The Generalized Minimum Spanning Tree Problem

Petrica C. Pop

2002

Ph.D. thesis
University of Twente

Twente University **Press**

Also available in print:
http://www.tup.utwente.nl/catalogue/book/index.jsp?isbn=9036517850

The Generalized Minimum Spanning Tree Problem

Twente University **Press**

Publisher: Twente University Press,
P.O. Box 217, 7500 AE Enschede, the Netherlands, www.tup.utwente.nl

Cover design: Jo Molenaar, [deel4 ontwerpers], Enschede
Print: Océ Facility Services, Enschede

ISBN 9036517850

THE GENERALIZED MINIMUM SPANNING TREE PROBLEM

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de rector magnificus,
prof.dr. F.A. van Vught,
volgens besluit van het College voor Promoties
in het openbaar te verdedigen
op vrijdag 13 december 2002 te 15.00 uur

door

Petrica Claudiu Pop
geboren op 30 oktober 1972
te Baia Mare (Roemenië)

Dit proefschrift is goedgekeurd door de promotor
prof.dr. U. Faigle

en de assistent-promotoren
dr. W. Kern
dr. G. Still

To the memory of my mother
Pop Ana

Preface

I want to thank to all the people who helped me, directly and indirectly with the realization of this thesis.

First of all I want to thank my supervisors prof. U. Faigle and prof. G. Woeginger; to prof. U. Faigle also for offering me the opportunity to become a PhD student in the Operations Research group. Besides prof. Faigle and prof. Woeginger there were two persons without whom this work could not be carried out in the manner it has been done: Walter Kern and Georg Still. Their advice and support during the project were very valuable for me and I enjoyed very much working with them.

A special word of thanks goes to my daily supervisor Still. The door of his office was always open for me and he has always found time to answer my questions and to hear my problems.

I am very happy that I met prof. G. Woeginger. I am grateful to him for the useful discussions we had and his comments regarding my research.

I want to express my gratitude to prof. C. Hoede, prof. R. Dekker, prof. A. van Harten and prof. M. Labbé for participating in the promotion committee and for their interest in my research.

A word of thanks goes to my colleagues from the Department of Discrete Mathematics, Operations Research and Stochastic and in particular to my former office-mates: Cindy Kuipers and Daniële Paulusma who offered me a pleasant working environment.

During the time spent in Twente I made many good friends and I am sure that without them my life in the Netherlands would not be so comfortable. Special thanks to Octav, Rita, Natasa, Goran, Adriana, Marisela, Valentina and Andrei.

At last but not at least I want to thank to my family.

In particular I am most grateful to my wife, my parents and my brother. I would not have been where I am now without their unconditioned love and support.

Baia Mare, 10 November 2002

Petrica C. Pop

Contents

Chapter 1

Introduction

In this thesis we use several combinatorial optimization techniques to solve the Generalized Minimum Spanning Tree problem. For an overview of general combinatorial optimization techniques, we refer to the books of Nemhauser and Wolsey [55], Papadimitriou and Steiglitz [57] and Schrijver [69].

In this chapter we present fundamental concepts of Combinatorial and Integer optimization (Section 1.1), Polyhedral theory (Section 1.2) and Graph theory (Section 1.3). We end the chapter with an outline of the thesis.

1.1 Combinatorial and Integer Optimization

Combinatorial Optimization is the process of finding one or more best (optimal) solutions in a well defined discrete problem space, i.e. a space containing a finite set of possible solutions, that optimizes a certain function, the so-called *objective function*. The finite set of possible solutions can be described by inequality and equality constraints, and by integrality constraints. The integrality constraints force the variables to be integers. The set of points that satisfy all these constraints is called the *(feasible) solution set*.

Such problems occur in almost all fields of management (e.g. finance, marketing, production, scheduling, inventory control, facility location, etc.), as well as in many engineering disciplines (e.g. optimal design of waterways or bridges, design and analysis of data networks, energy resource-planning models, logistic of electrical power generation and transport, etc). A survey of applications of combinatorial optimization is given by Grötschel in [30].

Combinatorial Optimization models are often referred to as integer programming models where some or all of the variables can take on only a finite number of alternative possibilities.

In this thesis we consider combinatorial optimization problems for which the objective function and the constraints are linear and the variables are integers. These problems are called *integer programming problems*:

$$(IP) \quad \begin{array}{lrcl} \min & c^T x & & \\ s.t. & Ax & \leq & b \\ & x & \in & \mathbb{Z}^n \end{array}$$

where $\mathbb{Z}^n$ is the set of integral n-dimensional vectors, $x = (x_1, ..., x_n)$ is an integer n-vector and c is an integer n-vector. Furthermore, we let m denote the number of inequality contraints, A an $m \times n$ matrix and b an m-vector. If we allow some variables x_i to be continous instead of integer, i.e. $x_i \in \mathbb{R}$ instead of $x_i \in \mathbb{Z}$, then we obtain a *mixed integer programming problem*, denoted by (MIP). For convenience, we discuss integer linear programs that are *minimization* problems with *binary* variables, i.e. the integer variables are restricted to values 0 or 1. In Chapter 3 we present several integer programming and mixed integer programming models of the Generalized Minimum Spanning Tree problem.

Solving integer programming problems can be a difficult task. The difficulty arises from the fact that unlike linear programming, where the feasible region is a convex set, in integer problems one must search a lattice of feasible points or, in the mixed integer case a set of disjoint halflines or line segments to find an optimal solution. Therefore, unlike linear programming where, due to the convexity of the problem, we can exploit the fact that any local solution is a global optimum, integer programming problems may have many ”local optima” and finding a global optimum to the problem requires one to prove that a particular solution dominates all feasible points by arguments other than the calculus-based derivative approaches of convex programming.

When optimizing combinatorial problems, there is always a trade-off between the computational effort (and hence the running time) and the quality of the obtained solution. We may either try to solve the problem to optimality with an *exact algorithm*, or choose for an *approximation* or *heuristic algorithm*, which uses less running time but does not guarantee optimality of the solution.

1.1.1 Complexity theory

The first step in studying a combinatorial problem is to find out whether the problem is ”easy” or ”hard”. Complexity theory deals with this kind of problem classification. In this subsection we summarize the most important concepts of complexity theory. Most approaches are taken from the books mentioned before and from the book of Grötschel, Lovász and Schrijver [31].

An *algorithm* is a list of instructions that solves every *instance* of a problem in a finite number of steps. (This means also: the algorithm detects that the problem instance has no solution).

The *size* of a problem is the amount of information needed to represent the instance. The instance is assumed to be described (encoded) by a string of symbols. Therefore, the size of an instance equals the number of symbols in the string.

The *running time* of a combinatorial optimization algorithm is measured by an upper bound on the number of elementary arithmetic operations (adding, subtracting, multiplying, dividing and comparing numbers) it needs for any valid input, expressed as a function of the input size. The *input* is the data used to represent a problem instance. If the input size is measured by s, then the running time of the algorithm is expressed as $O(f(s))$, if there are

constants b and s_0 such that the number of steps for any instance with $s \geq s_0$ is bounded from above by $bf(s)$. We say that the running time of such an algorithm is of order $f(s)$.

An algorithm is said to be a *polynomial time algorithm* when its running time is bounded by a polynomial function, $f(s)$. An algorithm is said to be an *exponential time algorithm* when its running time is bounded by an exponential function (e.g. $O(2^{p(s)})$).

The theory of complexity concerns in the first place decision problems. A *decision problem* is a question that can be answered only by "yes" or "no". For example in the case of the integer programming problem (IP) the decision problem is:

> Given an instance of (IP) and an integer L is there a feasible solution x such that $c^T x \leq L$?

For a combinatorial optimization problem we have the following: if one can solve the decision problem efficiently, then one can solve the corresponding opimization problem efficiently.

Decision problems that are solvable in polynomial time are considered to be "easy" , the class of these problems is denoted by $\mathcal{P}$. $\mathcal{P}$ includes for example linear programming and the minimum spanning tree problem.

The class of decision problems solvable in exponential time is denoted by EXP. Most combinatorial optimization problems belong to this class. If a problem is in $EXP \setminus \mathcal{P}$, then solving large instances of this problem will be difficult. To distinguish between "easy" and "hard" problems we first describe a class of problems that contains $\mathcal{P}$.

The complexity class $\mathcal{NP}$ is defined as the class of decision problems that are solvable by a so-called *non-deterministic algorithm*. A decision problem is said to be in $\mathcal{NP}$ if for any input that has a positive answer, there is a certificate from which the correctness of this answer can be derived in polynomial time.

Obviously, $\mathcal{P} \subseteq \mathcal{NP}$ holds. It is widely assumed that $\mathcal{P} = \mathcal{NP}$ is very unlikely. The class $\mathcal{NP}$ contains a subclass of problems that are considered to be the hardest problems in $\mathcal{NP}$. These problems are called $\mathcal{NP}$*-complete* problems. Before giving the definition of an $\mathcal{NP}$*-complete* decision problem we explain the technique of polynomially transforming one problem into another. Let Π_1 and Π_2 be two decision problems.

Definition 1.1 *A* **polynomial transformation** *is an algorithm $\mathcal{A}$ that, for every instance σ_1 of Π_1 produces in polynomial time an instance σ_2 of Π_2 such that the following holds: for every instance σ_1 of Π_1, the answer to σ_1 is "yes" if and only if the answer to instance σ_2 of Π_2 is "yes".* □

Definition 1.2 *A decision problem Π is called $\mathcal{NP}$-complete if Π is in $\mathcal{NP}$ and every other $\mathcal{NP}$ decision problem can be polynomially transformed into Π.* □

Clearly, if an *$\mathcal{NP}$-complete* problem can be solved in polynomial time, then all problems in $\mathcal{NP}$ can be solved in polynomial time, hence $\mathcal{P} = \mathcal{NP}$. This explains why the *$\mathcal{NP}$-complete* problems are considered to be the hardest problems in $\mathcal{NP}$. Note that polynomially transformability is a transitive relation, i.e if Π_1 is polynomially transformable to Π_2 and Π_2 is polynomially transformable to Π_3, then Π_1 is polynomially transformable to Π_3. Therefore, if we want to prove that a decision problem Π is *$\mathcal{NP}$-complete*, then we only have to show that

(i) Π is in $\mathcal{NP}$.

(ii) Some decision problem already known to be *$\mathcal{NP}$-complete* can be polynomially transformed to Π.

Now we want to focus on combinatorial optimization problems. Therefore we extend the concept of polynomially transformability. Let Π_1 and Π_2 be two problems (not necessarily decision problems).

Definition 1.3 *A* **polynomial reduction** *from Π_1 to Π_2 is an algorithm $\mathcal{A}_1$ that, solves Π_1 by using an algorithm $\mathcal{A}_2$ for Π_2 as a subroutine such that, if $\mathcal{A}_2$ were a polynomial time algorithm for Π_2, then $\mathcal{A}_1$ would be a polynomial time algorithm for Π_1.* □

Definition 1.4 *An optimization problem Π is called $\mathcal{NP}$-hard if there exists an $\mathcal{NP}$-complete decision problem that can be polynomially reduced to Π.* □

Cearly, an optimization problem is *$\mathcal{NP}$-hard* if the corresponding decision problem is *$\mathcal{NP}$-complete*. In particular, the Generalized Minimum Spanning Tree problem is *$\mathcal{NP}$-hard* (see Section 2.5).

1.1.2 Heuristic and Relaxation Methods

As we have seen, once established that a combinatorial problem is $\mathcal{NP}$-*hard*, it is unlikely that it can be solved by a polynomial algorithm.

Finding good solutions for hard minimization problems in combinatorial optimization requires the consideration of two issues:

- calculation of an upper bound that is as close as possible to the optimum;
- calculation of a lower bound that is as close as possible to the optimum.

General techniques for generating good upper bounds are essentially heuristic methods such as Simulated Annealing, Tabu Search, Genetic Algorithms, etc., which we are going to consider in chapter 7 of the thesis. In addition, for any particular problem, we may well have techniques which are specific to the problem being solved.

On the question of lower bounds, we present in Chapter 5 the following techniques:

- Linear Programming (LP) relaxation
  In LP relaxation we take an integer (or mixed-integer) programming formulation of the problem and relax the integrality requirement on the variables. This gives a linear program which can either be solved exactly using a standard algorithm (simplex or interior point); or heuristically (dual ascent). The solution value obtained for this linear program gives a lower bound on the optimal solution to the original minimization problem.
- Lagrangian relaxation
  The general idea of Lagrangian relaxation is to ”relax” (dualize) some (or all) constraints by adding them to the objective function using Lagrangian multipliers. Choosing ”good” values for the Lagrangian multipliers is of key importance in terms of quality of the lower bound generated.
- Semidefinite programming (SDP) relaxation
  Combinatorial optimization problems often involve binary (0,1 or +1,-1) decision variables. These can be modelled using quadratic constraints $x^2 - x = 0$, and $x^2 = 1$, respectively. Using the positive semidefinite matrix $X = xx^T$, we can lift the problem into a matrix space and

obtain a Semidefinite Programming Relaxation by ignoring the rank one restriction on X, see e.g. [49]. These semidefinite relaxations provide tight bounds for many classes of hard problems and in addition can be solved efficiently by interior-point methods.

The connection between heuristics and relaxations in the case of a minimization problem can be summarized by the diagram of Figure 1.1.

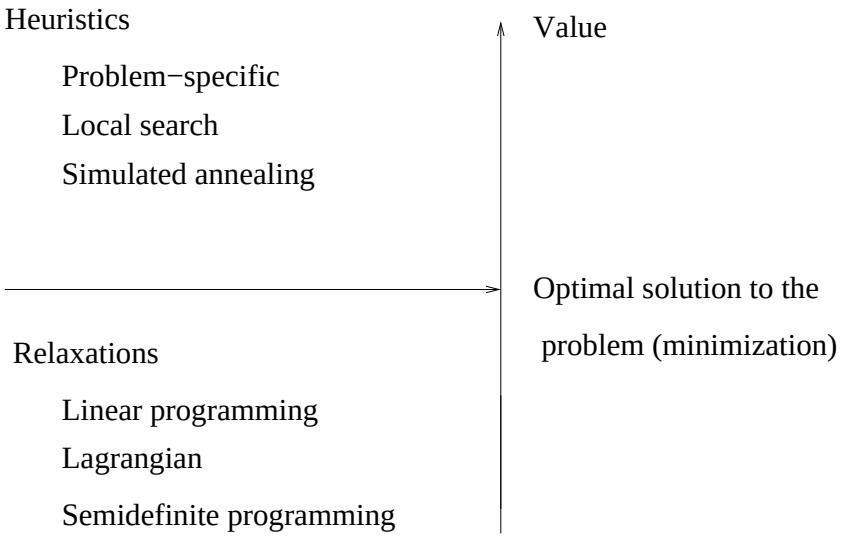

Figure 1.1: Connection between heuristics and relaxations

1.1.3 Dynamic programming

Dynamic programming is a decomposition technique that first decomposes the problem into a nested family of subproblems. The solutions to the original problem are obtained by either working backward from the end to the beginning (*backward dynamic programming*) or forward from the beginning to the end (*forward dynamic programming*).

Five characteristics can be distinguished that are common to dynamic programming applications:

1. The problem can be divided into *stages* t, with a *decision* required at each stage.

2. Each stage t has a set of *states* $\{i_t\}$ associated with it. At any stage, a state holds all the information that is needed to make a decision.

3. The decision chosen at any stage determines how the state at the current stage is transformed into the state at the next stage, as well as the immediately earned reward or cost.

4. Given the current state, the optimal decision for each remaining stages must not depend on previously reached states or previously chosen decisions. This is the so-called *principle of optimality* for dynamic programming (Bellman, [3]).

5. If the states for the problem have been classified into one of T stages, there must be a *recursion* that relates the cost or reward earned during stages $t, t+1, ..., T$ to the cost or reward earned from stages $t+1, t+2, ..., T$.

Dynamic programming algorithms are computationally efficient, as long as the state space does not become too large. However, when the state space becomes too large implementation problems may occur (e.g. insufficient computer memory), or excessive computational time may be required to use dynamic programming to solve the problem. For more information about dynamic programming we refer to Bellman [3] and Dreyfus and Law [12].

1.2 Polyhedral theory

The theory we discuss in this section is derived from the the books of Schrijver [69] and Grötschel, Lovász and Schrijver [31].

Consider a set of points $X = \{x^1, \ldots, x^k\} \subseteq \mathbb{R}^n$ and a vector $\lambda \in \mathbb{R}^k$. The linear combination $x = \sum_{i=1}^k \lambda_i x^i$ is an *affine combination* if $\sum_{i=1}^k \lambda_i = 1$, and x is called a *convex combination* if in addition to $\sum_{i=1}^k \lambda_i = 1$ we have $\lambda_i \geq 0$. X is called *linearly (affinely) independent* if no point $x^i \in X$ can be written as a linear (affine) combination of the other points in X.

A subset $S \subseteq \mathbb{R}^n$ is *convex* if for every finite number of points $x^1, \ldots, x^k \in S$ any convex combination of these points is a member of S.

A nonempty set $C \subseteq \mathbb{R}^n$ is called a *convex cone* if $\lambda x + \mu y \in C$ for all $x, y \in C$ and for all real numbers $\lambda, \mu \geq 0$.

A convex set $P \subseteq \mathbb{R}^n$ is a *polyhedron* if there exists an $m \times n$ matrix A and a vector $b \in \mathbb{R}^m$ such that

$$P = P(A, b) = \{x \in \mathbb{R}^n \mid Ax \leq b\}.$$

We call $Ax \leq b$ a *system of linear inequalities*.

A polyhedron $P \subseteq \mathbb{R}^n$ is *bounded* if there exist vectors $l, u \in \mathbb{R}^n$ such that $l \leq x \leq u$ for all $x \in P$. A bounded polyhedron is called a *polytope*.

For our purposes, only rational polyhedra are of interest. A polyhedron is *rational* if it is the solution set of a system $Ax \leq b$ of linear inequalities, where A and b are rational. From now on we will implicitly assume a polyhedron to be rational.

A point $x \in P$ is called a *vertex* of P if x cannot be written as a convex combination of other points in P.

The *dimension* of a polyhedron $P \subseteq \mathbb{R}^n$ is equal to the maximum number of affinely independent points in P minus 1. An *implicit equality* of the system $Ax \leq b$ is an inequality $\sum_{j=1}^{n} a_{ij}x_j \leq b_i$ of that system such that $\sum_{j=1}^{n} a_{ij}x_j = b_i$ for all vectors $x \in P(A, b)$. For $Ax \leq b$ we denote the (sub)system of implicit equalities by $A^{=}x \leq b^{=}$. Let rank(A) denote the *rank of a matrix*, i.e., the maximum number of linearly independent row vectors. We have the following standard result.

Theorem 1.1 *The dimension of a polyhedron $P(A, b) \subseteq \mathbb{R}^n$ is equal to $n -$ rank$(A^{=})$.* □

A subset $H \subseteq \mathbb{R}^n$ is called a *hyperplane* if there exists a vector $h \in \mathbb{R}^n$ and a number $\alpha \in \mathbb{R}$ such that

$$H = \{x \in \mathbb{R}^n \mid h^T x = \alpha\}.$$

A *separating hyperplane* for a convex set S and vector $y \notin S$ is a hyperplane given by a vector $h \in \mathbb{R}^n$ and a number $\alpha \in \mathbb{R}$ such that $h^T x \leq \alpha$ and $h^T y > \alpha$ holds for all $x \in S$.

The *separation problem* for a polyhedron $P \subseteq \mathbb{R}^n$ is, given a vector $x \in \mathbb{R}^n$, to decide whether $x \in P$ or not, and, if $x \notin P$, to find a separating hyperplane for P and x. A *separation algorithm* for a polyhedron P is an algorithm that solves the separation problem for P.

Linear programming (LP) deals with maximizing or minimizing a linear function over a polyhedron. If $P \subseteq \mathbb{R}^n$ is a polyhedron and $d \in \mathbb{R}^n$, then we call the optimization problem

$$(LP) \quad \max\{d^T x \mid x \in P\}$$

a *linear program*. A vector $x \in P$ is called a *feasible solution* of the linear program and x^* is called an *optimal solution* if x^* is feasible and $d^T x^* \geq d^T x$ for all feasible solutions x. We say that (LP) is *unbounded* if for all $\lambda \in \mathbb{R}$ there exists a feasible solution $x^* \in P$ such that $d^T x^* \geq \lambda$, If (LP) has no optimal solution then it is either infeasible or unbounded.

The first polynomial time algorithm for LP is the so-called *ellipsoid algorithm*, proposed by Khachiyan [43]. Although of polynomial running time, the algorithm is impractical for LP. Nevertheless, it has extensive theoretical applications in combinatorial optimization: the stable set problem on perfect graphs for example can be solved in polynomial time using the ellipsoid algorithm. Grötschel, Lóvasz and Schrijver [32] refined this method in such a way that the computational complexity of optimizing a linear function over a convex set S depends on the complexity of the separation problem for S. In 1984, Karmarkar [42] presented another polynomial time algorithm for LP. His algorithm avoids the combinatorial complexity (inherent in the simplex algorithm) of the vertices, edges and faces of the polyhedron by staying inside the polyhedron. His algorithm lead to many other algorithms for LP based on similar ideas. These algorithms are known as *interior point methods*.

1.3 Graph theory

In this section we include some terminology for readers not familiar with graph theory. For more details on graph theory we refer to the book of Bondy and Murty [7].

A *graph* G is an ordered pair (V, E), where V is a nonempty, finite set called the *node set* and E is a set of (unordered) pairs (i, j) with $i, j \in V$ called the *edge set*.

The elements of V are called *nodes* and the elements of E are called *edges*. If $e = (i, j) \in E$ we say that node i and node j are *adjacent*. In such a case i and j are called the *end points* of e or *incident* with e. Furthermore, we say

that e is *incident* with i and j. Two edges are called *adjacent* if they have a common incident node. A node j for which there is an edge $(i,j) \in E$ is a *neighbor* of i. The set of neighbors of a node i is denoted by $\delta(i)$ and a node with no neighbors is called an *isolated node*. For $U \subseteq V$, we define the *cutset*, denoted by $\delta(U)$ as follows:

$$\delta(U) = \{\, e = (i,j) \in E \mid i \in U,\ j \notin U\}$$

A *multigraph* is a graph with possibly more than one edge between two nodes.

We speak of a *weighted* graph if a *weight function* $w : E \to \mathbb{R}$ is defined on the edge set E of a graph G. The number $w(e)$ is the *weight* of an edge $e \in E$. (It can usually be interpreted as a certain profit or cost.) The *weight of a subset* $E' \subseteq E$ is equal to the sum of the weights of its edges and denoted by $w(E')$.

A *complete graph* is a graph with an edge between every pair of nodes. The complete graph on n nodes is denoted by K_n.

A *subgraph* of G is a graph $G' = (V', E')$ with $V' \subseteq V$ and $E' \subseteq E$. G' is called the subgraph *induced* by V', denoted by $G|_{V'}$, if $E' = \{(i,j) \in E \mid i,j \in V'\}$. If $V' \subseteq V$ then we let $G \backslash V'$ denote the graph obtained by removing V' (and the edges incident with nodes in V'). If $E' \subseteq E$ then $G \backslash E'$ denotes the graph obtained by removing the edges in E'. We say that a graph G *contains* a graph G' if G has G' as subgraph.

A *path* from i to j is a graph $P = (V, E)$ with a node set that can be ordered as $v_0, \ldots, v_n$ with $v_0 = i$ and $v_n = j$ such that $E = \{(v_k, v_{k+1}) \mid k = 0, \ldots, n-1\}$. The nodes i and j are called the *end points* of the path and n is the *length* of the path. We also write $P = v_0 v_1 \ldots v_n$.

A *cycle* is a graph for which the node set can be ordered as $v_0, \ldots, v_n$ such that $E = \{(v_k, v_{k+1}) \mid k = 0, \ldots, n-1\} \cup \{(v_n, v_0)\}$. We denote a cycle on n nodes by C_n.

A graph G is called *connected* if G contains a path from i to j for each two nodes $i, j \in V$. A *component* G' of G is a maximal connected subgraph of G, i.e., if $\hat{G}$ is a connected subgraph of G and G' is a subgraph of $\hat{G}$, then $\hat{G} = G'$. The *size of a component* is its number of nodes. We denote the size of a component G' by $|G'|$. A component is called *even* or *odd* if it has an even respectively odd number of nodes.

A *tree* is a connected graph T that does not contain any cycle. A node $i \in V$ is called a *leaf* of a tree $T = (V, E)$, if i has exactly one neighbor. A *forest* is a

graph (not necessarily connected) that does not contain any cycle. A *spanning tree* of $G = (V, E)$ is a tree (V', E') with $V' = V$.

A *node cover* in a graph $G = (V, E)$ is a subset V' of V such that every edge in E is incident with a node in V'.

If the pairs (i, j) in the edge set of a graph are ordered, then we speak of a *directed graph* or *digraph* and we call such an ordered pair (i, j) an *arc*. In this case the edge set is usually denoted by A. The arc $a = (i, j) \in A$ is an *outcoming arc* of node i and is an *incoming arc* of node j. The set of outcoming arcs of a node i is denoted by $\delta^+(i)$ and the set of incoming arcs of a node i is denoted by $\delta^-(i)$. To this end, call a subset A' of A an $s - t$ *cut* if $A' = \delta^+(U)$ for some subset U of V satisfying $s \in U$ and $t \notin U$, where for every $U \subseteq V$, $\delta^+(U)$ is defined as follows:

$$\delta^+(U) = \{\ (i, j) \in A \mid i \in U,\ j \notin U\}$$

Let $D = (V, A)$ be a directed graph and let $r, s \in V$. A function $f : A \to \mathbb{R}$ is called an $r - s$ *flow* if

(i) $f(a) \geq 0$ for each $a \in A$,

(ii) $\sum_{a \in \delta^+(v)} f(a) = \sum_{a \in \delta^-(v)} f(a)$ for each $v \in V \setminus \{r, s\}$.

The value of an $r - s$ flow f is, by definition:

$$value(f) := \sum_{a \in \delta^+(r)} f(a) - \sum_{a \in \delta^-(r)} f(a).$$

So the value is the net amount of flow leaving r. It is also equal to the net amount of flow entering s.

Let $c : A \to \mathbb{R}_+$, be a *capacity function*. We say that a flow f is *under* c if

$$f(a) \leq c(a) \qquad \text{for each } a \in A.$$

The *maximum flow problem* now is to find an $r - s$ flow under c, of maximum value. To formulate the so-called min-max relation, we define the *capacity* of a cut $\delta^+(U)$ by

$$c(\delta^+(U)) := \sum_{a \in \delta^+(U)} c(a).$$

Then the following result known as *max-flow min-cut* theorem, holds

Theorem 1.2 *(Ford and Fulkerson (1956))*
For any directed graph $D = (V, A)$, $r, s \in V$, and $c : A \to \mathbb{R}_+$, the maximum value of an $r - s$ flow under c is equal to the minimum capacity of an $r - s$ cut:

$$\max_{f\ r-s\ flow} value(f) = \min_{\delta^+(U)\ r-s\ cut} c(\delta^+(U)).$$

□

1.4 Outline of the thesis

In this thesis we apply several techniques of combinatorial optimization to the *Generalized Minimum Spanning Tree problem* denoted by GMST. This combinatorial optimization problem was introduced by Myung *et al.* [54] and finds many interesting applications.

In Chapter 2 we introduce the concept of *generalization* in the context of combinatorial optimization, describe the Minimum Spanning Tree problem which is a special case of the Generalized Minimum Spanning Tree problem and two efficient algorithms for constructing minimum spanning trees, namely, Kruskal's algorithm and Prim's algorithm. Finally, we define the Generalized Minimum Spanning Tree problem, present results regarding its complexity and cases when the problem is solvable in polynomial time and applications.

Chapter 3 concentrates on several integer and mixed integer programming formulations of the GMST problem. We compare the polyhedra defined by the LP relaxations of these formulations. Based on a new formulation we give a solution procedure which solves the problem to optimality for all numerical instances considered in the related literature so far. Computational results are reported for many instances of the problem.

