D I S S E R T A T I O N

On the Minimum Label Spanning Tree Problem

Solution Methods and Applications

ausgeführt zum Zwecke der Erlangung des akademischen Grades des Doktors der technischen Wissenschaften unter der Leitung von

ao. Univ.-Prof. Dipl.-Ing. Dr. Günther Raidl
Institut für Computergraphik und Algorithmen E186
Technische Universität Wien

und

ao. Univ.-Prof. Dipl.-Ing. Dr. Ulrich Pferschy
Institut für Statistik und Operations Research
Universität Graz

eingereicht an der Technischen Universität Wien
Fakultät für Informatik von

Mag. Dipl.-Ing. Andreas M. Chwatal
Matrikelnummer 9725777
Heinrich-Collin-Straße 8-14/13/14, 1140 Wien

Wien, am 28. Mai 2010

Andreas M. Chwatal

Kurzfassung

Das *Minimum Label Spanning Tree Problem* ist ein kombinatorisches Optimierungsproblem mit Anwendungen im Bereich des Designs von Telekommunikationsnetzwerken mit dem Ziel ein möglichst einheitliches Netzwerk hinsichtlich der verwendeten Übertragungseinrichtungen zu finden. Gegeben ist ein zusammenhängender Graph, wobei jeder Kante ein oder mehrere Label zugewiesen sind. Das Ziel ist die Bestimmung eines Spannbaumes welcher eine minimale Anzahl an Labels benötigt, sodass für jede gewählte Kante mindestens ein Label ausgewählt wird. Das Problem ist $\mathcal{NP}$-vollständig.

Die bisherige Forschung in Bezug auf das gegebene Problem war primär auf die Entwicklung approximativer und metaheuristischer Algorithmen ausgerichtet, aber auch einige exakte Verfahren wurden vorgeschlagen. Es wurde gezeigt, dass kein polynomieller Algorithmus mit konstantem Approximationsfaktor existieren kann (außer $\mathcal{P} = \mathcal{NP}$), jedoch sind heuristische und metaheuristische Algorithmen in der Lage hochqualitative Lösungen in angemessener Laufzeit zu finden. Exakte Verfahren können nur relativ kleine Probleminstanzen zu lösen, jedoch kann die Entwicklung fortgeschrittener Methoden durchaus dazu in der Lage sein die Grenze der praktisch lösbaren Instanzen deutlich in Richtung größerer Instanzen zu verschieben, und somit deren praktische Anwendung zu ermöglichen.

In dieser Dissertation werden exakte und heuristische Methoden für das Minimum Label Spanning Tree Problem und einige seiner Varianten betrachtet. Bezüglich heuristischer Verfahren stellt die bisher noch nicht untersuchte Anwendung von *Ant Colony Optimization* auf das gegebene Problem den Schwerpunkt dar. Weiters wird die Verwendung der heuristischen Verfahren als primale Heuristik zur Beschleunigung exakter Verfahren untersucht. Dann werden exakte Verfahren basierend auf gemischt-ganzzahliger linearer Programmierung genauer betrachtet. Hierbei werden einerseits existierende Formulierungen anhand neuer Klassen von gültigen Ungleichungen gestärkt, und des Weiteren neue Formulierungen vorgeschlagen. Weiters wird gezeigt wie fraktionale Lösungen der Relaxierung durch Weglassen der Ganzzahligkeitsbedingungen bezüglich der Label-Variablen mittels *Odd-Hole*-Ungleichungen separiert werden können. Für letztere wird

zur Verwendung in einem Schnittebenenverfahren eine heuristische Separierungsmethode basierend auf einem gemischt-ganzzahligen Programm mit Miller-Tucker-Zemlin Ungleichungen vorgestellt. Alle vorgestellten heuristischen und exakten Methoden wurden implementiert und mittels zahlreicher computationaler Tests ausgewertet. Die präsentierten Ergebnisse dokumentieren die Unterschiede zu bestehenden Verfahren, sowie die erzielten Verbesserungen. Insbesondere stellt ein *Branch-and-Cut* Algorithmus basierend auf einer neuen Formulierung die eine schnelle Schnittebenen-Separierung ermöglicht eine deutliche Verbesserung zu bestehenden exakten Verfahren dar. Die Anwendung der Odd-Hole Schnittebenen erweist sich für spezielle Instanzen als nützlich.

Der letzte Teil der Dissertation widmet sich einem neuen Ansatz zur Datenkompression, basierend auf Minimum Label Spanning Trees. Ziel ist die kompakte Repräsentation einer ungeordneten Menge von mehrdimensionalen Punkten. Der betrachtete Anwendungshintergrund kommt aus der Biometrie, wo Fingerabdrücke oftmals durch deren charakteristische Punkte ("Minutien") dargestellt werden. Um diese Daten als zusätzliches Sicherheitsmerkmal, z.B. in Reisepässen, verwenden zu können, werden spezielle Kompressionsverfahren benötigt um diese als Wasserzeichen in Passbilder einbetten zu können. Dafür kodieren wir eine Teilmenge der Minutien als Spannbaum, dessen Kanten wiederum durch eine Referenz auf ein Element einer kleinen Menge an "Template-Arcs" und kleinen Korrekturvektoren dargestellt werden. Dadurch werden geometrische Eigenschaften der Punkte für die Kompression benutzt indem ein Baum mit möglichst ähnlichen Kanten hinsichtlich der relativen Position von Anfangs- und Endknoten bestimmt wird. Dabei entsprechen mögliche Template-Arcs einer Kante den Labels des Minimum Label Spanning Tree Problems. Da nur eine Teilmenge der gegebenen Punkte an den Baum angeschlossen wird, besteht ein weiterer Zusammenhang zum *k-cardinality Tree Problem*, welches ebenfalls $\mathcal{NP}$-schwer ist. Das resultierende Optimierungsproblem wird als *k-node Minimum Label Spanning Tree* (k-MLSA) Problem bezeichnet.

Vor der Lösung des k-MLSA Problemes wird die Menge der *Candidate Template-Arcs* mittels eines Preprocessing-Schrittes von den Eingabedaten abgeleitet, was ebenfalls ein nicht-triviales Problem darstellt. Zu diesem Zweck wird ein Algorithmus basierend auf begrenzter Enumeration vorgestellt. Für das resultierende k-MLSA Problem werden Heuristiken wie *Greedy Randomized Adaptive Search Procedures* oder Genetischen Algorithmen, sowie exakten Verfahren wie Branch-and-Cut vorgschlagen. Weiters wird gezeigt, wie der relativ zeitaufwändige Preprocessing-Schritt direkt in das Branch-and-Cut Verfahren integriert werden kann, indem neue Template-Arcs während der Ausführung des Algorithmus dynamisch zu einem ursprünglich eingeschränkten Modell hinzugefügt werden. Derartige Ansätze sind bekannt als *Branch-and-Cut-and-Price*. Das *Pricing-Problem* entspricht der Bestimmung einer Template-Arc Variable welche die Zielfunktion potentiell verbessern kann, und wird anhand eines Algorithmus basierend auf einer k-d Tree Datenstruktur gelöst. Computationale Tests belegen die Eignung des präsentierten Kompressionsmodelles für die vorgeschlagene Anwendung. Darüber hinaus kann es als Ausgangspunkt für weitere Untersuchungen hinsichtlich der Kompression von intrinsisch ungeordneten Datenmengen, wie zum Beispiel ungleich verteilten Messwerten, dienen.

Abstract

The *minimum label spanning tree* problem is a combinatorial optimization problem with applications in telecommunication network design with the goal of deriving a network that is preferably uniform w.r.t. to the used communication facilities. For the minimum label spanning tree problem we are given a connected graph with labels associated to its edges. The goal is to derive a spanning tree requiring a minimum amount of labels in the sense that for each edge an according label must be selected. The problem has been shown to be $\mathcal{NP}$-complete.

So far, research has primarily been devoted to the development and analysis of approximation algorithms and heuristics, but also some exact solution methods have been proposed. It has been shown that no polynomial constant-factor approximation algorithms do exist (unless $\mathcal{P} = \mathcal{NP}$), but however, heuristic and metaheuristic algorithms are able to provide high quality solutions within a reasonable amount of computation time. Exact methods are only capable of solving relatively small instances in practice, but the development of elaborate methods may significantly shift the border of exactly solvable instances towards larger ones, enabling their application for practical purposes.

Within this thesis exact and heuristic methods for the minimum label spanning tree problem and some variations are investigated. Regarding metaheuristics, main emphasis is given to the application of ant colony optimization, which has not yet been considered for the given problem. Moreover, it is studied how these methods can be used as primal heuristics to speed up exact solution methods. In the following, exact methods based on mixed integer programming are investigated. Within this context existing formulations are strengthened by new classes of inequalities and new formulations are proposed. In particular it is shown how odd-hole inequalities can be used to cut-off fractional label solutions in order to tighten the linear programming relaxation. For the latter ones a heuristic separation procedure based on a mixed integer linear program using Miller-Tucker-Zemlin inequalities is proposed, to be used within a cutting-plane algorithm. All the presented heuristic and exact methods have been implemented and evaluated by comprehensive

computational experiments. Reported computational results show the differences and improvements to existing methods. In particular the branch-and-cut algorithm based on the new connectivity formulation, which enables a fast cutting-plane separation, significantly outperforms all existing exact methods. The application of odd-hole cutting-planes turned out to be beneficial for particular classes of instances.

The last part of this thesis is dedicated to a newly developed compression model, which is primarily based on the minimum label spanning tree problem. The particular goal is to derive a compact representation of a set of unordered multi-dimensional points. The considered application scenario comes from the field of biometrics, where fingerprint data is often encoded as a set of characteristic points ("minutiae") of the fingerprint pattern. In order to use this data as an additional security feature e.g. in passports, strong compression is required to be able to embed this data into images by watermarking techniques. For this purpose a subset of the minutiae is encoded by a spanning tree, whose edges are in turn represented by references into a small set of "template arcs" and small correction vectors. By this approach it is possible to extract geometric structure within the given points and obtain compression by deriving a tree that contains a maximum number of arcs that are similar w.r.t. the relative geometric position of their incident nodes. By identification of the template arcs with labels we obtain the correspondence to a variant of the minimum label spanning tree problem. As only a subset of nodes is required to be connected, the problem is also related to the k-cardinality tree problem, which is also $\mathcal{NP}$-hard. We call the resulting optimization problem k-node minimum label arborescence (k-MLSA) problem.

Prior to solving the k-MLSA problem a set of candidate template arcs needs to be derived in a preprocessing step from the input data, which is itself a non-trivial task. For this purpose an algorithm based on restricted enumeration is proposed. For the resulting k-MLSA problem heuristic methods including greedy randomized adaptive search procedures and genetic algorithms as well as exact methods including branch-and-cut are proposed. Finally it is shown, how the relatively time-consuming preprocessing step can be directly incorporated into the branch-and-cut algorithm by dynamically adding new promising template arcs to the initially restricted model during the branch-and-cut algorithm. Such approaches are well known as branch-and-cut-and-price. The pricing problem, i.e. the determination of a template arc variable potentially improving the current objective function, is solved by an algorithm based on a k-d tree data structure. The presented compression model is shown to be beneficial for the outlined application background, but may also serve as source for further investigations into the direction of compression models that benefit from intrinsically unordered data sets like for instance unequally distributed scientific measurements.

Acknowledgments

First of all, I want to express my gratitude to my supervisor Prof. Günther Raidl, who gave me the opportunity to do my PhD at the Vienna University of Technology. He introduced me into the field of combinatorial optimization and mixed integer programming, and gave me a great support and provided numerous valuable ideas and helpful advises. He also attained to form a great working atmosphere within the Algorithms and Data Structures Group of the institute. I am further grateful to Prof. Ulrich Pferschy who immediately agreed to be the second assessor of this thesis.

Furthermore I owe my gratitude to all of my colleagues at the Algorithms and Data Structures Group. During the last four years we had many fruitful discussions as well as a lot of fun! In particular I want to thank those colleagues who provided any kind of help and support for my work – which probably holds for all of them!

Of course I also want to thank all other people who gave my any kind of support for this theses, in particular to all my friends for their encouragement. Last, but not least, I owe deep thank to my family, for their support and and encouragement during the last years!

Contents

Chapter 1

Introduction

Dear friend, all theory is gray,
And green the golden tree of life.

Faust in J. W. Goethe's Faust, Part 1

his thesis deals with the application and analysis of exact and heuristic methods for the solution of *combinatorial optimization problems* (COPs). The central theme is the *minimum label spanning tree* (MLST) problem and related problems, which are combinatorial optimization problems falling into the category of *network design*. In general, the notion of network design refers to various problems emerging in the context of industrial and economic optimization scenarios, with the common goal of deriving an optimal configuration w.r.t. a potential infrastructure, minimizing particular costs given by a specific objective function. As optimization scenarios resulting from particular real world problems usually consist of many properties and issues requiring specific solution concepts being solely important for one particular application, academic research within this area is usually concerned with prototypical problems arising as subproblem or being basis for many real world optimization scenarios. Concepts and solutions methods devised for these problems can then be more or less easily be applied to particular application-specific problem variants.

