using Lemma 32, the true gain of component C_i, with respect to T_{i-1}, may be written as

$$\begin{aligned} tg_{T_{i-1}}(C_i) &= gain_{T_{i-1}}(C_i) + com(C_i) \\ &= gain_{T_{i-1}}(C_i) + com(T_{i-1} \cup C_i) \\ &= gain_{T_{i-1}}(C_i) + w(\mathrm{MST}(T_{i-1} \cup C_i)) - w(\mathrm{MST}(T_{i-1} \cup ¢(C_i))) \\ &= gain_{T_{i-1}}(C_i) + w(\mathrm{MST}(T_{i-1} \cup C_i)) - w_e(T_i) \\ &\overset{Lem\ 32}{=} w_e(T_{i-1}) - w(\mathrm{MST}(T_{i-1} \cup C_i)) + w(\mathrm{MST}(T_{i-1} \cup C_i)) - w_e(T_i) \\ &= w_e(T_{i-1}) - w_e(T_i). \end{aligned}$$

Observe that the true gain of component C_i represents by how much the weight of terminal spanning tree T_{i-1} decreases if we select C_i. Analogously, we define $G_i := tg_{T_i}(\mathrm{Opt}_k^q)$. This value is the difference between the weight of T_i and the weight of the commit-contracted optimal Steiner tree $¢(\mathrm{Opt}_k^q)$. Indeed,

$$\begin{aligned} G_i &= tg_{T_i}(\mathrm{Opt}_k^q) \\ &= gain_{T_i}(\mathrm{Opt}_k^q) + com(\mathrm{Opt}_k^q) \\ &= w_e(T_i) - w(\mathrm{Opt}_k^q) + w(\mathrm{Opt}_k^q) - w_e(¢(\mathrm{Opt}_k^q)) \end{aligned}$$

$$= w_e(T_i) - w_e(\not{c}(\mathrm{Opt}_k^q)).$$

Notice that G_i is not necessarily positive, as the weight of T_i decreases between successive iterations.

The next lemma considers the ratio between the commit weight of selected full components and the commit weight of the optimal solution Opt_k^q.

Lemma 33. *(Robins and Zelikovsky 2005 [19]) If G_t is positive, we have* $\ln\left(\frac{G_0}{G_t}\right) \geq \frac{\sum_{i=1}^{t} com(C_i)}{com(\mathrm{Opt}_k^q)}$.

Proof. Let $X_j \in \mathrm{Opt}_k^q$ represent a full component Opt_k^q, then

$$\begin{aligned}
\frac{G_0}{com(\mathrm{Opt}_k^q)} &= \frac{tg_{T_0}(\bigcup_{X_j \in \mathrm{Opt}_k^q} X_j)}{\sum_{X_j \in \mathrm{Opt}_k^q} com(X_j)} \\
&\overset{Lem\ 28}{\leq} \frac{\sum_{X_j \in \mathrm{Opt}_k^q} tg_{T_0}(X_j)}{\sum_{X_j \in \mathrm{Opt}_k^q} com(X_j)} \\
&= \frac{\sum_{X_j \in \mathrm{Opt}_k^q} (gain_{T_0}(X_j) + com(X_j))}{\sum_{X_j \in \mathrm{Opt}_k^q} com(X_j)} \\
&\leq 1 + \max_{X_j \in \mathrm{Opt}_k^q} \left\{ \frac{gain_{T_0}(X_j)}{com(X_j)} \right\} \\
&\leq 1 + \frac{gain_{T_0}(C_1)}{com(C_1)} \\
&= \frac{tg_{T_0}(C_1)}{com(C_1)}.
\end{aligned}$$

By induction on i, one can show that $G_{i-1}\frac{com(C_i)}{com(\mathrm{Opt}_k^q)} \leq tg_{T_{i-1}}(C_i)$, for $i = 1, 2, \ldots, t$.