Chapter 4 deals with approximation algorithms. We present an in-approximability result for the GMST problem: under the assumption $\mathcal{P} \neq \mathcal{NP}$, there is no approximation algorithm for the GMST problem. However, under the following assumptions:

- the graph has bounded cluster size,
- the cost function is strictly positive and satisfies the triangle inequality, i.e. $c_{ij} + c_{jk} \geq c_{ik}$ for all $i, j, k \in V$,

we present a polynomial approximation algorithm for the GMST problem.

In Chapter 5 we discuss the basic theory of Lagrangian and semidefinite programming relaxations and we study different possible dualizations for a general 0-1 program.

In Chapter 6 we present an algorithm based on Lagrangian relaxation of a bidirectional multicommodity flow formulation of the GMST problem. The subgradient optimization algorithm is used to obtain lower bounds. Computational results are reported for many instances of the problem.

Chapter 7 deals with heuristic algorithms. We present the basic theory of local search and other concepts underlying many modern heuristic techniques, such as Simulated Annealing, Tabu Search Genetic Algorithms, etc., and we solve the GMST problem with Simulated Annealing. Computational results are reported.

Chapter 2

The Generalized Minimum Spanning Tree Problem

The aim of this chapter is to introduce the Generalized Minimum Spanning Tree problem. In the first section we discuss the concept of *generalization* and provide a list of problems defined on graphs that have a *generalized* structure. In the second section we describe the Minimum Spanning Tree problem which is a special case of the Generalized Minimum Spanning Tree problem and two efficient algorithms for constructing minimum spanning trees, namely, Kruskal's algorithm and Prim's algorithm. The next sections focus on the GMST problem: we give the definition of the problem, present results regarding its complexity and cases when the problem is solvable in polynomial time and applications. As well as in the last section we present a variant of the GMST problem.

2.1 Generalized Combinatorial Optimization Problems

In this section we present a list of problems defined on graphs that have a *generalized* structure. We begin this section by introducing the concept of generalization.

Given a graph $G = (V, E)$ and a cost function $c : E \to \mathbb{R}$, following the definition of Nemhauser and Wolsey [55], a *combinatorial optimization* problem consists of determining among a finite set of feasible solutions those that minimize the cost function. If we let $\mathcal{F}$ be a family of subsets of the edge set E and denote by $c(F) = \sum_{e \in F} c_e$ for $F \subseteq E$, a combinatorial optimization problem in its minimization form is:

$$\min \{c(F) \ : \ F \in \mathcal{F}\}.$$

Classical combinatorial optimization problems can often be *generalized* in a natural way by considering a related problem relative to a given *partition* $V = V_1 \cup V_2 \cup ... \cup V_m$ of the nodes into *clusters* $V_k \subseteq V$, $k \in \{1, ..., m\}$ such that the classical problem corresponds to the trivial partition $V_k = \{k\}$ into singletons.

For example,

- the Generalized Minimum Spanning Tree Problem

as introduced by Myung, Lee and Tcha [54] asks for a cost-minimal tree T in G which spans exactly one node $i_k \in V_k$ in each cluster. The Generalized Minimum Spanning Tree Problem is the subject of this thesis. In what it follows we will present several results regarding its complexity, approximability, solvability, etc.

- the Generalized Steiner Tree Problem

Given an complete undirected graph $G = (V, E)$ and a subset of nodes $S \subseteq V$ such that S is partitioned into m clusters and a positive cost function defined on the edge set E, the *Generalized Steiner Tree Problem* denoted GSTP asks for a minimum cost tree of G that contains *at least* one node from each cluster.

The Generalized Steiner Tree problem was introduced by Reich and Widmayer [65] and is also known as the *Group Steiner Tree Problem* or the *Class Steiner Tree Problem*.

- the Generalized Traveling Salesman Problem

Given an undirected graph $G = (V, E)$ with the nodes partitioned into m clusters and edges defined between nodes from different clusters with a positive cost, the *symmetric Generalized Traveling Salesman Problem* asks for a minimum cost cycle that visits *exactly* one node in each cluster. This problem was introduced by Henry-Labordere [36], Saskena [68] and Srivastava [72].

Consider the directed graph $D = (V, A)$ obtained by replacing each edge $e = (i, j) \in E$ by the opposite arcs (i, j) and (j, i) in A with the same weight as the edge $(i, j) \in E$. The directed version of the Generalized Traveling Salesman Problem called the *asymmetric Generalized Traveling Salesman Problem* was introduced by Laporte, Mercure and Nobert [46], [47] and asks for a minimum cost oriented cycle that visits exactly one node from each cluster.

For more examples of generalized combinatorial optimization problems we refer the reader to Feremans [17] and Dror and Haouari [13].

2.2 Minimum Spanning Trees

The aim of this section is to describe the well-known algorithms of Kruskal and Prim for finding the minimum spanning tree.

Let $G = (V, E)$ be a connected graph with a positive cost function $c \geq 0$ defined on the edge set E.

Definition 2.1 *(Minimum spanning tree)*
A minimum spanning tree (MST) of G is a spanning tree T^* of G that has minimal cost, i.e.,

$$c(E(T^*)) = \min \{c(E(T)) \mid T \textit{ is a spanning tree of } G\} \tag{2.1}$$

□

Recall that a spanning tree is a connected subgraph of G that has no cycles and contains all nodes V.

We present two efficient algorithms for constructing minimum spanning trees, namely, Kruskal's algorithm [45] and Prim's algorithm [58].

Kruskal's algorithm is also called the *greedy algorithm*. Given a set of objects, the greedy algorithm attempts to find a feasible subset with minimum objective value by repeatedly choosing an object of minimum cost among the unchosen ones and adding it to the current subset provided the resulting subset is feasible. In particular, Kruskal's algorithm works by repeatedly choosing an edge of minimum cost among the edges not chosen so far, and adding this edge to the "current spanning forest" provided this does not create a cycle. The algorithm terminates when the current spanning forest becomes connected.

Kruskal's Minimum Spanning Tree Algorithm

Input: A connected graph $G = (V, E)$ with a positive cost function on the edges.

Output: Edge set $F \subset E$ of minimum spanning tree of G.

$F := \emptyset$; (F is the edge set of the current spanning forest)
linearly order the edges in E according to nondecreasing cost;
let the ordering be: $e_1, e_2, ..., e_{|E|}$;

for each edge e_i, $i = 1, 2, ..., |E|$, **do**
  if $F \cup \{e_i\}$ has no cycle
  then $F := F \cup \{e_i\}$; (add the edge to the current forest)
    if $|F| = |V| - 1$ **then stop** and **output** F; **end**;
  end; (if)
end; (for)

Theorem 2.1 *Kruskal's algorithm is correct and it finds a minimum spanning tree. Its running time is* $O(|V||E|)$.

Proof: See for example [55]. □

Remark 2.1 *By using appropriate data structures, the running time of the Kruskal's algorithm can be improved to* $O(|E| \log |V|)$. □

Prim's algorithm starts with a "current tree" T that consists of a single node. In each iteration, a minimum-cost edge in the "boundary" $\delta(V(T))$ of T is added to T, and this is repeated till T is a spanning tree.

Prim's Minimum Spanning Tree Algorithm

Input: A connected graph $G = (V, E)$ with a positive cost function on the edges.

Output: Minimum spanning tree $T = (S, F)$ of G.

$F := \emptyset$; (F is the edge set of the current tree T)
$S := \{v\}$, where v is an arbitrary node; (S is the node set of T)

while $S \neq T$ **do**
  among all the edges having exactly one end in S, find an edge $(i, j) \in E$ of minimum cost;
  $F := F \cup (i, j)$;
  $S := S \cup (\{i, j\} \setminus S)$; (add the end node in $V \setminus S$)
end; (while)

Theorem 2.2 *Prim's algorithm is correct and it finds a minimum spanning tree. Its running time is* $O(|V|^2)$.

Proof: See for example [55]. □

Remark 2.2 *By using the Fibonacci heaps data structure, the running time of the Prim's algorithm can be improved to* $O(|E| + |V| \log |V|)$. □

2.3 Definition of the GMST problem

The *Generalized Minimum Spanning Tree Problem* is defined on an undirected graph $G = (V, E)$ with the nodes partitioned into m node sets called *clusters*. Let $|V| = n$ and $K = \{1, 2, ..., m\}$ be the node index of the clusters. Then, $V = V_1 \cup V_2 \cup \ldots \cup V_m$ and $V_l \cap V_k = \emptyset$ for all $l, k \in K$ such that $l \neq k$. We assume that edges are defined only between nodes belonging to different clusters and each edge $e = (i, j) \in E$ has a nonnegative cost c_e.

The *Generalized Minimum Spanning Tree Problem*, denoted by GMST is the problem of finding a minimum cost tree spanning a subset of nodes which includes *exactly* one node from each cluster (see Figure 2.2 for a feasible solution of the GMST problem). We will call a tree containing exactly one node from each cluster a *generalized spanning tree*.

The GMST problem was introduced by Myung, Lee and Tcha [54]. Feremans, Labbé and Laporte [18] present several integer formulations of the GMST problem and compare them in terms of their linear programming relaxations, and in [19] they study the polytope associated with the GMST problem.

The *Minimum Spanning Tree Problem* is a special case of the GMST problem where each cluster has exactly one node. The MST problem can be solved by a polynomial time algorithm, for instance the algorithm of Kruskal [45] or the algorithm of Prim [58] as we have seen in the previous section. However, as we will show the GMST problem is $\mathcal{NP}$*-hard*.

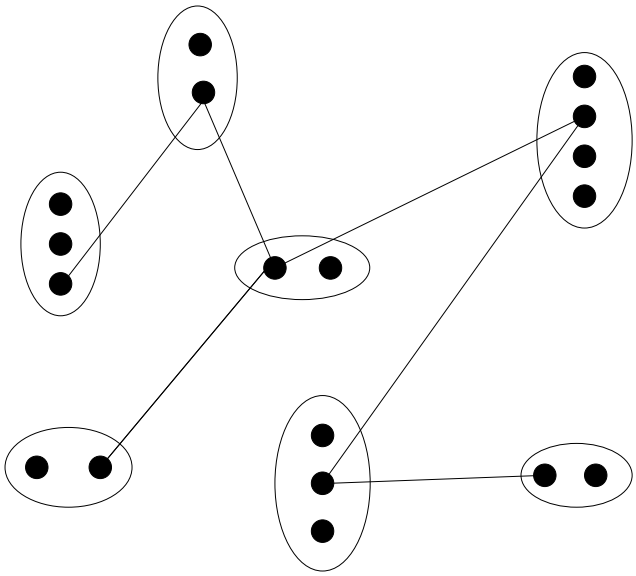

Figure 2.1: A feasible solution for the GMST problem

2.4 The GMST problem on trees

We present in this section a special case of the GMST problem: the case when the graph $G = (V, E)$ is a tree.

Garey and Johnson [21] have shown that for certain combinatorial optimization problems, the simple structure of trees can offer algorithmic advantages for efficiently solving them. Indeed, a number of problems that are $\mathcal{NP}$-*complete*, when are formulated on a general graph, become polynomially solvable when the graph is a tree. Unfortunately, this is not the case for the GMST problem. We will show that on trees the GMST problem is $\mathcal{NP}$-*hard*.

Let us consider the case when the graph $G = (V, E)$ is a tree. For any tree $T \subset G$ and node $i \in V$ set $y_i = 1$ if $i \in T$ and 0 otherwise. We will regard G as a rooted tree with root $r \in V$. Let $\pi(i)$ be the parent of node $i \in V \setminus \{r\}$. Since each edge joins some node i and its parent, we can set $e_i = (i, \pi(i))$ and $c_i = c_{e_i} = c_{(i,\pi(i))}$, for all $i \in V \setminus \{r\}$.

We can formulate the GMST problem on G with $\{r\} = V_1$ forming a cluster as the following integer linear program:

$$
\begin{aligned}
\min \quad & \sum_{i \in V} c_i y_i & & \\
s.t. \quad & y(V_k) = 1, & \forall\, k \in K = \{1, ..., m\} \quad & (2.2)\\
& y_i \leq y_{\pi(i)}, & \forall\, i \in V \setminus \{0\} \quad & (2.3)\\
& y_i \in \{0, 1\}, & \forall\, i \in V \quad & (2.4)
\end{aligned}
$$

Constraints (2.2) assure that from every cluster we select a node. Constraints (2.3) guarantee that the selected nodes form a tree.

The structure of this integer program is particulary simple because of the fact that graph G is a tree. The general case (see Chapter 3) is more complicated.

To show that the GMST problem on trees is $\mathcal{NP}$-*hard* we introduce the so-called *set cover problem* which is known to be $\mathcal{NP}$-*complete* (see [21]).

Given a finite set $X = \{x_1, ..., x_a\}$, a collection of subsets, $S_1, ..., S_b \subseteq X$ and an integer $k < |X|$, the *set cover problem* is to determine whether there exists a subset $Y \subseteq X$ such that $|Y| \leq k$ and

$$S_c \cap Y \neq \emptyset,\ \forall\, c \text{ with } 1 \leq c \leq b.$$

We call such a set Y a *set cover* for X.

After a discussion with G. Woeginger we came with the following result:

Theorem 2.3 *The Generalized Minimum Spanning Tree problem on trees is $\mathcal{NP}$-hard.*

Proof: In order to prove that the GMST problem on trees is *$\mathcal{NP}$-hard* it is enough to show that there exists an *$\mathcal{NP}$-complete* problem that can be polynomially reduced to GMST problem.

We consider the set cover problem for a given finite set $X = \{x_1, ..., x_a\}$, a collection of subsets of X, $S_1, ..., S_b \subseteq X$ and an integer $k < |X|$.

We show that we can construct a graph $G = (V, E)$ having a tree structure such that there exists a set cover $Y \subseteq X$, $|Y| \leq k$ if and only if there exists a generalized spanning tree in G of cost at most k.

The constructed graph G contains the following $m = a + b + 1$ clusters $V_1, ..., V_m$:

- V_1 consists of a single node denoted by r
- $V_2, ..., V_{a+1}$ node sets (corresponding to $x_1, x_2, ..., x_a \in X$) each of which has two nodes: one 'expensive' (see the construction of the edges) say $\overline{x_i}$ and one 'non-expensive' say $\hat{x_i}$, for $i = 2, ..., a$, and
- b node sets, $V_{a+2}, ..., V_m$ with $V_\nu = S_{\nu-(a+1)}$, for $\nu = a + 2, ..., m$.

Edges in G are constructed as follows:

(i) Each 'expensive node', say $\overline{x_t}$ of V_t for all $t = 2, ..., a + 1$, is connected with r by an edge of cost 1 and each 'non-expensive' node, say $\hat{x_t}$ of V_t for all $t = 2, ..., a + 1$, is connected with r by an edge of cost 0.

(ii) Choose any node $j \in V_t$ for any $t \in \{a + 2, ..., m\}$. Since $V_t \subset X$, then j coincides with a node in X, say $j = x_l$. We construct an edge between j and (the expensive node) $\overline{x_l} \in V_l$ with $l \in \{2, ..., a\}$. The cost of the edges constructed in this way is 0.

By construction the graph $G = (V, E)$ has a tree structure.

Suppose now that there exists a generalized spanning tree in G of cost at most k then by choosing

$$Y := \{x_l \in X \mid \text{the expensive vertex } \overline{x_l} \in V_{l+1} \text{ corresponding to } x_l \text{ is a vertex of the generalized spanning tree in } G\}$$

we see that Y is a set cover of X.

On the other hand, if there exists a set cover $Y \subseteq X$, $|Y| \leq k$ then according to the construction of G there exists a generalized spanning tree in G of cost at most k.

□

2.5 Complexity of the GMST problem

The following theorem due originally to Myung et al. [54] is an easy consequence of Theorem 2.3.

Theorem 2.4 *(Myung, Lee and Tcha [54])*
The Generalized Minimum Spanning Tree problem is $\mathcal{NP}$-hard.

□

Remark 2.3 To show that the GMST problem is *$\mathcal{NP}$-hard*, Myung, Lee and Tcha [54] used the so-called *node cover problem* which is known that is *$\mathcal{NP}$-complete* (see [21]) and showed that it can be polynomially reduced to GMST problem. Recall that given a graph $G = (V, E)$ and an integer $k < |V|$, the *node cover problem* is to determine whether a graph has a set C of at most k nodes such that all the edges of G are adjacent to at least one node of C. We call such a set C a *node cover* of G. □

2.6 Polynomially solvable cases of the GMST problem

As we have seen the GMST problem is *$\mathcal{NP}$-hard*. In this section we present some cases when the GMST problem can be solved in polynomial time.

A special case in which the GMST problem can be solved in polynomial time is the following:

Remark 2.4 If $|V_k| = 1$, for all $k = 1, ..., m$ then the GMST problem trivially reduces to the classical Minimum Spanning Tree problem which can be solved in polynomial time, by using for instance the algorithm of Kruskal or the algorithm of Prim, presented in the first section of this chapter. □

Another case in which the GMST problem can be solved in polynomial time is given in the following proposition:

Proposition 2.1 *If the number of clusters m is fixed then the GMST problem can be solved in polynomial time (in the number of nodes n).*

Proof: We present a polynomial time procedure based on dynamic programming which solves the GMST problem in this case.

We contract all the nodes from each cluster into one, resulting in a graph with vertex set $\{V_1, ..., V_m\}$ which we assume to be complete.

Given a global spanning tree, i.e. a tree which spans the clusters, we use dynamic programming in order to find the best (w.r.t. minimization of the cost) generalized spanning tree.

Fix an arbitrary cluster V_{root} as the root of the global spanning tree and orient all the edges away from vertices of V_{root} according to the global spanning tree.

The "subtree" rooted at a vertex v, $v \in V_k$ with $k \leq m$, denoted by $T(v)$ includes all the vertices reachable from v under this orientation of the edges. The *children* of v denoted by $C(v)$ are all those vertices u with a directed edge (v, u). Leaves of the tree have no children.

Let $W(T(v))$ denote the minimum weight of a generalized "subtree" rooted at v. We want to compute:

$$\min_{r \in V_{root}} W(T(r)).$$

We give now the dynamic programming recursion to solve the subproblem $W(T(v))$. The initialization is:

$$W(T(v)) = 0 \text{ if } v \in V_k \text{ and } V_k \text{ is a leaf of the global spanning tree.}$$

The recursion for $v \in V$ an interior vertex is then as follows:

$$W(T(v)) = \sum_{l, C(v) \cap V_l \neq \emptyset} \min_{u \in V_l} \{c(v,u) + W(T(u))\},$$

where by $c(v,u)$ we denoted the cost of the edge (v,u).

For computing $W(T(v))$, i.e. find the optimal solution of the subproblem $W(T(v))$, we need to look at all the vertices from the clusters V_l such that $C(v) \cap V_l \neq \emptyset$. Therefore for fixed v we have to check at most n vertices. So the overall complexity of this dynamic programming algorithm is $O(n^2)$, where $n = |V|$.

Notice that the above procedure leads to an $O(m^{m-2}n^2)$ time exact algorithm for GMST problem, obtained by trying all the global spanning trees, i.e. the possible trees spanning the clusters, where m^{m-2} represents the number of distinct spanning trees of a completely connected undirected graph of m vertices given by Cayley's formula [8].

□

2.7 Applications of the GMST problem

The following two applications of the GMST problem in the real world were described in [54]:

- Determining the location of the regional service centers.
  There are m market segments each containing a given number of marketing centers. We want to connect a number of centers by building links. The problem is to find a minimum cost tree spanning a subset of centers which includes exactly one from every market segment.

- Designing metropolitan area networks [24] and regional area networks [59].
  We want to connect a number of local area networks via transmission links such as optical fibers. In this case we are looking for a minimum cost tree spanning a subset of nodes which includes exactly one from each local network. Then, such a network design problem reduces to a GMST problem.

2.8 A variant of the GMST problem

A variant of the GMST problem is the problem of finding a minimum cost tree spanning a subset of nodes which includes *at least* one node from each cluster. In comparison with the GMST problem, in this case we define edges between all the nodes of V. We attach to each edge $e \in E$ a nonnegative cost c_e .

We denote this variant of the GMST problem by L-GMST problem (where L stands for Least) as in [17]. The L-GMST problem was introduced by Ihler, Reich and Widmayer [40] as a particular case of the *Generalized Steiner Tree Problem* under the name *Class Tree Problem*.

In Figure 2.2 we present a feasible solution of the L-GMST problem.

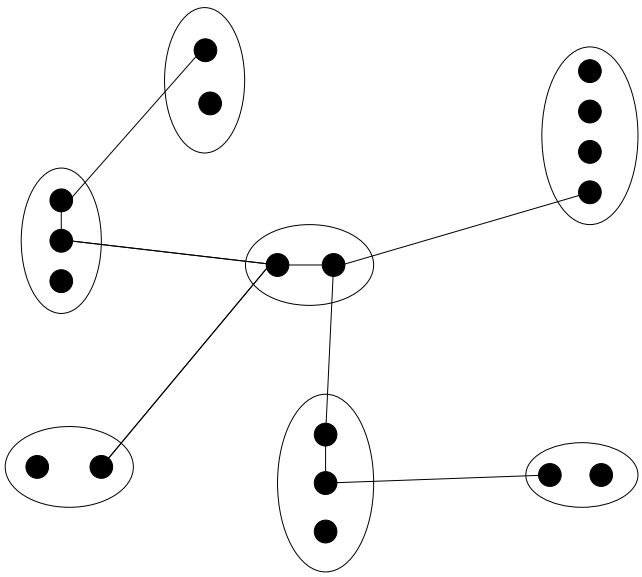

Figure 2.2: A feasible solution for the L-GMST problem

Remark 2.5 For the particular case when the costs of c_e of the edges $e = (i, j)$ with $i, j \in V_k$, $k \in \{1, ..., m\}$ are zero then L-GMST problem becomes the MST on the "contracted graph" with vertex set $\{V_1, ..., V_m\}$ and the cost of the (global) edges given by

$$C(V_k, V_l) = \min\{c(i, j) \mid i \in V_k, j \in V_l\},$$

for all $1 \leq k < l \leq m$. □

In [14], Dror, Haouari and Chaouachi provided five heuristic algorithms including a genetic algorithm for the L-GMST problem and an exact method is described in [17] by Feremans.

Dror, Haouari and Chaouachi in [14] used the L-GMST problem to solve an important real life problem arising in desert environments: given m parcels having a polygonal shape with a given number of vertices and a source of water the problem is to construct a minimal irrigation network (along edges of the parcels) such that each of the parcels has at least one irrigation source.

Chapter 3

Polyhedral aspects of the GMST Problem

In this chapter we describe different integer programming formulations of the GMST problem and we establish relationships between the polytopes of their linear relaxations.

In the first Section 3.1 we introduce some notations which are common to all formulations of the GMST problem. The next two sections contain formulations of the GMST problem with an exponential number of constraints: in Section 3.2 we present three formulations in the case of an undirected graph, two of them already introduced by Myung, Lee and Tcha [54] and the corresponding formulations for the directed graph are described in Section 3.3. The last formulations contain a polynomial number of constraints: we present different flow formulations in Section 3.4 and finally in Section 3.5 we describe a new formulation of the GMST problem based on distinguishing between *global* and *local* connections. Based on this formulation we present in Section 3.6 a solution procedure and computational results and we discuss the advantages of our new approach in comparison with earlier methods. A conclusion follows in Section 3.7. Roughly, this chapter can be found in Pop *et al.* [60] and [61].

3.1 Introduction

Given an undirected graph $G = (V, E)$, for all $S \subseteq V$ we define

$$E(S) = \{\, e = (i, j) \in E \mid i, j \in S\}$$

the subset of edges with the end nodes in S.

We consider the directed graph $G' = (V, A)$ associated with G obtained by replacing each edge $e = (i, j) \in E$ by the opposite arcs (i, j) and (j, i) in A with the same weight as the edge $(i, j) \in E$. We define

$$\delta^+(S) = \{\, (i, j) \in A \mid i \in S,\ j \notin S\}$$

the subset of arcs leaving the set S,

$$\delta^-(S) = \{\, (i, j) \in A \mid i \notin S,\ j \in S\}$$

the subset of arcs entering the set S, and

$$A(S) = \{\, (i, j) \in A \mid i, j \in S\}$$

the subset of arcs with the end nodes in S. For simplicity we write $\delta^-(i)$ and $\delta^+(i)$ instead of $\delta^-\{(i)\}$ and $\delta^+\{(i)\}$.

In order to model the GMST problem as an integer programming problem we define the following binary variables:

$$x_e = x_{ij} = \begin{cases} 1 & \text{if the edge } e = (i, j) \in E \text{ is included in the selected subgraph} \\ 0 & \text{otherwise} \end{cases}$$

$$z_i = \begin{cases} 1 & \text{if the node } i \text{ is included in the selected subgraph} \\ 0 & \text{otherwise} \end{cases}$$

$$w_{ij} = \begin{cases} 1 & \text{if the arc } (i, j) \in A \text{ is included in the selected subgraph} \\ 0 & \text{otherwise.} \end{cases}$$

We use the vector notations $x = (x_{ij})$, $z = (z_i)$, $w = (w_{ij})$ and the notations $x(E') = \sum_{\{i,j\} \in E'} x_{ij}$, for $E' \subseteq E$, $z(V') = \sum_{i \in V'} z_i$, for $V' \subseteq V$ and $w(A') = \sum_{(i,j) \in A'} w_{ij}$, for $A' \subseteq A$.

3.2 Formulations for the undirected problem

Let $G = (V, E)$ be an undirected graph. A feasible solution to the GMST problem can be seen as a cycle free subgraph with $m - 1$ edges, one node selected from every cluster and connecting all the clusters. Therefore the GMST problem can be formulated as the following 0-1 integer programming problem:

$$
\begin{aligned}
\min \quad & \sum_{e \in E} c_e x_e & & \\
s.t. \quad & z(V_k) = 1, & \forall\, k \in K = \{1, ..., m\} & \quad (3.1)\\
& x(E(S)) \leq z(S - i), & \forall\, i \in S \subset V, 2 \leq |S| \leq n - 1 & \quad (3.2)\\
& x(E) = m - 1 & & \quad (3.3)\\
& x_e \in \{0, 1\}, & \forall\, e \in E & \quad (3.4)\\
& z_i \in \{0, 1\}, & \forall\, i \in V. & \quad (3.5)
\end{aligned}
$$

For simplicity we used the notation $S - i$ instead of $S \setminus \{i\}$. In the above formulation, constraints (3.1) guarantee that from every cluster we select exactly one node, constraints (3.2) eliminate all the subtours and finally constraint (3.3) guarantees that the selected subgraph has $m - 1$ edges.

This formulation, introduced by Myung [54], is called the *generalized subtour elimination formulation* since constraints (3.2) eliminate all the cycles.

We denote the feasible set of the linear programming relaxation of this formulation by P_{sub}, where we replace the constraints (3.4) and (3.5) by $0 \leq x_e, z_i \leq 1$, for all $e \in E$ and $i \in V$.

We may replace the subtour elimination constraints (3.5) by connectivity constraints, resulting in the so-called *generalized cutset formulation* introduced in [54]:

$$
\begin{array}{ll}
\min & \displaystyle\sum_{e\in E} c_e x_e \\
s.t. & (3.1), (3.3), (3.4), (3.5) \text{ and} \\
& x(\delta(S)) \geq z_i + z_j - 1,\ \forall\, i \in S \subset V, j \notin S \qquad (3.6)
\end{array}
$$

where the set $\delta(S)$ was defined in Section 1.3.

We denote the feasible set of the linear programming relaxation of this formulation by P_{cut} .

Theorem 3.1 *The following properties hold:*
a) We have $P_{sub} \subset P_{cut}$.
b) The polyhedra P_{cut} and P_{sub} may have fractional extreme points.