Network design problems usually share the common property that they can be naturally formulated on *graphs*, which are general concepts in the field of discrete mathematics. A graph is given by a set of *nodes* (or *vertices*) that are connected by undirected *edges* or directed *arcs*. Both entities may have associated additional information like *costs/weights*, *capacities* or *labels/colors*. According to the considered network design problem these entities usually have corresponding real world objects, e.g. edges corresponding to streets and nodes to crossings, or edges and nodes corresponding to communication infrastructure and network hubs. The latter example emphasizes the importance for the whole telecommunication sector.

The well known *minimum spanning tree* (MST) problem is the notorious archetype of network design problems, being closely related to the MLST problem as well. For the MST problem we are given a set of nodes and a set of edges, each connecting two nodes. We assume that the given graph is connected, i.e. there are no isolated nodes. Furthermore we are given *costs* that are assigned to the given edges. The objective is to find a minimum weight subgraph connecting all the given nodes. It is easy to see, that the resulting subgraph is a *tree*, i.e. it contains no *cycles*. The MST problem has the desirable property that it can be efficiently solved, i.e. optimal solutions can be derived within polynomial time by prominent algorithms of Kruskal and Prim [70, 88]. Unfortunately, most network design problems do not have this property – on the contrary it can be shown that they cannot be optimally solved within polynomial time with a deterministic algorithm (unless $\mathcal{P} = \mathcal{NP}$).

1.1 Minimum Label Spanning Tree

The minimum label spanning tree (MLST) problem was first introduced in [16]. For the MLST problem we are given a graph $G = (V, E, l)$ with nodes $v \in V$ and edges $e \in E$ connecting pairs of nodes of the node set V. In addition a labelling function $l : E \to L$ is given, assigning to each edge an element "label" from a finite set L. The objective is to find a minimum cardinality label subset $L_T \subseteq L$ inducing a feasible spanning tree in the sense that for each edge in the spanning tree its corresponding label is selected. If $l : E \to 2^L$, i.e. more than one label can be assigned to an edge, the problem is also called *generalized* MLST which we will abbreviate by GMLST, cf. [17]. However, the original problem formulation from [16] does not explicitly forbid parallel edges (i.e. multigraphs), which corresponds to multiple labels assigned to single edges.

The MLST problem is known to be $\mathcal{NP}$-complete (the concept of $\mathcal{NP}$-completeness will be discussed in Section 2.3), the non-existence of a polynomial-time constant-factor approximation (unless $\mathcal{P} = \mathcal{NP}$) has been further shown in [16, 69], which will be both addressed in more detail in Section 3.1.

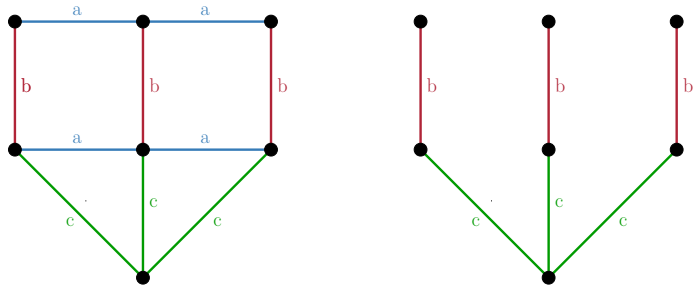

Figure 1.1: MLST instance (left) with corresponding optimal solution (right), with edges being colored in correspondence to their (single) assigned label.

By $L(e)$ we denote the subset of labels that are associated to edge $e \in E$, and $E(l)$ the edges having assigned label $l \in L$ accordingly. We call $|E(l)|$ the *label frequency*. For the nodes we define $L(v)$ to be the set of labels associated to all edges incident to node v, and $V(l)$ denoting the set of nodes that are incident to edges with label l. Further notation will be introduced on demand in the subsequent chapters.

Practical applications of the MLST problem do for instance exist within the context of telecommunication network design. Within these networks the particular nodes are connected by various types of communication medium like optical fiber, cable, microwave, telephone line [16]. The goal is to derive a network of maximal uniformity, i.e. a minimum amount of different connection facilities should be used. Another application scenario is proposed within this thesis: The goal is to compress an unordered set of multi-dimensional data, which is reduced to an extended version of the MLST problem. In Chapter 5 this topic is treated in detail.

1.2 Outline of the Thesis

The remainder of this thesis is organized as follows. Chapter 2 first gives a short introduction to graph theory, combinatorial optimization and complexity theory. Afterwards, prevalent methods and algorithms within the field of combinatorial optimization are presented, with particular focus on the techniques applied in the thesis. The applied methods can primarily be subdivided in the categories *exact* and *heuristic* algorithms.

Chapter 3 deals with heuristic solution methods, with primary focus on *metaheuristic* techniques applied to the MLST problem. Incipiently, previous work is reviewed and discussed. The heuristic methods considered within this thesis are then described in detail. Existing algorithms from literature have been implemented and tested with minor modifications and improvements, furthermore the application of a metaheuristic not yet applied to the MLST problem, namely *ant colony optimization*, is reported. Finally, results of computational experiments are presented and discussed. This chapter includes the ant colony optimization approach published in [21].

Chapter 4 then focuses on exact solution methods, in particular techniques based on *mathematical programming*. The chapter is based on [22]. Again, existing methods, primarily based on *mixed integer programming* are reviewed and discussed. New mixed integer programming formulations, based on *cutting-plane-separation* are proposed, the corresponding polyhedra are compared w.r.t. their linear relaxation and aptitude for *cutting-plane* algorithms. It is furthermore shown, how *odd-hole* inequalities can be applied to strengthen the MIP formulations and heuristic separation procedures for these inequalities are presented. Computational results and their discussion finally close this chapter.

Chapter 5 starts with a short discussion of potential real world applications of the MLST problem and its variants. The main part of this chapter is devoted to a new compression model based on the MLST problem. In particular, this compression model is adequate for small sets of unordered weakly structured data. The particular application background arises within the context of biometrics: For many purposes fingerprint data is represented

by *minutiae* which are characteristic points within the fingerprint image. In order to use such minutiae data as additional security feature for e.g. passport images, highly specialized compression mechanisms are required. After describing the particular compression model, various exact and heuristic methods for the solution of the resulting optimization problem are presented and evaluated. The chapter is mainly based on [24], the section regarding branch-and-cut-and-price is based on [25]. Earlier stages of research related to the topic of this chapter have been published in [23, 19, 92].

Conclusive remarks in Chapter 6 finally complete the thesis.

Chapter 2

Theory and Methodologies

There is nothing as practical as a good theory.
Kurt Lewin

Our best theories are not only truer than common sense,
they make far more sense than common sense does.
David Deutsch, The fabric of reality (1997)

As a profound basis for the subsequent chapters, underlying theory and methodologies are reviewed in this chapter. Starting point is a short introduction to basic concepts from graph theory with a definition of according notation. Subsequently a brief introduction to combinatorial optimization and complexity theory is given. The main part of this chapter is devoted to solution methods constituting the primary toolbox for solving intractable problems within the field of combinatorial optimization. These methods can be subdivided into *exact* and *heuristic methods*, each being addressed in a distinct section. Primary focus is given to the methods that are used and applied within the remainder of this thesis.

2.1 Graphs

The probably first introduction of the concept of graphs dates back to the 16th century, where Leonhard Euler modeled the famous problem of finding a walk through the city Königsberg[1], which is situated on the Pregel river, crossing every bridge exactly once. Euler simplified the problem by reducing all land masses to *nodes*, and bridges to *edges* connecting these nodes accordingly. By analyzing the resulting formal model he showed

[1] Today, the city is named Kaliningrad and is part of Russia.

the inexistence of a walk through the graph that uses each edge exactly once [45]. Sequences of edges fulfilling this property are nowadays called *Eulerian Trails* or *Tours* (if they have the same start and end node). Within the beginning of the 20th century graphs and their properties attained more interest, and the mathematical discipline of *graph theory* emerged, with Dénes König being one of the first pioneers. Nowadays graph theory is an important subdiscipline of discrete mathematics, as many problems having practical and theoretical importance respectively, can be formulated by means of graphs. In particular for computer science, graphs are an integral part for theory as well as for practical applications as many algorithms are based on graph data structures.

Formally, a graph $G = (V, E)$ is given by a finite set of *nodes* or *vertices* $v \in V$ together with a set of edges $e = \{v_1, v_2\} \in E$, $v_1, v_2 \in V$. If edges are oriented, they are called *arcs* and are denoted by ordered pairs $a = (v_1, v_2) \in A$, $A \subseteq V \times V$. Within many contexts it is more convenient to denote the vertices by their indices, i.e. $i \in V$ instead of $v_i \in V$ and therefore $a = (i, j)$ instead of $a = (v_i, v_j)$. If a graph contains only edges it is called an *undirected graph*, if it contains only arcs, it is called an *directed graph* (or simply *digraph*), if it contains both edges and arcs it is called a *mixed graph*. Within this thesis only *simple graphs* containing no self-loops $\{v_i, v_i\}$ or (v_i, v_i) and not having multiple edges or arcs between every pair of nodes are considered. However, for directed graphs the existence of arcs (v_i, v_j) and (v_j, v_i), $v_i, v_j \in V$ is permitted. Two nodes are called *adjacent* if an edge or arc connecting these two nodes exists. A node v is called *incident* to an edge $e \in E$ or arc $a \in A$ and vice versa if $v \in e$, or $v \in a$ respectively. Besides the specification of a graph as a set of nodes and edges/arcs, it can also be described by an $|V| \times |V|$ *adjacency matrix*, say M, where each element m_{ij} of M is one if an edge/arc from node v_i to node v_j exists, and zero otherwise. As this matrix is symmetric for undirected graphs it is sufficient to provide an (upper) triangular matrix.

A graph $G' = (V', E')$ with $V' \subseteq V$, $E' \subseteq E$ and V' containing all incident nodes of E' is called a *subgraph* of graph $G = (V, E)$. The same definition remains valid for directed graphs.

Graphs containing only relatively few edges in the order of magnitude of the number of nodes are called *sparse*, otherwise they are called *dense*. If the edges or arcs fully connect all nodes $v \in V$, the graph is called *complete*, and it is often denoted by K_n with $n = |V|$. Hence, a complete undirected graph has $n \cdot (n-1)/2$ edges, a digraph $n \cdot (n-1)$ arcs. A graph is called *bipartite* if the node set V can be partitioned in two disjoint sets $V_1 \cup V_2 = V$ such that each edge or arc connects a node from V_1 to one from V_2, but no edges connecting nodes within one particular set do exist.

A sequence of nodes $v_0, v_1, \ldots, v_k, k \geq 1$, is called a *walk* if $\{v_{i-1}, v_i\} \in E$ for $i = 1, \ldots, k$. If all edges are distinct the walk is called *trail*. If furthermore no node repetitions do occur on the trail, we have a *path*. The *length* k of a walk/trail/path is the number of its edges. The definitions of walk, trail and path do also naturally apply for directed graphs. A walk $v_0, v_1, \ldots, v_k$ is *closed* if $v_0 = v_k$. A closed path with $k \geq 2$ is called a *cycle*. A graph that does not contain cycles is called *acyclic*.

A *connected component* is a subset $V' \subseteq V$ with the property that a path exists between all pair of nodes of V'. In the case of directed graphs we need to distinguish between

weakly and *strongly connected components*. A subset V' is said to be weakly connected if the underlying undirected graph obtained by ignoring the directions of the arcs, is connected. It is further said to be strongly connected, if for all pairs of vertices $v_1, v_2 \in V'$ directed paths from v_1 to v_2 and vice versa exist. By κ we denote the number of connected components, isolated vertices are counted as component as well. If $\kappa = 1$ the graph is called *connected*.

The number of incident edges to a vertex v is denoted by its degree $d(v)$. For directed graphs we further distinguish the indegree $d^-(v)$ and outdegree $d^+(v)$ denoting the number of incoming and outgoing edges, respectively. A complete undirected graph therefore has $d(v) = |V| - 1$ for all $v \in V$, and $d^-(v) = d^+(v) = |V| - 1$ in the directed case.

A *cut* C is defined by a subset $S \subset V,\ S \neq \emptyset$, that induces a partitioning $C = (S, \bar{S})$ of the graph, where $\bar{S} = V \backslash S$. The set of edges or arcs with one node from set S and the other one from set $\bar{S}$ is called *cutset*.

An acyclic graph is also called a *forest*, if it further is connected it is called a *tree*. Given a connected graph G it is often intended to find a tree $T = (V, E'),\ E' \subseteq E,\ |E'| = |V| - 1$, that is optimal w.r.t. some objective function. Such trees are called *spanning trees*. All nodes with $d(v) = 1$ of a tree are called *leaves*. A directed acyclic graph is called *arborescence* if it has $|V| - 1$ arcs and there exists a *root* node such that there is a directed path to each of the other vertices. A complete undirected graph K_n has exactly n^{n-2} spanning trees. This result is known as *Cayley's formula* [14, 26]. The number of spanning trees for the more general case $|E| < {}^{n\cdot(n-1)}/_2$ is, by *Kirchhoff's matrix tree theorem*, given by the absolute value of the determinant of any $(|V| - 1) \times (|V| - 1)$ submatrix of the Laplacian-matrix which is given by the difference of the degree-matrix and the adjacency matrix of the graph [57].