When GREEDYALG$_k$ selects a component C_i, it decreases the cost of T_i by $tg_{T_{i-1}}(C_i)$. Thus, the true gain of Opt_k^q also decreases by this same value, because $G_{i-1} - G_i = w_e(T_{i-1}) - w_e(T_i) = tg_{T_{i-1}}(C_i)$. It follows that

$$G_i = G_{i-1} - tg_{T_{i-1}}(C_i) \leq G_{i-1}\left(1 - \frac{com(C_i)}{com(\mathrm{Opt}_k^q)}\right),$$

for $i = 1, 2, \ldots, t$. Because $G_t > 0$ and G_i is decreasing in i, we have that every G_i, $i = 0, \ldots, t$, is positive, and thus:

$$\frac{G_t}{G_0} \leq \frac{G_{t-1}}{G_0}\left(1 - \frac{com(C_t)}{com(\mathrm{Opt}_k^q)}\right) \leq \cdots \leq \prod_{i=1}^{t}\left(1 - \frac{com(C_i)}{com(\mathrm{Opt}_k^q)}\right).$$

Then, by applying the natural logarithm on both sides, we get:

$$\begin{aligned}
\ln\left(\frac{G_t}{G_0}\right) &\leq \ln\left(\prod_{i=1}^{t}\left(1 - \frac{com(C_i)}{com(\mathrm{Opt}_k^q)}\right)\right) \\
&= \sum_{i=1}^{t} \ln\left(1 - \frac{com(C_i)}{com(\mathrm{Opt}_k^q)}\right)
\end{aligned}$$

$$\leq \sum_{i=1}^{t} -\frac{com(C_i)}{com(\mathrm{Opt}_k^q)}$$

The last inequality used $\ln(1-x) \leq -x$ for $x < 1$, and the fact that $\frac{com(C_i)}{com(\mathrm{Opt}_k^q)} < 1$. Now, multiplying both sides by -1 yields the desired result. □

Recall that the cost of returned solution is given by the cost of T_{last} plus the sum of commit weights for all selected full components, $\sum_{i=1}^{last} com(C_i)$. To bound the cost of T_{last}, we use Corollary 31, which gives $w_e(T_{last}) \leq w(\mathrm{Opt}_k^q)$. To bound $\sum_{i=1}^{last} com(C_i)$, one might try to use Lemma 33. However, it requires that G_i be positive, and thus this lemma cannot be applied directly. To bound the cost of the solution returned by GREEDYALG$_k$, we consider an alternative algorithm, GREEDYALG$_k^{\eta+1}$, which only iterates $\eta+1$ times. We choose η as the unique index such that $w_e(T_{\eta+1}) < w(\mathrm{Opt}_k^q) \leq w_e(T_\eta)$. Since the cost of a partial solution only gets cheaper after each iteration, the cost of the solution returned by this alternative algorithm is an upper bound on the cost of the solution returned by GREEDYALG$_k$.

The general strategy to bound the solution cost is as follows. Since $G_{\eta+1}$ is not positive, we cannot use Lemma 33, and thus we will derive an analogous of this lemma. First we write $com(C_{\eta+1})$ as the sum of two terms, $c^1_{\eta+1}$, $c^2_{\eta+1}$, i.e.,

$$com(C_{\eta+1}) = c^1_{\eta+1} + c^2_{\eta+1}.$$

Second, the value of $tg_{T_\eta}(C_{\eta+1})$ is also split in the sum of two terms, $g^1_{\eta+1}$, $g^2_{\eta+1}$ as

$$tg_{T_\eta}(C_{\eta+1}) = g^1_{\eta+1} + g^2_{\eta+1}.$$

Recall that the costs of $T_{\eta+1}$ and T_η differ by exactly $tg_{T_\eta}(C_{\eta+1})$. Since we want to bound $w_e(T_{\eta+1})$ by $w(\mathrm{Opt}_k^q)$, we need to decrease $w_e(T_\eta)$ just as much as needed to obtain equality, thus we define

$$g^1_{\eta+1} := w_e(T_\eta) - w(\mathrm{Opt}_k^q),$$

Informally, one can think of this split as if iteration $\eta+1$ is performed only "partially". Thus, the first part of $com(C_{\eta+1})$ is defined by the same proportion of $g^1_{\eta+1}$, i.e., define

$$c^1_{\eta+1} := \frac{g^1_{\eta+1}}{tg_{T_\eta}(C_{\eta+1})} com(C_{\eta+1}).$$