Proof:
a) (See also [17]) Let $(x, z) \in P_{sub}$ and $i \in S \subset V$ and $j \notin S$. Since $E = E(S) \cup \delta(S) \cup E(V \setminus S)$, we get

$$
\begin{array}{rcl}
x(\delta(S)) & = & x(E) - x(E(S)) - x(E(V \setminus S)) \\
& \geq & z(V) - 1 - z(S) + z_i - z(V \setminus S) + z_j = z_i + z_j - 1.
\end{array}
$$

To show that the inclusion is strict we consider the following example provided by Magnanti and Wolsey in [51]:

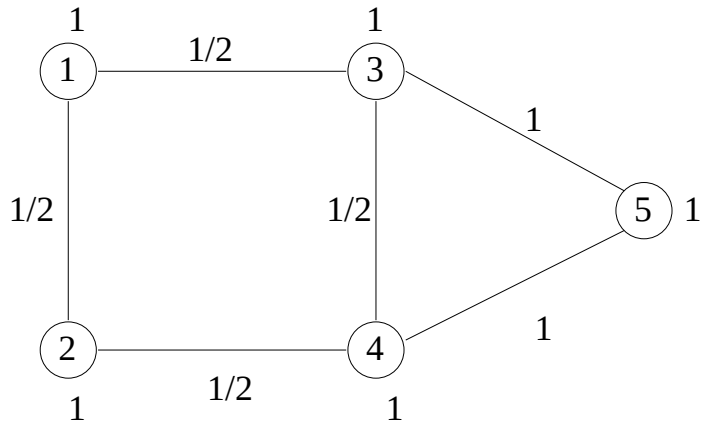

Figure 3.1: Example showing that $P_{sub} \subset P_{cut}$

In Figure 3.1 we consider five clusters $V_1, ..., V_5$ as singletons. The positive values x_{ij} and z_i are shown on the edges respectively on the nodes. It easy to see that constraints (3.1), (3.3) and (3.6) are satisfied, while (3.2) is violated for $S = \{3, 4, 5\}$.

b) To show that P_{cut} may have fractional extreme points we consider Figure 3.1 to have five singletons and we set the cost of the edges $\{1, 2\}$, $\{1, 3\}$ and $\{2, 4\}$ to 1 and the cost of the edges $\{3, 4\}$, $\{4, 5\}$ and $\{3, 5\}$ to 0. Then the cost of an optimal solution over the P_{cut} is $\frac{3}{2}$.

In comparison to the MST problem, the polyhedron P_{sub}, corresponding to the linear programming relaxation of the generalized subtour elimination formulation of the GMST problem, may have fractional extreme points. See Figure 3.2 for such an example.

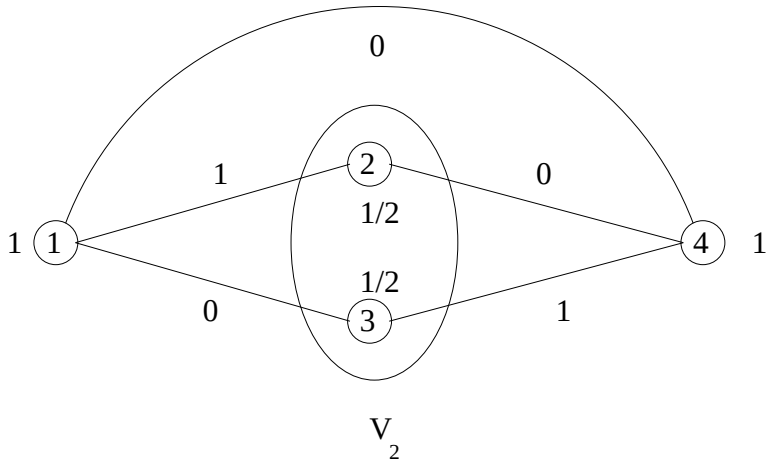

Figure 3.2: Example of a graph in which P_{sub} has fractional extreme points

In Figure 3.2 we consider the clusters V_1 and V_3 as singletons and the cluster V_2 to have two nodes. By setting the cost of the edges $\{1, 3\}$, $\{1, 4\}$ and $\{2, 4\}$ to 2 and the cost of the edges $\{1, 2\}$ and $\{3, 4\}$ to value 1, we get a fractional optimal solution over P_{sub} given by $x_{12} = x_{34} = 1$, $z_1 = z_4 = 1$, $z_2 = z_3 = \frac{1}{2}$ and all the other variables 0.

□

Our next model, the so-called *generalized multicut formulation*, is obtained by replacing simple cutsets by multicuts. Given a partition of the nodes $V = C_0 \cup C_1 \cup \ldots \cup C_k$, we define the multicut $\delta(C_0, C_1, ..., C_k)$ to be the set of edges connecting different C_i and C_j. The generalized multicut formulation for the GMST problem is:

$$\begin{aligned} \min \quad & \sum_{e \in E} c_e x_e \\ s.t. \quad & (3.1), (3.3), (3.4), (3.5) \text{ and} \\ & x(\delta(C_0, C_1, ..., C_k)) \geq \sum_{j=0}^{k} z_{i_j} - 1, \ \forall\, C_0, C_1, ..., C_k \text{ node partitions} \\ & \qquad\qquad \text{of } V \text{ and } \forall\, i_j \in C_j \text{ for } j = 0, 1, ..., k. \end{aligned} \tag{3.7}$$

Let P_{mcut} denote the feasible set of the linear programming relaxation of this model. Clearly, $P_{mcut} \subseteq P_{cut}$. Generalizing the proof in the case of the minimum spanning tree problem [51], we show that:

Proposition 3.1 $P_{sub} = P_{mcut}$.

Proof: The proof of $P_{sub} \subseteq P_{mcut}$ is analoguos to the proof of $P_{sub} \subseteq P_{cut}$.

Conversely, let $(x, z) \in P_{mcut}$, $i \in S \subset V$ and consider the inequality (3.7) with $C_0 = S$ and with $C_1, ..., C_k$ as singletons with union is $V \setminus S$. Then

$$x(\delta(S, C_1, ..., C_k)) \geq \sum_{j=0}^{k} z_{i_j} - 1 = z_i + z(V \setminus S) - 1,$$

where $i \in S \subset V$. Therefore

$$\begin{aligned} x(E(S)) &= x(E) - x(\delta(S, C_1, ..., C_k)) \\ &\leq z(V) - 1 - z_i - z(V \setminus S) + 1 = z(S - i). \end{aligned}$$

□

3.3 Formulations for the directed problem

Consider the directed graph $D = (V, A)$ obtained by replacing each edge $e = (i, j) \in E$ by the opposite arcs (i, j) and (j, i) in A with the same weight as the edge $(i, j) \in E$. The directed version of the GMST problem introduced by Myung et al. [54] and called *Generalized Minimum Spanning Arborescence problem* is defined on the directed graph $D = (V, A)$ rooted at a given cluster, say V_1 without loss of generality, and consists of determining a minimum cost arborescence which includes exactly one node from every cluster.

The two formulations that we are going to present in this section, were presented by Feremans *et al.* [18] under the names of directed cutset formulation, respectively directed subpacking formulation.

We consider first a *directed generalized cutset formulation* of the GMST problem. In this model we consider the directed graph $D = (V, A)$ with the cluster V_1 chosen as a root, without loss of generality, and we denote $K_1 = K \setminus \{1\}$.

$$
\begin{aligned}
\min \quad & \sum_{e \in E} c_e x_e & & \\
s.t. \quad & z(V_k) = 1, & \forall\, k \in K = \{1, ..., m\} & \\
& x(E) = m - 1 & & \\
& w(\delta^-(S)) \geq z_i, & \forall\, i \in S \subseteq V \setminus V_1 & (3.8) \\
& w_{ij} \leq z_i\ , & \forall\, i \in V_1, j \notin V_1 & (3.9) \\
& w_{ij} + w_{ji} = x_e\ , & \forall\, e = (i, j) \in E & (3.10) \\
& x, z, w \in \{0, 1\}. & & (3.11)
\end{aligned}
$$

In this model constraints (3.8) and (3.9) guarantee the existence of a path from the selected root node to any other selected node which includes only the selected nodes.

Let P_{dcut} denote the projection of the feasible set of the linear programming relaxation of this model into the (x, z)-space.

Another possible directed generalized cutset formulation considered by Myung *et al.* in [54], was obtained by replacing (3.3) with the following constraints:

$$
\begin{aligned}
w(\delta^-(V_1)) &= 0 & & (3.12) \\
w(\delta^-(V_k)) &\leq 1, \quad \forall\, k \in K_1. & & (3.13)
\end{aligned}
$$

To show the above equivalence it is enough to use the following result proved by Feremans, Labbé and Laporte [18].

Lemma 3.1 *(Feremans, Labbé and Laporte [18])*
a) Constraints (3.1), (3.3), (3.8) *and* (3.10) *imply*

$$\begin{aligned} w(\delta^-(i)) &= 0, \quad \forall\, i \in V_1 \\ w(\delta^-(i)) &= z_i, \quad \forall\, i \in V \setminus V_1 \end{aligned}$$

b) Constraints (3.1), (3.8), (3.10), (3.12) *and* (3.13) *imply* (3.3).

□

We introduced now a formulation of the GMST problem based on branchings. Consider, as in the previous formulation, the digraph $D = (V, A)$ with V_1 chosen as the cluster root. We define the *branching model* of the GMST problem to be:

$$\begin{aligned} \min \quad & \sum_{e \in E} c_e x_e \\ s.t. \quad & z(V_k) = 1, && \forall\, k \in K = \{1, ..., m\} \\ & x(E) = m - 1 \\ & w(A(S)) \le z(S - i), && \forall\, i \in S \subset V, 2 \le |S| \le n-1 \quad (3.14) \\ & w(\delta^-(V_k)) = 1\ , && \forall\, k \in K_1 \quad (3.15) \\ & w_{ij} + w_{ji} = x_e\ , && \forall\, e = (i,j) \in E \\ & x, z, w \in \{0, 1\}. \end{aligned}$$

Let P_{branch} denote the projection of the feasible set of the linear programming relaxation of this model into the (x, z)-space. Obviously, $P_{branch} \subseteq P_{sub}$.

The following result was established by Feremans [17]. We present a different proof of the proposition.

Proposition 3.2 *(Feremans [17])* $P_{branch} = P_{dcut} \cap P_{sub}$.

Proof: First we prove that $P_{dcut} \cap P_{sub} \subseteq P_{branch}$.

Let $(x, z) \in P_{dcut} \cap P_{sub}$. Using Lemma 3.1, it is easy to see that constraint (3.15) is satisfied. Therefore $(x, z) \in P_{branch}$.

We show that $P_{branch} \subseteq P_{dcut} \cap P_{sub}$.

It is obvious that $P_{branch} \subseteq P_{sub}$, therefore it remains to show $P_{branch} \subseteq P_{dcut}$.

Let $(x, z) \in P_{branch}$. For all $i \in V_1$ and $j \notin V_1$ take $S = \{i, j\} \subset V$. Then by (3.14) we have $w_{ij} + w_{ji} \leq z_i$ and this implies (3.9).

Now we show that $w(\delta^-(l)) \leq z_l$, for $l \in V_k, k \in K_1$.

Take $V^l = \{i \in V | (i, l) \in \delta^-(l)\}$ and $S^l = V^l \cup \{l\}$, then $w(\delta^-(l)) = w(A(S^l))$ and choose $i_l \in V^l$.

$$\begin{aligned} 1 &= \sum_{l \in V_k} w(\delta^-(l)) = \sum_{l \in V_k} w(A(S^l)) \leq \sum_{l \in V_k} z(S^l \backslash i_l) \\ &= \sum_{l \in V_k} z_l + \sum_{l \in V_k} \sum_{j \in V^l \backslash i_l} z_j = 1 + \sum_{l \in V_k} \sum_{j \in V^l \backslash i_l} z_j. \end{aligned}$$

Therefore, for all l there is only one $i_l \in V^l$ with $z_{i_l} \neq 0$ and

$$w(\delta^-(l)) = w(A(S^l)) \leq z(S^l \backslash i_l) = z_l.$$

For every $i \in S \subset V \backslash V_1$

$$w(A(S)) = \sum_{i \in S} w(\delta^-(i)) - w(\delta^-(S)) \leq z(S - i),$$

which implies that:

$$\begin{aligned} w(\delta^-(S)) &\geq \sum_{i \in S} w(\delta^-(i)) - z(S) + z_i \\ &= \sum_{i \in S} \left[1 - \sum_{l \in V_k \backslash \{i\}} w\left(\delta^-(l)\right) \right] - z(S) + z_i \\ &\geq \sum_{i \in S} \left[1 - \sum_{l \in V_k \backslash \{i\}} z_l \right] - z(S) + z_i = z(S) - z(S) + z_i = z_i. \end{aligned}$$

□

3.4 Flow based formulations

All the formulations that we have described so far have an exponential number of constraints. The formulations that we are going to consider next will have only a polynomial number of constraints but an additional number of variables. In order to give compact formulations of the GMST problem one possibility is to introduce 'auxiliary' flow variables beyond the natural binary edge and node variables.

We wish to send a flow between the nodes of the network and view the edge variable x_e as indicating whether the edge $e \in E$ is able to carry any flow or not. We consider three such flow formulations: a single commodity model, a multicommodity model and a bidirectional flow model. In each of these models, although the edges are undirected, the flow variables will be directed. That is, for each edge $(i, j) \in E$, we will have flow in the both directions i to j and j to i.

In the *single commodity model*, the source cluster V_1 sends one unit of flow to every other cluster. Let f_{ij} denote the flow on edge $e = (i, j)$ in the direction i to j. This leads to the following formulation:

$$
\begin{aligned}
\min \quad & \sum_{e \in E} c_e x_e & & \\
s.t. \quad & z(V_k) = 1, & & \forall\, k \in K = \{1, ..., m\} \\
& x(E) = m - 1 & & \\
& \sum_{e \in \delta^+(i)} f_e - \sum_{e \in \delta^-(i)} f_e = \begin{cases} (m-1)z_i & \text{for } i \in V_1 \\ -z_i & \text{for } i \in V \backslash V_1 \end{cases} & & (3.16) \\
& f_{ij} \le (m-1)x_e, & \forall\, e = (i, j) \in E & (3.17) \\
& f_{ji} \le (m-1)x_e, & \forall\, e = (i, j) \in E & (3.18) \\
& f_{ij}, f_{ji} \ge 0, & \forall\, e = (i, j) \in E & (3.19) \\
& x, z \in \{0, 1\}. & &
\end{aligned}
$$

In a discussion, Ravindra K. Ahuja drew my attention to this formulation of the GMST problem.

In this model, the mass balance equations (3.16) imply that the network defined by any solution (x, z) must be connected. Since the constraints (3.1)

and (3.3) state that the network defined by any solution contains $m-1$ edges and one node from every cluster, every feasible solution must be a generalized spanning tree. Therefore, when projected into the space of the (x,z) variables, this formulation correctly models the GMST problem.

We let P_{flow} denote the projection of the feasible set of the linear programming relaxation of this model into the (x,z)-space.

A stronger relaxation is obtained by considering multicommodity flows. This *directed multicommodity flow model* was introduced by Myung *et al.* in [54]. In this model every node set $k \in K_1$ defines a commodity. One unit of commodity k originates from V_1 and must be delivered to node set V_k. Letting f_{ij}^k be the flow of commodity k in arc (i,j) we obtain the following formulation:

$$
\begin{aligned}
\min \quad & \sum_{e \in E} c_e x_e & & \\
s.t. \quad & z(V_k) = 1, & & \forall\, k \in K = \{1,...,m\} \\
& x(E) = m-1 & & \\
& \sum_{a \in \delta^+(i)} f_a^k - \sum_{a \in \delta^-(i)} f_a^k = \begin{cases} z_i & , i \in V_1 \\ -z_i & , i \in V_k \\ 0 & , i \notin V_1 \cup V_k \end{cases} & & , k \in K_1 \quad (3.20) \\
& f_{ij}^k \le w_{ij}, & & \forall\, a=(i,j) \in A,\ k \in K_1 \quad (3.21) \\
& w_{ij} + w_{ji} = x_e\ , & & \forall\, e=(i,j) \in E \\
& f_a^k \ge 0, & & \forall\, a=(i,j) \in A,\ k \in K_1 \quad (3.22) \\
& x, z \in \{0,1\}. & &
\end{aligned}
$$

In [54], Myung *et al.* presented a branch and bound procedure to solve the GMST problem. The computational efficiency of such a procedure depends greatly upon how quickly it generates good lower and upper bounds. Their lower bounding procedure was based on the directed multicommodity flow model, since its linear programming relaxation not only provides a tight lower bound but also has a nice structure based on which an efficient dual ascent algorithm can be constructed. They developed also a heuristic algorithm which finds a feasible solution for the GMST problem using the obtained dual solution. Later we compare their numerical results with ours.

We let P_{mcflow} denote the projection of the feasible set of the linear programming relaxation of this model into the (x,z)-space.

Proposition 3.3 $P_{mcflow} \subseteq P_{flow}$.

Proof: Let $(w, x, z, f) \in P_{mcflow}, then$

$$0 \leq \sum_{k \in K_1} f_{ij}^k \leq |K_1| w_{ij} \leq (m-1) x_e.$$

With $f_{ij} = \sum_{k \in K_1} f_{ij}^k$ for every $e = (i, j) \in E$, we find

$$\begin{aligned} \sum_{e \in \delta^+(i)} f_a^k - \sum_{e \in \delta^-(i)} f_a^k &= \sum_{k \in K_1} \Big(\sum_{a \in \delta^+(i)} f_a^k - \sum_{a \in \delta^-(i)} f_a^k \Big) \\ &= \begin{cases} (m-1) z_i & for \quad i \in V_1 \\ -z_i & for \quad i \in V \backslash V_1. \end{cases} \end{aligned}$$

□

We obtain a closely related formulation by eliminating the variables w_{ij}. The resulting formulation consists of constraints (3.1), (3.3), (3.20), (3.22) plus

$$f_{ij}^h + f_{ij}^k \leq x_e \ , \ \forall \ h, k \in K_1 \text{ and } \forall \ e \in E \tag{3.23}$$

We refer to this model as the *bidirectional flow formulation* of the GMST problem and let P_{bdflow} denote its set of feasible solutions in (x, z)-space. Observe that since we have eliminated the variables w_a in constructing the bidirectional flow formulation, this model is defined on the undirected graph $G = (V, E)$, even though for each commodity k we permit flows f_{ij}^k and f_{ji}^k in both directions on edge $e = (i, j)$.

In the bidirectional flow formulation, constraints (3.23) which we are called the bidirectional flow inequalities, link the flow of different commodities flowing in different directions on the edge (i, j). These constraints model the following fact: in any feasible generalized spanning tree, if we eliminate edge (i, j) and divide the nodes in two sets; any commodity whose associated node lies in the same set as the root node set does not flow on edge (i, j); any two commodities whose associated nodes both lie in the set without the root both flow on edge (i, j) in the same direction. So, whenever two commodities h and k both flow on edge (i, j), they both flow in the same direction and so one of f_{ij}^h and f_{ij}^k equals zero.

Proposition 3.4 $P_{mcflow} = P_{bdflow}$.

Proof: If $(w, x, z, f) \in P_{mcflow}$, using (3.10) we have that

$$f_{ij}^h + f_{ji}^k \leq w_{ij} + w_{ji} = x_e \ , \ \ \forall \ e = (i, j) \in E \text{ and } \forall \ h, k \in K_1.$$

On the other hand, assume that $(x, z, f) \in P_{bdflow}$. By (3.23)

$$\max_h f_{ij}^h + \max_k f_{ji}^k \leq x_e \ \ \forall \ e = (i, j) \in E.$$

Hence we can choose w such that $\max_h f_{ij}^h \leq w_{ij}$ and $x_e = w_{ij} + w_{ji}$ for all $e = (i, j) \in E$. For example take

$$w_{ij} = \frac{1}{2}(x_e + \max_h f_{ij}^h - \max_k f_{ji}^k).$$

Clearly, $(w, x, z, f) \in P_{mcflow}$.

□

In Chapter 6, we present an algorithm based on a Lagrangian relaxation of the bidirectional flow formulation of the GMST problem in order to obtain "good" lower bounds.

We obtain another formulation, which we refer to as the *undirected multi-commodity flow model*, by replacing the inequalities (3.21) by the weaker constraints:

$$f_{ij}^k \leq x_e \ , \quad \text{for every } k \in K_1 \text{ and } e \in E.$$

3.5 Local-global formulation of the GMST problem

We present now a new formulation of the GMST problem. We shrink all the vertices from each cluster into one. Our formulation aims at distinguishing between *global*, i.e. inter-cluster connections, and *local* ones, i.e. connections between nodes from different clusters. We introduce variables y_{ij} $(i, j \in \{1, ..., m\})$ to describe the global connections. So $y_{ij} = 1$ if cluster V_i is connected to cluster V_j and $y_{ij} = 0$ otherwise and we assume that y represents a spanning tree. The convex hull of all these y-vectors is generally

known as the spanning tree polytope (on the contracted graph with vertex set $\{V_1, ..., V_m\}$ which we assume to be complete).

Following Yannakakis [78] this polytope, denoted by P_{MST}, can be represented by the following polynomial number of constraints:

$$\begin{aligned}
\sum_{\{i,j\}} y_{ij} &= m-1 \\
y_{ij} &= \lambda_{kij} + \lambda_{kji}, \text{ for } 1 \leq k,i,j \leq m \text{ and } i \neq j && (3.24)\\
\sum_{j} \lambda_{kij} &= 1, \qquad \text{for } 1 \leq k,i,j \leq m \text{ and } i \neq k && (3.25)\\
\lambda_{kkj} &= 0, \qquad \text{for } 1 \leq k,j \leq m && (3.26)\\
y_{ij}, \lambda_{kij} &\geq 0, \qquad \text{for } 1 \leq k,i,j \leq m.
\end{aligned}$$

where the variables λ_{kij} are defined for every triple of nodes k, i, j, with $i \neq j \neq k$ and their value for a spanning tree is:

$$\lambda_{kij} = \begin{cases} 1 \text{ if j is the parent of i when we root the tree at k} \\ \\ 0 \text{ otherwise.} \end{cases}$$

The constraints (3.26) mean that an edge (i, j) is in the spanning tree if and only if either i is the parent of j or j is the parent of i; the constraints (3.27) mean that if we root a spanning tree at k then every node other than node k has a parent and finally constraints (3.28) mean that the root k has no parent.

If the vector y describes a spanning tree on the contracted graph, the corresponding best (w.r.t. minimization of the costs) *local solution* $x \in \{0,1\}^{|E|}$ can be obtained by one of the following two methods described in the next subsections.

3.5.1 Local solution using dynamic programming

We use the same notations as in Proposition 2.1.

Given a global spanning tree, i.e. a tree which spans the clusters $\{V_1, ..., V_m\}$ we want to compute:

$$\min_{r \in V_{root}} W(T(r)),$$

where V_{root} is an arbitrary root of the global spanning tree and by $W(T(v))$ we denoted the minimum weight of the "subtree" rooted at v, $v \in V$.

The dynamic programming recursion which solves the subproblem $W(T(v))$ is the same as in Proposition 2.1.

Remember that for computing $W(T(v))$, i.e. find the optimal solution of the subproblem $W(T(v))$, we need to look at all the vertices from the clusters V_l such that $C(v) \cap V_l \neq \emptyset$. Therefore, for fixed v, we have to check at most n vertices. The overall complexity of the dynamic programming algorithm is $O(n^2)$, where $n = |V|$.

3.5.2 Local solution using integer linear programming

If the 0-1 vector y describes a spanning tree on the contracted graph, the corresponding local solution $x \in \{0,1\}^{|E|}$ that minimizes the costs can be obtained by solving the following integer linear programming problem:

$$
\begin{array}{lll}
\min & \displaystyle\sum_{e \in E} c_e x_e & \\
s.t. & z(V_k) = 1, & \forall\, k \in K = \{1, ..., m\} \\
& x(V_l, V_r) = y_{lr}, & \forall\, l, r \in K = \{1, ..., m\},\ l \neq r \\
& x(i, V_r) \leq z_i, & \forall\, r \in K, \forall\, i \in V \setminus V_r \\
& x_e, z_i \in \{0,1\}, & \forall\, e = (i,j) \in E,\ \forall\, i \in V,
\end{array}
$$

where $x(V_l, V_r) = \displaystyle\sum_{i \in V_l, j \in V_r} x_{ij}$ and $x(i, V_r) = \displaystyle\sum_{j \in V_r} x_{ij}$.

For given y, we denote the feasible set of the linear programming relaxation of this program by $P_{local}(y)$. The following result holds:

Proposition 3.5 *If y is the 0-1 incidence vector of a spanning tree of the contracted graph then the polyhedron $P_{local}(y)$ is integral.*

Proof: Suppose that the 0-1 vector y describes a spanning tree T of the contracted graph, then in order to prove that the polyhedron $P_{local}(y)$ is integral it is enough to show that every solution of the linear programming relaxation can be written as a convex combination of solutions corresponding to spanning trees: $(x, z) = \sum \lambda_T (x_T, z_T)$.

The proof is by induction on the support of x, denoted by $supp(x)$ and defined as follows:

$$supp(x) := \{x_e \mid x_e \neq 0,\ e \in E\}.$$

Suppose that there is a global connection between the clusters V_l and V_r (i.e. $y_{lr} = 1$) then

$$1 = x(V_l, V_r) = \sum_{i \in V_l} x(i, V_r) \leq \sum_{i \in V_l} z_i = 1,$$

which implies that $x(i, V_r) = z_i$.

We claim that $supp(x) \subseteq E$ contains a tree connecting all clusters. Assume the contrary and let $T^1 \subseteq E$ be a maximal tree in $supp(x)$. Since T^1 does not connect all clusters, there is some edge (l, r) with $y_{lr} = 1$ such that T^1 has some vertex $i \in V_l$ but no vertex in V_r. Then $z_i > 0$, and thus $x(i, V_r) = z_i > 0$, so T^1 can be extended by some $e = (i, j)$ with $j \in V_r$, a contradiction.

Now let x^{T^1} be the incidence vector of T^1 and let $\alpha := \min\{x_e \mid e \in T^1\}$. If $\alpha = 1$, then $x = x^{T^1}$ and we are done. Otherwise, let z^{T^1} be the vector which has $z_i^{T^1} = 1$ if T^1 covers $i \in V$ and $z_i^{T^1} = 0$ otherwise. Then

$$(\widehat{x}, \widehat{z}) := ((1-\alpha)^{-1}(x - \alpha x^{T^1}), (1-\alpha)^{-1}(z - \alpha z^{T^1}))$$

is again in $P_{local}(y)$ and, by induction, it can be written as a convex combination of tree solutions. The claim follows.

□

A similar argument shows that the polyhedron $P_{local}(y)$ is integral even in the case when the 0-1 vector y describes a cycle free subgraph in the contracted graph. If the 0-1 vector y contains a cycle of the contracted graph then $P_{local}(y)$ is in general not integral. In order to show this consider the following example:

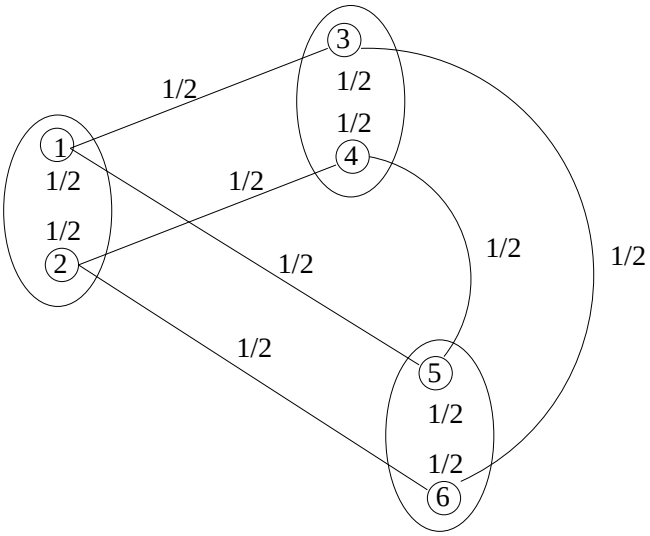

Figure 3.3: Example showing that $P_{local}(y)$ may have fractional extreme points

If the lines drawn in the above figure (i.e., $\{1,3\}$, $\{2,4\}$ etc.) have cost 1 and all the other lines (i.e., $\{1,4\}$, $\{2,3\}$ etc.) have cost $M \gg 1$, then $z \equiv \frac{1}{2}$ and $x \equiv \frac{1}{2}$ on the drawn lines is an optimal solution of $P_{local}(y)$, showing that the polyhedron $P_{local}(y)$ is not integral.