When given a weighting function $w : E \to \mathbb{R}$, we can define the *minimum spanning tree* problem, which asks for a spanning tree having minimum weight. Although the solution space is exponentially large it is possible to compute the optimal solution efficiently. However, considering other objective functions usually makes the problem intractable, which is addressed in more detail in Section 2.3. Other fundamental problems within the area of graph theory are concerned with traversing graphs, like shortest paths problems, the Hamilton-cycle problem, the problem of finding Eulerian circuits, and the Chinese Postman problem, to mention just some famous ones. Further problems deal with certain colorings of graphs, flows in graphs as well as the determination of certain subgraphs with special properties. For a comprehensive introduction to the topic the reader is referred to extensive literature of the topic, as for instance [4].

2.2 Combinatorial Optimization

Optimization is generally considered with solving the problem of finding extreme points like maxima and minima of certain functions. Extreme points of polynomial functions with one variable can be easily found by elementary mathematics. The more general case consists of an *objective function* of several variables, and additional constraints, restricting

the space of feasible solutions. The case of linear objective functions and restrictions defined for real variables, which can be solved within polynomial time, is discussed in more detail in Section 2.4.1. Contrary, no general efficient techniques exist for finding global optima of nonlinear multi-dimensional functions.

If the set of feasible solutions is finite, the problem is called a *discrete optimization problem*. If the objective function itself is restricted to a discrete domain like integers or elements from a finite set we have a *combinatorial optimization problem* (COP). Combinatorial optimization is thus a branch of on the one hand combinatorics, itself being a branch of discrete mathematics, and optimization on the other hand.

Although the number of solutions to a COP is finite, it is usually exponentially large in dependence of the size of the input instance. Therefore combinatorial optimization mainly deals with the development of algorithms that perform better than the complete enumeration of the solution space, which is usually not possible for problem-sizes of interest.

As graphs constitute objects of discrete mathematics, many problems defined on graphs naturally fall into the category of combinatorial optimization. Furthermore a plenitude of other problems are studied within this field. Scientific importance of the area is based on numerous COPs arising in industry like telecommunication network design, routing, facility location, assignment, scheduling, and manufacturing. Besides this, combinatorial optimization provides many insights and methodologies being important for mathematics, physics and computer sciences. Within the latter one, COPs arise within many subdisciplines dealing with e.g. programming languages, compilers, automata, cryptography, and automated proving.

The majority of COPs cannot be solved efficiently, as they are known to be $\mathcal{NP}$*-hard*, which is addressed in the following section. The remainder of this chapter is devoted to theoretical foundations and methodologies within the area of combinatorial optimization.

An extensive introduction to combinatorial optimization can be found in [84], further suggestions are [71, 87, 32, 100, 72].

2.3 Complexity Theory

In Section 2.1 we already introduced the minimum spanning tree (MST) problem. Despite the exponentially large number of feasible solutions, optimal solutions can be found *efficiently* for this particular problem. By *efficiently* we mean that the number of computational steps is bounded by a polynomial of the size of the input instance. However, this beneficial property does not hold for many other problems, as for them no algorithms are known that only require polynomial time. Complexity theory mainly deals with the classification of particular problems according to their *hardness*.

A first prerequisite for the analysis of the "complexity" is a hardware-independent model of computation. Alan Turing provided such a model, which is now known as *Turing machine* – not a machine in a physical sense, but only a gedankenexperiment. Such a machine consists of an infinite *tape* with discrete reading and writing positions, a *read-write head*

and a *finite state control*. A program for such a *deterministic one-tape Turing machine* (DTM) consists of a finite alphabet (finite set of symbols), a finite set of states (with distinguished start- and halt-state), and an according *transition function*. If the machine reads some particular symbol in some particular state, the transition function determines in which state to move next, which symbol to write on the tape, and if to proceed with the next leftmost or rightmost cell on the tape. The important thing about the Turing machine is that it can solve any problem that can be solved on computers, as all basic operations can be simulated by the basic operations of the read-write head. Based on this computational model we can now define the class $\mathcal{P}$ to be the class of all problems (stated in terms of decision problems) for which a polynomial time program for the DTM exists. More precisely, the number of steps is a polynomial within the length of the input sequence.

A further important class arises when introducing indeterminism to the Turing machine model. This is achieved by the introduction of a *guessing*-module, which in a first step writes a guessed string on the tape, such that the input string is not overwritten. If this process has stopped, the NDTM continues its execution in the same way as the DTM, which is called the *checking*-stage. To see that this nondeterministic (one-tape) Turing machine (NDTM) is more powerful than its deterministic counterpart we consider the situation where the probabilistic nature of the process comes to the correct decision in each step, i.e. the machine performs a correct *guess* within each step. Based on this model, the complexity class $\mathcal{NP}$ is defined as the class of all problems for which a polynomial time NDTM program does exist. In other words, all problems for which a polynomial-time nondeterministic *"guess-and-check"* algorithm does exist, fall into the class $\mathcal{NP}$. Hence, solutions to problems within $\mathcal{NP}$ can at least be *verified* in polynomial time, even if deterministic *solution* methods are unknown.

Central concepts of nowadays complexity theory have been proposed by S. A. Cook in the seminal paper [31]. Particular attention is paid to *decision problems*, which only have "yes" or "no" as result. Note that each optimization problem, say minimization problem, can be formulated as a decision problem, by asking the question: does a solution with objective value lower than some specified value exist? Furthermore Cook introduced the concept of *polynomial reduction*. A polynomial time transformation converting problem A into problem B therefore allows to use any algorithm for B for problem A. Cook proved that each problem within $\mathcal{NP}$ can be reduced to the satisfyability problem (SAT). Consequently, if SAT can be solved within polynomial time, every problem in $\mathcal{NP}$ can be solved in polynomial time. In [64] Karp proposed the famous 21 problems, which are in this sense equivalent to SAT. Such problems form the "hardest" class of problems within $\mathcal{NP}$ and are called $\mathcal{NP}$*-complete*. All problems to which $\mathcal{NP}$-complete problems can be reduced, even if they are not part of class $\mathcal{NP}$, are called $\mathcal{NP}$*-hard*[2]. It is still an unsolved problem to prove or disprove if $\mathcal{P} \neq \mathcal{NP}$. It is however widely believed that $\mathcal{P} \neq \mathcal{NP}$, as otherwise polynomial time algorithms for $\mathcal{NP}$-complete problems would exist, which have however not be found so far, even though much efforts have been undertaken.

[2]The famous Halting problem is $\mathcal{NP}$-hard and does not belong to class $\mathcal{NP}$.

Many problems within combinatorial optimization are $\mathcal{NP}$-complete. A comprehensive introduction into the area of complexity theory can be found in [49].

2.4 Exact Methods

Although some COPs can be solved to optimality within polynomial time, the vast majority falls in the category of $\mathcal{NP}$-complete or $\mathcal{NP}$-hard problems. For such problems, already for relatively small instance sizes the *brute-force* approach, usually performed by completely enumerating all possible solutions, fails due to memory and/or execution time limitations. However, performing such an enumerative approach in a more intelligent way, which permits to rule out certain branches of the enumeration tree soon, significantly shifts the border of solvable instances towards larger ones. Such an approach is usually called branch-and-bound (B&B), where in each intermediate node of the enumeration tree the problem is split into subproblems by restricting the domain of certain decision variables. The availability of methods to compute bounds is crucial in order to traverse the tree in a prospective way, and to effectively prune branches leading to suboptimal solutions. Branch-and-bound based on *linear programming* (LP) plays a crucial role within the area of combinatorial optimization, as it turned out to work efficiently for numerous problems.

After a short introduction to linear programming, the following sections are primarily devoted to (mixed) integer linear programming and related methods as LP-based branch-and-bound, branch-and-cut, branch-and-price, and branch-and-cut-and-price. In Section 2.4.3 we finally give a short overview about further methods being of high relevance within the combinatorial optimization area. For a broad introduction to the area of integer programming and related methods the reader is referred to [84, 9, 99, 72] which also have been primary resources for the remainder of this section.

2.4.1 Linear Programming

The general linear program (LP) is given by

$$\max \quad \sum_{j=1}^{n} c_j x_j \tag{2.1a}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} a_{ij} x_j \geq b_i \qquad \text{for } i = 1, \ldots, m \tag{2.1b}$$

$$x_j \geq 0 \qquad \text{for } j = 1, \ldots, n \tag{2.1c}$$

with constants $b_i, c_j \in \mathbb{R},\ i = 1, \ldots, m,\ j = 1, \ldots, n$, or equivalently in vector notation

$$\begin{aligned} \max \quad & \boldsymbol{c}^\mathsf{T}\boldsymbol{x} && \text{(2.2a)} \\ \text{s.t.} \quad & \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b} && \text{(2.2b)} \\ & \boldsymbol{x} \in \mathbb{R}^n_+, && \text{(2.2c)} \end{aligned}$$

where $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, $\boldsymbol{b} \in \mathbb{R}^m$, and $\boldsymbol{c} \in \mathbb{R}^n$ are given, and $\boldsymbol{x} \in \mathbb{R}^n$ should be found. As shorthand for the LP defined by (2.1a) or (2.2a) we will also write

$$z^{\mathrm{LP}} = \max\{\boldsymbol{c}^\mathsf{T}\boldsymbol{x} \mid \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \in \mathbb{R}^n_+\}, \tag{2.3}$$

with z^{LP} denoting the objective value of the optimal solution. The problem is *well-defined* if the objective function is bounded and feasible solutions do exists. In this case, the problem has an optimal solution. By (2.3) we have defined the *standard-form* of an LP, to which any other LP can be transformed. Maximization problems can be transformed into minimization problems by multiplying the objective function by -1. Equalities can be modeled by imposing two corresponding inequalities. It may also be convenient to formulate the LP in terms of equalities, rather then inequalities. This *slack form* (also called augmented form) can be obtained by the introduction of m additional *slack variables* $x_{n+i},\ i = 1, \ldots, m$, enabling to write each particular inequality as equality.

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} c_j x_j && && \text{(2.4a)} \\ \text{s.t.} \quad & b_i - \sum_{j=1}^{n} a_{ij} x_j = x_{n+i}, && \text{for } i = 1, \ldots, m, && \text{(2.4b)} \\ & \boldsymbol{x} \in \mathbb{R}^{n+m+1}_+. && && \text{(2.4c)} \end{aligned}$$

Duality

To each LP, a *dual problem* exists, being strongly connected to the original *primal* problem. A vivid interpretation of the dual problem comes from asking for an upper bound (or lower bound in the case of minimization) to the primal problem. This can be achieved by expressing $\boldsymbol{c}$ as a positive linear combination of Inequalities (2.18c), the restrictions, with coefficients $y_i \geq 0,\ i = 1, \ldots, m$. We thus obtain

$$\boldsymbol{c}^\mathsf{T}\boldsymbol{x} = \left(\sum_{i=1}^{m} y_i a_i\right)^\mathsf{T} \boldsymbol{x} = \sum_{i=1}^{m} y_i \left(a_i{}^\mathsf{T}\boldsymbol{x}\right) \leq \sum_{i=1}^{m} y_i b_i, \tag{2.5}$$

with a_i denoting the i-th column of matrix $\boldsymbol{A}$. We obtain the best bound by minimizing the last term in (2.5), again being a linear program. Hence, if the primal problem is given by

$$\boldsymbol{z}^{\mathrm{LP}} = \max\{\boldsymbol{c}^\mathsf{T}\boldsymbol{x} \mid \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \in \mathbb{R}^n_+\}, \tag{2.6}$$

the corresponding dual problem is

$$w^{\mathrm{LP}} = \min\{\boldsymbol{y}^\mathsf{T}\boldsymbol{b} \mid \boldsymbol{y}^\mathsf{T}\boldsymbol{A} \geq \boldsymbol{c}^\mathsf{T}, \boldsymbol{y} \in \mathbb{R}^m_+\}. \tag{2.7}$$

Proposition 1 *The dual of the dual problem is the primal problem.*

As we have already seen, solutions to one of these problems can be used to obtain bounds for the corresponding dual problem.

Proposition 2 (Weak Duality) *If $\boldsymbol{x}$ is primal feasible and $\boldsymbol{y}$ is dual feasible, then*

$$\boldsymbol{c}^\mathsf{T}\boldsymbol{x} \leq \boldsymbol{y}^\mathsf{T}\boldsymbol{b}.$$

It directly follows, that if the primal problem is unbounded, then the dual is infeasible. The following central theorem within the field of linear programming provides a mean of certification of optimality of a particular solution.

Theorem 1 (Strong Duality) *If z^{LP} or w^{LP} is finite, then both (the primal and the dual) problem have finite optimal value $z^{\mathrm{LP}} = w^{\mathrm{LP}}$.*

A further important property if the primal-dual pair is given by the following proposition.

Proposition 3 (Complementary Slackness) *Let $\boldsymbol{x}$ and $\boldsymbol{y}$ be feasible solutions for the primal and dual problem respectively, then $\boldsymbol{x}$ and $\boldsymbol{y}$ are optimal solutions if and only if*

$$y_i(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x})_i = 0, \qquad \textit{for all } i, \textit{ and}$$

$$x_j(\boldsymbol{y}^\mathsf{T}\boldsymbol{A} - \boldsymbol{c}^\mathsf{T})_j = 0, \qquad \textit{for all } j.$$

Polyhedra

Let $\boldsymbol{x}_k, k \in \mathbb{N}$, be a finite set of points in $\mathbb{R}^n$. A point $\boldsymbol{x} = \sum_{k\in N} \lambda_k \boldsymbol{x}_k,\ \lambda_k \in \mathbb{R}$ is called a *linear combination* of points $\boldsymbol{x}_k$. It is further called

- *conical*, if $\lambda_k \geq 0,$ for all $k \in N$,
- *affine*, if $\sum_{k\in N} \lambda_k = 1$, and
- *convex*, if it is both conical and affine.