From the definitions above, one can readily observe that

$$c^2_{\eta+1} = \frac{g^2_{\eta+1}}{tg_{T_\eta}(C_{\eta+1})} com(C_{\eta+1}).$$

Finally, we define $G^1_{\eta+1} := G_\eta - g^1_{\eta+1}$. The value $G^1_{\eta+1}$ corresponds to the cost of the commit of an optimal solution. Indeed,

$$G^1_{\eta+1} = G_\eta - g^1_{\eta+1}$$

$$
\begin{aligned}
&= (gain_{T_\eta}(\mathrm{Opt}_k^q) + com(\mathrm{Opt}_k^q)) - (w_e(T_\eta) - w(\mathrm{Opt}_k^q)) \\
&= (w_e(T_\eta) - w_e(T_\eta[\![\mathrm{Opt}_k^q]\!]) + com(\mathrm{Opt}_k^q)) - (w_e(T_\eta) - w(\mathrm{Opt}_k^q)) \\
&= (w_e(T_\eta) - w(\mathrm{Opt}_k^q) + com(\mathrm{Opt}_k^q)) - (w_e(T_\eta) - w(\mathrm{Opt}_k^q)) \\
&= com(\mathrm{Opt}_k^q).
\end{aligned}
$$

If $com(\mathrm{Opt}_k^q) = 0$, then a terminal spanning tree is an optimal solution, and the algorithm returns an optimal solution. Thus, assume that $com(\mathrm{Opt}_k^q) > 0$. It follows that $G^1_{\eta+1}$ is positive. This leads to the following lemma, whose proof is similar to the proof of Lemma 33.

Lemma 34. $\ln\left(\frac{G_0}{G^1_{\eta+1}}\right) \geq \frac{c^1_{\eta+1} + \sum_{i=1}^{\eta} com(K_i)}{com(\mathrm{Opt}_k^q)}$.

Proof. In the proof of Lemma 33, we showed that $G_{i-1}\frac{com(C_i)}{com(\mathrm{Opt}_k^q)} \leq tg_{T_{i-1}}(C_i)$, for $i = 1, 2, \ldots, \eta$. As a special case, we may also show

$$G_\eta \frac{c^1_{\eta+1}}{com(\mathrm{Opt}_k^q)} \leq g^1_{\eta+1}.$$

Now, repeating the same arguments as of the proof of Lemma 33, the result follows. □

Recall that $\alpha = w_v(\mathrm{Opt}^q)/w(\mathrm{Opt}^q)$, then we have the following lemma.

Lemma 35. *Let H be a Steiner tree, then $com(H) \leq w_v(H) + w_e(H)/2$. Moreover, $com(\mathrm{Opt}^q) \leq \frac{(1+\alpha)}{2} w(\mathrm{Opt}^q)$.*

Proof. Denote by $com_v(H)$ and $com_e(H)$ the vertex weight and edge weight of the commit a Steiner tree H. By adding dummy vertices and edges of zero weight, we may turn H into a regular binary tree H' where the root is a dummy vertex and the set of leaves correspond to set of terminals. For every inner vertex v of H', chose the lightest edge between the two edges to children of v, and denote the set selected edges by B. Note that $w_e(B) \leq w_e(H')/2 = w_e(H)/2$. Also, note that B induces a forest which spans every vertex of H' such that each connected component of contains exactly one terminal. Therefore, $w_e(B) \geq com_e(H') = com_e(H)$. Combining the previous inequalities, we obtain $com_e(H) \leq w_e(H)/2$. Since every inner vertex of H is in the commit of H, we have that $com_v(H) = w_v(H)$. Therefore