3.5.3 Local-global formulation of the GMST problem

The observations presented so far in Section 3.5 lead to our final formulation, called *local-global formulation* of the GMST problem as an 0-1 mixed integer programming problem, where only the global variables y are forced to be integral:

$$
\begin{array}{lll}
\min & \sum_{e \in E} c_e x_e & \\
s.t. & z(V_k) = 1, & \forall\, k \in K = \{1, ..., m\} \\
 & x(E) = m - 1 & \\
 & x(V_l, V_r) = y_{lr}, & \forall\, l, r \in K = \{1, ..., m\},\ l \neq r \\
 & x(i, V_r) \leq z_i, & \forall\, r \in K, \forall\, i \in V \setminus V_r \\
(P_0) \qquad & y_{ij} = \lambda_{kij} + \lambda_{kji}, & \forall\, 1 \leq k, i, j \leq m \text{ and } i \neq j \\
 & \sum_j \lambda_{kij} = 1, & \forall\, 1 \leq k, i, j \leq m \text{ and } i \neq k \\
 & \lambda_{kkj} = 0, & \forall\, 1 \leq k, j \leq m \\
 & \lambda_{kij} \geq 0, & \forall\, 1 \leq k, i, j \leq m \\
 & x_e, z_i \geq 0, & \forall\, e = (i, j) \in E,\ \forall\, i \in V \\
 & y_{lr} \in \{0, 1\}, & \forall\, 1 \leq l, r \leq m.
\end{array}
$$

This new formulation of the GMST problem was obtained by incorporating the constraints characterizing P_{MST}, with $y \in \{0, 1\}$, into $P_{local}(y)$.

In the next section we present a solution procedure for solving the GMST problem based on the local-global formulation and we report on our computational results for many instances of the problem.

3.6 Solution procedure and computational results

There are different ways to solve the GMST problem with the help of formulation (P_0). The first possibility is to consider the mixed integer program (P_0) and solve it directly (for example with CPLEX).

Secondly we considered the relaxation of (P_0) obtained by choosing randomly one cluster V_k and rooting the global tree only at the root k.

$$
\begin{array}{lll}
\min & \sum_{e \in E} c_e x_e & \\
s.t. & z(V_k) = 1, & \forall\, k \in K = \{1, ..., m\} \\
 & x(E) = m - 1 & \\
 & x(V_l, V_r) = y_{lr}, & \forall\, l, r \in K = \{1, ..., m\},\ l \neq r \\
 & x(i, V_r) \leq z_i, & \forall\, r \in K, \forall\, i \in V \setminus V_r \\
(P_0^k) & y_{ij} = \lambda_{kij} + \lambda_{kji}, & \forall\, 1 \leq k, i, j \leq m \text{ and } i \neq j,\ \underline{k \text{ fixed}} \\
 & \sum_j \lambda_{kij} = 1, & \forall\, 1 \leq k, i, j \leq m \text{ and } i \neq k,\ \underline{k \text{ fixed}} \\
 & \lambda_{kkj} = 0, & \forall\, 1 \leq k, j \leq m,\ \underline{k \text{ fixed}} \\
 & \lambda_{kij} \geq 0, & \forall\, 1 \leq k, i, j \leq m,\ \underline{k \text{ fixed}} \\
 & x_e, z_i \geq 0, & \forall\, e = (i, j) \in E,\ \forall\, i \in V \\
 & y_{lr} \in \{0, 1\}, & \forall\, 1 \leq l, r \leq m.
\end{array}
$$

If the optimal solution of this relaxation produces a generalized spanning tree, then we have given the optimal solution of the GMST problem. Otherwise we choose another root or add a second root and proceed in this way till we get the optimal solution of the GMST problem. We call this procedure the *rooting procedure*.

It turned out that the lower bounds computed by solving the linear programming relaxation of (P_0^k) are comparable with the lower bounds given in [54], but can be computed faster.

Upper bounds can be computed by solving $P_{local}(y)$ for certain 0-1 vector y corresponding to a spanning tree. (E.g., a minimum spanning tree w.r.t. the edge weights $d_{lr} = \min\{c_{ij} \mid i \in V_l, j \in V_r\}$.)

We compare the computational results for solving the problem using our rooting procedure with the computational results given by Myung *et al.* in [54].

According to the method of generating the edge costs, the problems generated are classified into three types:

- **structured Euclidean case**
- **unstructured Euclidean case**

- **non-Euclidean case**

For the instances in the structured Euclidean case m squares (clusters) are "packed in a square" and in each of these m clusters n_c nodes are selected randomly. The costs between nodes are the Euclidean distances between the nodes. So in this model the clusters can be interpreted as physical clusters. In the other models such an interpretation is not valid.

For the unstructured Euclidean case $n = mn_c$ nodes are generated randomly in $[0, 100]^2$ with costs given by the Euclidean distances. But then the clusters are choosen randomly among these points. Finally in the non-Euclidean model the edge costs are randomly generated on $[0, 100]$.

Our algorithms have been coded in C and compiled with a HP-UX cc compiler. For solving the linear and mixed integer programming problems we used CPLEX. The computational experiments were performed on a HP 9000/735 computer with a 125 Mhz processor and 144 Mb memory.

Tables 3.1, 3.2 and 3.3 show the computational results for the non-Euclidean, unstructured Euclidean and structured Euclidean problems.

Table 3.1: Computational Results for non-Euclidean problems
(average of five problems for each combination)

Pb. size		Rooting Procedure				Myung's results [54]		
m	n_c	LB/OPT	roots	iter.	CPU	LB/OPT	LB/UB	CPU
8	3	100	1	1	0.07	100	-	0.05
	4	100	1	1	0.09	96.1	-	0.57
	6	100	1	1	0.14	94.3	-	3.25
	10	100	1	1	0.69	94.9	-	17.68
10	3	100	1	1	0.10	89.1	-	0.05
	4	100	1	2	0.32	99.2	-	2.13
	6	100	1	1	0.76	87.8	-	3.25
	10	100	1	1	3.53	91.3	-	17.68
12	3	100	1	1	0.17	89.6	-	6.00
	4	100	1	1	0.21	92.8	-	7.63
	6	100	2	3	5.61	91.3	-	54.98
	10	100	1	2	14.57	-	-	-
15	3	100	1	1	0.29	89.0	-	17.45
	4	100	1	1	0.73	-	-	-
	6	100	1	1	5.99	-	86.2	7.65
	10	100	2	2	40.53	-	82.7	18.25
18	3	100	1	1	0.57	95.3	-	38.83
	6	100	1	1	9.45	-	-	-
	10	100	1	2	193.81	-	-	-
20	3	100	2	3	3.82	-	-	-
	6	100	1	1	1.46	-	81.4	13.7
	10	100	1	2	407.64	-	77.5	35.8
25	3	100	1	2	21.68	-	-	-
	6	100	1	1	25.12	-	80.5	44.4
	8	100	1	2	306.69	-	81.0	76.8
30	3	100	1	1	4.01	-	-	-
	6	100	2	2	84.05	-	74.3	45.2
	8	100	1	1	341.16	-	-	-
40	3	100	2	3	71.64	-	84.1	36.5
	4	100	1	2	1713.29	-	74.9	154.8

Table 3.2: Computational Results for structured Euclidean problems (average of five problems for each combination)

Pb. size		Rooting Procedure				Myung's results [54]		
m	n_c	LB/OPT	roots	iter.	CPU	LB/OPT	LB/UB	CPU
8	3	100	1	1	0.05	-	-	-
	4	100	2	3	0.09	-	-	-
	6	100	2	3	0.15	-	-	-
	10	100	2	3	0.85	-	-	-
10	3	100	2	3	0.06	-	-	-
	4	100	3	4	0.20	-	-	-
	6	100	3	4	0.54	-	-	-
	10	100	3	4	2.39	-	-	-
12	3	100	2	3	0.21	-	-	-
	4	100	3	4	0.42	-	-	-
	6	100	3	4	1.12	-	-	-
	10	100	3	4	5.43	-	-	-
15	3	100	3	4	0.72	-	-	-
	4	100	3	4	0.95	-	-	-
	6	100	4	5	3.80	-	-	-
	8	100	4	5	29.73	-	-	-
18	3	100	4	5	1.68	-	-	-
	6	100	4	5	18.25	-	-	-
	8	100	4	5	55.24	-	-	-
20	3	100	4	5	3.70	-	-	-
	6	100	5	6	20.18	-	-	-
	8	100	5	6	97.31	-	-	-
25	3	100	4	5	18.67	-	-	-
	6	100	4	5	198.63	-	-	-
	8	100	4	5	325.76	-	-	-
30	3	100	4	5	46.34	-	-	-
	4	100	4	5	51.33	-	-	-
	6	100	4	5	301.86	-	-	-
40	3	100	4	5	141.28	-	-	-
	4	100	4	5	283.72	-	-	-

Table 3.3: Computational Results for unstructured Euclidean problems (average of five problems for each combination)

Pb. size		Rooting Procedure				Myung's results [54]		
m	n_c	LB/OPT	roots	iter.	CPU	LB/OPT	LB/UB	CPU
8	3	100	2	3	0.08	93.4	-	0.51
	4	100	1	1	0.16	93.8	-	1.49
	6	100	2	3	0.59	94.1	-	1.78
	10	100	3	4	19.48	94.5	-	13.67
10	3	100	2	3	0.26	92.9	-	2.35
	4	100	2	3	0.43	91.4	-	3.44
	6	100	2	3	6.69	92.0	-	15.77
	10	100	3	4	30.23	90.3	-	95.14
12	3	100	2	3	0.31	95.1	-	2.81
	4	100	3	4	6.17	91.6	-	16.44
	6	100	3	4	17.95	91.1	-	35.42
	10	100	3	4	45.76	-	-	-
15	3	100	2	3	2.28	93.7	-	25.07
	4	100	3	4	60.35	90.5	-	104.46
	6	100	3	4	124.62	-	-	-
	8	100	3	4	205.73	-	86.2	7.29
18	3	100	4	5	15.12	-	-	-
	6	100	4	5	128.63	-	-	-
	8	100	4	5	259.24	-	-	-
20	3	100	4	5	80.92	-	-	-
	6	100	4	5	150.72	-	83.8	9.36
	8	100	4	5	397.31	-	80.8	21.67
25	3	100	4	5	98.63	-	-	-
	6	100	4	5	259.33	-	82.2	34.78
	8	100	4	5	477.86	-	74.6	23.46
30	3	100	4	5	136.34	-	-	-
	4	100	4	5	367.28	-	85.5	29.55
	6	100	4	5	509.36	-	81.0	52.17
40	3	100	4	5	277.39	-	88.2	22.44
	4	100	4	5	453.78	-	79.6	57.76

The first two columns in the tables give the number of clusters (m) and the number of nodes per cluster (n_c). The next four columns describe our rooting procedure and contain: (LB/OPT) the lower bounds obtained as a percentage of the optimal value of the GMST problem, (roots) the number of root clusters and (iter.) the number of mixed integer programs necessary to find the optimal solution and (CPU) the computation time in seconds for solving the GMST problem to optimality. The last columns give the numerical results reported by Myung *et al.* in [54]. The numerical results in [54] were obtained based on the dual of the linear programming relaxation of the multicommodity flow formulation. These columns evaluate the lower bounds obtained as a percentage of the optimal objective value, respectively the lower bounds obtained as a percentage of the upper bounds and finally the computation time is shown in the last column. In the last three columns '-' means that this information is not provided in [54].

As can be seen, in all the problems that we considered, the optimal solution of the GMST problem has been found by using our rooting procedure. These numerical experiences with the new formulation of the GMST problem are very promising.

The computational results so far could indicate that the exponential growth of the computation time for solving the GMST problem to optimality is moderate in comparison with the growth of computation time for solving the full problem directly, for example with CPLEX.

Finishing the thesis we became aware of the branch and cut numerical results provided by Feremans [17]. In comparison with our numerical results, in [17] in about 75% of the considered instances the optimal value of the GMST problem is obtained, but the computing times spent by the branch and cut code are in general larger than the times spent by the rooting procedure code.

3.7 Concluding remarks

We have provided several integer and mixed integer programming formulations for the GMST problem and established relationships between the polytopes of their linear relaxations.

Based on one of the new formulations that we introduced, i.e. the local-global formulation of the GMST problem, we present a new solution procedure called rooting procedure. Computational results show that the rooting

procedure performs better in comparison with earlier methods. For all the instances of the problem that we considered, the optimal solution of the GMST problem has been found by using our solution procedure.

Chapter 4

Approximation Algorithms

Throughout this chapter we will distinguish between so-called *positive results* which establish the existence of some approximation algorithms, and so-called *negative results* which disprove the existence of good approximation results for some combinatorial optimization problems under the assumption that $\mathcal{P} \neq \mathcal{NP}$.

We start with some basic results of the approximation theory. In Section 4.2 we focus on positive results in the area of approximation: design and analysis of polynomial time approximation algorithms. We present in Section 4.3 an in-approximability result for the GMST problem. Finally, in the last section of this chapter, under special assumptions, we give an approximation algorithm for the GMST problem with bounded cluster size. Roughly, the last section can be found in Pop, Kern and Still [62].

The basic results presented in the first two sections follow the descriptions of Goemans [27] and Schuurman and Woeginger [70].

4.1 Introduction

Many of the optimization problems we would like to solve are $\mathcal{NP}$*-hard*. Therefore it is very unlikely that these problems could be solved by a polynomial-time algorithm. However, these problems still have to be solved in practice. In order to do that, we have to relax some of the requirements. There are in general, three different possibilities.

- **Superpolynomial-time algorithms:** Even though an optimization problem is $\mathcal{NP}$*-hard*, there are "good" and "not so good" algorithms for solving it exactly. To the "not so good" algorithms certainly belong most of the simple enumeration methods where one would enumerate all the feasible solutions and then choose the one with the optimal value of the objective function. Such methods have very high time complexity. Among the "good" algorithms belong methods like *branch-and-bound* where an analysis of the problem at hand is used to discard most of the feasible solutions before they are even considered. These approaches allow one to obtain exact solutions of reasonably large problem instances, but their running time still depends exponentially on the size of the problem.

- **Average-case polynomial-time algorithms:** For some problems, it is possible to have algorithms which require superpolynomial-time on only a few instances and for the other instances run in polynomial time. A famous example is the *Simplex Method* for solving problems in Linear Programming.

- **Approximation Algorithms:** We may also relax the requirement of obtaining an exact solution of the optimization problem and content ourselves with a solution which is "not too far" from the optimum. This is partially justified by the fact that, in practice, it is usually enough to obtain a solution that is slightly sub-optimal.

Clearly, there are good approximation algorithms and bad ones as well. What we need is some means of determining the quality of an approximation algo-

rithm and a way of comparing different algorithms. There are a few criteria to consider:

Average-case performance: One has to consider some probability distribution on the set of all possible instances of a given problem. Based on this assumption, an expectation of the performance can then be found. Results of this kind strongly depend on the choice of the initial distribution and do not provide us any information about the performance on a particular instance.

Experimental performance: This approach is based on running the algorithm on a few "typical" instances. It has been used mostly to compare performance of several approximation algorithms. Of course the result depend on the choice of the "typical" instances and may vary from experiment to experiment.

Worst-case performance: This is usually done by establishing upper and lower bounds for approximate solutions in terms of the optimum value. In case of minimization problems, we try to establish upper bounds, in case of maximization problems, one wants to find lower bounds.

The advantage of the worst-case bounds on the performance of approximation algorithms is the fact that given any instance of the optimization problem, we are guaranteed that the approximate solution stays within these bounds. It should also be noted that approximation algorithms usually output solutions much closer to the optimum than the worst-case bounds suggest. Thus it is of independent interest to see how tight the bounds on the performance of each algorithm are; that is, how bad the approximate solution can really get. This is usually done by providing examples of specific instances for which the approximate solution is very far from the optimum solution.

Establishing worst-case performance bounds for even simple algorithms often requires a very deep understanding of the problem at hand and the use of powerful theoretical results from areas like linear programming, combinatorics, graph theory, probability theory, etc.

We consider an $\mathcal{NP}$*-hard* optimization problem for which as we have seen it is difficult to find the exact optimal solution within polynomial time. At the expense of reducing the quality of the solution by relaxing some of the requirements, we can often get considerable speed-up in the complexity. This leads us to the following definition:

Definition 4.1 *(Approximation algorithms)*
Let X be a minimization problem and $\alpha > 1$. An algorithm APP is called

an α-approximation algorithm for problem X, if for all instances I of X it delivers in polynomial time a feasible solution with objective value $APP(I)$ such that

$$APP(I) \leq \alpha OPT(I), \tag{4.1}$$

where by $APP(I)$ and $OPT(I)$ we denoted the values of an approximate solution and that of an optimal solution for instance I, respectively.

□

The value α is called the *performance guarantee* or the *worst case ratio* of the approximation algorithm APP. The closer α is to 1 the better the algorithm is.

4.2 Positive Results: the Design of the Approximation Algorithms

Positive results in the area of approximation concern the design and analysis of polynomial time approximation algorithms. This section concentrates on such positive results; we will outline the main strategy.

Assume that we need to find an approximation algorithm for an *$\mathcal{NP}$-hard* optimization problem X. How shall we proceed? We will concentrate on minimization problems, but the ideas apply equally well to maximization problems.

In what follows, given an instance I of an optimization problem and some algorithm for its approximation, we will denote by $OPT = OPT(I)$ the value of an optimum solution and by $APP = APP(I)$ the value of the solution output by the approximation algorithm. We are interested in an α-approximation algorithm, i.e. an algorithm APP such that

$$APP \leq \alpha OPT.$$

Relating APP to OPT directly can be difficult. Therefore instead we try to find a lower bound denoted by LB for the optimal solution satisfying the following conditions:

$$1.\ LB \leq OPT$$

$$2.\ APP \leq \alpha LB.$$

Consider now the following optimization problem:

$$OPT = \min_{x \in S} f(x).$$

A lower bound on OPT can be obtained by so-called *relaxation*. Consider the following related optimization problem:

$$LB = \min_{x \in R} g(x).$$

Definition 4.2 *(Lower bound)*

LB is called a lower bound on OPT and the optimization problem is called a relaxation of the original problem if the following two conditions hold:

$$\begin{aligned} S &\subseteq R \\ g(x) &\leq f(x) \text{ for all } x \in S. \end{aligned}$$

□

The above conditions imply the following:

$$LB = \min_{x \in R} g(x) \leq \min_{x \in S} f(x) = OPT.$$

showing that LB is a lower bound on OPT.

Many combinatorial optimization problems can be formulated as integer or mixed integer programs and the linear programming relaxations of this programs provide a lower bound for the optimal solution. Later we show how to use a linear programming relaxation of an integer programming formulation of the GMST problem to get an α-approximation algorithm for the GMST problem with bounded cluster size.

There are two main techniques to derive an approximately optimal solution from a solution of the relaxed problem:

1. **Rounding**

The idea is to solve the linear programming relaxation and then convert the fractional solution obtained into an integral solution. The approximation guarantee is establish by comparing the costs of the integral and fractional solutions.

Therefore we find an optimal solution x^* to the relaxation and "round" $x^* \in R$ to an element $x^{'} \in S$. Then prove that $f(x^{'}) \leq \alpha g(x^*)$ which implies that

$$f(x^{'}) \leq \alpha LB \leq \alpha OPT.$$

Often randomization is helpful. In this case $x^* \in R$ is randomly rounded to some element $x^{'} \in S$ so that $E[f(x^{'})] \leq \alpha g(x^*)$. These algorithms can sometimes be derandomized, in which case one finds an $x^{''}$ such that $f(x^{''}) \leq E[f(x^{'})]$.

An example where the rounding technique is applied, is the Minimum Weight Vertex Cover Problem, result due to Hochbaum [38].

2. **Primal-Dual**

We consider the linear programming relaxation of the primal program. This method constructs iteratively an integer solution to the primal program and a feasible solution to the dual program, $max\{h(y) : y \in D\}$. The approximation guarantee is establish by comparing the two solutions.

The weak duality property of the relaxation implies:

$$max\{h(y) : y \in D\} \leq min\{g(x) : x \in R\}$$

We construct $x \in S$ from $y \in D$ such that:

$$f(x) \leq \alpha h(y) \leq \alpha h(y_{max}) \leq \alpha g(x_{min}) \leq \alpha OPT.$$

Notice that y can be any element of D, not necessarily an optimal solution to the dual.

More details about primal-dual technique can be found in [1] and [75].

4.3 A negative result for the GMST problem

For some hard combinatorial optimization problems it is possible to show that they don't have an approximation algorithm unless $\mathcal{P} = \mathcal{NP}$. In order

to give a result of this form it is enough to show that the existence of an α-approximation algorithm would allow one to solve some decision problem, known to be $\mathcal{NP}$*-complete*, in polynomial time.

Applying this scheme to the GMST problem we obtain an in-approximability result. This result is a different formulation in terms of approximation algorithms of a result provided by Myung *et al.* [54] which says that even finding a near optimal solution for the GMST problem is $\mathcal{NP}$*-hard*. Our proof is slightly different to the proof provided in [54].

Theorem 4.1 *Under the assumption $\mathcal{P} \neq \mathcal{NP}$, there is no α-approximation algorithm for the GMST problem.*

Proof: Assume that there exists an α-approximation algorithm APP for the GMST problem, where α is a real number greater than or equal to 1. This means that

$$APP(I) \leq \alpha OPT(I),$$

for every instance I, where $OPT(I)$ and $APP(I)$ are the values of the optimal solution and of the solution found by the algorithm APP, respectively.

Then, we will show that APP also solves the node-cover problem for a given graph $G = (V, E)$ and an integer k such that $k < |V|$. This result contradicts the assumption that $\mathcal{P} \neq \mathcal{NP}$.

We construct a graph $G^{'} = (V^{'}, E^{'})$ and the edge cost function such that the algorithm APP finds a feasible solution with a value no greater than α times the optimal cost if and only if G contains C, where C is a node cover of G, i.e. a subset of V such that all the edges of G are adjacent to at least one node of C.

The graph $G^{'}$ contains the following $m = |E| + k + 1$ node sets (clusters), $V_1^{'}, ..., V_m^{'}$:

- $V_1^{'}$ consists of a single node denoted by r,
- $V_2^{'}, ..., V_{k+1}^{'}$ are identical node sets, each of which has $|V|$ nodes corresponding to the nodes of V, and
- $|E|$ node sets, $V_{k+2}^{'}, ..., V_m^{'}$, each of which contains a single node corresponding to an edge $e \in E$.

Edges in $G^{'}$ are constructed as follows:

(i) Each node of $V_t^{'}$ for all $t = 2, ..., k+1$ is connected to r by an edge. The set consisting of these edges is denoted by $E_1^{'}$.

(ii) Let i be a node of $V_t^{'}$ for any $t \in \{2, ..., k+1\}$ and j be a node of $V_t^{'}$ for any $t \in \{k+2, ..., m\}$. Then, an edge is constructed between i and j if the edge of G corresponding to j is incident to the node of G corresponding to i, and let $E_2^{'}$ denote the set of those edges.

(iii) We also construct an edge between i and j even though the edge of G corresponding to j is not incident to the node of G corresponding to i, and we let $E_3^{'}$ denote the set of those edges.

We let $E^{'} = E_1^{'} \cup E_2^{'} \cup E_3^{'}$.

The cost of each edge is defined as follows:

$$c_{ij} = \begin{cases} 0 & \text{for all } i,j \in E_1^{'} \\ 1 & \text{for all } i,j \in E_2^{'} \\ (|E|+1)\alpha & \text{for all } i,j \in E_3^{'}. \end{cases}$$

We claim that G contains C if and only if

$$APP(I) \leq \alpha |E|,$$

where instance I corresponds to $G^{'}$ and its cost function.

Note that there always exists a generalized spanning tree in $G^{'}$: all the clusters different from the identical clusters $V_2^{'}, ..., V_{k+1}^{'}$ have only one node and if we select k nodes from $V_2^{'}, ..., V_{k+1}^{'}$, one node from each cluster such that each node of C is included, then these k nodes together with the remaining nodes selected always uniquely form a generalized spanning tree of $G^{'}$ using edges in $E_1^{'} \cup E_2^{'}$, by the definition of $G^{'}$.

Suppose now that $G^{'}$ contains a generalized spanning tree and let C be a set of distinct nodes selected from the clusters, $V_2^{'}, ..., V_{k+1}^{'}$ in the tree, then C is a node cover of G.

Therefore, we showed that a generalized spanning tree only using edges in $E_1^{'} \cup E_2^{'}$ exists in $G^{'}$ if and only if G contains a node cover C.

If G contains C, then $OPT(I) = |E|$. Moreover, if G does not contain C, then any generalized spanning tree of G' should use at least one edge in E_3' and thus

$$OPT(I) \geq |E| - 1 + \alpha(|E| + 1) > \alpha|E|.$$

In particular if G contains a node cover C then the approximation algorithm APP will produce a solution with value

$$APP(I) \leq \alpha|E| = \alpha OPT(I),$$

i.e. the solution does not use any edge from E_3' and APP identifies a node cover.

□

However, under further assumptions, in the next section we present a positive result for the GMST problem.

4.4 An Approximation Algorithm for Bounded Cluster Size

As we have seen in the previous section there exists no α-approximation algorithm for the GMST problem under the assumption $\mathcal{P} \neq \mathcal{NP}$.

However under the following assumptions:

A1: the graph has bounded cluster size, i.e. $|V_k| \leq \rho$, for all $k = 1, ...m$

A2: the cost function is strict positive and satisfies the triangle inequality, i.e. $c_{ij} + c_{jk} \geq c_{ik}$ for all $i, j, k \in V$,

a polynomial approximation algorithm for the GMST problem is possible.

In this section under the above assumptions we present an approximation algorithm for the GMST problem with performance ratio 2ρ. The approximation algorithm is constructed following the ideas of Slavik [71] where the Generalized Traveling Salesman Problem and Group Steiner Tree Problem have been treated.

4.4.1 An integer programming formulation of the GMST problem

We consider the following integer programming formulation of the GMST problem:

Problem IP1:

$$
\begin{aligned}
Z_1 = \quad & \min \quad \sum_{e \in E} c_e x_e \\
& s.t. \quad z(V_k) = 1, && \forall\, k \in K = \{1, ..., m\} && (4.2)\\
& \qquad x(\delta(S)) \geq z_i, && \forall\, i \in S,\ \forall S \subset V \text{ such that for} && (4.3)\\
& && \text{some cluster } V_k, \text{ with } k \in K,\ S \cap V_k = \emptyset \\
& \qquad x(E) = m - 1 && && (4.4)\\
& \qquad x_e \in \{0, 1\}, && \forall\, e \in E && (4.5)\\
& \qquad z_i \in \{0, 1\}, && \forall\, i \in V. && (4.6)
\end{aligned}
$$

We used here the standard notations introduced in Section 3.1.

In the above formulation condition (4.2) guarantees that a feasible solution contains exactly one vertex from every cluster. Condition (4.3) guarantees that any feasible solution induces a connected subgraph. Condition (4.4) simply assures that any feasible solution has $m - 1$ edges and due to the fact that the cost function is strictly positive this constraint is redundant.

Consider now the linear programming relaxation of the integer programming formulation of the GMST problem. In order to do that, we simply replace conditions (4.5) and (4.6) in IP1 by new conditions:

$$
\begin{aligned}
0 \ \leq \ x_e \ \leq \ 1, \qquad & \text{for all } e \in E, && (4.7)\\
0 \ \leq \ z_i \ \leq \ 1, \qquad & \text{for all } i \in V. && (4.8)
\end{aligned}
$$

4.4.2 An Approximation Algorithm for the GMST problem

We assume that the assumptions A1 and A2 hold.

The algorithm for approximating the optimal solution of the GMST problem is as follows:

Algorithm "Approximate the GMST problem"

Input: A complete graph $G = (V, E)$ with strictly positive cost function on the edges satisfying the triangle inequality, and with the nodes partitioned into the clusters $V_1, ..., V_m$ with bounded size, $|V_k| \leq \rho$.

Output: A tree $T \subset G$ spanning some vertices $W^{'} \subset V$ which includes exactly one vertex from every cluster, that approximates the optimal solution to the GMST problem.

1. Solve the linear programming relaxation of the problem IP1 and let $(z^*, x^*, Z_1^*) = ((z_i^*)_{i=1}^n, (x_e^*)_{e \in E}, Z_1^*)$ be the optimal solution.
2. Set $W^* = \left\{ i \in V | z_i^* \geq \frac{1}{\rho} \right\}$ and consider $W^{'} \subset W^*$, with the property that $W^{'}$ has exactly one vertex from each cluster, and find a minimum spanning tree $T \subset G$ on the subgraph $G^{'}$ generated by $W^{'}$.
3. Output $APP = cost(T)$ and the generalized spanning tree T.