The points $\boldsymbol{x}_k$ are said to be *linearly independent*, if $\sum_{k\in N} \lambda_k \boldsymbol{x}_k = 0 \Rightarrow \lambda_k = 0,$ for all $k \in \mathbb{N}$, i.e. the zero vector has only the trivial representation. A set $X \subset \mathbb{R}^n$ is a *convex set*, if it is closed under finite convex combinations. The set of all finite convex combinations of points in X is called *convex hull* and denoted by $\mathrm{conv}(X)$. Analogously we can define the *linear span* ($\mathrm{sp}(X)$), *conical hull* ($\mathrm{cone}(X)$) and the *affine span* ($\mathrm{aff}(X)$).

Theorem 2 (Minkowski, Weyl) *For a subset $P \subset \mathbb{R}^n$, the following statements are equivalent.*

1. $P = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}$, *with* $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$.
2. $P = \text{conv}(V) + \text{cone}(E)$

Such a set P is called a polyhedron.

From this we can see, that every LP defines a polyhedron, as a set of points satisfying a finite number of linear inequalities is a polyhedron.

Definition 1 *If $P \subseteq \mathbb{R}^n$ is* bounded, *i.e. there exists an $\omega \in \mathbb{R}_+$ such that $\forall x \in P \mid -\omega \leq x_j \leq \omega$ for $j = 1, \ldots, n$, it is called a* polytope.

Proposition 4 *A* polytope *is the convex hull* $\text{conv}(V)$ *for some finite set $V \subset \mathbb{R}^n$.*

In the following we well consider the geometric entities, which can be used to describe the boundary of a polyhedron.

Definition 2 (Hyperplane) *The set $\{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{a}^\mathsf{T}\boldsymbol{x} = a_0\}$ for $\boldsymbol{a} \in \mathbb{R}^n \backslash \{0\}$, $a_0 \in \mathbb{R}$ is called a* hyperplane.

Definition 3 (Valid Inequality) *The inequality $\boldsymbol{a}^\mathsf{T}\boldsymbol{x} \leq a_0$ (also denoted by $(\boldsymbol{a}, a_0)$) is called a* valid inequality *for P if it is satisfied by all points of P.*

Hence, each valid inequality defines a hyperplane with the polyhedron P entirely lying on one side of it. Particularly important valid inequalities are the ones, directly describing the boundary of the polyhedron and are called *facets*.

Definition 4 (Face) *If $(\boldsymbol{a}, a_0)$ is a valid inequality for P, and $F = \{\boldsymbol{x} \in P \mid \boldsymbol{a}^\mathsf{T}\boldsymbol{x} = a_0\}$, F is called a* face *of P.*

Definition 5 (Facet) *A face F of P is a* facet *of P if* $\dim(F) = \dim(P) - 1$.

Important special cases of faces are *corner points*.

Definition 6 (Corner Point) *Each facet F with* $\dim(F) = 0$ *is called* corner point.

Using these definition, we can already deduce a nice geometric interpretation of finding optimal solutions to feasible LPs. We already showed, that each LP corresponds to a polyhedron. From Definition 2 we can see, that the objective function together with an arbitrary objective function value defines a hyperplane. As each interior point of the polyhedron corresponds to a feasible solution, each nonempty intersection with the objective function hyperplane $\boldsymbol{c}^\mathsf{T}\boldsymbol{x} = c_0$ corresponds to a feasible solution with objective value c_0.

Finding optimal solutions of an LP can thus be interpreted as shifting the objective function hyperplane towards larger values c_0, as long as the intersection with the polyhedron remains nonempty. From this we can already see that any optimal solution lies on the boundary of the polyhedron.

The main workhorse for solving LPs is the *Simplex* algorithm, to be described in the following section. Although the Simplex method has exponential worst case running time, it is usually preferred over other existing polynomial methods (like for instance the Ellipsoid method), as it shows far superior performance for a plenitude of problems in practice, and the worst case barely occurs.

Simplex Algorithm

The main principle of the Simplex algorithm is to iteratively move from one corner point of the polyhedron to a neighboring one having higher objective value. This is basically achieved by changing the *basis* within each step. We start our description by first presenting the main steps of the algorithm and then discussing some related issues in more detail afterwards, essentially following the description in [72].

Let us consider the LP brought into standard slack form, only consisting of equality constraints. For this purpose also the objective function is modeled as an equality, by introducing the additional variable x_0.

$$\begin{aligned}
\max \quad & x_0 && && \text{(2.8a)}\\
\text{s.t.} \quad & x_0 - \sum_{j=1}^{n} c_j x_j = 0 && && \text{(2.8b)}\\
& \sum_{j=1}^{n} a_{ij} x_j = b_i && \text{for } i = 1, \ldots, m && \text{(2.8c)}\\
& x_j \geq 0 && \text{for } j = 1, \ldots, n && \text{(2.8d)}\\
& && && \text{(2.8e)}
\end{aligned}$$

This is no loss of generality, as each LP can easily be transformed in this particular form. In order to model the objective function as an equality, variable x_0 has been introduced. The indices $\{1, \ldots, n\}$ can now be partitioned into basic indices $\beta = (\beta_1, \ldots, \beta_n)$ and nonbasic indices $\eta = (\eta_1, \ldots, \eta_{n-m})$.

Definition 7 (Basic Solution) *A basic solution $\boldsymbol{x}^*$ is a feasible solution, where $n-m$ variables $x^*_{\eta_1} = x^*_{\eta_2} = \ldots = x^*_{\eta_{n-m}}$ are set to zero.*

It follows, that $x^*_{\beta_1}, x^*_{\beta_2}, \ldots, x^*_{\beta_m}$ is the unique solution of the remaining system

$$\sum_{j=1}^{m} a_{i\beta_j} x_{\beta_j} = b_i, \quad \text{for } i = 1, \ldots, m,$$

or, equivalently in matrix notation $\boldsymbol{x}^*_\beta = \boldsymbol{A}_\beta^{-1}\boldsymbol{b}$.

The dual program of (2.8) is given by

$$\begin{aligned} \min \quad & \boldsymbol{y}^\mathsf{T}\boldsymbol{b} && \text{(2.9a)} \\ \text{s.t.} \quad & \boldsymbol{y}^\mathsf{T}\boldsymbol{A} \geq \boldsymbol{c}^\mathsf{T}. && \text{(2.9b)} \end{aligned}$$

The *basic dual solution* associated with basis β is uniquely defined by

$$\boldsymbol{y}^{*\mathsf{T}} = \boldsymbol{c}_\beta{}^\mathsf{T}\boldsymbol{A}_\beta^{-1}. \tag{2.10}$$

Definition 8 (Reduced Costs)

$$\bar{\boldsymbol{c}}_\eta := \boldsymbol{c}_\eta - \boldsymbol{y}^{*\mathsf{T}}\boldsymbol{A}_\eta \tag{2.11}$$

If all reduced costs $\bar{c}_{\eta_j}$ are nonpositive, then $\boldsymbol{y}^*$ is feasible to the dual problem. If $\boldsymbol{x}^*$ is primal feasible, we can write the objective in terms of basic variables $\boldsymbol{c}^\mathsf{T}\boldsymbol{x}^* = \boldsymbol{c}_\beta{}^\mathsf{T}\boldsymbol{x}^*_\beta$. Due to Equations (2.8d) we can write $\boldsymbol{x}^*_\beta$ as $\boldsymbol{A}_\beta{}^{-1}$. Due to Equation 2.10 we further obtain $\boldsymbol{c}_\beta{}^\mathsf{T}\boldsymbol{A}_\beta{}^{-1}\boldsymbol{b} = \boldsymbol{y}^{*\mathsf{T}}\boldsymbol{b}$. This reasoning yields to the following theorem:

Theorem 3 (Weak Optimal-Basis Theorem) *If the basis is both primal feasible and dual feasible, then it is optimal.*

It is further possible to derive the following result:

Theorem 4 (Strong Optimal-Basis Theorem) *If the primal and corresponding dual problem are both feasible, then there exists a basis that is both primal and dual feasible, and therefore optimal.*

The Simplex algorithm starts with a primal feasible solution, and within each step determines a new element to enter the basis, and one to leave the basis. This approach corresponds to following a path moving from one corner point of the polyhedron to another one, having equal or better objective value, finally ending up in the optimal corner point. The operations are carried out on the *Simplex tableau*. Linear program (2.8) in tableau form is given by:

$$\begin{array}{ccc} x_0 & x & \text{r.h.s.} \\ \hline 1 & -c & 0 \\ 0 & \boldsymbol{A} & \boldsymbol{b} \end{array} \tag{2.12}$$

In this table "r.h.s" stands for right hand side. If we write this tableau in terms of basic and non-basic variables we obtain:

$$\begin{array}{cccc} x_0 & x_\beta & x_\eta & \text{r.h.s.} \\ \hline 1 & 0 & -c_\eta & \boldsymbol{c}_\beta{}^\mathsf{T}\boldsymbol{x}^*_\beta \\ 0 & \mathbb{1} & \boldsymbol{A}_\beta^{-1}\boldsymbol{A}_\eta & \boldsymbol{A}_\beta^{-1}\boldsymbol{b} \end{array} \tag{2.13}$$

Here we have moved $\boldsymbol{c}_\beta{}^\mathsf{T}\boldsymbol{x}^*_\beta$ to the right hand side, and denote the unity matrix, being the submatrix of $\boldsymbol{A}$ w.r.t. the basic variables, by $\mathbb{1}$. We can now solve for the basic variables and replace $-c_\eta$ with the reduced costs $\bar{\boldsymbol{c}}_\eta := \boldsymbol{c}_\eta - \boldsymbol{y}^{*\mathsf{T}}\boldsymbol{A}_\eta = \boldsymbol{c}_\beta\boldsymbol{A}_\beta^{-1}\boldsymbol{b}$.

$$\begin{array}{cccc} x_0 & x_\beta & x_\eta & \text{r.h.s.} \\ \hline 1 & 0 & -c_\eta + \boldsymbol{c}_\beta\boldsymbol{A}_\beta^{-1}\boldsymbol{A}_\eta & \boldsymbol{c}_\beta\boldsymbol{A}_\beta^{-1}\boldsymbol{b} \\ 0 & \mathbb{1} & \boldsymbol{A}_\beta^{-1}\boldsymbol{A}_\eta & \boldsymbol{A}_\beta^{-1}\boldsymbol{b} \end{array} \tag{2.14}$$

With $\bar{\boldsymbol{A}}_\eta := \boldsymbol{A}_\beta{}^{-1}\boldsymbol{A}_\eta$ we further obtain:

$$\begin{array}{cccc} x_0 & x_\beta & x_\eta & \text{r.h.s.} \\ \hline 1 & 0 & -\bar{c}_\eta & \boldsymbol{c}_\beta\boldsymbol{x}^*_\beta \\ 0 & \mathbb{1} & \bar{\boldsymbol{A}}_\eta & \boldsymbol{x}^*_\beta \end{array} \tag{2.15}$$

In this particular form it is easy to check for feasibility, i.e. β is primal feasible if $\boldsymbol{x}^*_\beta \geq 0$, and dual feasible if $\bar{\boldsymbol{c}}_\eta \leq 0$. As already mentioned, the Simplex algorithm starts with a feasible basis β. Then, *pivot* steps are performed iteratively, until an optimal basic solution is found. The pivot consists of the determination of one element β_i to leave the basis, and one (non-basic) element η_j with positive reduced costs $\bar{c}_{\eta_j}$ to enter the basis. The element β_i to leave the basis is determined by

$$i = \text{argmin}_k \left\{ \frac{x^*_{\beta_k}}{\bar{a}_{k,\eta_j}} \mid \bar{a}_{k,\eta_j} > 0 \right\}. \tag{2.16}$$

The element to enter the basis η_j with $\bar{c}_j > 0$ is chosen such that the new basis $\beta' = (\beta_1, \beta_2, \ldots, \beta_{i-1}, \eta_i, \beta_{i+1}, \ldots, \beta_m)$ remains feasible. The leaving variable is set to zero

(according to the definition of non-basic variables), and the entering variable takes on the value $x^*_{\beta'_i} = \frac{x^*_{\beta_i}}{\bar{a}_{i,\eta_j}}$. We obtain a (positive) increase of the objective function value by $\bar{c}_{\eta_j} x^*_{\beta'_i}$ as long as $x^*_{\beta'_i} > 0$.

So far, we presented the general algorithmic approach, but certain issues have not been addressed yet. Some of these aspects are now briefly discussed, without going too much into detail. For a comprehensive description the reader is referred to [36].

The Simplex method described so far requires an initial feasible basic solution. Such a solution can in turn be derived with the Simplex method itself by the introduction of additional auxiliary variables, which directly provide a feasible solution. If subsequent pivoting operations are successful to transform the initial solution into one having all auxiliary variables equalling zero, we have found a basic feasible solution to the original problem, otherwise infeasibility has been proven. These initial steps are called *Phase-I* Simplex, in contrast to the subsequently performed *Phase-II* Simplex, which has already been discussed. The overall approach is hence called *Two-Phase-Simplex*.