$$com(H) = com_e(H) + com_v(H) \leq \frac{w_e(H)}{2} + w_v(H).$$

In particular, if $H = \mathrm{Opt}^q$, we have

$$com(\mathrm{Opt}^q) \leq \frac{w_e(\mathrm{Opt}^q)}{2} + w_v(\mathrm{Opt}^q) = \frac{w(\mathrm{Opt}^q) + w_v(\mathrm{Opt}^q)}{2} = \frac{(1+\alpha)w(\mathrm{Opt}^q)}{2}.$$ □

Now we are ready to bound the cost of returned solution. Recall that Opt^q denotes an optimal solution to an instance I of q-MNSTP and that $\alpha := w_v(\mathrm{Opt}^q)/w(\mathrm{Opt}^q)$. The approximation factor depends on α; in the worst case, $\alpha = 0$, and the factor converges to 1.54931 as k grows.

Theorem 36. *For any $\delta > 0$, there is a constant k such that* GREEDYALG$_k$ *runs in polynomial time and has approximation factor at most $1 + \frac{1+\alpha}{2} \cdot \ln\left(3\frac{1-\alpha}{1+\alpha}\right) + \delta < 1.55$.*

Proof. We first verify that, for constant k, GREEDYALG$_k$ runs in polynomial time. Constructing set $\mathscr{C}_k$ takes polynomial time, since $|\mathscr{C}_k| \in O(|V|^k)$. Also, the gain of an already selected full component is not positive, thus each full component is selected at most once, and the loop is executed at most $|\mathscr{C}_k|$ times. Finally, observe that the gain of a full component C can be computed by executing a minimum spanning tree algorithm, and the commit cost of C can be computed in constant time, because C has constant size. Therefore one can select the full component C which maximizes $gain_{T_{t-1}}(C)/com(C)$ efficiently, and each iteration takes polynomial time.

Let S be the solution returned by algorithm GREEDYALG$_k$ and $S^{\eta+1}$ be the solution returned by GREEDYALG$_k^{\eta+1}$. Since $w(S) \leq w(S^{\eta+1})$, to analyze the approximation factor, we bound $w(S^{\eta+1})$. We obtain

$$\begin{aligned}
w(S^{\eta+1}) &= w(\text{MST}(T_0 \cup C_1 \cup C_2 \cup \cdots \cup C_{\eta+1})) \\
&\leq w(\text{MST}(\text{MST}(T_0 \cup C_1) \cup C_2 \cup \cdots \cup C_{\eta+1})) \\
&\leq com(C_1) + w(\text{MST}(T_1 \cup C_2 \cup \cdots \cup C_{\eta+1})) \\
&\leq \cdots \\
&\leq \sum_{i=1}^{\eta} com(C_i) + w(\text{MST}(T_\eta \cup C_{\eta+1})) \\
&\leq \sum_{i=1}^{\eta} com(C_i) + com(C_{\eta+1}) + w_e(T_{\eta+1}).
\end{aligned}$$

The first inequality holds as $\text{MST}((T_0 \cup C_1) \cup C_2 \cup \cdots \cup C_{\eta+1})$ is a spanning subgraph of $T_0 \cup C_1 \cup C_2 \cup \cdots \cup C_{\eta+1}$. The second inequality holds because $T_1 = MST(T_0 \cup \not{c}(C_1))$. The inequalities that follow are similar.