Even though the linear programming relaxation of the problem IP1 has exponentially many constraints, it can still be solved in polynomial time either using the ellipsoid method with a min-cut max-flow oracle [32] or using Karmarkar's algorithm [42] since the linear programming relaxation can be formulated "compactly" (the number of constraints polynomially bounded) using flow variables see Section 3.4.

4.4.3 Auxiliary results

We start this subsection with a result established by Goemans and Bertsimas [29] and called *parsimonious property*.

Given a complete undirected graph $G = (V, E)$. We associate with each edge $(i, j) \in E$ a cost c_{ij} and for any pair (i, j) of vertices, let r_{ij} be the connectivity requirement between i and j (r_{ij} is assumed to be symmetric, i.e. $r_{ij} = r_{ji}$). A network is called *survivable* if it has at least r_{ij} edge disjoint paths between any pair (i, j) of vertices.

The *survivable network design problem* consists in finding the minimum cost

survivable network. This problem can be formulated by the following integer program:

$$
\begin{aligned}
(IP_\emptyset(r)) \quad IZ_\emptyset(r) = \quad \min \quad & \sum_{e\in E} c_e x_e \\
s.t. \quad & x(\delta(S)) \geq \max_{(i,j)\in\delta(S)} r_{ij},\ S \subset V,\ S \neq \emptyset \quad (4.9)\\
& 0 \leq x_e, \qquad e \in E, \\
& x_e \text{ integral}, \qquad e \in E.
\end{aligned}
$$

We denote by $IZ_\emptyset(r)$ the optimal value of the above integer program. Let $(P_\emptyset(r))$ denote the linear programming relaxation of $(IP_\emptyset(r))$ obtained by dropping the integrality restrictions and let $Z_\emptyset(r)$ be its optimal value.

By definition the degree of vertex $i \in V$ is $d_x(i) = x(\delta(i))$, for any feasible solution x, either to $(IP_\emptyset(r))$ or to $(P_\emptyset(r))$. Because of constraints (4.9) for $S = \{i\}$, the degree of vertex i is at least equal to $\max_{j\in V\setminus\{i\}} r_{ij}$. If $d_x(i) = \max_{j\in V\setminus\{i\}} r_{ij}$, then we say that x is *parsimonious* at vertex i. If we impose that the solution x is parsimonious at all the vertices of a set $D \subseteq V$, we get some interesting variations of $(IP_\emptyset(r))$ and $(P_\emptyset(r))$, denoted by $(IP_D(r))$ and $(P_D(r))$ respectively. The formulation of $(IP_D(r))$ as an integer program is:

$$
\begin{aligned}
(IP_D(r)) \quad IZ_D(r) = \quad \min \quad & \sum_{e\in E} c_e x_e \\
s.t. \quad & x(\delta(S)) \geq \max_{(i,j)\in\delta(S)} r_{ij},\ S \subset V,\ S \neq \emptyset \\
& x(\delta(i)) = \max_{j\in V\setminus\{i\}} r_{ij},\ i \in D,\ D \subseteq V \\
& 0 \leq x_e, \qquad e \in E \\
& x_e \text{ integral}, \qquad e \in E.
\end{aligned}
$$

We denote by $IZ_D(r)$ the optimal value of the above integer program. Let $(P_D(r))$ denote the linear programming relaxation of $(IP_D(r))$ obtained by dropping the integrality restrictions and let $Z_D(r)$ be its optimal value.

Theorem 4.2 *(parsimonious property, Goemans and Bertsimas [29]) If the costs c_{ij} satisfy the triangle inequality, then*

$$Z_\emptyset(r) = Z_D(r)$$

for all subsets $D \subseteq V$. □

The proof of this theorem is based on a result on connectivity properties of Eulerian multigraphs, due to Lovász [50].

Let now $W \subset V$ and consider the following linear program:

Problem LP2:

$$\begin{aligned} Z_2^*(W) = \quad & \min \quad \sum_{e \in E} c_e x_e \\ & s.t. \quad x(\delta(S)) \geq 1, \ S \subset V, \text{ s.t. } W \cap S \neq \emptyset \neq W \setminus S \quad (4.10) \\ & \qquad x(\delta(i)) = 0, \ \ i \in V \setminus W, \quad (4.11) \\ & \qquad 0 \leq x_e \leq 1, \ \ e \in E. \quad (4.12) \end{aligned}$$

Replacing constraints (4.12) with the integrality constraints $x_e \in \{0, 1\}$, the formulation obtained is the formulation of the minimum tree spanning the subset of nodes $W \subset V$.

Consider the following relaxation of the problem LP2.

Problem LP3:

$$\begin{aligned} Z_3^*(W) = \quad & \min \quad \sum_{e \in E} c_e x_e \\ & s.t. \quad x(\delta(S)) \geq 1, \ S \subset V, \text{ s.t. } W \cap S \neq \emptyset \neq W \setminus S \\ & \qquad 0 \leq x_e, \qquad e \in E. \quad (4.13) \end{aligned}$$

Thus we omitted constraint (4.11) and relaxed constraint (4.12).

The following result is a straightforward consequence of the parsimonius property, if we choose $r_{ij} = 1$, if $i, j \in W$, and 0 otherwise, and $D = V \setminus W$.

Lemma 4.1 *The optimal solution values to problems LP2 and LP3 are the same, that is*

$$Z_2^*(W) = Z_3^*(W).$$

□

Consider the following problem:

Problem IP4:

$$
\begin{aligned}
Z_4 = \quad & \min \quad \sum_{e \in E} c_e x_e \\
& s.t. \quad x(\delta(S)) \geq 1, \; S \subset V, \text{ s.t. } S \neq \emptyset \neq V \qquad (4.14) \\
& \qquad x_e \in \{0,1\}, \qquad e \in E. \qquad (4.15)
\end{aligned}
$$

Clearly, it is the integer programming formulation of the MST (minimum spanning tree) problem. Let LP4 be the LP relaxation of this formulation, that is, we simply replace the constraint (4.15) by the constraint $0 \leq x_e \leq 1$, for all $e \in E$.

Denote by Z_4^* the value of the optimal solution of the LP4. The following known result for minimum spanning trees holds:

Proposition 4.1

$$L^T(V) \leq (2 - \frac{2}{|V|}) Z_4^*,$$

where $L^T(V)$ denotes the cost of the minimum spanning tree on V.

Proof: See for example [71]. □

Let $W \subset V$, then Proposition 4.1 can be easily modified to obtain:

Proposition 4.2

$$L^T(W) \leq (2 - \frac{2}{|W|}) Z_2^*(W).$$

Proof: Let (x_e) be a feasible solution to LP2. If $e \notin E(W) = \{(i,j) \mid i,j \in W\}$ implies $x_e = 0$ and using Proposition 4.1 we prove the inequality. □

4.4.4 Performance Bounds

Let $(z^*, x^*, Z_1^*) = ((z_i^*)_{i=1}^n, (x_e^*)_{e \in E}, Z_1^*)$ be the optimal solution to the linear programming relaxation of the GMST problem. Define

$$\widehat{x}_e = \rho x_e^*$$

$$\widehat{z}_i = \begin{cases} 1 \text{ if } z_i^* \geq \dfrac{1}{\rho} \\ \\ 0 \text{ otherwise.} \end{cases}$$

$W^* = \left\{ i \in V \mid z_i^* \geq \dfrac{1}{\rho} \right\} = \{ i \in V \mid \widehat{z}_i = 1 \}$. Because we need only one vertex from every cluster we delete extra vertices from W^* and consider $W^{'} \subset W^*$ such that $|W^{'}|$ = m and $W^{'}$ consists of exactly one vertex from every cluster.

Since LP1 is the linear programming relaxation of the problem IP1, we have

$$Z_1^* \leq Z_1.$$

Now let us show that $(\widehat{x}_e)_{e \in E}$ is a feasible solution to LP3 with $W = W^{'}$. Indeed, $\widehat{x}_e \geq 0$ for all $e \in E$, hence condition (4.13) is satisfied. Let $S \subset V$ be such that $W^{'} \cap S \neq \emptyset \neq W^{'} \setminus S$ and choose some $i \in W^{'} \cap S$. Hence $\widehat{z}_i$ = 1 and $z_i^* \geq \frac{1}{\rho}$. Then we have

$$\widehat{x}(\delta(S)) = \sum_{e \in \delta(S)} \widehat{x}_e = \rho \sum_{e \in \delta(S)} x_e^* \geq \rho z_i^* \geq \rho \frac{1}{\rho} = 1,$$

by definition of $\widehat{x}_e$ and the fact that the (x_e^*) solve LP1. Hence the $(\widehat{x}_e)$ satisfy constraint (4.10) in LP3.

Therefore, for the algorithm APP on page 65

$$\begin{aligned} APP &= L^T(W^{'}) \leq (2 - \frac{2}{|W^{'}|}) Z_2^*(W^{'}) = (2 - \frac{2}{|W^{'}|}) Z_3^*(W^{'}) \\ &\leq (2 - \frac{2}{|W^{'}|}) \sum_{e \in E} c_e \widehat{x}_e = (2 - \frac{2}{|W^{'}|}) \rho \sum_{e \in E} c_e x_e^* = (2 - \frac{2}{|W^{'}|}) \rho Z_1^* \\ &\leq (2 - \frac{2}{|W^{'}|}) \rho Z_1 = (2 - \frac{2}{|W^{'}|}) \rho OPT. \end{aligned}$$

And since $W^{'} \subset V$, that is, m = $|W^{'}| \leq |V|$ = n, we have proved the following.

Theorem 4.3 *The performance ratio of the algorithm "Approximate GMST problem" for approximating the optimum solution to the GMST problem satisfies:*

$$\frac{APP}{OPT} \leq (2 - \frac{2}{n})\rho.$$

□

In what follows we generalize the algorithm "Approximate GMST problem" and its analysis to the case when, in addition to the cost c_{ij} associated with each edge $e = (i, j) \in E$, there is a cost, say d_i , associated with each vertex $i \in V$.

In this case the GMST problem can be formulated as the following integer program:

$$\begin{aligned} OPT = \quad & \min \quad \sum_{e \in E} c_e x_e + \sum_{i \in V} d_i z_i \\ & s.t. \quad (4.2) - (4.6). \end{aligned}$$

Suppose that $(\overline{x}, \overline{z})$ is an optimal solution. Then the optimal value OPT of this integer program consists of two parts:

$$L_{OPT} := \sum_{e \in E} c_e \overline{x_e} \text{ and } V_{OPT} := \sum_{i \in V} d_i \overline{z_i}.$$

Under the same assumptions A1 and A2, the algorithm for approximating the optimal solution of the GMST problem in this case, is as follows:

---

1. Solve the linear programming relaxation of the previous integer program and let $(z^*, x^*) = ((z_i^*)_{i=1}^n, (x_e^*)_{e \in E})$ be the optimal solution.
2. Set $W^* = \left\{ i \in V | z_i^* \geq \frac{1}{\rho} \right\}$ and consider $W^{'} \subset W^*$ with the property that $W^{'}$ has exactly one vertex from each cluster, and find a minimum spanning tree $T \subset G$ on the subgraph $G^{'}$ generated by $W^{'}$.
3. Output $APP = ecost(T) + vcost(T)$ and the generalized spanning tree T.

---

where by $ecost(T)$ and $vcost(T)$ we denoted the cost of the tree T with respect to the edges, respectively to the nodes.

Regarding the performance bounds of this approximation algorithm, using the same auxiliary results and defining $(\widehat{x}_e, \widehat{z}_i)$ as we did at the beginning of this subsection, the following inequalities hold:

$$L^T(W^{'}) \leq \rho(2 - \frac{2}{n})L_{OPT},$$

$$V^T(W^{'}) \leq \rho V_{OPT},$$

where $L^T(W^{'})$, $V^T(W^{'})$ denote the cost of the tree T spanning the nodes of $W^{'}$ with respect to the edges, respectively to the nodes and as before $W^{'} \subset W^*$ = $\left\{ i \in V \mid z_i^* \geq \frac{1}{\rho} \right\}$, such that |$W^{'}$| = m and $W^{'}$ consists of exactly one vertex from every cluster.

For the approximation algorithm proposed in this case, the following holds:

$$\begin{aligned} APP &= L^T(W^{'}) + V^T(W^{'}) \leq \rho(2 - \frac{2}{n})L_{OPT} + \rho V_{OPT} \\ &\leq \rho(2 - \frac{2}{n})(L_{OPT} + V_{OPT}) = \rho(2 - \frac{2}{n})OPT. \end{aligned}$$

4.5 Concluding remarks

We have distinguished between so-called positive results and negative results in the area of approximation algorithms.

We presented an in-approximability result: there is no α-approximation algorithm for the GMST problem unless $\mathcal{P} = \mathcal{NP}$. However, under special assumptions (see Section 4.4) we give an approximation algorithm for the GMST problem.

Chapter 5

Linear Programming, Lagrangian and Semidefinite Relaxations

As was remarked in Chapter 1 of this thesis, finding good solutions for hard minimization problems in combinatorial optimization requires the consideration of two issues:

- calculation of an upper bound that is as close as possible to the optimum;
- calculation of a lower bound that is as close as possible to the optimum.

General techniques for generating good upper bounds are essentially heuristic methods which we are going to consider in Chapter 7 of the thesis. In addition, for any particular problem, we may well have techniques which are specific to the problem being solved.

To obtain lower bounds, one well-known technique is Linear Programming (LP) relaxation. In LP relaxation we take an integer (or mixed-integer) programming formulation of the problem and relax the integrality requirement on the variables. This gives a linear program which can either be:

- solved exactly using a standard algorithm (simplex or interior point); or
- solved heuristically (dual ascent).

The solution value obtained for this linear program gives a lower bound on the optimal value of the original minimization problem.

Another well-known technique to find lower bounds is Lagrangian relaxation. The general idea is to ”relax” (dualize) some (or all) constraints by adding them to the objective function using Lagrangian multipliers. Choosing values for the Lagrangian multipliers is of key importance in terms of quality of the lower bound generated.

Semidefinite programming (SDP) is currently a very exciting and active area of research. Combinatorial optimization problems often involve binary (0,1 or +1,-1) decision variables. These can be modelled using quadratic constraints $x^2 - x = 0$, and $x^2 = 1$, respectively. Using the positive semidefinite matrix relationship $X = xx^T$, we can lift the problem into matrix space and obtain a Semidefinite Programming Relaxation by ignoring the rank 1 restriction on X, see e.g. [49]. These semidefinite relaxations provide tight bounds for many classes of hard problems and in addition can be solved efficiently by interior-point methods.

5.1 Lagrangian Relaxation

5.1.1 Basic principles

The description of this section follows the presentations of Bertsimas and Tsitsiklis [5] and Faigle *et al.* [16].

We consider the following optimization problem:

$$z_1^* = \min \{f(x) \mid g(x) \leq 0,\ h(x) \leq 0,\ x \in \mathcal{X}\} \tag{5.1}$$

Dualizing the constraints $g(x) \leq 0$ with Lagrangian multipliers λ, $\lambda \geq 0$, yields the lower bound for z_1^*

$$L(\lambda) = \min_{x \in \mathcal{X}} \{f(x) + \lambda^T g(x) \mid h(x) \leq 0\}. \tag{5.2}$$

Here the function L (function of λ) is called *Lagrangian dual function* (or just dual function). Since the dual function is the pointwise minimum of a family of affine functions, $L(\lambda)$ is concave, even when the problem (5.1) is not convex.

The best bound of this type arises from the solution of the *Lagrangian dual problem*

$$\max_{\lambda \geq 0} L(\lambda). \tag{5.3}$$

The difference between the optimal value of problems (5.1) and (5.3) is called *duality gap*, which is always a nonnegative number. We prove this last result in Theorem 5.1, in the special case of integer linear programming.

The crucial question is which constraints to dualize. The more constraints are dualized, the weaker the bound becomes. On the other hand, dualizing more constraints facilitates the computation of $L(\lambda)$. In the extreme case, if all constraints are dualized, the computation of $L(\lambda)$ has no side constraints to consider. There is a trade off between the quality of the bounds we obtain and the effort necessary for their computation. Generally, one would dualize only the ”nasty” constraints, i.e. those that are difficult to deal with directly.

Held and Karp [33], [34] were the first to apply the idea of Lagrangian relaxation to integer linear programs. Assume we are given an integer program of the form

$$z^*_{IP} = \min \{c^T x \mid Ax \geq b,\ Bx \geq d,\ x \in \mathbb{Z}^n\} \tag{5.4}$$

for given integral matrices A, B and vectors b, c, d and let z^*_{IP} be the optimum value of (5.4). Dualizing the constraints $b - Ax \leq 0$ with multipliers $u \geq 0$ yields the lower bound

$$\begin{aligned} L(u) &= \min \{c^T x + u^T(b - Ax) \mid Bx \geq d,\ x \in \mathbb{Z}^n\} \\ &= u^T b + \min \{(c^T - u^T A)x \mid Bx \geq d,\ x \in \mathbb{Z}^n\} \end{aligned}$$

for z^*_{IP} and thus the Lagrangian dual problem is

$$z^*_D = \max_{u \geq 0} L(u). \tag{5.5}$$

where by z^*_D we denoted the optimal value of the Lagrangian dual problem.

The following general result holds (cf. e.g. [16]):

Theorem 5.1 *We have* $z^*_D \leq z^*_{IP}$.

Proof: Let x^* denote the optimal solution to (5.4). Then $b - Ax^* \leq 0$ and therefore

$$L(u) \leq c^T x^* + u^T(b - Ax^*) \leq c^T x^* = z^*_{IP}$$

for all $u \geq 0$, and therefore

$$z^*_D = \max_{u \geq 0} L(u) \leq z^*_{IP} .$$

□

The previous theorem represents the weak duality theory of integer programming. Unlike common linear programming, linear integer programming does not have a strong duality theorem.

We next investigate the quality of the bound z^*_D in comparison to the one provided by the linear programming relaxation

$$z^*_{LP} = \min \{c^T x \mid Ax \geq b,\ Bx \geq d\}, \tag{5.6}$$

which we obtain by dropping the integrality constraints $x \in \mathbb{Z}^n$.

The following general result holds (cf. e.g. [16]):

Theorem 5.2 *We have* $z^*_{LP} \leq z^*_D$.

Proof: The proof follows from the fact that the Lagrangian dual of a linear program equals the linear programming dual. We obtain the inequality by applying linear programming duality twice:

$$\begin{aligned}
z^*_D &= \max_{u \geq 0} L(u) \\
&= \max_{u \geq 0} \{u^T b + \min\{(c^T - u^T A)x \mid Bx \geq d,\ x \in \mathbb{Z}^n\}\} \\
&\geq \max_{u \geq 0} \{u^T b + \min\{(c^T - u^T A)x \mid Bx \geq d\}\} \\
&= \max_{u \geq 0} \{u^T b + \max\{d^T v \mid v^T B = c - u^T A,\ v \geq 0\}\} \\
&= \max\{u^T b + v^T d \mid u^T A + v^T B = c,\ u \geq 0\ v \geq 0\} \\
&= \min\{c^T x \mid Ax \geq b,\ Bx \geq d\} = z^*_{LP}.
\end{aligned}$$

□

Remark 5.1 As the proof of Theorem 5.2 shows, $z^*_{LP} = z^*_D$ holds if and only if the integrality constraint $x \in \mathbb{Z}^n$ is redundant in the Lagrangian dual problem defining z^*_D. In this case, the Lagrangian is said to have the *integrality property*, cf [23]. □

It turns out that solving the Lagrangian dual problem amounts to maximizing a "piecewise linear" function of a certain type. We say that a function $f : \mathbb{R}^n \to \mathbb{R}$ is *piecewise linear concave* if f is obtained as a minimum of a finite number of affine functions $f_i : \mathbb{R}^n \to \mathbb{R}$ (see Figure 5.2).

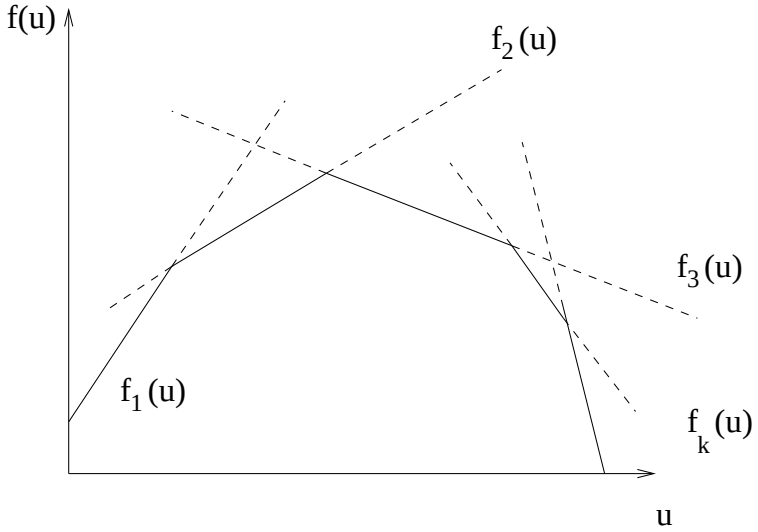

Figure 5.2: $f(u) = \min\{f_i(u) \mid 1 \leq i \leq k\}$

5.1.2 Solving the Lagrangian dual

In this subsection, we outline the subgradient method for computing the best possible lower bound $L(u^*)$ and solving the Lagrangian dual

$$z_D^* = \max_{u \geq 0} L(u).$$

We start with some definitions:

Definition 5.1 *Let f be a concave function on $\mathbb{R}^n$. A vector γ such that*

$$f(x) \leq f(\overline{x}) + \gamma^T(x - \overline{x}),$$

for all $x \in \mathbb{R}^n$, is called a **subgradient** *of f at $\overline{x}$. The set of all subgradients of f at $\overline{x}$ is denoted by $\partial f(\overline{x})$ and is called the* **subdifferential** *of f at $\overline{x}$.* □

If the function f is differentiable at $\overline{x}$, then it can be verified that $\partial f(\overline{x})$ contains only one element: the gradient of f at $\overline{x}$, i.e. $\partial f(\overline{x}) = \{\nabla^T f(\overline{x})\}$.

The Lagrangian function $L(u)$ is subdifferentiable everywhere, i.e., for all $u \geq 0$ there exists a subgradient of $L(u)$ at u. For every optimal solution $x_{\overline{u}}$ of $L(\overline{u})$, the vector $b - Ax_{\overline{u}}$ is a subgradient of L at $\overline{u}$:

$$\begin{aligned} L(u) &= c^T x_u + u^T(b - Ax_u) \\ &\leq c^T x_{\overline{u}} + u^T(b - Ax_{\overline{u}}) \\ &= c^T x_{\overline{u}} + \overline{u}^T(b - Ax_{\overline{u}}) + (b - Ax_{\overline{u}})^T(u - \overline{u}) \\ &= L(\overline{u}) + (b - Ax_{\overline{u}})^T(u - \overline{u}). \end{aligned}$$

Any other subgradient is a convex combination of these subgradients (cf [5]). It is clear that u^* is an optimal solution of the Lagrangian dual problem if and only if $0 \in \partial L(u^*)$.

Remark 5.2 It can be shown that if the Lagrangian problem $L(u)$ can be solved in polynomial time for each $u \geq 0$, then the Lagrangian dual problem can be solved in polynomial time (see, Schrijver [69], p. 368) □

Calculating the exact value may cost too much time, although it might be done in polynomial time. Therefore, one often only approximates the optimal solution of the Lagrangian dual problem. The subgradient method is an algorithm that is used very often in practice to approximate the value of the Lagrangian dual.

The subgradient optimization algorithm

1. Choose a starting point $u^1 \geq 0$ and a sequence θ_k; let $k = 1$.
2. Given u^k, choose a subgradient $\gamma^k \in \partial L(u^k)$. If $\gamma^k = 0$, then u^k is an optimal solution and the algorithm stops. Else define u^{k+1} by
$$u^{k+1} = Proj_U(u^k + \theta_k \gamma^k).$$
3. If a stopping criterion is satisfied, then stop. Else set $k \leftarrow k + 1$ and goto Step 2.

In Step 2, because we want to have $u^{k+1} \geq 0$, we project the resulting vector u^{k+1} onto the set of feasible multipliers $U = \{u \geq 0\}$ and obtain

$$u^{k+1} = Proj_U(u^k + \theta_k \gamma^k),$$

where θ_k is a positive stepsize parameter.

Often a limit on the number of iterations is used as a stopping criterion. The maximum number of iterations is determined by practical experiments. The convergence of the subgradient method depends essentially on the procedure of choosing the step sizes θ_k. The appropriate choice of the step size is a delicate problem both in theory and practice. It can be proved that the sequence of best values generated by the subgradient optimization algorithm

$$\overline{L}_k := \max \{L(u^i) : \ i = 1, ..., k\}$$

converges to the maximum value of $L(\cdot)$, for any stepsize sequence θ_k such that

$$\lim_{k \to \infty} \theta_k = 0 \quad and \quad \sum_{k=0}^{\infty} \theta_k = \infty$$

(see [37]). An example of such a sequence is $\theta_k = \frac{1}{k}$. However, this leads to slow convergence, and other choices for the step sizes, which are more efficient in practice, often are used. An example is

$$\theta_k = \theta_0 \alpha^k, \quad k = 1, 2, ...,$$

where α is a scalar satisfying $0 < \alpha < 1$. A more sophisticated and popular rule is to let

$$\theta_k = \frac{\hat{z}_D - L(u^k)}{\|\gamma^k\|^2} \alpha^k,$$

where α satisfies $0 < \alpha < 1$ and $\hat{z}_D$ is an appropriate estimate of the optimal value z_D^*.

5.2 Semidefinite Programming

Semidefinite programming (SDP) is an extension of the linear programming (LP). In comparison to standard LP the vectors are replaced by matrix variables and nonnegativity elementwise is replaced by positive semidefiniteness. In addition, LP and SDP are special cases of the conic programming (CP) problem:

$$\begin{array}{lrcl} \min & < c, x > & & \\ s.t. & Ax & = & b \\ & x & \in & K, \end{array}$$

where K is a convex cone ($K + K \subseteq K$ and $\alpha K \subseteq K,\ \forall \alpha \geq 0$), A is a linear operator $A : \mathbb{R}^n \to \mathbb{R}^m$, vectors $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ and $<, >$ is an inner product on $\mathbb{R}^n$. Let $\mathcal{S}^n$ denote the set of $n \times n$ symmetric matrices .

LP is obtained with $K = \mathbb{R}^n_+$ (the non-negative orthant) whereas SDP is obtained by letting $K = \mathcal{P}$ (the cone of positive semidefinite matrices). The cone of positive semidefinite matrices induces a partial order on the set $\mathcal{S}^n$, called the $L\ddot{o}wner\ partial\ order$

$$\text{for } A, B \in \mathcal{S}^n \quad A \succeq B \ \text{ if } \ A - B \in \mathcal{P}$$

This is the origin of the notation $A \succeq 0$ for positive semidefinite matrices.

We now discuss in more detail the similarities and differences between LP and SDP.

5.2.1 Geometry

The geometry of polyhedral sets developed for linear programming can be extended to semidefinite programming. This was studied as early as 1948 by Bohnenblust [6] and later by Taussky [73] and also Barker and Carlson [2].

Some results similar to LP are: the SDP cone is $selfpolar$, i.e.

$$\mathcal{P} = \mathcal{P}^+ := \{Y \mid X \bullet Y \ \geq 0,\ \forall X \in \mathcal{P}\},$$

where $X \bullet Y = Trace\ XY$ is the trace inner product. In the space $\mathbb{R}^n$, by definition the trace inner product is the usual inner product, i.e. $x \bullet y = x^T y$.

Also, $\mathcal{P}$ is $homogeneous$, i.e. for any $X, Y \in int(\mathcal{P})$, there exists an invertible linear operator $\mathcal{A}$ that leaves $\mathcal{P}$ invariant and $\mathcal{A}(X) = Y$ (see [76]).

5.2.2 Duality and Optimality Conditions

Extensions of the optimality conditions and duality theory from linear programming to semidefinite programming appeared in [4]. However, unlike the LP case, strong duality theorems require a Slater-type constraint qualification.

This Slater-type constraint qualification means strict feasibility, i.e., there exists a feasible point in the interior of $\mathcal{P}$ (see Definition 5.2).