A detailed discussion regarding the pivoting-rule (2.16) is also beyond the scope of this brief introduction. However, it should be mentioned that this particular pivoting-rule comprises several problems: Following this selection rule may yield to degenerated tableaus and cycling, which may lead to situations where the algorithm does not terminate. These issues can be easily circumvented, though, by the application of *Bland's* pivoting rule.

A further important variation of the Simplex algorithm is the *Dual Simplex* method, which is particularly important for situations where additional constraints have been added which render the current solution infeasible. Such situations are important with respect to *cutting-plane methods*, which are discussed in Section 2.4.2 in the context if *integer* linear programs, but are also of high importance for the solution of merely linear programs. In contrast to the Primal Simplex method, Dual Simplex works by moving from one dual feasible basis to another, trying to obtain a primal feasible one.

2.4.2 Integer Linear Programming

Numerous problems within the combinatorial optimization area can be modeled as linear programs with the additional constraints that some or all of the decision variables must be integer or even Boolean. If all variables are restricted to an integer domain we have an *integer linear program* (ILP)

$$\begin{aligned} \max \quad & \boldsymbol{c}^\mathsf{T}\boldsymbol{x} && \text{(2.17a)} \\ \text{s.t.} \quad & \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} && \text{(2.17b)} \\ & \boldsymbol{x} \in \mathbb{Z}^n_+, && \text{(2.17c)} \end{aligned}$$

if this restriction is only imposed for some variables, we have a *mixed integer program* (MIP)

$$\max \quad \boldsymbol{c}^\top \boldsymbol{x} + \boldsymbol{c'}^\top \boldsymbol{x'} \tag{2.18a}$$
$$\text{s.t.} \quad \boldsymbol{A}\boldsymbol{x} + \boldsymbol{A'}\boldsymbol{x'} \leq \boldsymbol{b} \tag{2.18b}$$
$$\boldsymbol{x} \in \mathbb{Z}^n_+, \ \boldsymbol{x'} \in \mathbb{R}^{n'}_+. \tag{2.18c}$$

In some particular cases, linear programming provides a direct way for solving ILPs.

Definition 9 (Total Unimodularity) *Matrix $\boldsymbol{A}$ is called* totally unimodular*, if every square nonsingular submatrix of $\boldsymbol{A}$ has determinant ± 1.*

Theorem 5 *If $\boldsymbol{A}$ is totally unimodular and $\boldsymbol{b}$ is integer valued, then every corner point of the polyhedron is integer valued.*

Hence, the availability of a polyhedral description with this property implies the problem to be polynomially solvable. For $\mathcal{NP}$-hard problems such descriptions are generally not available. Explicitly restricting variable domains to integral numbers for polyhedral descriptions given by linear inequalities is generally a much harder problem than linear programming. In particular it is in general not possible to derive optimal ILP-solutions from optimal LP-solutions by simple rounding methods or other transformations, as shown in Figure 2.1.

Theorem 6 *0-1-Integer Programming is $\mathcal{NP}$-complete.*

As integer programming is more general than 0-1-integer Programming, a direct consequence is that it is also $\mathcal{NP}$-hard. Furthermore the following result can be shown.

Corollary 1 *Integer Programming is $\mathcal{NP}$-complete.*

Hence, in contrast to linear programming, no polynomial time algorithms for ILP do exist, unless $\mathcal{P} = \mathcal{NP}$. Nevertheless, LP-techniques are very important tools for solving ILPs for two reasons. First, LP-solutions of an ILP, i.e. solutions of the ILP where integrality constraints have been omitted, provide a *bound* for the objective value z^{ILP} of the ILP. Such a solution with objective function value z^{LP} is called *LP-relaxation* and for maximization problem it holds that $z^{\text{ILP}} \leq z^{\text{LP}}$. Second, effective algorithms for solving ILPs can be built upon these bounds by integrating the process of LP-solving into an enumerative framework, which is to be discussed in more detail in the following section.

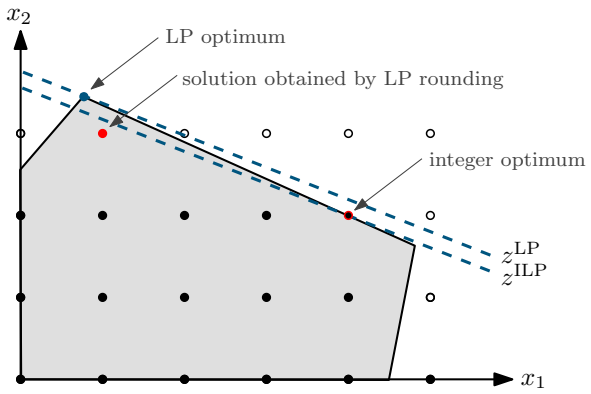

Figure 2.1: Example where rounding of LP-solution does not yield the optimal ILP-solution. The dashed lines correspond to the objective funcion.

Branch-and-Bound

Branch-and-bound (B&B) is an algorithmic framework for solving combinatorial optimization problems by performing a restricted enumeration of the solution space, using in each step upper and lower bounds to restrict search paths that cannot lead to a global optimum. Branch-and-bound basically follows the *"divide and conquer"* principle as all problems within the enumeration tree that cannot be directly solved are split into smaller subproblems. Let $b(P_i)$ denote the bound for subproblem P_i, i.e. a lower bound for minimization problems and an upper bound for maximization problems, and $obj(P_i)$ denote the objective function value for a feasible solution P_i. Let further G denote the global bound which is an upper bound for minimization problems, and a lower bound for maximization problems. The generic branch-and-bound framework is described in Algorithm 1.

Algorithm 1: Generic Branch-and-Bound

1 Put initial problem on the list of active subproblems.
2 Select an active subproblem P_i. If none exists, G is the global optimum.
3 If P_i is infeasible, delete P_i and goto line 2; otherwise compute $b(P_i)$.
4 If $b(P_i)$ is equal to or worse than the global bound G, delete P_i.
5 **if** P_i *directly solvable* **then**
6 solve P_i
7 **if** P_i *better than* G **then** $G \leftarrow obj(P_i)$
8 **else**
9 Split P_i into further subproblems, and add them to the list
10 of active problems. Goto 2
11 **end**

To give a precise specification of a B&B algorithm certain aspects need to be detailed. An important design decision is how to choose the next subproblem. Possibilities are

depth-first search plus backtracking, *breath-first search* or *quick improvement* (see [84] for details). Lines 3 and 4 of Algorithm 1 perform the *pruning* of the search tree. Besides infeasibility or optimality *value dominance* according to the global bound G is used to prune the search tree, and may be applied with some small tolerance $\epsilon > 0$ to avoid numeric issues. Finally, it needs to be specified, how considered problems are partitioned into subproblems. The efficiency of a B&B algorithm usually heavily depends on the strength of the bounds and the ability to compute them fast.

Besides its general ability to be used for solving COPs, B&B is also the main algorithmic framework for solving MIPs. The branch-and-bound algorithm starts with solving the LP-relaxation of the considered problem in the *root node*. The obtained solution generally will consist of many *fractional* variables, i.e. variables not fulfilling the integrality constraints of the MIP. As the LP-relaxation has not directly provided a solution in this case, the problem needs to be split up into subproblems. Let $\boldsymbol{x}^*$ denote the solution of the LP-relaxation of subproblem, consisting of at least one fractional element x_i^*. Two new subproblems are usually created by adding constraints

$$x_i^* \leq \lfloor x_i^* \rfloor, \tag{2.19}$$

and

$$x_i^* \geq \lceil x_i^* \rceil, \tag{2.20}$$

respectively. Such a situation is depicted in Figure 2.2.

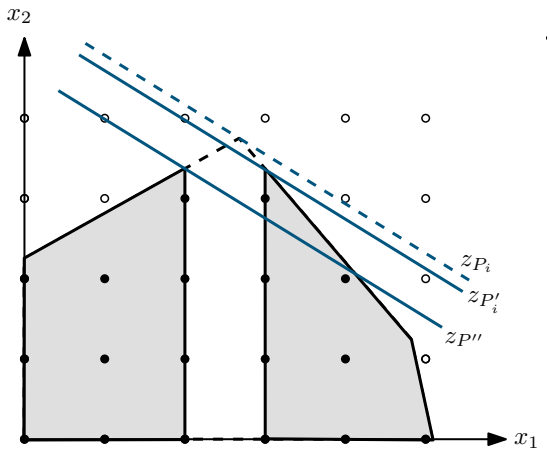

Figure 2.2: Example for splitting problem P_i into subproblems P' and P'' by adding constraints $x_1 \geq 3$ and $x_1 \leq 2$.

Cutting-Plane Method and Branch-and-Cut

The idea of branch-and-cut (B&C) originates from the *cutting-plane method* for solving COPs by linear programming. We have already discussed that it is generally not possible to describe the convex hull of the solution vectors to an $\mathcal{NP}$-hard COP by a polynomial

number of constraints w.r.t. the instance size. This can be achieved by an exponentially large set of constraints, which is, however, usually not available. Even in the case such a formulation is available, it is usually not possible to directly pass an exponentially large set of constraints to an LP-solver. However, usually only a relatively small subset of the exponentially many constraints is necessary in order to obtain an optimal integer solution of the LP. This observation led to the development of the cutting-plane algorithm. The procedure starts by solving the LP-relaxation of a restricted problem. This restricted problem can either be the LP-relaxation of a feasible MIP, or even an incomplete formulation, having solutions that violate certain constraints of the problem under consideration. Given a solution to the restricted problem, it needs to be identified, which inequalities of the polyhedral description of the convex hull of the solution vectors of the initial problem are *violated*. The problem of finding such inequalities is called *separation* problem.

Definition 10 (Separation problem) *Given a point $\tilde{\boldsymbol{x}} \in \mathbb{R}^n$ and a polyhedron $P \subseteq \mathbb{R}^n$. Decide, whether $\tilde{\boldsymbol{x}}$ belongs to P or not, and, in the latter case find a valid inequality $(\boldsymbol{\alpha}, \alpha_0)$ for P which is violated by $\tilde{\boldsymbol{x}}$.*

The solution to the separation problem provides a valid inequality $(\boldsymbol{\alpha}, \alpha_0)$, if one exists, which then can be added to the LP. Resolving the LP, which can efficiently be performed by the Dual Simplex algorithm (see Section 2.4.1), then yields another solution, having the same or better objective function value than the previous one. As this solution again might be fractional or infeasible to the initial problem, the process needs to be iteratively performed until a valid integer solution is obtained. The name cutting-plane method arises from the interpretation of cutting away certain parts of the polyhedron by the addition of a valid inequality. See Figure 2.3 for an illustration of cutting-planes. The challenging part within this approach is to efficiently solve the separation problem, and in particular to find good valid inequalities in the sense that preferably large parts of the polyhedron are cut away. An important result of Grötschel, Lovasz, and Schrijver in 1981 shows that separation and optimization are equivalent to a certain extend, as a linear optimization problem defined by a polyhedron can be solved in polynomial time if and only if the corresponding separation problem can be solved in polynomial time. In this case, polynomial time refers to the number of variables and the logarithm of some bound on the magnitude of entries of $\boldsymbol{A}$ and $\boldsymbol{b}$.

We now address the topic of how to obtain cutting plains, which can be divided into *generic* cutting-planes and *template*-based cutting-planes. The first class can be generally applied to any ILP, whereas the second class constitutes problem specific cutting-planes being closely related to specific templates or components a valid solution must or must not contain.

Chvátal-Gomory Cutting-Planes Let P denote the polyhedron defined by (2.17b) and (2.17c) and nonnegativity constraints $\boldsymbol{x} \in \mathbb{R}^n_+$. Let further further $\mathcal{F}$ denote the set of feasible integer solutions to ILP (2.17). The Chvátal-Gomory (C-G) procedure describes a general framework for deriving valid inequalities for $\text{conv}(\mathcal{F})$ which are not valid for P and therefore strengthen the formulation.

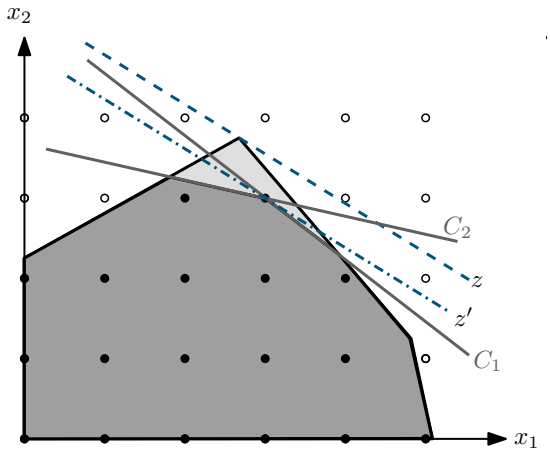

Figure 2.3: Additional inequalities (cutting planes) depicted by dark gray lines are added to the initial polyhedron (area shaded in light gray). The resulting (dark gray) polyhedron provides a tighter description of the underlying ILP model, in this case all fractional solutions w.r.t. the depicted objective function are cut-off and the LP-solution z' is integral.