By using the identity $tg_{T_\eta}(C_{\eta+1}) = w_e(T_\eta) - w_e(T_{\eta+1})$, and replacing $tg_{T_\eta}(C_{\eta+1}) = g^1_{\eta+1} + g^2_{\eta+1}$ and $com(C_{\eta+1}) = c^1_{\eta+1} + c^2_{\eta+1}$ in the expression above, we get

$$w(S^{\eta+1}) \leq \sum_{i=1}^{\eta} com(C_i) + (c^1_{\eta+1} + c^2_{\eta+1}) + w_e(T_\eta) - (g^1_{\eta+1} + g^2_{\eta+1}).$$

Since, from the definition, $tg_{T_\eta}(C_{\eta+1}) = gain_{T_\eta}(C_{\eta+1}) + com(C_{\eta+1}) \geq com(C_{\eta+1})$, we obtain $c^2_{\eta+1} = \frac{g^2_{\eta+1}}{tg_{T_\eta}(C_{\eta+1})} com(C_{\eta+1}) \leq g^2_{\eta+1}$. Replacing in the last inequality, it follows that

$$w(S^{\eta+1}) \leq \sum_{i=1}^{\eta} com(C_i) + c^1_{\eta+1} + w_e(T_\eta) - g^1_{\eta+1}.$$

By using $g^1_{\eta+1} = w_e(T_\eta) - w(\text{Opt}^q_k)$, we get

$$w(S^{\eta+1}) \leq \sum_{i=1}^{\eta} com(C_i) + c^1_{\eta+1} + w(\text{Opt}^q_k).$$

Now, use Lemma 34 to obtain

$$\begin{aligned} w(S^{\eta+1}) &\leq \ln\left(\frac{G_0}{G^1_{\eta+1}}\right) com(\mathrm{Opt}^q_k) + w(\mathrm{Opt}^q_k) \\ &= \ln\left(\frac{w_e(T_0) - w(\mathrm{Opt}^q_k) + com(\mathrm{Opt}^q_k)}{com(\mathrm{Opt}^q_k)}\right) com(\mathrm{Opt}^q_k) + w(\mathrm{Opt}^q_k) \\ &= \ln\left(1 + \frac{w_e(T_0) - w(\mathrm{Opt}^q_k)}{com(\mathrm{Opt}^q_k)}\right) com(\mathrm{Opt}^q_k) + w(\mathrm{Opt}^q_k). \end{aligned}$$

The first equality comes from $G^1_{\eta+1} = com(\mathrm{Opt}^q_k)$, and from the definition of G_0.

To rewrite this bound in terms of $w(\mathrm{Opt}^q_k)$, we use Lemma 35, which implies $com(\mathrm{Opt}^q_k) \leq \frac{(1+\alpha)}{2} w(\mathrm{Opt}^q_k)$, and Theorem 13, which implies that $w_e(T) \leq 2(1-\alpha_k) w(\mathrm{Opt}^q_k)$.

$$\begin{aligned} w(S^{\eta+1}) &\overset{Lem\ 35}{\leq} \ln\left(1 + \frac{w_e(T_0) - w(\mathrm{Opt}^q_k)}{\frac{(1+\alpha)}{2} w(\mathrm{Opt}^q_k)}\right) \frac{(1+\alpha)}{2} w(\mathrm{Opt}^q_k) + w(\mathrm{Opt}^q_k) \\ &\overset{Thm\ 13}{\leq} \ln\left(1 + \frac{2(1-\alpha) w(\mathrm{Opt}^q_k) - w(\mathrm{Opt}^q_k)}{\frac{(1+\alpha)}{2} w(\mathrm{Opt}^q_k)}\right) \frac{(1+\alpha)}{2} w(\mathrm{Opt}^q_k) + w(\mathrm{Opt}^q_k) \\ &= \left[\ln\left(3\frac{1-\alpha}{1+\alpha}\right)\frac{(1+\alpha)}{2} + 1\right] w(\mathrm{Opt}^q_k). \end{aligned}$$

In the first inequality we used the fact that function $\ln\left(1 + \frac{w_e(T_0) - w(\mathrm{Opt}^q_k)}{com(\mathrm{Opt}^q_k)}\right) com(\mathrm{Opt}^q_k)$ is increasing with respect to $com(\mathrm{Opt}^q_k)$.