Now, consider the typical primal semidefinite program

$$\begin{array}{rll} & \min & C \bullet X \\ (PSDP) & s.t. & \mathcal{A}(X) = b \\ & & X \succeq 0, \end{array}$$

where $C \in \mathcal{S}^n$, $b \in \mathbb{R}^m$, and $\mathcal{A} : \mathcal{S}^n \to \mathbb{R}^m$, is a linear operator, i.e. the components are defined as

$$(\mathcal{A}(X))_i = A_i \bullet X = b_i,$$

for some given symmetric matrices A_i.

In order to derive the dual of this program we need the adjoint operator of $\mathcal{A}$. By definition, it is the operator $\mathcal{A}^T : \mathbb{R}^m \to \mathcal{S}^n$, satisfying

$$\mathcal{A}(X) \bullet y = X \bullet \mathcal{A}^T(y),\ \forall X \in \mathcal{S}^n,\ \forall y \in \mathbb{R}^m,$$

where by definition, $\mathcal{A}(X) \bullet y = \mathcal{A}^T(X)y$, i.e. in the $\mathbb{R}^m$ space the trace inner product is the usual inner product.

Since,

$$\mathcal{A}(X) \bullet y = \sum_{i=1}^{m} y_i\ Trace(A_i X) = Trace(X \sum_{i=1}^{m} y_i A_i) = X \bullet \sum_{i=1}^{m} y_i A_i,$$

we obtain

$$\mathcal{A}^T(y) = \sum_{i=1}^{m} y_i A_i.$$

For constructing the dual we use a Lagrangian approach. The primal equality constraints are lifted into the objective by means of a Lagrange multiplier $y \in \mathbb{R}^m$ so that the primal problem reads $\min_{X \succeq 0} \max_{y \in \mathbb{R}^m} C \bullet X + y^T(b - \mathcal{A}(X))$. The dual of the (PSDP) is obtained by interchanging min and max,

$$\min_{X \succeq 0} \max_{y \in \mathbb{R}^m} C \bullet X + y^T(b - \mathcal{A}(X)) \geq \max_{y \in \mathbb{R}^m} \min_{X \succeq 0} y^T b + (C - \mathcal{A}^T(y)) \bullet X.$$

The construction implies that the right hand side value cannot exceed the value of the primal problem (see, e.g., Rockafellar [66], Lemma 36.1). For the maximum on the right hand side to be finite, the inner minimization over $X \succeq 0$ must remain finite for some $\hat{y} \in \mathbb{R}^m$. This requires $C - \mathcal{A}^T(\hat{y})$ to be positive semidefinite. We write this condition by introducing a slack matrix Z,

$$\begin{array}{lll} & \max & b^T y \\ (DSDP) & s.t. & \mathcal{A}^T y + Z = C \\ & & y \in \mathbb{R}^m,\ Z \succeq 0. \end{array}$$

This is the standard formulation of the dual semidefinite program to (PSDP).

The gap between a dual feasible solution (y, Z) and a primal feasible solution X is

$$Trace\, CX - b^T y = Trace(\mathcal{A}^T y + Z)X - \mathcal{A}(X) \bullet y = Trace\, ZX \geq 0.$$

This shows that the duality gap is always nonnegative. The last inequality follows from the following well-known lemma:

Lemma 5.1 *Let $A, B \in \mathcal{S}^n$, $A, B \succeq 0$. Then $A \bullet B \geq 0$. Moreover $A \bullet B = 0$ if and only if $AB = 0$.*

Proof: $AB = 0$ directly implies $A \bullet B = Trace\ (AB) = 0$. To prove the converse, consider the transformation of $A, B \succeq 0$ to diagonal form

$$A = \sum_{i=1}^{n} \alpha_i q_i q_i^T, \quad B = \sum_{j=1}^{n} \beta_j v_j v_j^T,$$

where q_i, v_j are the orthonormal eigenvectors and α_i, β_j the corresponding eigenvalues of A, B. Then with $A \bullet B = Trace\ (AB)$ we find using $\alpha_i \beta_j \geq 0$

$$A \bullet B = \sum_{i,j=1}^{n} \alpha_i \beta_j (q_i q_i^T v_j v_j^T) = \sum_{i,j=1}^{n} \alpha_i \beta_j (v_j^T q_i q_i^T v_j) = \sum_{i,j=1}^{n} \alpha_i \beta_j (q_i^T v_j)^2 \geq 0$$

Moreover, $A \bullet B = 0$ implies $\alpha_i \beta_j (q_i^T v_j)^2 = 0$ or $\alpha_i \beta_j (q_i^T v_j) = 0$ for all i, j and then

$$AB = \sum_{i,j=1}^{n} \alpha_i \beta_j q_i q_i^T v_j v_j^T = \sum_{i,j=1}^{n} \alpha_i \beta_j (q_i^T v_j)(q_i v_j^T) = 0.$$

□

In the following we state that the gap between optimal primal and dual objective value is guaranteed to be zero (i.e. strong duality holds) if at least one of (PSDP) and (DSDP) has a strictly feasible point.

Definition 5.2 *A point X is strictly feasible for (PSDP) if it is feasible for (PSDP) and satisfies $X \succ 0$ (i.e. X is positive definite).*
A pair (y, Z) is strictly feasible for (DSDP) if it is feasible for (DSDP) and satisfies $Z \succ 0$ (i.e. Z is positive definite). □

Theorem 5.3 *(Strong Duality)*
Assume that there exists a strictly feasible solution $(\hat{y}, \hat{Z})$ for (DSDP) and let

$$\begin{aligned} p^* &= \min \{C \bullet X \mid \mathcal{A}(X) = b,\ X \succeq 0\} \text{ and} \\ d^* &= \max \{b^T y \mid \mathcal{A}^T(y) + Z = C,\ Z \succeq 0\}. \end{aligned}$$

Then $p^* = d^*$ and if p^* is finite it is attained for some $X \in \{X \succeq 0 \mid \mathcal{A}(X) = b\}$.

Proof: See for example [76] or [63]. □

The optimality conditions for (PSDP) and (DSDP) are

$$\begin{array}{lcll} \mathcal{A}^T(y) + Z - C & = & 0 & dual\ feasibility \\ b - \mathcal{A}(X) & = & 0 & primal\ feasibility \\ XZ & = & 0 & complementary\ slackness \\ X, Z \succeq 0. & & & \end{array}$$

$X, (y, Z)$ satisfying these conditions are called a primal-dual optimal pair, Z is the (dual) slack variable. In the case that complementary slackness may fail, the variables are called a primal-dual feasible pair. Feasible solutions X and Z satisfying the last equality constraint are called *complementary*. The complementary slackness constraint can be written in the equivalent form $Z \bullet X = 0$ (conform Lemma 5.1). The above conditions imply a zero duality gap since

$$Trace\ ZX = Trace(C - \mathcal{A}^T y)X = Trace\ CX - y^T b.$$

If we perturb the complementary slackness condition,

$$ZX = \mu I, \quad \mu > 0,$$

then we get the optimality conditions for a log-barrier problem (see [67]). These are the equations that are used in interior-point methods. However, unlike linear programming there is an interesting subtle complication. One cannot apply Newton's method directly, since ZX is not necessarily symmetric (i.e. $XZ \neq ZX$), and so we end up with an overdetermined system of equations. There are various ways of modifying this system in order to get good search directions, see e.g. [53], [74]. Many of these directions work very well in practice. The algorithms that currently work well are the primal-dual interior-point algorithms.

5.3 Lagrangian duality for quadratic programs

The material introduced in Sections 5.1 and 5.2 will be used now to apply Lagrange duality to quadratic optimization problems. The remaining of this chapter follows closely the paper of Lemaréchal and Oustry [48]. In addition at page 93, we have added a comparison of the lower bounds obtained by linear and complete dualizations with the bounds given by the linear programming relaxation.

5.3.1 Dualizing a quadratic problem

We start with a well-know lemma, which will be used throughout this chapter:

Lemma 5.2 *(Schur's Lemma)*
For $Q \in \mathcal{S}^p$, $P \in \mathcal{S}^n$ and $S^T = (s_1, ..., s_p) \in \mathbb{R}^{n \times p}$ the following two statements are equivalent:
i) The matrix $\begin{pmatrix} P & S^T \\ S & Q \end{pmatrix}$ is positive definite (respectively positive semidefinite).
ii) The matrices Q and $P - S^T Q^\dagger S$ are both positive definite (respectively positive semidefinite) and $s_i \in \mathcal{R}(Q)$, for $i = 1, ..., p$; these last range conditions are automatically satisfied when $Q \succ 0$ (then $\mathcal{R}(Q) = \mathbb{R}^p$).

□

In the above lemma we denoted by $Q^\dagger$ the pseudo-inverse of the matrix Q.

Given $m+1$ quadratic functions defined on $\mathbb{R}^n$:

$$q_j(x) := x^T Q_j x + b_j^T x + c_j,\ j = 0, ..., m, \tag{5.7}$$

with $Q_j \in \mathcal{S}^n$, $b_j \in \mathbb{R}^n$ and $c_j \in \mathbb{R}$, for $j = 0, ..., m$; we assume $c_0 = 0$. We consider the quadratic problem

$$\begin{array}{ll} \min\limits_{x \in \mathbb{R}^n} & q_0(x), \\ s.t. & q_j(x) = 0,\ j = 1, ..., m\ . \end{array} \tag{5.8}$$

Dualizing the costraints (5.8) with Lagrangian multipliers $u \in \mathbb{R}^m$ yields the following Lagrangian dual function:

$$L(u) = \min_{x \in \mathbb{R}^n} \{x^T Q(u) x + b(u)^T x + c(u)\},$$

where $Q(u) := Q_0 + \sum_{j=1}^m u_j Q_j$, $b(u) := b_0 + \sum_{j=1}^m u_j b_j$, $c(u) := \sum_{j=1}^m u_j c_j = c^T u$. The dual problem is to maximize L. In fact the Lagrangian dual function can be computed explicitly:

Proposition 5.1 *(Lemaréchal and Oustry [48])*

$$L(u) \ = \ \begin{cases} c(u) - \dfrac{1}{4} b(u)^T Q(u)^\dagger b(u) & \textit{if } Q(u) \succeq 0 \textit{ and } b(u) \in \mathcal{R}(Q(u)), \\ -\infty & \textit{otherwise}. \end{cases}$$

where $Q^\dagger$ is the pseudo-inverse of Q and $\mathcal{R}(Q)$ is the range (column space) of Q.

Proof: The proof comes directly from the theory regarding the infimum of a quadratic function (see [48]). □

We want to compute the best lower bound and hence to solve the Lagrangian dual $\max\limits_{u} L(u)$, which we reformulate as follows (with $u \in \mathbb{R}^m$ and $r \in \mathbb{R}$):

$$\begin{aligned}
\max_{u} L(u) &= \max_{u,r} \{r \mid L(u) \geq r\} \\
&= \max_{u,r} \{r \mid x^T Q(u)x + b(u)^T x + c(u) \geq r,\ \forall x \in \mathbb{R}^n\} \\
&= \max_{u,r} \{r \mid (1, x^T) \begin{pmatrix} c(u) - r & \frac{1}{2}b(u)^T \\ \frac{1}{2}b(u) & Q(u) \end{pmatrix} \begin{pmatrix} 1 \\ x \end{pmatrix} \geq 0,\ \forall x \in \mathbb{R}^n\} \\
&= \max_{u,r} \{r \mid \begin{pmatrix} c(u) - r & \frac{1}{2}b(u)^T \\ \frac{1}{2}b(u) & Q(u) \end{pmatrix} \succeq 0\}.
\end{aligned}$$

Therefore the dual problem is equivalent to the following SDP problem with variables $u \in \mathbb{R}^m$ and $r \in \mathbb{R}$:

$$\begin{aligned}
&\max_{u \in \mathbb{R}^m, r \in \mathbb{R}} && r \\
&s.t. && \begin{pmatrix} c(u) - r & \frac{1}{2}b(u)^T \\ \frac{1}{2}b(u) & Q(u) \end{pmatrix} \succeq 0.
\end{aligned} \tag{5.9}$$

Applying duality again to the semidefinite problem (5.9) which is the same with bidualizing the quadratic problem (5.8). we obtain that the dual of (5.9), i.e., bidual of (5.8) is:

$$\begin{array}{lll}
\min & Q_0 \bullet X + b_0^T x, & X \in \mathcal{S}^n,\ x \in \mathbb{R}^n \\
s.t. & Q_j \bullet X + b_j^T x + c_j = 0, & j = 1, ..., m \\
& \begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix} \succeq 0. &
\end{array} \qquad (SDP)$$

Applying the weak duality theory twice, we see that

$$val(5.8) \geq val(5.9) \leq val(SDP),$$

where by $val(\cdot)$ we denote the optimal value of an optimization problem $(\cdot)$.

5.3.2 The lifting procedure

There exists a strong similarity between the quadratic problem (5.8) and the semidefinite program denoted by (SDP) these two problems differ by the introduction of the new variable X.

Using Schur's Lemma, (SDP) can be written in the equivalent form:

$$\begin{array}{ll} \min & Q_0 \bullet X + b_0^T x,\ X \in \mathcal{S}^n,\ x \in \mathbb{R}^n \\ s.t. & Q_j \bullet X + b_j^T x + c_j = 0,\ j = 1, ..., m \\ & X \succeq xx^T. \end{array} \tag{5.10}$$

This new form gives us a direct interpretation of (SDP) without passing by the intermediate problem (5.9):

- A quadratic form $x^T Q x$ can equivalently be written as

$$x^T Q x = Qx \bullet x = Q \bullet xx^T.$$

- By setting $X := xx^T$, the quadratic problem (5.8) can be written as

$$\begin{array}{ll} \min & Q_0 \bullet X + b_0^T x,\ X \in \mathcal{S}^n,\ x \in \mathbb{R}^n \\ s.t. & Q_j \bullet X + b_j^T x + c_j = 0,\ j = 1, ..., m \\ & X = xx^T. \end{array} \tag{5.11}$$

- In this last formulation of the quadratic problem (5.8), everything is linear except the last constraint, which is nonconvex.

- Relax this last constraint $X = xx^T$ to $X \succeq xx^T$, which is now convex with respect to (x, X).

The above procedure called *lifting procedure* was introduced in [49] in the context of 0-1 programming.

Summarizing in terms of duality gaps the merits of the relaxations introduced in this subsection we have:

$$val(5.9) \leq val(SDP) = val(5.10) \geq val(5.8).$$

The first inequality is weak duality between (5.9) and its dual (SDP). The equality is Schur's Lemma. The last inequality holds because the feasible domain in (5.10) is larger than in (5.8).

It is worth to mentione that the first inequality often holds as an equality (for example if an appropriate constraint qualification is satisfied).

5.4 Application to 0-1 programming

Consider the following combinatorial optimization problem:

$$\begin{array}{ll} \min & c^T x \\ s.t. & Ax = b \\ & x_i^2 - x_i = 0, \;\; i = 1, ..., n. \end{array} \tag{5.12}$$

In this program the boolean constraints $x_i \in \{0, 1\}, \; i = 1, ..., n$ were modelled using the quadratic constraints $x_i^2 - x_i = 0, \; i = 1, ..., n$, and are the only quadratic constraints.

Folllowing [48] we present three possible dualizations for the above 0-1 program and compare the optimal values of the corresponding Lagrangian dual problems.

5.4.1 Linear dualization

The first possibility is to lift the linear constraints into the objective by means of a Lagrange multiplier $u \in \mathbb{R}^m$. This kind of dualization is called linear dualization. We obtain the following Lagrangian problem:

$$L_1(u) = \min_{x \in \{0,1\}^n} \{c^T x + u^T (Ax - b)\}.$$

This minimization over $\mathcal{X} = \{0, 1\}^n$ is trivial, the result is a piecewise linear dual function:

$$L_1(u) = \sum_{i=1}^{n} \min \{0, (c + A^T u)_i\} - b^T u. \tag{5.13}$$

Maximizing L_1 is therefore a nonsmooth optimization problem, which can be done by a number of methods, see e.g. [37].

5.4.2 Complete dualization

Another possible dualization consists in taking $\mathcal{X} = \mathbb{R}^n$ and in dualizing all the constraints: the linear as well as the boolean ones. This dualization, which is called complete dualization, produces another Lagrangian problem, which now depends on two dual variables $(u, v) \in \mathbb{R}^m \times \mathbb{R}^n$:

$$\begin{aligned} L_2(u,v) &= \min_{x\in\mathbb{R}^n} \{c^T x + u^T(Ax-b) + \sum_{i=1}^n v_i(x_i^2 - x_i)\} \\ &= \min_{x\in\mathbb{R}^n} \{(c + A^T u - v)^T x + x^T D(v) x\} - b^T u. \end{aligned}$$

Here $D(v)$ denotes the diagonal matrix constructed from the vector v.

Minimizing the above Lagrangian with respect to x (on the whole $\mathbb{R}^n$) can be done using elementary calculations and we obtain:

$$L_2(u,v) = -b^T u + \sum_{i=1}^n L_i(u, v_i),$$

where

$$L_i(u,v_i) = \begin{cases} -\dfrac{(c + A^T u - v)_i^2}{4v_i} & \text{if } v_i > 0 \\ 0 & \text{if } v_i = 0 \text{ and } (c + A^T u)_i = 0 \\ -\infty & \text{otherwise.} \end{cases}$$

Theorem 5.4 *(Lemaréchal and Oustry [48])*
The linear and complete dualizations produce the same duality gap. More precisely:

$$L_1(u) = \max_{v\in\mathbb{R}^n} L_2(u,v). \tag{5.14}$$

□

Therefore the complete dualization is of moderate interest as compared to linear dualization: it increases the number of dual variables and produces a more complicated dual function, all this without improving the duality gap.

Remark 5.3 *Applying the lifting procedure to (5.12), we obtain*

$$\begin{array}{ll} \min\limits_{X \in \mathcal{S}^n} & c^T d(X), \\ s.t. & Ad(X) = b,\ j = 1, ..., m \\ & X - d(X)d(X)^T \succeq 0, \end{array} \tag{5.15}$$

where $d(X)$ is the vector obtained by extracting the diagonal of $X \in \mathbb{R}^{n \times n}$. Despite the appearances, this is a linear program in $\mathbb{R}^n$: X plays no role except through the diagonal, therefore we can set $X_{ij} = 0$, for $i \neq j$, and replace $d(X)$ by $x \in \mathbb{R}^n$. The semidefinite constraint becomes

$$X - xx^T \succeq 0 \Leftrightarrow x_i \geq x_i^2,\ i = 1, ..., n \Leftrightarrow 0 \leq x_i \leq 1,\ i = 1, ..., n.$$

□

In addition to the results obtained in [48], applying the results from Subsection 5.3, we were able to compare the linear programming relaxation bound of (5.12) with the bounds provided by the linear and complete dualizations:

$$val(5.12) \geq val(5.15) = val(LP) \geq \max_{u,v}\ L_2(u, v).$$

where $val(LP)$ denotes the optimal value of the linear programming relaxation of problem (5.12). Using Theorems 5.2 and 5.4, we have

$$val(5.12) \geq val(5.15) = val(LP) = \max_{u,v}\ L_2(u, v) = \max_{u}\ L_1(u).$$

This last result shows that the best bounds obtained using linear and complete dualizations are equal to the optimal value of the linear programming relaxation.

5.4.3 Boolean dualization

A third possibility for (5.12) is to dualize the boolean constraints. Then the corresponding dual function is the optimal value of the problem

$$L_3(v) = \min_{Ax=b} \{(c - v)^T x + x^T D(v) x\}. \tag{5.16}$$

We will call this dualization scheme boolean dualization. Before computing $L_3(v)$, we check that this complication is worth the effort.

Theorem 5.5 *(Lemaréchal and Oustry [48])*
There holds $\max_v L_3(v) \geq \max_{u,v} L_2(u,v) = \max_u L_1(u)$.

Proof: The equality is Theorem 5.3. The inequality is a general result: when more constraints are dualized, the duality gap can only increase (see for example [16]). □

We want to compute $\max_v\ L_3(v)$. If $b \notin \mathcal{R}(A)$ (i.e. $Ax = b$ has no solution in $\mathbb{R}^n$) then $val(5.12) = +\infty$ and $\max_v\ L_3(v) = +\infty$; so there is no duality gap. We therefore assume that $b = A\hat{x}$ for some $\hat{x} \in \mathbb{R}^n$. Assume $m' = rank(A)$ (i.e. the rank of A), let Z be a matrix with $n - m'$ columns that form a basis of $\mathcal{N}(A)$, where $\mathcal{N}(A)$ denote the null space of A. Thus

$$Ax = b\ \ with\ x \in \mathbb{R}^n \quad \Longleftrightarrow \quad x = \hat{x} + Zy\ \ with\ y \in \mathbb{R}^{n-m'},$$

so that (5.12) is equivalent to the following problem in y

$$\begin{array}{ll} \min & c^T Zy + c^T \hat{x}, \\ s.t. & y^T Z_i^T Z_i y + (2\hat{x}_i^T - 1) Z_i y + \hat{x}_i^2 - \hat{x}_i = 0,\ i = 1, ..., n\ . \end{array} \quad (5.17)$$

Here $Z_i \in \mathbb{R}^{n-m'}$ denotes the ith row of Z.

Theorem 5.6 *(Lemaréchal and Oustry [48])*
The dual value $\max_v L_3(v)$*, associated with (5.12), (5.16), is the value of the SDP problem*

$$\begin{array}{ll} \max & r \\ s.t. & \begin{pmatrix} \hat{c}(v) - r & \frac{1}{2}\hat{b}(v)^T \\ \frac{1}{2}\hat{b}(v) & Q(v) \end{pmatrix} \succeq 0, \end{array} \quad (5.18)$$

where

$$\begin{array}{lcl} \hat{c}(v) & := & \hat{x}^T D(v)\hat{x} + (c - v)^T \hat{x}, \\ \hat{b}(v) & := & 2D(v)\hat{x} + c - v, \\ Q(v) & := & ZD(v)Z^T. \end{array}$$

□

5.4.4 Inequality constraints

Suppose that the general optimization problem (5.12) contains inequality constraints; say

$$\begin{array}{ll} \min & c^T x \\ s.t. & Ax \leq b \\ & x_i \in \{0,1\} \;\; i = 1, ..., n \end{array} \tag{5.19}$$

or even a mixture of equalities and inequalities.

The new formulation of the general optimization problem (5.19) does not affect the linear dualization. The only change is that the dual problem now has positivity constraints and becomes $\max_{u \geq 0} L_1(u)$; but the duality gap still corresponds to replacing $\{0,1\}$ by $[0,1]$.

Remark 5.4 *Due to the general result: the more constraints are dualized, the weaker the bound becomes, one possibility to obtain a better lower bound than the bound provided by the linear dualization is to dualize just some of the linear constraints. One would dualize only the constraints that are difficult to deal with directly.* □

The complete dualization is also unaffected. The u-part of the dual variables is again constrained to $\mathbb{R}^m_+$, and we again obtain a scheme equivalent to linear dualization:

Theorem 5.7 *(Lemaréchal and Oustry [48])*
For (5.20), linear and complete dualizations produce the same duality gap.
□

By contrast, the boolean dualization breaks down: the dual fuction becomes

$$\min_{Ax \leq b} \{(c - v)^T x + x^T D(v) x\}.$$

Since $D(v)$ has no reason to be positive semidefinite, the above problem is a difficult nonconvex problem.

Therefore the boolean dualization can treat only equality constraints; as for the inequalities, they have to be treated by a linear strategy.

5.5 Concluding remarks

We presented three possible dualizations of 0-1 programming problem (containing only equality constraints): linear, complete and boolean dualizations and we compared the bounds obtained using these dualizations with the bound provided by the linear programming relaxation.

We showed that the bounds obtained by linear and complete dualizations are equal to the optimal value of the linear programming relaxation of the initial 0-1 program. In addition the bounds obtained by using boolean dualization may be better.

The results obtained for linear and complete dualizations hold also in the case when the initial 0-1 program contains inequality constraints, but break down in the case of boolean dualization. Therefore the boolean dualization can treat only equality constraints; as for the inequalities, they have to be treated by a linear strategy.

Chapter 6

Solving the GMST problem with Lagrangian Relaxation

As we have seen in the previous chapter, we may obtain better lower bounds than the bound provided by linear and complete dualizations if we dualize only the ”nasty” linear constraints, i.e. the constraints that are difficult to deal with directly.

In this chapter we present an algorithm to obtain ”good” lower bounds (better than the bounds provided by the LP relaxation) for the GMST problem. The algorithm is based on a Lagrangian relaxation of a bidirectional multicommodity flow formulation of the problem. The subgradient optimization algorithm is used to obtain lower bounds. Computational results are reported for many instances of the problem.

6.1 A strong formulation of the GMST problem

In Section 3.4, we proposed a mixed integer programming formulation of the GMST problem, called the *bidirectional multicommodity flow formulation*. In that model a cluster $V_1 \subset V$ is chosen to be the *source* offering $|K| - 1 = m - 1$ commodities, one demanded by each of the remaining $m - 1$ clusters. Variables f_{ij}^k, $(i,j) \in A$ and $k = 1, ..., m-1$ indicate the flow amount of commodity k going through the arc (i,j). Binary variables x_{ij}, with $(i,j) \in E$ and z_i, with $i \in V$ control the inclusion ($x_{ij} = 1$) or not ($x_{ij} = 0$) of edge (i,j), respectively the inclusion ($z_i = 1$) or not ($z_i = 0$) of node i in the solution. The GMST problem can be formulated as follows:

$$\min \quad \sum_{(i,j)\in E} c_{ij} x_{ij}$$

$$s.t. \quad f_{ij}^k + f_{ji}^{k'} \leq x_e, \qquad \forall\, e = (i,j) \in E,\ \forall\, k, k' \in K_1 \tag{6.1}$$

$$x(E) = m - 1 \tag{6.2}$$

$$z(V_k) = 1, \qquad \forall\, k \in K = \{1, ..., m\} \tag{6.3}$$

$$\sum_{(i,j)\in A} f_{ij}^k \geq z_i, \qquad \forall\, k \in K_1,\ \forall\, i \in V_1 \tag{6.4}$$

$$\sum_{(j,i)\in A} f_{ji}^k \geq z_i, \qquad \forall\, k \in K_1,\ \forall\, i \in V_k \tag{6.5}$$

$$\sum_{(i,j)\in A} f_{ij}^k - \sum_{(h,i)\in A} f_{hi}^k \geq 0,\ \forall\, k \in K_1,\ \forall\, i \in V \setminus (V_1 \cup V_k) \tag{6.6}$$

$$f_{ij}^k \in \{0,1\}, \qquad \forall\, k \in K_1,\ \forall\, (i,j) \in A \tag{6.7}$$

$$x_{ij}, z_i \in \{0,1\}, \qquad \forall\, (i,j) \in E,\ \forall\, i \in V. \tag{6.8}$$

Constraints (6.1) allow a non-zero flow f_{ij}^k or $f_{ji}^{k'}$ of commodity k or k' through an edge $e = (i,j)$ only if the latter is included in the solution. Constraints (6.2) and (6.3) guarantee that any feasible solution has $m - 1$ edges and contains exactly one vertex from every cluster. Equations (6.4), (6.5) and (6.6), for a given $k \in K_1$, are the network flow equations for the problem of sending a flow of value 1 from cluster V_1 to cluster V_k. Note here that in this program we implicitly set $f_{ij}^k = 0$, $\forall\, k \in K_1$, $\forall\, (i,j) \in A$ with $j \in V_1$ (i.e. delete all the arcs in cluster V_1).

6.2 Defining a Lagrangian problem

We relax equations (6.4), (6.5) and (6.6) in a Lagrangian fashion. Let t_{ik} (≥ 0, $\forall\, i \notin V_1 \cup V_k$, $\forall\, k \in K_1$) be the Lagrange multipliers corresponding to (6.6), $t_{ik}^{(1)}$ (≥ 0, $\forall\, i \in V_1$, $\forall\, k \in K_1$) be the Lagrange multipliers corresponding to (6.4) and let $t_{ik}^{(k)}$ (≥ 0, $\forall\, i \in V_k$, $\forall\, k \in K_1$) be the Lagrange multipliers corresponding to (6.5).