We can now choose a $\boldsymbol{u} \in \mathbb{R}^m_+$. All $\boldsymbol{x}$ satisfying (2.17c) also satisfy

$$\sum_{j=1}^{n} (\mathbf{u}^\mathsf{T} \mathbf{A}_j) x_j \leq \mathbf{u}^\mathsf{T} \mathbf{b}.$$

It directly follows, that

$$\sum_{j=1}^{n} \lfloor \mathbf{u}^\mathsf{T} \mathbf{A}_j \rfloor x_j \leq \mathbf{u}^\mathsf{T} \mathbf{b}. \tag{2.21}$$

By using the integrality of $\boldsymbol{x}$ we obtain the C-G cutting-planes.

Definition 11 (Chvátal-Gomory Cutting Planes)

$$\sum_{j=1}^{n} \lfloor \mathbf{u}^\mathsf{T} \mathbf{A}_j \rfloor x_j \leq \lfloor \mathbf{u}^\mathsf{T} \mathbf{b} \rfloor \tag{2.22}$$

By variation of vector $\boldsymbol{u}$ we obtain the first Chvátal closure of the polyhedron P, given by

$$P_1 = \left\{ \boldsymbol{x} \in \mathbb{R}^n_+ \mid \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \sum_{j=1}^{n} (\lfloor \boldsymbol{u}^\mathsf{T} \boldsymbol{A}_j \rfloor) x_j \leq \lfloor \boldsymbol{u}^\mathsf{T} \boldsymbol{b} \rfloor, \text{ for all } \boldsymbol{u} \in \mathbb{R}^n_+ \right\}. \tag{2.23}$$

It can be shown that P_1 is indeed a polyhedron, and in some cases (as for instance for the matching problem) indeed corresponds to $\text{conv}(\mathcal{F})$.

Gomory Cutting-Planes Gomory cutting-planes are a particular manifestation of C-G cutting-planes, which can be directly extracted from basic-feasible solutions. A row of the Simplex tableau (compare (2.15)) is given by

$$x_{\beta_i} + \sum_{j=1}^{n-m} \bar{a}_{\beta_i \eta_j} x_{\eta_j} = x^*_{\beta_i}, \quad i = 0, \ldots, m. \tag{2.24}$$

By rounding down the coefficients we obtain the *Gomory (fractional) cutting planes*:

$$x_{\beta_i} + \sum_{j=1}^{n-m} \lfloor \bar{a}_{\beta_i \eta_j} \rfloor x_{\eta_j} = \lfloor x^*_{\beta_i} \rfloor, \quad i = 0, \ldots, m \tag{2.25}$$

By subtracting (2.24) from (2.26) we obtain an equivalent inequality

$$x_{\beta_i} + \sum_{j=1}^{n-m} (\lfloor \bar{a}_{\beta_i \eta_j} \rfloor - \bar{a}_{\beta_i \eta_j}) x_{\eta_j} = \lfloor x^*_{\beta_i} \rfloor - x^*_{\beta_i}, \quad i = 0, \ldots, m. \tag{2.26}$$

Template-based Cutting-Planes In contrast to the C-G or Gomory cutting-planes, the cutting-planes discussed in the following cannot be generally applied to any MIP. In contrast, they are defined by a particular template that must either be fulfilled or must not occur in a solution of the considered problem. Such cutting-planes are often very strong, such that they define facets of $\text{conv}(\mathcal{F})$ or faces of reasonable high dimension. Furthermore it is often possible to develop efficient separation algorithms for such families of valid inequalities. First exciting results in the context of template-based cutting-planes have been obtained for the symmetric traveling salesman problem, by using *subtour-elimination* constraints, *2-matching* inequalities and *comb* inequalities. To consider the subtour-elimination constraints in more detail, they state that no subgraph defined by $V' < V$ is allowed to contain a tour, as the overall goal is to find a minimum cost tour on the whole graph. The corresponding inequalities are given by

$$\sum_{e \in E(S)} x_e \leq |S| - 1 \quad \text{for all } S \subset V,\ 2 \leq |S| \leq |V| - 1, \tag{2.27}$$

with x_e denoting the decision variables for the edges. However, there are exponentially many such constraints, which therefore cannot be directly added to a solver (except for very small instances). Hence, such inequalities are only separated in the case they are violated by a LP-solution. This is done by inspecting each obtained optimal solution of the LP-relaxation in order to find a partial solution violating the corresponding constraints, or to show that no such violation exists. In the first case the corresponding inequalities are added to the model, which is in turn resolved.

A lot of research has been devoted to the development and analysis of such template-based cutting-planes for many different combinatorial optimization problems. In this paragraph we have just mentioned a few of them, in order to present the general idea. For solving spanning-tree problems *directed connection cuts* and *cycle-elimination cuts* are of particular importance, and will be introduced and discussed in more detail in Chapter

4 in Section 4.2, the corresponding separation algorithms are discussed in Section 4.2.2. In Chapter 4 we further show how to apply *odd-hole inequalities* to the minimum label spanning tree problem, and propose a MIP-based heuristic separation method in Section 4.2.3.

Combining Branch-and-Bound with the Cutting-Plane Method Embedding cutting-plane methods into the branch-and-bound framework (Section 2.4.2) leads to the *branch-and-cut* (B&C) approach. Cutting-planes may be used for two purposes. First, to obtain feasible solutions within each node of the branch-and-bound tree, which is particularly the case when the problem was modeled with exponentially many constraints, and the initial LP-model is therefore not feasible to the original problem. Well-known examples for such situations are formulations for spanning trees based on connectivity or cycle-elimination constraints. The second reason to use cutting-planes within the B&B framework is to strengthen the LP-relaxation in each node of the B&B-tree, yielding better bounds, likely permitting a more effective pruning of the search tree. However, as opposed to the advantage of smaller B&B-trees, there is the evident drawback of the separation-problem to be solved frequently, again emphasizing the importance of efficient separation algorithms. Whereas Gomory cuts were not to be thought to be of practical importance in the early 90's of the last century [34], they are nowadays part of almost every LP-solver, as they turned out to significantly speed up the branch-and-bound process, in particular when all cuts from the optimal tableau are generated. The additional separation of template cuts leads in many cases to efficient state-of-the-art exact algorithms.

Column Generation

As opposed to the cutting-plane method, where new *constraints* are dynamically added to the model, *column generation* proceeds by adding new *variables* to the model. The name originates from considering each variable in the Simplex tableau (2.12) as a column. The method is particularly important for problems having a huge, mostly exponential number of variables (as opposed to constraints). Many problems like the *cutting-stock problem*, which was the first application of column generation (see [51]), naturally imply such formulations, but however, many other problems can be transformed to a model with a larger number of variables by *Dantzig-Wolfe decomposition* [37]. Let us consider the MIP

$$\begin{aligned} \min \quad & z = \boldsymbol{c}^\mathsf{T}\boldsymbol{x} && \text{(2.28a)} \\ \text{s.t.} \quad & \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b} && \text{(2.28b)} \\ & \boldsymbol{D}\boldsymbol{x} \geq \boldsymbol{d} && \text{(2.28c)} \\ & \boldsymbol{x} \in \mathbb{Z}^n_+, && \text{(2.28d)} \end{aligned}$$

within this context known as the *original* or *compact formulation*. From Theorem 2 we know that we can write each $\boldsymbol{x} \in P$ as a convex combination of extreme points $\{\boldsymbol{p}_q\}, q \in Q$

plus a nonnegative combination of extreme rays $\{\boldsymbol{p}_r\}, r \in R$ of P, i.e.

$$\boldsymbol{x} = \sum_{q \in Q} \boldsymbol{p}_q \lambda_q + \sum_{r \in R} \boldsymbol{p}_r \lambda_r, \quad \sum_{q \in Q} \lambda_q = 1, \quad \boldsymbol{\lambda} \in \mathbb{R}_+^{|Q|+|R|}. \tag{2.29}$$

With the linear transformations $c_j = \boldsymbol{c}^\mathsf{T} \boldsymbol{p}_j$ and $\boldsymbol{a}_j = \boldsymbol{A}\boldsymbol{p}_j$, $j \in Q \cup R$, we obtain the *extensive formulation*, which is equivalent to (2.28).

$$\min \quad z = \sum_{q \in Q} c_q \lambda_q + \sum_{r \in R} c_r \lambda_r \tag{2.30a}$$

$$\text{s.t.} \sum_{q \in Q} \boldsymbol{a}_q \lambda_q + \sum_{r \in R} \boldsymbol{a}_r \lambda_r \geq \boldsymbol{b} \tag{2.30b}$$

$$\sum_{q \in Q} \lambda_q = 1 \tag{2.30c}$$

$$\boldsymbol{\lambda} \geq \boldsymbol{0} \tag{2.30d}$$

This reformulation (2.30) usually has less rows than (2.28), but a larger number of variables. It can be shown, that the resulting polyhedra are not combinatorially equivalent. Constraints (2.31c) are known as *convexity-constraints*.

Decomposition will be particularly useful if the considered MIP contains a *block-diagonal* structure, which is often the case for COPs, as certain groups of "connecting" or "coupling" constraints, having many non-zero coefficients, can often be identified. The blocks can be interpreted as relatively independent subproblems which are then, on a higher level, linked together (by $\boldsymbol{\lambda}$).

Column Generation is an algorithmic approach for solving such structured problems as given by (2.30). Hence, we call

$$\min \quad z = \sum_{j \in J} c_j \lambda_j \tag{2.31a}$$

$$\text{s.t.} \sum_{j \in J} \boldsymbol{a}_j \lambda_j \geq \boldsymbol{b} \tag{2.31b}$$

$$\lambda_j \geq 0 \qquad \text{for all } j \in J \tag{2.31c}$$

the *master problem* (MP). In each iteration of the Simplex algorithm a new non-basic variable to enter the basis is determined based on the reduced costs. This particular step of the Simplex provides the main motivation for the CG approach. Instead of directly starting with solving the MP, we consider the *restricted master problem* (RPM), just consisting of a small subset $J' \subseteq J$ of columns. New columns are just added on demand, according to the selection rule (compare with Definition 8)

$$\operatorname{argmin} \left\{ \bar{c}_j := c_j - \boldsymbol{y}^{*\mathsf{T}} \boldsymbol{A}_j \mid j \in J \right\}. \tag{2.32}$$

The dual optimal solution $\boldsymbol{y}^*$, whereupon this decision is based on, is provided by the solution of the RPM by the Simplex algorithm. The determination of such a column to add, having negative reduced costs is called the *pricing problem*. Each such variable could potentially improve the objective function, furthermore, if no such variable with negative reduced costs could be found, no further improvements are possible. It is, however, desirable not to have to explicitly enumerate the whole index set. By exploiting the structure of the considered problem it is often possible to formulate the pricing problem in terms of an optimization problem.

$$\bar{c}^* = \min \left\{ c(\boldsymbol{a}) - \boldsymbol{y}^{*\mathsf{T}} \boldsymbol{a} \mid \boldsymbol{a} \in \mathcal{A} \right\} \tag{2.33}$$

In this case it is not necessary to explicitly enumerate the space of all possible columns $\mathcal{A}$, but rather to determine the best such column in a more direct way, or to prove that none such column does exist.

Branch-and-Price and Branch-and-Cut-and-Price

Using column generation within the branch-and-bound framework directly leads to the *branch-and-price* (B&P) algorithm, which was for the first time proposed by Desrosiers [40], and later on described in a generic algorithm in [105]. Within each node of the B&B-tree new variables are priced in, until no further potentially improving variable is found. If the solution still contains fractional variables branching is performed. A further combination with the cutting-plane method is usually called *branch-and-cut-and-price* (BCP).

Although it is generally believed that such combinations of cutting-plane methods with B&B and B&C are very powerful tools, these approaches show only moderate performance if applied in a straightforward way. Frequently observed problems are the *heading-in effect*, which refers to slow progression in the beginning due to poor information provided by the dual variables, and the *tailing-off effect* referring to poor convergence after a certain stage of fast progression. Dual solutions are frequently observed to jump between extreme values (*bang-bang effect*), intermediate Lagrangean dual bounds not to converge monotonically (*yo-yo effect*). Advanced techniques like *stabilized column generation* [79] significantly improve the overall performance.

For a comprehensive description of column generation and related topics the reader is referred to [39] and [74].

2.4.3 Further Methods

There are many further popular and frequently used methods and approaches to modeling, which are, however beyond the scope of this work, as this introductory chapter is primarily focused on methods going to be applied in the remainder of this thesis. For the sake of completeness we finish this section covering exact solution methods for combinatorial optimization problems by briefly mentioning some further, important approaches.