To complete the proof, notice that from Theorem 18, for any $\delta > 0$, there exists (a sufficiently large value) k such that $w(\mathrm{Opt}^q_k) \leq (1+\delta) w(\mathrm{Opt}^q)$. □

Chapter 6

Final Remarks

There is a plentiful set of practical and important problems related to the Steiner Tree problem, and many of them are NP-hard. A natural generalization of the classical STP is the variant in which both node and edge sets have associated weight functions. In this work, we considered this problem from the perspective of approximation algorithms. Unlike STP, for which the metric and non-metric versions are equivalent, when considering the node weighted version, the restriction that the input graph is metric is central to obtain a constant-factor approximation; indeed, the non-metric version encodes the Set Cover Problem, and it has no approximation factor smaller than $O(\log n)$ unless P = NP [18].

For the metric version, we showed that even a terminal minimum spanning tree is a 2-approximation. Breaking the barrier of factor 2, however, seems as challenging as it was for the for the STP. Unfortunately, many of the ideas and techniques used in the STP case cannot be applied to the MNSTP. When we consider the particular – yet relevant – case in which the ratio between vertices and edges weights is bounded (the q-MNSTP), we can extend the algorithms and techniques from STP to MNSTP, and obtain improved factors, even matching the factor for one algorithm for the standard STP.

Recall that $\alpha := w_v(\text{Opt}^q)/w(\text{Opt}^q)$ is the ratio of vertex weight over the total weight of an optimal solution. The described results for MNSTP and q-MNSTP are summarized as follows.

- A minimum terminal spanning tree is $(2-\alpha)$-approximation for MNSTP.
- The optimal value of the k-restricted version of q-MNSTP is not much larger then the optimal value of q-MNSTP. This does not hold in the MNSTP case, and thus enumerating all the full components does not seem to be a viable option to improve the factor for MNSTP.
- The *iterative randomized rounding* technique in [6] can be extended for the q-MNSTP to obtain an expected approximation factor of $4 - \frac{3}{\sqrt[3]{2-2\alpha}} + \delta(q,k) < 1.62$. Interestingly, the analysis in [6] which leads to a 1.39-approximation for STP cannot be extended to q-MNSTP.
- The greedy algorithm by [19] can be generalized to solve instances of q-MNSTP, and we show that this adapted algorithm has approximation factor of $1 + \frac{1+\alpha}{2} \cdot \ln\left(3\frac{1-\alpha}{1+\alpha}\right) + \delta(q,k) < 1.55$. This matches the algorithm's factor for STP.

We remark that carefully considering the ratio α is essential for the analysis of the approximation factors in the proposed algorithms; if one considers the particular cases where the value of α is at least a constant, the obtained approximation factors decrease. In Figure 6.1, we draw the approximation factors as functions of α.