Then the coefficient C_{ij}^k of f_{ij}^k in the objective function of the Lagrangian dual program is given by

$$C_{ij}^k = \begin{cases} -t_{ik}^{(1)} - t_{jk}^{(k)} & \text{if } i \in V_1 \text{ and } j \in V_k \\ -t_{ik} - t_{jk}^{(k)} & \text{if } i \notin V_1 \cup V_k \text{ and } j \in V_k \\ -t_{ik}^{(1)} + t_{jk} & \text{if } i \in V_1 \text{ and } j \notin V_1 \cup V_k \\ -t_{ik} + t_{jk} & \text{if } i \notin V_1 \cup V_k \text{ and } j \notin V_1 \cup V_k \\ 0 & \text{otherwise,} \end{cases}$$

the coefficient of z_i, p_i is given by

$$p_i = \begin{cases} \sum_{k=2}^{m} t_{ik}^{(1)} & \text{if } i \in V_1 \\ \sum_{k=2}^{m} t_{ik}^{(k)} & \text{if } i \in V_k, \end{cases}$$

and the Lagrangian dual program is:

$$\begin{aligned} \min \quad & \sum_{(i,j)\in E} c_{ij}x_{ij} + \sum_{(i,j)\in A}\sum_{k\in K_1} C_{ij}^k f_{ij}^k + \sum_{i\in V} p_i z_i \\ s.t. \quad & f_{ij}^k + f_{ji}^{k'} \leq x_e, && \forall\, e=(i,j) \in E,\ \forall\, k, k' \in K_1 \\ & x(E) = m-1 \\ & z(V_k) = 1, && \forall\, k \in K = \{1,...,m\} \\ & f_{ij}^k \in \{0,1\}, && \forall\, k \in K_1,\ \forall\, (i,j) \in A \\ & x_{ij}, z_i \in \{0,1\}, && \forall\, (i,j) \in E,\ \forall\, i \in V. \end{aligned}$$

From constraint (6.1), it is simple to deduce that the best contribution to the dual objective function is given by:

$$b_{ij} := c_{ij} + \sum_{k \in K_1} \min\{0, C_{ij}^k, C_{ji}^k\}$$

and therefore the Lagrangian dual program becomes:

$$\begin{aligned} \min \quad & \sum_{(i,j) \in E} b_{ij} x_{ij} + \sum_{i \in V} p_i z_i \\ & x(E) = m - 1 \\ & z(V_k) = 1, \qquad \forall\, k \in K = \{1, ..., m\} \\ & x_{ij}, z_i \in \{0, 1\}, \qquad \forall\, (i,j) \in E,\ \forall\, i \in V. \end{aligned}$$

Let (X_{ij}), (Z_i), (F_{ij}^k), represent the optimum values of (x_{ij}), (z_i), (f_{ij}^k) in the solution of the Lagrangian dual program; then the optimal value of the Lagrangian dual program Z_D (a lower bound on the optimal solution of the GMST problem) is given by:

$$Z_D = \sum_{\{i,j\} \in E} b_{ij} X_{ij} + \sum_{i \in V} p_i Z_i.$$

6.3 The Subgradient Procedure

As we have seen in the previous chapter, choosing values for the Lagrangian multipliers is of key importance in terms of quality of lower bound generated by solving the Lagrangian dual problem.

In this section we use the subgradient method in an attempt to maximize the lower bounds obtained from the Lagrangian relaxation of the GMST problem. The procedure is as follows:

- **Step 1.** Set the initial values for the Lagrangian multipliers:

$$t_{ik}^{(1)} = 0,\ \forall\, i \in V_1,\ \forall\, k \in K_1,$$

$$t_{ik}^{(k)} = 0,\ \forall\, i \in V_k,\ \forall\, k \in K_1,$$

$$t_{ik} = 0,\ \forall\ i \notin V_1 \cup V_k,\ \forall\ k \in K_1,$$

and initialize Z_{UB}, the upper bound of the problem (e.g. from some heuristic for the problem).

- **Step 2.** Solve the Lagrangian dual program with the current set of multipliers and let the solution be Z_D, (X_{ij}), (Z_i), (F_{ij}^k).

- **Step 3.** If the Lagrangian solution (X_{ij}), (Z_i), (F_{ij}^k) is a feasible solution to the original problem then update Z_{UB}, the upper bound on the problem corresponding to a feasible solution, accordingly. Update Z_{max} at each subgradient iteration using $Z_{max} = \max\{Z_{max}, Z_D\}$, where Z_{max} denotes the maximum lower bound found over all subgradient iterations. Initially $Z_{max} = -\infty$.

- **Step 4.** Stop if $Z_{UB} = Z_{max}$ since then Z_{UB} is the optimal solution, else go to Step 5.

- **Step 5.** Calculate the subgradients

$$H_{ik}^{(1)} = Z_i - \sum_{(i,j)\in A} F_{ij}^k, \qquad \forall\ i \in V_1,\ \forall\ k \in K_1$$

$$H_{ik}^{(k)} = Z_i - \sum_{(j,i)\in A} F_{ji}^k, \qquad \forall\ i \in V_k,\ \forall\ k \in K_1$$

$$H_{ik} = \sum_{(j,i)\in A} F_{ji}^k - \sum_{(i,j)\in A} F_{ij}^k, \forall\ i \notin V_1 \cup V_k,\ \forall\ k \in K_1.$$

- **Step 6.** Define a step size T by

$$T = \alpha \frac{(Z_{UB} - Z_D)}{\|H\|},$$

where $0 < \alpha \leq 2$, and $\|H\|$ is defined by

$$\|H\| = \sum_{i \in V_1} \sum_{k \in K_1} (H_{ik}^{(1)})^2 + \sum_{i \in V_k} \sum_{k \in K_1} (H_{ik}^{(k)})^2 + \sum_{i \notin V_1 \cup V_k} \sum_{k \in K_1} (H_{ik})^2.$$

This step size depends upon the gap between the current lower bound Z_D and the upper bound Z_{UB} and the user defined parameter α with $\|H\|$ being a scaling factor.

- **Step 7.** Update the Lagrange multipliers by

$$t_{ik}^{(1)} = \max\{0, t_{ik}^{(1)} + TH_{ik}^{(1)}\}, \ \forall \ i \in V_1, \ \forall \ k \in K_1,$$

$$t_{ik}^{(k)} = \max\{0, t_{ik}^{(k)} + TH_{ik}^{(k)}\}, \ \forall \ i \in V_k, \ \forall \ k \in K_1,$$

$$t_{ik} = \max\{0, t_{ik} + TH_{ik}\}, \ \forall \ i \notin V_1 \cup V_k, \ \forall \ k \in K_1,$$

- **Step 8.** Go to Step 2 to resolve the Lagrangian dual program with this new set of multipliers unless a stopping criterium is met.

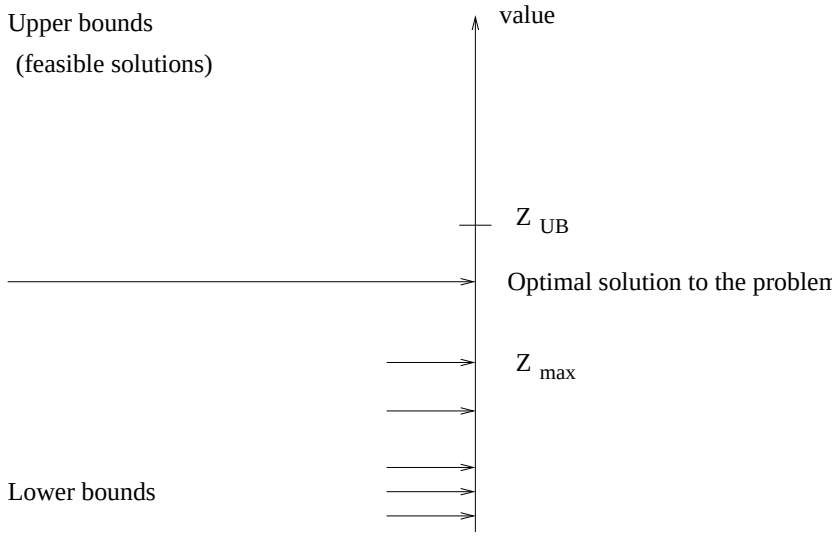

Figure 6.1: Subgradient optimization

Figure 6.1 illustrates the situation during the subgradient iterations. As shown in this figure, we plot the lower bound found at each subgradient iteration on the value line. The best (maximum) of these lower bounds is Z_{max}.

Initially we set $\alpha = 2$. If Z_{max} is not improved (i.e. increased) in the last N subgradient iterations with the current value of α then we halve α. Based on experimenting (computationally) with different values of N, it seems to be reasonable to choose the value $N = 30$.

In our experiments we set $Z_{UB} = Z_{OPT}$, where Z_{OPT} is the optimal value of the GMST problem calculated using the rooting procedure (see Section 3.6).

6.4 Computational Results

We used the same classes of instances as in Subsection 3.6. According to the method of generating the edge costs, the problems generated are classified into three types:

- **structured Euclidean case**
- **unstructured Euclidean case**
- **non-Euclidean case**.

For the instances in the structured Euclidean case m squares (clusters) are "packed in a square" and in each of these m clusters n_c nodes are selected randomly. The costs between nodes are the Euclidean distances between the nodes. So in this model the clusters can be interpreted as physical clusters. In the other models such an interpretation is not valid.

For the unstructured Euclidean case $n = mn_c$ nodes are generated randomly in $[0, 100]^2$ with costs given by the Euclidean distances. But then the clusters are choosen randomly among these points. Finally in the non-Euclidean model the edge costs are randomly generated on $[0, 100]$.

Our computational experiments were performed on a HP 9000/735 computer with a 125 Mhz processor and 144 Mb memory. Our Lagrangian relaxation scheme is written in C and compiled with a HP-UX cc compiler.

Table 6.1: Computational Results for unstructured Euclidean problems

Problem number	m	n_c	OPT	LB	Gap(%)	CPU(s)	# subgradient iterations
1	8	3	90	89.998	0	0.09	25
2		4	88	87.996	0	0.12	30
3		6	60	59.999	0	0.43	65
4		8	46	45.941	0.13	36.93	160
5		10	45	44.953	0.10	48.18	180
6	10	3	108	107.998	0	7.08	35
7		4	91	90.976	0.03	16.14	50
8		6	85	84.972	0.03	20.73	65
9		8	67	66.951	0.07	41.83	125
10		10	62	61.943	0.09	55.27	140
11	12	3	116	115.947	0.05	13.03	45
12		4	107	106.916	0.08	24.94	75
13		6	89	88.836	0.18	78.53	150
14		8	75	74.759	0.32	174.43	275
15	15	3	135	134.987	0.01	13.72	30
16		4	124	123.985	0.01	16.91	45
17		6	96	95.899	0.11	72.88	105
18		8	91	90.869	0.14	134.29	175
19	18	3	153	152.991	0.01	15.29	40
20		4	132	131.985	0.01	25.65	45
21		6	127	126.909	0.07	84.28	110
22		8	112	111.849	0.14	204.12	215
23	20	3	175	174.981	0.01	16.39	35
24		4	147	146.972	0.02	44.71	75
25		6	129	128.959	0.04	65.78	95
26		8	108	107.795	0.19	220.43	200
27	25	3	182	181.975	0.01	23.79	60
28		4	159	158.969	0.02	61.55	90
29		6	142	141.893	0.08	92.37	125
30		8	127	126.825	0.14	289.78	250
31	30	3	196	195.926	0.04	41.27	75
32		4	174	173.889	0.06	101.35	150

Table 6.2: Computational Results for structured Euclidean problems

Problem number	m	n_c	OPT	LB	Gap(%)	CPU(s)	# subgrad. iterations
1	8	3	983	982.998	0	0.08	25
2		4	966	965.993	0	0.18	30
3		6	960	959.988	0	0.49	65
4		8	934	933.953	0.01	36.91	160
5		10	922	921.949	0.01	48.72	180
6	10	3	1251	1250.998	0	7.23	35
7		4	1243	1242.980	0	16.59	50
8		6	1240	1239.975	0	20.93	65
9		8	1225	1224.956	0	41.98	125
10		10	1208	61.943	0	55.31	140
11	12	3	1616	1615.951	0	14.63	45
12		4	1545	1544.908	0.01	25.76	75
13		6	1487	1486.841	0.01	78.96	150
14		8	1458	1457.745	0.02	174.81	275
15	15	3	1977	1976.985	0	14.79	30
16		4	1966	1965.981	0	18.85	45
17		6	1946	1945.909	0	75.81	105
18		8	1932	1931.899	0.01	138.77	175
19	18	3	2384	2383.992	0	17.16	40
20		4	2365	2364.985	0	27.70	45
21		6	2352	2351.904	0	89.22	110
22		8	2338	2337.863	0.01	225.08	215
23	20	3	2765	2764.972	0	17.32	35
24		4	2741	2740.968	0	46.81	75
25		6	2728	2727.956	0	69.77	95
26		8	2709	2708.797	0.01	231.48	200
27	25	3	3112	3111.967	0	24.72	60
28		4	3094	3093.958	0	64.51	90
29		6	3075	3074.795	0.01	97.38	125
30		8	3058	3057.715	0.01	298.89	250
31	30	3	3478	3477.832	0	52.13	75
32		4	3452	3451.787	0.01	138.96	150

Table 6.3: Computational Results for non-Euclidean problems

Problem number	m	n_c	OPT	LB	Gap(%)	CPU(s)	# subgradient iterations
1	8	3	28	27.998	0.01	0.05	25
2		4	19	18.996	0.02	0.11	25
3		6	11	10.995	0.05	0.42	65
4		8	13	12.952	0.37	35.02	150
5		10	9	8.953	0.52	40.12	175
6	10	3	29	28.998	0.01	7.03	35
7		4	24	23.986	0.06	15.01	50
8		6	16	15.985	0.09	20.17	65
9		8	12	11.956	0.37	40.73	110
10		10	11	10.941	0.54	43.28	120
11	12	3	34	33.999	0	12.25	45
12		4	21	20.998	0.01	24.73	75
13		6	20	19.866	0.67	76.28	150
14		8	14	13.759	1.75	172.32	250
15		10	12	11.825	1.48	177.23	275
16	15	3	28	27.996	0.01	12.89	30
17		4	25	24.993	0.01	15.23	45
18		6	19	18.967	0.17	70.12	105
19		8	15	14.823	1.19	135.95	180
20		10	14	13.792	1.46	142.37	210
21	18	3	34	33.994	0.02	14.61	45
22		4	34	33.995	0.01	21.75	40
23		6	22	21.974	0.07	83.27	110
24		8	18	17.921	0.14	165.38	200
25		10	17	16.823	0.14	171.82	215
26	20	3	37	36.989	0.03	16.78	35
27		4	28	27.972	0.10	43.66	75
28		6	27	26.982	0.07	59.23	85
29		8	19	18.899	0.53	201.83	175
30		10	19	18.792	1.11	200.25	180

Problem number	m	n_c	OPT	LB	Gap(%)	CPU(s)	# subgradient iterations
31	25	3	46	45.978	0.05	51.29	75
32		4	35	34.981	0.05	60.07	80
33		6	24	23.829	0.72	169.23	180
34		8	24	23.779	0.93	202.78	230
35	30	3	43	42.969	0.07	53.21	75
36		4	32	31.978	0.07	63.72	80
37		6	29	23.852	0.62	174.55	170
38		8	29	28.872	0.44	209.33	220
39	40	3	44	43.967	0.08	82.51	80
40		4	41	40.915	0.21	101.73	90

Each line corresponds to an instance. The first column is the problem number. The next two columns give the number of clusters m and the number of nodes per cluster n_c. The fourth column (OPT) contains the optimal value of the GMST problem found by using the rooting procedure (see Section 3.6). The fifth column (LB) gives the lower bound obtained using the subgradient method. The next column (GAP %) gives the gap in percentage defined by $100(OPT - LB)/LB$. The last two columns give the CPU time and the number of subgradient iterations necessary to find the lower bound.

Comparing these results with the lower bounds provided by the LP relaxation of the GMST problem (we did numerical experiments which are not given here, solving the LP relaxation of the GMST problem by CPLEX), the lower bounds obtained using our Lagrangian relaxation scheme are in general better.

Remark 6.1 Here our experiments were made by setting $Z_{UP} = Z_{OPT}$, where Z_{OPT} is the optimal value of the GMST problem calculated using the rooting procedure (see Section 3.6). In practice the optimal value Z_{OPT} is unknown, therefore we only may have an upper bound $Z_{UP} \approx Z_{OPT}$. But numerical experiments have shown that even in this case the quality of the gap is maintained. The values of the gap in percentage in the special non-Euclidean case $m = 15$ and $n_c = 6$, using different upper bounds, are reported in the next table. □

m	n_c	OPT	UB	Gap(%)	CPU(s)	# subgrad. iterations
15	6	19	-	0.17	70.12	105
15	6	-	19.5	0.19	70.33	105
15	6	-	20	0.20	70.67	105
15	6	-	21	0.23	71.09	105

The first two columns give the number of clusters m and the number of nodes per cluster n_c. The third column (OPT) contains the optimal value of the GMST problem found by using the rooting procedure (see Section 3.6). The forth column (UB) gives the upper bound. The next column (GAP %) gives the gap in percentage defined by $100(UB - LB)/LB$. The last two columns give the CPU time and the number of subgradient iterations.

6.5 Concluding remarks

We presented a Lagrangian relaxation of a bidirectional multicommodity flow formulation of the GMST problem. The subgradient optimization algorithm was used to obtain lower bounds.

Computational experiments (by setting $Z_{UP} = Z_{OPT}$) show that using our Lagrangian relaxation scheme are in general better than the lower bounds provided by the linear programming relaxation of the GMST problem. This result holds even in the case when we may have available an upper bound $Z_{UP} \approx Z_{OPT}$.

Chapter 7

Heuristic Algorithms

The techniques described in this chapter are the ones which are known by the term heuristics, i.e. techniques which seek near-optimal solutions at a reasonable computational cost without being able to guarantee optimality, or even in many cases to state how close to optimality a particular solution is.

In the first sections we present some basic theory of heuristic algorithms and local search. In Section 7.4 we present the Simulated Annealing algorithm. Finally, in the last section we solve the GMST problem with Simulated Annealing.

7.1 Introduction

We have mentioned that many combinatorial optimization problems are $\mathcal{NP}$-*hard* [21], and the theory of $\mathcal{NP}$-completeness has reduced hopes that $\mathcal{NP}$-*hard* problems can be solved within polynomially bounded computation times. Nevertheless, sub-optimal solutions are sometimes easy to find. Consequently, there is much interest in approximation and heuristic algorithms that can find near optimal solutions within reasonable running time.

In mathematical programming, a *heuristic method* or *heuristic* for short is a procedure that determines good or near-optimal solutions to an optimization problem. As opposed to exact methods, heuristics carry no guarantee that an optimal solution will be found.

Practically, for many realistic optimization problems good solutions can be found efficiently, and heuristics are typically among the best strategies in terms of efficiency and solution quality for problems of realistic size and complexity. Heuristics can be classified as either *constructive* (greedy) or as *local search* heuristics. The former are typically one-pass algorithms whereas the latter are strategies of *iterative improvement*. We will be concerned exclusively with local search heuristics here.

Useful references on heuristic methods can be found in Osman and Laporte [56] and Reeves [64].

7.2 Local and global optima

The following concepts are taken from [57].

Definition 7.1 *An* **instance of an optimization problem** *is a pair* (S, c)*, where* S *is any set, the domain of feasible points;* c *is the cost function*

$$c : S \to \mathbb{R}$$

The problem is to find an $s \in S$ *for which*

$$c(s) \leq c(y) \quad \textit{for all } y \in S.$$

Such a point is called **globally optimal solution** *to the given instance or, when no confusion can rise, simply an* **optimal solution**. □

Given a feasible point $s \in S$ in a particular problem, it is useful in many situations to define a set $N(s)$ of points that are "close" in some sense to the point s.

Definition 7.2 *Given an optimization problem with instances (S, c), a* **neighbourhood** *is a mapping*

$$N : S \to 2^S$$

defined for each instance. □

In many combinatorial problems, there are quite natural choices for N suggested by the structure of S. It is often possible to find a solution s which is best in the sense that there is nothing better in its neighbourhood $N(S)$.

Definition 7.3 *Given an instance (S, c) of an optimization problem a neighbourhood N, a feasible solution $s \in S$ is called* **locally optimal with respect to N** *if*

$$c(s) \leq c(g) \quad \textit{for all } g \in N(s).$$

□

7.3 Local Search

In the next two sections we present the basic theory of local search and simulated annealing. We follow the description of Reeves [64].

Local search (LS) is a term for the extremely simple concept underlying many modern heuristic techniques, such as Simulated Annealing, Tabu Search and Genetic Algorithms.

Suppose that we have a minimization problem over a set of feasible solutions S and a cost function $c : S \to \mathbb{R}$. The optimal solution could be obtained by calculating $c(s)$ for each $s \in S$ and selecting the minimum. This method is impracticable when the set S is big which is the case of the most real-life problems. Local search overcomes this difficulty by searching only a small subset of the solution space. This is achieved by defining a neighbourhood structure on the solution space and searching the neighbourhood of the current solution for an improvement. If there is no neighbour which results in an

improvement to the cost function, the current solution is taken as an approximation to the optimum. If an improvement is found, the current solution is replaced by the improvement and the process is repeated.

Therefore the local search process can be described as follows:

Local search for a problem with solution space S, cost function c and neighbourhood structure N

---

1. Select a starting solution $s_0 \in S$;

2. Repeat
Select s such that $c(s) < c(s_0)$ by a suitable method;
Replace s_0 by s;
Until $c(s) > c(s_0)$ for all $s \in N(s_0)$.

s_0 is the approximation of the optimal solution.

A central problem for local search is the occurence of *local optima*, i.e. nodes in the search space where no neighbour strictly improves over the current node in terms of the cost function, but which are not global optima. Many strategies have been proposed that address the problem of how to overcome the local optima. In many cases, non-improving local moves are admitted based on a probabilistic decision or based on the history of the search. To obtain guiding principles for designing effective optimization strategies, a number of conceptual meta-level strategies have been employed with local search. These strategies are referred to as *metaheuristic*, a term introduced by Glover [25].

In contrast to individual heuristic algorithms that are designed to solve a specific problem, metaheuristics are strategic problem solving frameworks that can be adapted to solve a wide variety of problems. A metaheuristic framework includes a set of problem-solving principles that are sometimes based on natural or physical processes, and a set of control parameters. Application of a metaheuristic stategy to a specific problem involves choosing, and sometimes fine-tuning, these parameters.

Metaheuristic methods are not normally expected to outperform specialized heuristics on a problem. Their main purpose is to deal with difficult problems

for which no specialized algorithm is known. However, some studies have demonstrated that certain metaheuristics perform as well or better than special heuristics for some well-known difficult problems such as traveling salesman, graph partitioning and facility layout problems.

The most important metaheuristics for optimization are given in the following:

Simulated Annealing, introduced for optimization by Kirkpatrick *et al.* [44] in 1983. The central idea is to accept a candidate move that increases the solution quality based on a probabilistic decision. During the time of the search, the probability of acceptance of deteriorating moves is decreased according to a given *annealing schedule*.

Genetic Algorithms were invented by Holland and model optimization as an evolutionary process. The strategy is to have a pool of ”chromosomes” and iteratively apply the principles of mutation, mating and selection to attain ”survival of the fittest” [39]. Genetic algorithms extend the basic local search scheme to populations of solutions.

Tabu Search was first suggested by Glover [25] and uses information based on the history of the search [26]. It has been succesfully applied to obtain optimal or sub-optimal solutions for many combinatorial optimization problems such as scheduling, time-tabling, travelling salesman, etc.. The central idea is the use of adaptive memory to overcome local optima by driving the search to different parts of the search space (diversification) or back to promising parts (intensification).

Artificial Neural Networks make use of the metaphor of the neuron and are organized in network structures. The network of nodes is iteratively modified by adjusting the interconnections between neurons according to various schemes [77].

For particular application problems, the metaheuristics require concrete implementation and their success varies for different application domains.

7.4 Simulated Annealing

7.4.1 Introduction

The ideas that form the basis of simulated annealing were first published by Metropolis *et al*. [52] in 1953 as an algorithm to simulate the cooling of material in a heat bath, a process known as annealing. If solid material is heated past its melting point and then cooled back into a solid state, the structural properties of the cooled solid depend on the rate of cooling. The annealing process can be simulated by regarding the material as a system of particles. Essentially, Metropolis' algorithm simulates the change in energy of the system during the cooling process, as it converges to a steady 'frozen' state. Kirkpatrick *et al*. [44] and Cerny [9] suggested that this type of simulation could be used to search the feasible solutions of an optimization problem. Their approach can be regarded as a variant of the local search technique.

In the case of a minimization problem the local search employ a descent strategy, in which the search always moves in a direction of improvement. However, such a strategy often results in convergence to a local rather than a global optimum.

In order to overcome the local optima, in the simulated annealing heuristic non-improving local moves are allowed, but their frequency is governed by a probability function which changes as the algorithm progresses.

The inspiration for this form of control was Metropolis' work in statistical thermodynamics. The laws of thermodynamics state that at temperature T, the probability of an increase in energy of magnitude δE is given by

$$p(\delta E) = \exp(-\delta E/kT) \tag{7.1}$$

where k is a physical constant known as Boltzmann's constant.

Metropolis' simulation generates a perturbation and calculates the resulting energy change. If energy has decreased the system moves to this new state. If energy has increased, the new state is accepted according to the probability given in Equation 7.1. The process is repeated for a predetermined number of iterations at each temperature, after which the temperature is decreased until the system freezes into a steady state.

Kirkpatrick *et al*. [44] and Cerny [9] independently showed that Metropolis' algorithm could be applied to optimization problems by mapping the elements

of the physical cooling process onto the elements of a combinatorial optimization problem as shown in the following table:

Thermodynamic simulation	Combinatorial optimization
System states	Feasible solutions
Energy	Cost
Change of state	Change of solution
Temperature	Control parameter
Frozen state	Heuristic solution

Therefore any local search algorithm can be converted into an annealing algorithm by sampling the neighbourhoods randomly and allowing the acceptance of an inferior solution with respect to cost minimization according to the probability given in Equation 7.1. The level of acceptance then depends on the magnitude of the increase in the cost function and the current temperature.

7.4.2 The annealing algorithm

As we have seen the main disadvantage of local search is its likelihood of finding a local, rather than global, optimum. By allowing some non-imroving local moves in a controlled manner, simulated annealing overcomes this problem. The annealing algorithm can be stated as follows:

Simulated annealing for a minimization problem with solution space S, objective function c and neighbourhood structure N

Select a starting solution $s_0 \in S$, an initial temperature $T_0 > 0$ and a temperature reduction function α;

Repeat
  Repeat
    Randomly select $s \in N(s_0)$ and compute $\delta = c(s) - c(s_0)$;
    If $\delta < 0$
    then $s_0 = s$
    else generate random p uniformly in the range $(0, 1)$;
    if $p < \exp(-\delta/T)$ then $s_0 = s$
  Until $iterationcount = L$;
Set $T = \alpha(T)$;
Until stopping condition is true.

s_0 is the approximation of the optimal solution.

Here L represents the number of repetitions at each temperature, i.e. the number of randomly chosen candidate solutions in the neighborhood of the current solution that are evaluated.

At each stage, L randomly chosen candidate solutions in the neighborhood of the current solution are evaluated. If a candidate solution improves on the current solution, it is accepted. Otherwise, it is accepted with a probability $p(T, \delta) = \exp(-\delta/T)$, which depends on the control parameter T (the temperature in the physical equivalent) and the amount δ by which a move worsens the current solution. This relation ensures that the probability of accepting a move to a very poor solution is very small. At the completition of each stage, the temperature T is reduced at cooling rate r. Given a relatively high temperature T at the beginning of the process, the probability of accepting non-improving moves is fairly high. As the process continues, the temperature decreases and non-improving moves become less likely. The search is continued until there is some evidence to suggest that there is a very low probability of improving on the best solution found so far. At this stage, the system is said to be frozen.

The algorithm given is very general, and a number of decisions must be made in order to implement it for the solution of a particular problem. These can be divided into two categories:

- generic decisions which are concerned with parameters of the annealing algorithm itself. These include factors such as the initial temperature, the cooling schedule (governed by the parameter L, i.e. number of repetitions and the choice of the temperature reduction function α) and the stopping condition.