Lagrangean Relaxation

Lagrangean relaxation (LR) is a powerful relaxation technique, where certain hard constraints are moved to the objective function as a penalty term. Let us consider the following linear program:

$$\begin{aligned} \min \quad & \boldsymbol{c}^{\mathsf{T}}\boldsymbol{x} && \text{(2.34a)} \\ \text{s.t.} \quad & \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b} && \text{(2.34b)} \\ & \boldsymbol{B}\boldsymbol{x} \geq \boldsymbol{d} && \text{(2.34c)} \\ & \boldsymbol{0} \leq \boldsymbol{x} \leq \boldsymbol{1} && \text{(2.34d)} \end{aligned}$$

To compute a lower bound, we may now relax a subset of constraints, say $\boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b}$, by adding it to the objective function with coefficients $\boldsymbol{\lambda} \geq \boldsymbol{0}$. Hence, the LR is given by:

$$\begin{aligned} \min \quad & \boldsymbol{c}^{\mathsf{T}}\boldsymbol{x} + \boldsymbol{\lambda}(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}) && \text{(2.35a)} \\ \text{s.t.} \quad & \boldsymbol{B}\boldsymbol{x} \geq \boldsymbol{d} && \text{(2.35b)} \\ & \boldsymbol{0} \leq \boldsymbol{x} \leq \boldsymbol{1} && \text{(2.35c)} \end{aligned}$$

It can be easily verified that (2.35a) gives a lower bound for the optimal solution of (2.34a). In order to obtain the best available bound, the *Lagrangean dual program*

$$\max_{\boldsymbol{\lambda} \geq \boldsymbol{0}} \left\{ \begin{array}{ll} \min & \boldsymbol{c}^{\mathsf{T}}\boldsymbol{x} + \boldsymbol{\lambda}(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}) \\ \text{s.t.} & \boldsymbol{B}\boldsymbol{x} \geq \boldsymbol{d} \\ & \boldsymbol{0} \leq \boldsymbol{x} \leq \boldsymbol{1} \end{array} \right\} \tag{2.36}$$

must be solved. It can be shown that the bound obtained by solving (2.36) to optimality strictly dominates the LP-bound for problem (2.35a). The determination of the Lagrange-multipliers within (2.36) can be performed by *subgradient method* or the *volume algorithm* [5]. A good introduction to *Lagrangean relaxation* and related topics can be found in [7].

Constraint Programming

Constraint programming (CP) can to a certain extend be regarded as a complementary approach to solving COPs, as opposed to traditional LP-based techniques. It is of particular importance for *constraint satisfaction problems* (CSPs), where the goal is not to optimize over a given objective function, but to find any feasible assignment for the decision variables, fulfilling all specified constraints. As opposed to strictly linear constraints, also logical constraints, or even more complex functions not necessarily having closed mathematical forms can be specified.

The term *programming* in CP is in close correspondence to the term programming in for instance object oriented programming or functional programming, and has less in common with mathematical *programming*. It should be rather understood as programming technique [75]. Algorithms for solving CP problems are often based on *constraint propagation* and *domain reduction*. The term constraint propagation refers to the communication of any modification of a variables domain when it is modified, to other interacting constraints. In contrast, domain reduction is about the modification of further variables within one constraint in the case a particular variable is modified. To find a feasible solution to a CSP a *search strategy* needs to be specified, which is often based on depth-first search. At each node of the search tree the problem is split into subproblems based on a specified *goal*. This process is continued until a feasible solution has been found, or the problem could be shown to be infeasible. Hence the process is closely related to solving ILP by branch and bound, however with the difference that non-linear constraints can be handled, and subproblem generation is performed in a more problem-specific way.

Constraint programming turned out to be very effective in the context of scheduling problems, but also for many other problems where finding feasible solutions is of primary interest. On the other hand, it is not suitable for many types of problems, as no relaxations are used for effectively pruning the search tree. The topic of CP is covered in great detail in [96].

Convex Optimization

Some problems can be modeled by means of linear constraints, but have convex objective functions. A well known special case is *quadratic programming* (QP), given by

$$\min \quad \boldsymbol{c}^\mathsf{T}\boldsymbol{x} + \frac{1}{2}\boldsymbol{x}^\mathsf{T}\boldsymbol{Q}\boldsymbol{x} \tag{2.37a}$$

$$\text{s.t.} \quad \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b} \tag{2.37b}$$

$$\boldsymbol{x} \geq \boldsymbol{0}, \tag{2.37c}$$

with $\boldsymbol{Q}$ being a symmetric, positive semi-definite $n \times n$ matrix. With these properties the objective function is also convex. Such QPs can be solved by a slightly modified variant of the simplex algorithm [111], but also other algorithms like interior point method are applicable.

Having now reviewed the most important mathematical programming techniques and algorithms which are frequently applied for exactly solving combinatorial optimization problems, we now focus on heuristic and approximative methods in the remainder of this chapter.

2.5 Heuristic and Approximative Methods

With respect to $\mathcal{NP}$-complete problems, exact methods are usually only applicable to small to moderate sized problem instances, and typically require relatively long running times ranging from a couple of minutes to a couple of hours and even more. Consequently, heuristic methods, not necessarily yielding optimal solutions, are an auspicious alternative, in particular when running times are limited and near optimal solutions are of importance. Algorithms providing a guarantee of not exceeding the optimal objective function value by a certain value or multiplicative factor in the worst-case are called *approximation algorithms*. If the performance bound is unknown or infinite an algorithm is simply called a *heuristic*. Algorithms building a feasible solution from scratch are called *construction algorithms*.

If each decision of the algorithm is performed by selecting the locally best extension component, and these decisions are not revised later on, the algorithm is said to be *greedy*. Approximation and construction algorithms will be discussed in Sections 2.5.1 and 2.5.2.

Algorithms iteratively modifying solutions in order to obtain better ones are called *improvement algorithms*. During the last couple of decades numerous "general purpose" algorithmic templates emerged, which can be more or less easily adapted for particular optimization problems. Such algorithms are called *metaheuristics*, and are to be discussed in Section 2.6.

2.5.1 Approximation Algorithms

If algorithm $\mathcal{A}$ can be shown not to exceed the optimum in the worst case by more than a certain amount, it is called an *approximation algorithm*. Let I denote an instance of the considered problem, and $\mathcal{A}(I)$ denote the objective function delivered by algorithm $\mathcal{A}$. Let further $\mathrm{Opt}(I)$ denote the optimal objective function value of instance I. Algorithm $\mathcal{A}$ is said to have an *absolute performance guarantee* ρ if

$$|\mathrm{Opt}(I) - A(I)| \leq \rho, \quad \text{for all instances } I. \tag{2.38}$$

Definition 12 (Constant Factor Approximation Algorithm) *Algorithm $\mathcal{A}$ has a* relative performance guarantee $(1+\varepsilon)$ *for minimization problem I if there exists a constant $\varepsilon > 0$ such that*

$$\mathcal{A}(I) \leq (1+\varepsilon)\ \mathrm{Opt}(I), \quad \textit{for all} \text{ instances } I, \tag{2.39}$$

and a relative performance guarantee of $(1-\varepsilon)$ for a maximization problem

$$\mathcal{A}(I) \geq (1-\varepsilon)\ \mathrm{Opt}(I), \quad \textit{for all} \text{ instances } I. \tag{2.40}$$

Such algorithms are called $(1+\varepsilon)$-approximation algorithms, and $(1-\varepsilon)$-approximation algorithms respectively.

The performance bound is *tight*, if the inequalities are fulfilled with equality for at least one instance I. As a consequence this bound cannot be improved by further analysis.

If ε can be specified as a parameter for algorithm $\mathcal{A}$, such that $\mathcal{A}$ is a $(1+\varepsilon)$ or $(1-\varepsilon)$ approximation algorithm respectively, and furthermore algorithm $\mathcal{A}$ has polynomial runtime w.r.t. the size of the instance when ε is regarded as a constant, the resulting family of algorithms is called *polynomial-time approximation scheme*. If the algorithm has further polynomial runtime w.r.t. the instance size *and* parameter $1/\varepsilon$ it is called a *fully polynomial-time approximation scheme* (FPTAS).

The complexity class $\mathcal{APX}$ refers to all optimization problems in $\mathcal{NP}$, for which polynomial-time ε-approximation algorithms do exist. Unfortunately, many $\mathcal{NP}$-complete problems cannot be approximated within a constant factor within polynomial time. Let $\mathcal{PTAS}$ denote the complexity class (denoted by calligraphic font) of $\mathcal{NP}$-hard problems for which PTAS exists, and $\mathcal{FPTAS}$ be the class of $\mathcal{NP}$-hard problems for which FPTAS exists. It can be shown, that unless $\mathcal{P} = \mathcal{NP}$, $\mathcal{FPTAS} \subsetneq \mathcal{PTAS} \subsetneq \mathcal{APX}$, and in particular strict inclusion holds for this hierarchy. Hence, for $\mathcal{APX}$-hard problem, no PTAS can exist.

See [107] for a comprehensive introduction to approximation algorithms.

2.5.2 Construction Algorithms

Construction algorithms are methods that start with an empty solution, and then iteratively add further components to the partial solution, until a feasible solution is obtained. For many problems the main purpose of construction heuristics is to create feasible starting solutions for subsequent application of metaheuristics (see Section 2.6). However, for huge problem instances, the application of fast construction heuristics may be the only way to obtain any feasible solutions, as the application of metaheuristics is not practicable due to memory and running-time limitations. Due to these two major fields of application construction algorithms are usually primarily designed to work fast and efficient.

Many construction algorithms follow the *greedy*-principle. Due to their usual simplicity greedy algorithms often can be relatively easily analyzed and for many problems performance guarantees can be derived. Hence, in fact, many greedy algorithms are approximation algorithms. For some COPs it is also possible to develop greedy algorithms that always yield the optimal solution. Of course, this is only possible for problems not being $\mathcal{NP}$-complete or harder, unless $\mathcal{P} = \mathcal{NP}$. A well known example is the minimum spanning tree problem, which can be solved optimally with the well known algorithms by Prim and Kruskal. Optimal solutions can be generally found by greedy algorithms if the problem has matroid structure [71].

2.6 Metaheuristic Methods

During the last decades *metaheuristics* have become rather popular approaches for solving various kinds of optimization problems. The term *metaheuristic* was introduced by F. Glover in [52] for the first time. In contrast to problem specific heuristic algorithms, metaheuristics are high-level generic templates which can, in principle, be applied to any optimization problem. They primarily provide a description of the interaction and application of various low-level components, to be specifically designed for the application to a particular problem. Designing and integrating these low-level components in an appropriate way, as well as the adjustment of certain parameters, is usually the crucial part for obtaining efficient metaheuristic algorithms w.r.t. a particular problem. As these tasks cannot be described in a generic way, the application of metaheuristics to particular optimization problems is an ongoing and active field of research. Although many attempts have been made into the direction of the development of robust black-box optimization tools, best results can mostly be achieved by the incorporation of problem specific operators and methods into the general algorithmic framework of metaheuristics. A plenitude of different metaheuristics has been developed, and some of them are to be discussed in the following sections, as it is generally not possible to identify one particular approach that is superior over the other ones. Over all possible optimization problems, all optimization algorithms essentially show the same average performance. This result is essentially known as the *No Free Lunch Theorem*, which has been presented and analyzed in [112]. As a consequence, designing algorithms for particular optimization problems seldomly follows a strict methodology or recipe. In fact, it can be seen as the art of combining analysis, intuition, computational testing and experience in a beneficial way to obtain algorithms having desirable properties like providing high average solution quality, robustness, and low runtime and memory requirements. These conflicting properties frequently make design choices and analysis of the resulting algorithms even more difficult.

In the following some of the most popular and frequently used metaheuristics are described in more detail. Main emphasis is given to the methods going to be used in the remainder of this thesis. In Section 2.6.1 we consider methods having in common that they iteratively modify a single solution by considering its *neighbors* w.r.t. to some neighborhood structure. The metaheuristics covered in Sections 2.6.2 and 2.6.3 have in common that they simultaneously operate on a *set* of candidate solutions, and that they are inspired by certain mechanisms from nature. In Section 2.7.1 we finally turn our attention to the combination ("hybridization") of metaheuristics, which showed to be profitable for many applications and problems.

2.6.1 Neighborhood Search Algorithms

Many metaheuristics are built upon the principle of *neighborhood search*. All these algorithms have in common that they start from an initial solution $x \in \mathcal{S}$, where $\mathcal{S}$ denotes the search space, i.e. set of allowed solutions.

Definition 13 (Neighborhood Structure) *A neighborhood structure $\mathcal{N} : \mathcal{S} \rightarrow 2^{\mathcal{S}}$ is a function assigning a set of neighbors $\mathcal{N}(x) \subseteq S$, called neighborhood of x to every solution $x \in \mathcal{S}$.*

The basic algorithmic building block for exploiting neighborhood structures is called *local search* (LS). The principle is to start with a solution $x \in \mathcal{S}$ and then move through the solution space by successively proceeding with better solutions found by exploring the neighborhood of the current one. We assume the objective to be minimization according to objective function $f : x \rightarrow \mathbb{R}$. The basic local search algorithm is presented in the following Algorithm 2.

Algorithm 2: Local Search

```
1 Input: Solution x ∈ S
2 repeat
3     choose x' ∈ N(x)
4     if f(x') < f(x) then
5         x ← x'
6     end
7 until termination criterion is fulfilled
```

In Algorithm 2 it remains unspecified how to choose x' in line 3. One of these strategies is usually applied:

- *Random Neighbor:* A random neighbor is chosen.
- *Next Improvement:* The neighboring solutions are traversed, and as soon as an improvement to the current solution is found, the algorithm proceeds with this better solution.
- *Best Improvement:* An overall best solution from the neighborhood is used to proceed with.

Within LS usually next or best improvement is performed, whereas random neighbor has applications in simulated annealing and tabu search, to be presented in the following Sections 2.6.1 and 2.6.1. Local search is usually terminated if a *local optimum* has been found.

Definition 14 (Local Minimum) *A solution x is a local minimum for $\mathcal{N}(x)$ if $f(x) \leq f(x')$ for all $x' \in \mathcal{N}(x)$.*

A local maximum is defined analogously for maximization problems. Single application of LS to complex COPs generally yields only solutions of moderate quality. Straightforward improvements are, however, to perform multiple LS runs starting from different initial solutions. Such approaches are known as *multi-start local search* . In the following sections we consider metaheuristics which are extensions to the outlined local search procedure.