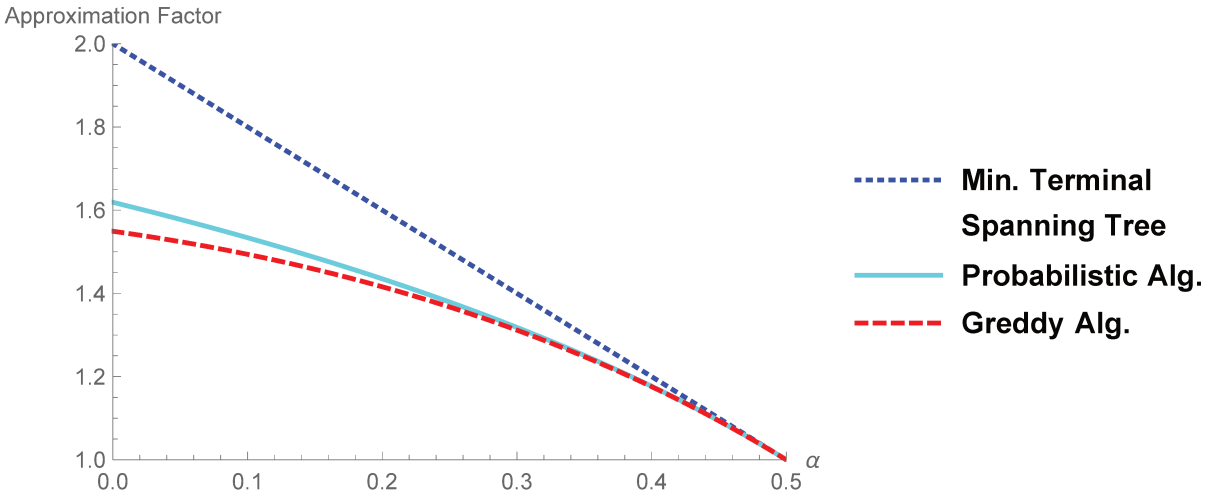


Figure 6.1: Approximation factors for the q-MNSTP in function of $\alpha := \frac{w_v(\text{Opt}^q)}{w(\text{Opt}^q)}$.

Bibliography

[1] A. Archer, M. Bateni, M. Hajiaghayi, and H. Karloff. Improved approximation algorithms for prize-collecting Steiner tree and TSP. *SIAM Journal on Computing*, 40(2):309–332, 2011.

[2] S. Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *J. ACM*, 45(5):753–782, September 1998.

[3] P. Berman, M. Karpinski, and A. Zelikovsky. *1.25-approximation algorithm for Steiner tree problem with distances 1 and 2*, pages 86–97. Springer Berlin Heidelberg, 2009.

[4] J.A. Bondy and U.S.R. Murty. *Graph Theory*. Springer-Verlag London, 2008.

[5] A. Borchers and D.Z. Du. The k-Steiner ratio in graphs. *SIAM Journal on Computing*, 26(3):857–869, 1997.

[6] J. Byrka, F. Grandoni, T. Rothvoss, and L. Sanità. Steiner tree approximation via iterative randomized rounding. *J. ACM*, 60(1):6:1–6:33, February 2013.

[7] M. Chlebík and J. Chlebíková. The Steiner tree problem on graphs: Inapproximability results. *Theoretical Computer Science*, 406(3):207–214, 2008.

[8] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[9] J. Edmonds. Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71B(4):233–240, 1967.

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

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

[12] S. Guha and S. Khuller. Improved methods for approximating node weighted Steiner trees and connected dominating sets. *Information and Computation*, 150(1):57–74, 1999.

[13] S.Y. Hsieh and H.M. Gao. On the partial terminal Steiner tree problem. *The Journal of Supercomputing*, 41(1):41–52, July 2007.

[14] C.W. Huang, C.W. Lee, H.M. Gao, and S.Y. Hsieh. The internal Steiner tree problem: Hardness and approximations. *Journal of Complexity*, 29(1):27–43, 2013.

[15] X. Li, F. Zou, Y. Huang, D. Kim, and W. Wu. A better constant-factor approximation for selected-internal Steiner minimum tree. *Algorithmica*, 56(3):333–341, March 2010.

[16] F.V. Martinez, J.C. de Pina, and J. Soares. Algorithms for terminal Steiner trees. *Theoretical Computer Science*, 389(1):133–142, 2007.

[17] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[18] D. Moshkovitz. The projection games conjecture and the np-hardness of $\ln n$-approximating set-cover. *Theory of Computing*, 11(7):221–235, 2015.

[19] G. Robins and A. Zelikovsky. Tighter bounds for graph Steiner tree approximation. *SIAM Journal on Discrete Mathematics*, 19(1):122–134, 2005.

[20] V.V. Vazirani. *Approximation Algorithms*. Springer-Verlag Berlin Heidelberg, 2003.

[21] C.C. Wei, S.Y. Hsieh, C.W. Lee, and S.L. Peng. An improved approximation algorithm for the partial-terminal Steiner tree problem with edge cost 1 or 2. *Journal of Discrete Algorithms*, 35:62–71, 2015.

[22] A.Z. Zelikovsky. An 11/6-approximation algorithm for the Steiner problem on graphs. *Annals of Discrete Mathematics*, 51:351–354, 1992.