- problem specific decisions and involves the choice of the space of feasible solutions, the form of the cost function and the neighborhood structure employed.

Simulated annealing has the advantage that it is simple in concept and implementation. The procedure improves on descent methods by reducing the probability of getting stuck at a poor but locally optimal solution. Simulated annealing has been applied successfully to many hard optimization problems; see e.g., the bibliography by Osman and Laporte [56].

Since Kirkpatrick's original paper there has been much research into the theoretical convergence of the annealing algorithm. The theoretical results are based on the fact that the behavior of the simulated annealing algorithm can be modelled using Markov chains (see e.g. [5]).

7.5 Solving the GMST problem with Simulated Annealing

7.5.1 The local-global formulation of the GMST problem

In order to solve the GMST problem with simulated annealing we use the local-global formulation of the problem (see Section 3.5), which arises from distinguishing between **global**, i.e., inter-cluster connections, and **local** ones. We introduced the variables y_{ij} $(i, j \in \{1, ..., m\})$ to describe the global connections: $y_{ij} = 1$ if cluster V_i is connected to cluster V_j and $y_{ij} = 0$ otherwise and we assumed that y represents a spanning tree. The convex hull of all these y-vectors is generally known as the spanning tree polytope on the contracted graph with vertex set $\{V_1, ..., V_m\}$ (which we assume to be complete) and can be represented by the following constraints:

$$
\begin{aligned}
\sum_{\{i,j\}} y_{ij} &= m-1 \\
y_{ij} &= \lambda_{kij} + \lambda_{kji}, \text{ for } 1 \le k,i,j \le m \text{ and } i \ne j \\
\sum_{j} \lambda_{kij} &= 1, \qquad \text{for } 1 \le k,i,j \le m \text{ and } i \ne k \\
\lambda_{kkj} &= 0, \qquad \text{for } 1 \le k,j \le m \\
y_{ij}, \lambda_{kij} &\ge 0, \qquad \text{for } 1 \le k,i,j \le m.
\end{aligned}
$$

where the variables λ_{kij} were defined in Section 3.5.

We have seen that if the 0-1 vector y describes a spanning tree on the contracted graph, the corresponding best (w.r.t. minimization of the cost) "local solution" $x \in \{0,1\}^{|E|}$ can be obtained as we have shown in Section 3.5 by using dynamic programming or by solving the following 0-1 programming problem:

$$
\begin{aligned}
\min \quad & \sum_{\{i,j\} \in E} c_{ij} x_{ij} \\
s.t. \quad & z(V_k) = 1, && \forall\, k \in K = \{1,...,m\} \\
& x(V_l, V_r) = y_{lr}, && \forall\, l,r \in K = \{1,...,m\},\ l \ne r \\
& x(i, V_r) \le z_i, && \forall\, r \in K, \forall\, i \in V \setminus V_r \\
& x_{ij}, z_i \in \{0,1\}, && \forall\, \{i,j\} \in E,\ \forall\, i \in V.
\end{aligned}
$$

In Proposition 3.5 we showed that the polyhedron $P_{local}(y)$, i.e. the feasible set of the linear programming relaxation of the previous 0-1 program, is integral.

Feasible solutions for the GMST problem can be computed by solving $P_{local}(y)$ for certain 0-1 vector y corresponding to spanning trees. (E.g., minimum spanning trees w.r.t. the edge weights $d_{lr} := \min\{c_{ij} \mid i \in V_l,\ j \in V_r\}$.)

7.5.2 Generic decisions

The generic decisions involve the choice of the following parameters:

- initial value of the control parameter, T_0;

- final value of the control parameter, T_f (stopping criterion);
- the number of repetitions at each temperature;
- decrement of the control parameter.

In our implementation of the simulated annealing algorithm for the GMST problem we chose the following cooling schedule:

1. **Initial value of the control parameter,** T_0.
   Following Johnson *et al.* [41] we determine T_0 by calculating the average increase in cost δ^+, for a number of random transitions, and find T_0 from

   $$\chi_0 = \exp(-\delta^+/T_0) \tag{7.2}$$

   Equation 7.2 leads to the following choice for T_0:

   $$T_0 = \frac{\delta^+}{\ln(\chi_0^{-1})} \tag{7.3}$$

   where $\chi_0 = 0.8$ is a given acceptance rate.

2. **Final value of the control parameter**.
   A stop criterion, determining the final value of the control parameter, is either by fixing the number of values T_k, for which the algorithm is to be executed or based on the argument that execution of the algorithm can be terminated if the improvement in cost, to be expected in the case of continuing execution of the algorithm is small.

3. **Number of repetitions at each temperature**.
   The number of iterations at each temperature which is related to the size of the neighbourhoods may vary from temperature to temperature. It is important to spend a long time at lower temperatures to ensure that a local optimum has been fully explored. Therefore we increased the value of L arithmetically (by adding a constant factor).

4. **Decrement of the control parameter**.
   We used the following decrement rule:

$$\alpha(T) = rT \tag{7.4}$$

where r is a constant smaller than but close to 1, called the cooling rate. We used, as in [41], $r = 0.95$. This corresponds to a fairly slow cooling.

7.5.3 Problem specific decisions

The space of feasible solutions and the form of the cost function are given in the definition of the GMST problem. Therefore it remains to present a neighbourhood structure.
If the problem consists of m clusters, by Cayley's formula [8] we know that the total number of global spanning trees (i.e. trees connecting the clusters) is equal to m^{m-2}. All these possibilities of global trees form the solution space.

As we have seen feasible solutions to the GMST problem can be computed by solving $P_{local}(y)$, for any 0-1 vector y corresponding to spanning trees.

In Figure 7.1 we present an initial global solution in the case when the GMST problem consists of $m = 8$ clusters.

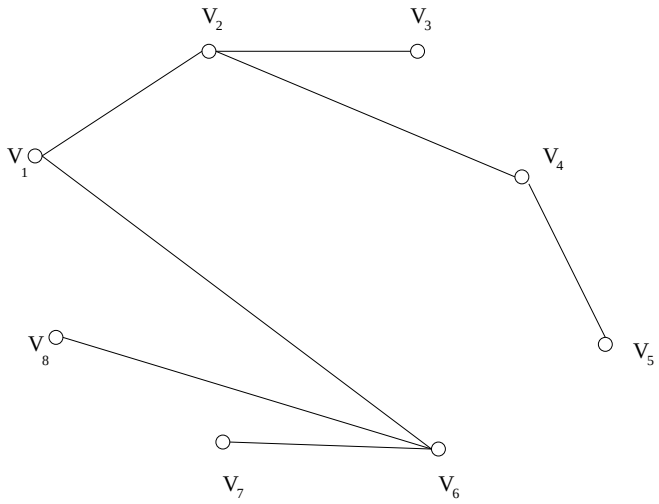

Figure 7.1: An initial global solution

A *k-neighbourhood* of any given global spanning tree is defined by those trees obtained by removing k links and replacing them by a different set of k links in such a way to maintain the feasibility, i.e. a global spanning tree.

We consider the case $k = 1$. For example if we remove the edge connecting the clusters V_2 and V_4.

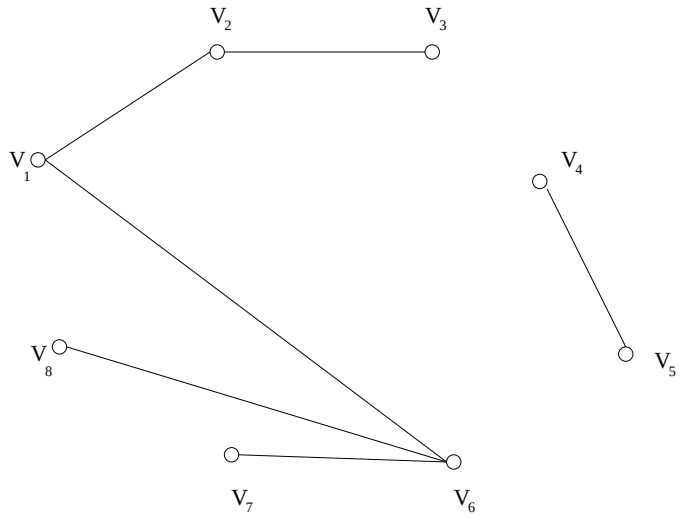

Figure 7.2: After edge removal

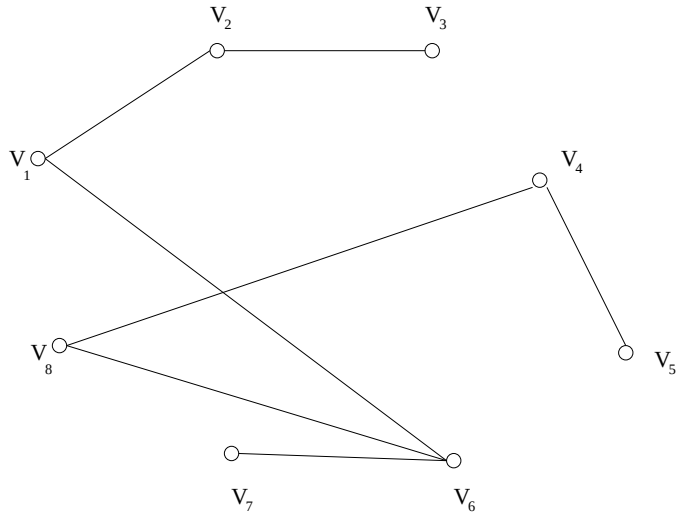

Figure 7.3: A neighbouring tree

There are several possibilities of reconnecting the two resulting sections to form a valid global tree, for example by connecting V_1 with V_4 or V_2 with V_5, etc. All these possibilities of reconnecting give the size of the neighbourhood.

Solving $P_{local}(y)$ with the vector y corresponding to the current neighbouring tree we get a feasible solution for the GMST problem.

Therefore the combination of the definition of feasible solution and neighbourhood structure appears to be a good candidate for annealing.

7.5.4 A computational example

Our computational experiment was performed on a HP 9000/735 computer with a 125 Mhz processor and 144 Mb memory. The Simulated Annealing scheme is written in C and compiled with a HP-UX cc compiler. For solving the linear programming problems we used CPLEX.

We consider the GMST problem with $m = 10$ clusters and $n = 3$ nodes per cluster as an illustration of the simulated annealing method.

We start with the temperature $T_0 = 100$. The cooling rate $r = 0.95$ will be used to reduce the temperature in each stage, and $L = 50$ moves evaluated in each stage. Table 7.1 shows the first twelve iterations of the simulated annealing procedure.

Table 7.1

sol.	swap		δ	$p \in U[0;1]$	$\exp^{-\frac{\delta}{T}}$	current solution	best solution
0	-	-	-	-	-	228	228
1	(V_5, V_9)	(V_1, V_9)	-66	-	-	152	152
2	(V_3, V_{10})	(V_3, V_8)	46	0.05	0.63	198	152
3	(V_7, V_9)	(V_4, V_7)	-45	0.05	1.57	153	152
4	(V_1, V_6)	(V_1, V_8)	-18	0.05	1.20	135	135
5	(V_6, V_8)	(V_5, V_6)	-9	0.05	1.09	126	126
6	(V_9, V_{10})	(V_5, V_{10})	-31	0.05	1.36	95	95
7	(V_5, V_{10})	(V_4, V_{10})	7	0.41	0.93	102	95
8	(V_1, V_9)	(V_2, V_9)	-14	0.41	1.15	88	88
9	(V_2, V_3)	(V_2, V_5)	16	0.98	0.85	104	88
10	(V_2, V_3)	(V_2, V_6)	8	0.63	0.92	96	88
11	(V_4, V_{10})	(V_2, V_{10})	22	0.02	0.80	118	88
12	(V_2, V_6)	(V_2, V_7)	-1	0.02	1.01	117	88

The first column is the solution number. The second column "swap" gives the global connection that is removed and the new global connection that maintains the feasibility. Next column give the cost of the move δ, the probability of accepting a move $p \in U[0;1]$, where $U[0;1]$ denote a uniform distribution between 0 and 1 from which random numbers are drawm, the value of $\exp^{-\frac{\delta}{T}}$, the current solution and the best solution.

The first move is to replace the connection of the clusters (V_5, V_9) with (V_1, V_9). The cost of this move is $\delta = -66$, therefore improving and is accepted automatically. The cost of the second move is $\delta = 46$. Since it is not an improving move, a random number is generated, here $p = 0.05 < 0.63 = \exp^{-\frac{-46}{100}} = \exp^{-\frac{\delta}{T}}$, and the move is accepted. The next four moves are downhill (i.e. improving) moves and are automatically accepted. The seventh move is not an improving move, a random number is generated, here $p = 0.41 < 0.93 = \exp^{-\frac{-7}{100}} = \exp^{-\frac{\delta}{T}}$, and the move is accepted. The eighth move is improving, and is automatically accepted. The nineth move, again an uphill move, is rejected since the random number drawn $p = 0.98 > 0.85 = \exp^{-\frac{-16}{100}} = \exp^{-\frac{\delta}{T}}$. The next two moves are not improving moves, but are accepted because $p < \exp^{-\frac{\delta}{T}}$. Finally the last move is an improving move and is accepted automatically.

It is important to observe the large number of non-improving moves that are accepted in the first stage. In the second stage, the temperature control parameter T is reduced to $T = 0.95(100) = 95$. This will reduce the probability of accepting non-improving moves by five percent. In later stages, T will become almost zero and nonimproving moves will be very unlikely to be accepted at all.

It is generally agreed that, relative to special purpose heuristics and other metaheuristics, simulated annealing requires long runs to reach good solutions. Choices of the control parameters and the stopping rule determine the computational effort.

Bibliography

[1] R. Bar-Yehuda and S. Even, A linear time approximation algorithm for the weighted vertex cover problem, *Journal of Algorithms*, 2, 198-203, (1981).

[2] G.P. Barker and D. Carlson, Cones of diagonally dominant matrices, *Pacific J. of Math.*, 57, 15-32, (1975).

[3] R. Bellman, *Dynamic Programming*, Princeton University Press, (1957).

[4] R. Bellman and K. Fan, *On systems of linear inequalities in Hermitian matrix variables*, Proceedings of Symposia in Pure Mathematics, Vol. 7, AMS, (1963).

[5] D. Bertsimas and J.N. Tsitsiklis, *Introduction to Linear Optimization*, Athena Scientific, Belmont, Massachusetts, (1997).

[6] F. Bohnenblust, *Joint positiveness of matrices*, Manuscript, (1948).

[7] J.A. Bondy and U.S.R. Murty, *Graph Theory with Applications*, Macmillan, London and Elsevier, New York, (1976).

[8] A. Cayley, On the mathematical theory of isomers, *Philosophical Magazine, 67, 444*, (1874).

[9] V. Cerny, A thermodinamical approach to the traveling salesman problem: an efficient simulated annealing algorithm, *J. of Optimization Theory and Applications*, 45, 41-55, (1985).

[10] N. Christofides, Worst-Case Analysis of a New Heuristic for the Traveling Salesman Problem, *Management Science Research Report*, 388, Carnegie Mellon University, Pittsburg, PA, (1976).

[11] P. Crescenzi and V. Kann, *A compendium of NP-optimization problems*, http://www.nada.kth.se/nada/theory/problemlist.html.

[12] S.E. Dreyfus and A.M. Law, *The Art and Theory of Dynamic Programming*, Academic Press, Inc. Ltd., (1977).

[13] M. Dror and M. Haouari, Generalized Steiner Problems and Other Variants, *Journal of Combinatorial Optimization*, 4, 415-436, (2000).

[14] M. Dror, M. Haouari and J. Chaouachi, Generalized Spanning Trees, *European Journal of Operational Research*, 120, 583-592, (2000).

[15] J. Edmonds, Minimum partition of matroid into independent sets, *Journal of Research of the National Bureau of Standards B*, 69B, 67-72, (1965).

[16] U. Faigle, W. Kern and G. Still, *Algorithmic Principles of Mathematical Programming*, To appear in Kluwer Academic Publishers, (2002).

[17] C. Feremans, *Generalized Spanning Trees and Extensions*, Ph.D Thesis, Universite Libré de Bruxelles, Belgium, (2001).

[18] C. Feremans, M. Labbé and G. Laporte, *A Comparative Analysis of Several Formulations for the Generalized Minimum Spanning Tree Problem*, Technical Report IS-MG 2000/22, Universite Libré de Bruxelles, Belgium, (2000).

[19] C. Feremans, M. Labbé and G. Laporte, *Polyhedral Analysis of the Generalized Minimum Spanning Tree Problem*, Technical Report IS-MG 2000/01, Universite Libré de Bruxelles, Belgium, (2000).

[20] D.R. Fulkerson, Blocking and anti-blocking polyhedra, *Mathematical Programming*, 1(2), 168-194, (1971).

[21] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, California, (1979).

[22] N. Garg, G. Konjevod and R. Ravi, A Polylogarithmic Approximation Algorithm for the Group Steiner Tree Problem, *Journal of Algorithms*, 37, 66-84, (2000).

[23] A.M. Geoffrion, Lagrangian relaxation for integer programming, *Mathematical Programming Study*, 2, 82-114, (1974).

[24] M. Gerla and L. Frata, Tree structured fiber optics MAN's, *IEEE J. Select. Areas Comm.*, SAC-6, 934-943, (1988).

[25] F. Glover, Heuristics for Integer Programming using Surrogate Constraints, *Decision Sciences*, Vol. 8, 156-166, (1977).

[26] F. Glover, E. Taillard and D. Werra, A User's Guide to Tabu Search, *Annals of Operations Research*, Vol. 41, 3-28, (1993).

[27] M.X. Goemans, *Course Notes: Approximation Algorithms*, ftp://theory.lcs.mit.edu/pub/classes1.approx.ps, (1994).

[28] M.X. Goemans and Y.S. Myung, A Catalog of Steiner Tree Formulations, *Networks*, 23, 19-28, (1993).

[29] M.X. Goemans and D.J. Bertsimas, Survivable networks, linear programming relaxations and the parsimonious property, *Mathematical Pro-*

gramming, 60, 145-166, (1993).

[30] M. Grötschel, *Discrete mathematics in manufacturing*, Preprint SC 92-3, ZIB, (1992).

[31] M. Grötschel, L. Lovász and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, Springer-Verlag, Berlin, (1988).

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

[33] M. Held and R.M. Karp, The traveling salesman problem and minimum spanning trees, *Operations Research*, 18, 1138-1162, (1970).

[34] M. Held and R.M. Karp, The traveling salesman problem and minimum spanning trees: part II, *Mathematical Programming*, 1, 6-25, (1971).

[35] C.S. Helvig, G. Robins and A. Zelikovsky, An Improved Approximation Scheme for the Group Steiner Problem, *Networks*, 37, Issue 1, 8-20, (2001).

[36] A.L. Henry-Labordere, The Record Balancing Problem: A Dynamic Programming Solution of a Generalized Traveling Salesman Problem, *RIRO*, B-2, 43-49, (1969).

[37] J.B. Hiriart-Urruty and C. Lemaréchal, *Convex Analysis and Minimization Algorithms II*, Springer-Verlag, Berlin, (1993).

[38] D. Hochbaum, Approximation algorithms for set covering and vertex cover problems, *SIAM Journal of Computing*, 11, 555-556, (1982).

[39] J.H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, (1975).

[40] E. Ihler, G. Reich and P. Widmayer, Class Steiner Trees and VLSI-design, *Discrete Applied Mathematics*, 90, 173-194, (1999).

[41] D. Johnson, C. Aragon, L. McGeoch and C. Schevon, Optimization by Simulated Annealing: An Experimental Evaluation, Part I, Graph Partitioning, *Operations Research*, 37, 865-892, (1989).

[42] N. Karmarkar, A new polynomial-time algorithm in linear programming, *Combinatorica*, 4, 373-395, (1979).

[43] L.G. Khachiyan, A polynomial algorithm in linear programming, *Soviet Mathematics Doklady*, 191-194, (1979).

[44] S. Kirkpatrick, C.D.. Gelatt and M.D. Vecchi, Optimization by Simulated Annealing, *Science*, Vol. 220, No. 4598, 671-680, (1983).

[45] J.B. Kruskal, *On the shortest spanning subtree of a graph and the traveling salesman problem*, Proceedings of the American Mathematical Society 7, 48-50, (1956).

[46] G. Laporte, H. Mercure and Y. Norbert, Finding the Shortest Hamiltonian Circuit through n Clusters: a Lagrangian Approach, *Congressus Numerantium*, 48, 277-290, (1985).

[47] G. Laporte, H. Mercure and Y. Norbert, Generalized Traveling Salesman Problem through n Sets of Nodes: The asymmetrical Case, *Discrete Applied Mathematics*, 18, 185-197, (1987).

[48] C. Lemaréchal and F. Oustry, *Semidefinite relaxations and Lagrangian duality with application to combinatorial optimization*, Rapport de recherche, N. 3710, INRIA Rhône-Alpes, France, (1999).

[49] L. Lovász and A. Schrijver, Cones of matrices and set-functions and 0-1 optimization, *SIAM J. Optim.*, 1(2): 166-190, (1991).

[50] L. Lovász, On some connectivity properties of Eulerian graphs, *Acta Mathematica Academiae Scientiarum Hungaricae*, 28, 129-138, (1976).

[51] T.L. Magnanti and L.A. Wolsey, *Optimal Trees*, Handbooks in Operations Research & Management Science, M.O. Ball, T.L. Magnanti, C.L. Monma, G.L. Nemhauser (Editors), Elsevier Science, Amsterdam, Vol. 7, chap. 9, 503-615, (1995).

[52] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller and E. Teller, Equations of State Calculations by Fast Computing Machines, *Journal of Chemical Physics*, Vol. 21, No. 6, 1087-1092, (1953).

[53] R.D.C. Monteiro and S. Zhang, A unified analysis for a class of long-step primal-dual path-following interior-point algorithms for semidefinite programming, *Math. Programming*, 81, 281-299, (1998).

[54] Y.S. Myung, C.H. Lee and D.w. Tcha, On the Generalized Minimum Spanning Tree Problem, *Networks*, 26, 231-241, (1995).

[55] G.L. Nemhauser and L.A. Wolsey, *Integer and Combinatorial Optimization*, John Wiley & Sons, New York, Reprint of the 1988 original, (1999).

[56] I.H. Osman and G. Laporte, Metaheuristics: A Bibliography, *Annals of Operations Research*, 63, 513-623, (1996).

[57] C.H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, New Jersey, (1982).

[58] R.C. Prim, Shortest connection networks and some generalizations, *Bell Systems Technical Journal*, 36, 1389-1401, (1957).

[59] J.J. Prisco, Fiber optic regional area networks in New York and Dallas, *IEEE J. Select. Areas Comm.*, SAC-4, 750-757, (1986).

[60] P.C. Pop, U. Faigle, W. Kern and G. Still, *Relaxation methods for the*

generalized minimum spanning tree problem, To be submitted for publication.

[61] P.C. Pop, U. Faigle, W. Kern and G. Still, *The Generalized Minimum Spanning Tree Problem*, Proceedings of the 17th International Symposium on Mathematical Programming, Atlanta, Georgia, USA, 7-11 August, (2000).

[62] P.C. Pop, W. Kern and G. Still, *An Approximation Algorithm for the Generalized Minimum Spanning Tree Problem with bounded cluster size*, EIDMA 2001 Symposium, Oostende, Belgium, 25-26 October, 2001, Memorandum 1577, University of Twente, Enschede, the Netherlands, http://www.math.utwente.nl/dos/ormp/preprints.htm, (2001).

[63] P.C. Pop and G. Still, *An easy way to obtain strong duality results in linear, linear semidefinite and linear semi-infinite programming*, Memorandum 1493, University of Twente, Enschede, the Netherlands, http://www.math.utwente.nl/dos/ormp/preprints.htm, (1999).

[64] C.R. Reeves, *Modern Metaheuristics Techniques for Combinatorial Problems*, Blackwell, Oxford, (1993).

[65] G. Reich and P. Widmayer, *Beyond Steiner's Problem: A VLSI Oriented Generalization*, Lecture Notes in Computer Science 411, Springer, 196-210, (1990).

[66] R.T. Rockafellar, *Convex Analysis*, Princeton University Press, Princeton, N.J., (1970).

[67] C. Roos, T. Terlaky and J.-Ph. Vial, *Theory and Algorithms for Linear Optimization: An interior point approach*, John Wiley & Sons, New York, (1997).

[68] J.P. Saskena, Mathematical Model of Scheduling Clients Welfare Agencies, *Journal of the Canadian Operational Research Society*, 8, 185-200, (1970).

[69] A. Schrijver, *Theory of Linear and Integer Programming*, John Wiley & Sons, Chichester, (1986).

[70] P. Schuurman and G.J. Woeginger, *Approximation schemes - A tutorial*, Report, TU Graz, Austria, (2001).

[71] P. Slavik, *On the Approximation of the Generalized Traveling Salesman Problem*, Working Paper, University of Buffalo, (1997).

[72] S.S. Srivastava, S. Kumar, R.C. Garg and P. Sen, Generalized Traveling Salesman problem through n sets of nodes, *Journal of the Canadian Operational Research Society*, 7, 97-101, (1969).

[73] O. Taussky, Positive-definite matrices and their role in the study of the

characteristic roots of general matrices, *Advances in Math.*, 2, 175-186, (1968).

[74] M.J. Todd, A study of search directions in primal-dual interior-point methods for semidefinite programming, *Optim. Methods Softw.*, 11&12, 1-46, (1999).

[75] V.V. Vazirani, *Approximation Algorithms*, Springer Verlag, Berlin, (2001).

[76] H. Wolkowicz, R. Saigal, and L. Vandenberghe (editors). *Handbook on Semidefinite Programming*, Kluwer, (2000).

[77] B.J. Wythoff, Backpropagation Neural Networks. A Tutorial, *Chemometrics and Intelligent Laboratory Systems*, 18, 115-155, (2001).

[78] M. Yannakakis, Expressing combinatorial optimization problems by linear programs, *Journal of Computer and System Sciences*, 43, 441-466, (1991).

Subject Index

Summary

Classical combinatorial optimization problems can often be generalized in a natural way by considering a related problem relative to a given partition of the nodes into clusters.

We study various relaxation and heuristic techniques in the special case of the so-called Generalized Minimum Spanning Tree (GMST) problem. In the GMST problem given an undirected graph with nodes partitioned into clusters and edges defined between nodes belonging to different clusters with a nonnegative cost, we search for a minimum cost tree spanning a subset of nodes which includes exactly one node from each cluster. This problem was introduced by Myung, Lee and Tcha [54], they also proved that the problem is $\mathcal{NP}$*-hard*. The last result is a direct consequence of one of our results in which we prove that the GMST problem on trees is $\mathcal{NP}$*-hard*. We present also two cases when the GMST problem is solvable in polynomial time.

We present new integer and mixed integer programming formulations for the GMST problem and we compare them in terms of their linear programming relaxations with the formulations already proposed in the literature.

Based on a new formulation of the problem called local-global formulation, we present a new solution procedure. Our procedure called rooting procedure is compared with the procedures developed by Myung *et al.* [54] and by Feremans [17]. For all instances of the problem that we considered, the optimal solution of the GMST problem has been found. As far as we know, our method is the second method, after Feremans [17], which solves instances with more than 120 nodes to optimality, but the computing times spent by the branch and cut code proposed in [17] are in general larger than the times spent by our rooting procedure code.

We compare the bound provided by the linear programming relaxation of a general 0-1 programming problem with the bounds obtained by considering

the linear, complete and boolean dualizations suggested in [48]. We apply the theoretical results obtained to the GMST problem. In the Lagrangian relaxation that we considered, a set of inequalities in the bidirectional multicommodity flow formulation of the problem is dualized and added to the objective function. We used the subgradient optimization algorithm to obtain lower bounds. Computational results are reported for many instances of the problem.

We distinguish in Chapter 4 between so-called positive and negative results in the area of approximation algorithms. We present an in-approximability result: there is no α-approximation algorithm for the GMST problem unless $\mathcal{P} = \mathcal{NP}$. Our result is a formulation in terms of approximation algorithms of a result of Myung *et al.* [54]. However, under special assumptions (see Section 4.4) we give an approximation algorithm for the GMST problem.

In the last chapter of this thesis we present basic theory of heuristic algorithms and local search and solve the GMST problem with Simulated Annealing.