Simulated Annealing

The origin of the name *simulated annealing* (SA) is due to the annealing process for crystals, whereas annealing is a thermal process for obtaining low-energy states of solids. In a first step the temperature of a heat bath is increased to an amount at which melting occurs, then temperature is decreased again to allow the particles to arrange themselves, with the goal to achieve a nearly perfectly structured lattice in the ground state. The metaheuristic simulated annealing, proposed in [66], closely follows this procedure.

The SA algorithm can be interpreted as an extension to LS with random neighbor step-function typically applied. Differences are, however, that inferior solutions are accepted with certain probability, depending on the difference of the objective values and temperature T. A neighboring solution $x' \in \mathcal{N}(x)$ with $f(x') > f(x)$ is accepted if

$$r < e^{|f(x')-f(x)|/T}, \tag{2.41}$$

where $r \in [0, 1)$ is a uniformly distributed random number. Lower gaps of the objective values and higher temperature thus increase the likelihood of worse solutions being accepted. The decision rule given by (2.41) is called *Metropolis criterion*. Hence, objective values correspond to the particles' energy state in the annealing process. The temperature T is initially set to a high value in a problem specific way, for instance such that moves to worse solutions are accepted with a specified probability. The temperature is then successively decreased during the execution of the algorithm, making moves to worse solutions less likely. In comparison to LS, the probability of ending up in a local optimum is significantly reduced. Due to its simplicity and effectiveness SA is a frequently applied metaheuristic.

Tabu Search

Another popular extension of local search is *tabu search* [53]. Again, the basic algorithm follows the local search paradigm, the neighborhood is typically explored with a best improvement strategy, and the best neighbor is usually *always* accepted. Major difference to local search is the usage of *tabu lists*, with the main intention to prevent cycling (visiting the same solution over and over) and therefore support the ability to escape local optima. The tabu list(s) keep track of either entire solutions or, more commonly, attributes of performed moves for a specified number of iterations. New solutions are only accepted, if they are compliant with the tabu list(s), i.e. recently visited solutions are avoided. An important parameter is the lenght of the tabu list, also called *tabu tenure*. If too small values are used, cycling can still occur. On the other hand too long tabu lists impose too strong restrictions on the further search process. Hence, the determination of the best value to be used for a particular problem is often a difficult task. Due to this issue, self-adaptive approaches have been proposed. For instance, the length of the tabu lists is dynamically adjusted within *reactive tabu search* [6].

Greedy Randomized Adaptive Search Procedures

A straightforward extension to iterated local search, proposed in [95], is called *greedy randomized adaptive search procedure* and is widely known by its acronym GRASP. Within this approach, multiple starting points for subsequent local search are created by a randomized greedy construction heuristic. The greedy decisions are randomized by means of a *restricted candidate list* (RCL) which is filled with potential extension candidates. In each construction step one of the elements of the RCL is randomly chosen. How the RCL is built up, is an important design issue for GRASP. Small RCLs imply less randomization, which leads to less diversity amongst the solutions used as starting points for the LS. Longer RCLs will on the contrary produce solutions of weaker expected quality on average, which may also be not desirable. Again, attempts to dynamically adjust this parameter have been made, for instance known as *Reactive GRASP*.

Variable Neighborhood Search Methods

The basic idea of *variable neighborhood search* methods is to use more than one neighborhood structure within a local search framework. This approach has been introduced by Mladenović and Hansen [82, 56]. If a solution x is a local optimum w.r.t. a particular neighborhood structure, another one may yield further improvements. Hence, the probability of getting stuck in local optima can be significantly reduced. Let $\mathcal{N}_1(x), \mathcal{N}_2(x), \ldots, \mathcal{N}_{k_{\max}}(x)$ denote the $k_{\max}$ considered neighborhood structures. It is generally reasonable to consider these neighborhood structures ordered increasingly w.r.t. their complexity, i.e. the number of neighboring solutions defined by it. Algorithm 3 shows a natural extension of LS using more than one neighborhood structure, called *variable neighborhood descent* (VND).

Algorithm 3: Variable Neighborhood Descent

```
1  Input: Solution x ∈ S
2  k ← 1
3  repeat
4      choose x' ∈ N_k(x)
5      if f(x') < f(x) then
6          x ← x'
7          k ← 1
8      else
9          k ← k + 1
10     end
11 until k = k_max
```

In line 4 the algorithm tries to find a better solution in the current neighborhood $\mathcal{N}_k$. If this attempt was successful, the algorithm proceeds with $\mathcal{N}_1$, otherwise the algorithm switches to the next neighborhood. However, abilities of leaving local optima are still limited within VND. This problem is circumvented by *(basic) variable neighborhood search* (VNS), presented in Algorithm 4.

Algorithm 4: (Basic) Variable Neighborhood Search

```
1  Input: Solution x ∈ S
2  repeat
3      k ← 1
4      repeat
5          Shaking: choose random solution x' from N'_k(x)
6          x' ← Local Search(x')
7          if f(x') < f(x) then
8              x ← x'
9              k ← 1
10         else
11             k ← k + 1
12         end
13     until k = k_max
14 until termination criterion is fulfilled
```

Primary difference to VND is the *shaking* operation in line 5, which chooses a random solution of the current neighborhood $\mathcal{N}'_k$. The algorithm then proceeds with a local search for this solution in line 6. Again, if an improvement compared to x could be found, the algorithm proceeds with this new incumbent solution and switches to first neighborhood structure, otherwise the next neighborhood structure is used.

Many variations of the basic VNS scheme of Algorithm 4 have been proposed. If the local search performed in line 6 is omitted, we obtain *reduced variable neighborhood search* (R-VNS). If alternatively local search is replaced by VND, the resulting algorithm is called *general variable neighborhood search* (G-VNS). Note that the neighborhood structures $\mathcal{N}'_k$ used within VND must not correspond to the ones used in the outer loop for the shaking. These neighborhood structures $\mathcal{N}'_k$ should be significantly larger than the one used within VND, in order to provide an appropriate mechanism for escaping local optima w.r.t. the neighborhoods used in VND. The decision of which variable neighborhood search variant to apply, and the design of the neighborhood structures to be used within the framework, must be carefully decided based upon the properties of the considered problem, characteristics of the data instances and allowed running times.

2.6.2 Methods Based on Swarm Intelligence

Many metaheuristic techniques are based on parallel construction or modification of a pool of candidate solutions and frequently are inspired by nature. The common concept is the observation of emergent intelligence at high levels in a system of “low level components”. The collective behavior of a fish school or flock birds would not be expected, if only low-level components, i.e. individual animals, are considered. Despite the absence of a centralized control by a group leader or a predefined schedule for all individuals, the whole group is able to commonly perform certain tasks, quickly adopt to changing environments

as a collectively behaving super-individual. For instance a fish school is able to split to avoid a predator and reunite afterwards.

Principles facilitating such collective behaviour in a decentralized system are usually subsumed under the term *self-organization*. The emergent intelligence on the level of the entire system is often called *swarm intelligence*. This astonishing phenomenon was inspiration for many metaheuristic optimization algorithms, which to a certain extend imitate the underlying principles and mechanisms of swarm intelligence.

The above mentioned examples of bird flocks and fish schools gave rise to a metaheuristic called *particle swarm optimization* (PSO), proposed in [65], which is particularly suited for continuous optimization problems. Within this approach a group of particles traverses the search space in order to find near optimal solutions. Within each discrete time step the motion of each individual particle undergoes a change of velocity and direction, based on the motion of surrounding particles and best solutions discovered so far.

A further example of biologically inspired algorithms are *artificial immune systems*, outlined for the first time in the seminal work [46]. Main properties of such approaches are to imitate the immune system's ability of adaptive learning and memory. Recently, these techniques have been also successfully applied to optimization problems. Probably the most popular approach among biologically inspired techniques is stimulated by the behaviour of ant colonies, and therefore subject of the following section.

Ant Colony Optimization

Ant colony optimization (ACO) is based on the foraging behaviour of ants. Despite the ants disability to survey the region around the anthill, or to make educated decisions, almost all ants run on near-optimal paths from the anthill to the food source and back. This "optimization task" is mainly achieved by *stigmergy*, i.e. communication by means of modification of the common environment. Along their way, each ant deposits *pheromone*, a volatile chemical factor, further ants follow trails of high pheromone concentration with high probability, but may also leave this trails to explore further ones. The latter mechanism is important to retain flexibility as well as to avoid premature convergence to a local optimum, i.e. all ants walking on a suboptimal path. The first metaheuristic based on these principles has been introduced by M. Dorigo, V. Maniezzo and A. Colorni [42, 43].

Algorithm 5: Generic-ACO

```
1 for it iterations do
2     for each ant m, 1 ≤ m ≤ M do
3         solution construction (of ant m)
4         optional local search
5     end
6     pheromone update
7 end
```

Ant colony optimization is a stochastic model-based search procedure, where construction steps are influenced by local information and global pheromone values. The correspondence to ants are artificial agents constructing the solution, mostly by traversing some kind of construction graph. The general algorithmic template works as specified by Algorithm 5. In each iteration, all ants construct solutions, usually one after another. Each decision in the constructive process is influenced by *local information* and *pheromone values*. Let i denote the current position of ant m in the construction graph. The set of feasible extension candidates is denoted by $F(i)$. To each edge $\{i, j\}$ in the construction graph a pheromone value τ_{ij} is associated. Local information is reflected by η_{ij} and usually corresponds to a simple function evaluating the contribution of the element corresponding to edge $\{i, j\}$. The decision of the next component to be added is usually performed according to the following probabilities

$$p_{ij}^m = \begin{cases} \dfrac{\tau_{ij}^\alpha \eta_{ij}^\beta}{\sum_{k \in F(i)} \tau_{ik}^\alpha \eta_{ik}^\beta} & \text{for all } j \in F(i), \\ 0 & \text{otherwise,} \end{cases} \tag{2.42}$$

Parameters α and β control the relative contribution of pheromone trails and local information to the resulting probabilities. If $\alpha = 0$ the construction process degenerates to a randomized greedy heuristic. If on the other hand $\beta = 0$ the whole process will quickly converge to an arbitrary solution (constructed by all ants).

After the solution construction process pheromone update takes place. This involves two steps: *deposition* and *evaporation*. Each ant deposits pheromone on the trails it has traversed in the construction graph during the solution construction process. The amount of pheromone to be deposited is usually dependent on the solution quality obtained by ant m. Evaporation is the primary mean to "forget" information that has been aquired by the pheromone model so far, and enables to explore other areas of the solution space during the search process.

Various algorithmic variations with differences according to pheromone deposition and evaporation do exist, like $\mathcal{MAX} - \mathcal{MIN}$ Ant System, Ant Colony System, Elitist Ant System or Rank-based Ant System. For a comprehensive introduction to ACO see [44].

2.6.3 Evolutionary Algorithms

Optimization algorithms that are inspired by biological evolution are called *evolutionary algorithms* (EAs). In particular, the principle components of Charles Darwin's theory of evolution and Mendel's laws of inheritence are reproduced as algorithmic building blocks. Frequently used variants of EAs are *genetic algorithms* (GAs), *evolution strategies* (ESs), *evolutionary programming* (EP) and *genetic programming* (GP). All these variants have in common that the algorithm operates on a set of *candidate solutions* or *individuals*. Within each iteration ("generation") these individuals are combined and modified in order to create new offspring solutions. Following the biological analog, these operations are called *recombination* and *mutation*. Primary goal of recombination is to merge beneficial properties of different individuals in a reasonable but essentially randomly decided

way. Mutation provides a mean of exploration of new path of evolution by small random changes. Having created the offspring individuals, *evaluation* is performed, based on the objective function for the given problem. As a result, following the terminology of biological evolution, *fitness* values are assigned to each individual, with higher fitness values mark better solutions. The next generation population is then obtained by the application of *selection*, which favors high quality solutions over inferior ones. A generic template for EAs is given in Algorithm 6.

Algorithm 6: Generic-EA

```
1  create initial population P
2  evaluate(P)
3  t ← 0
4  while t < max. generations do
5      P' ← recombine(P(t))
6      P'' ← mutate(P')
7      evaluate(P'')
8      P(t + 1) ← select(P'' ∪ P(t))
9      t ← t + 1
10 end
```

Variable t denotes discrete time-steps, for each such generation a new population $P(t)$ of individuals is created. All variants of GAs can be described with this template, differences are how individuals are encoded, and the functionality of the evolutionary operators recombination, mutation and selection.

The most famous variant of EAs probably are the *genetic algorithms*, which have been applied to numerous types of optimization problems. The idea of GAs goes back to John Holland et al. [60]. A comprehensive introduction to GAs can be found in [54]. Genetic algorithms differ to other EAs in that they facilitate an encoding of the solution that is in close correspondence to the biological archetype. Individuals are usually represented by a string (*"chromosome"*) of elements from a finite set (*"genes"*). These genes might have a strong correspondence to the problems decision variables, but also weaker couplings of the encoded candidate solution (*"genotype"*) to the decoded candidate solution (*"phenotype"*) might exist. This is for instance the case when real values are encoded with binary numbers. The simplest crossover strategy is to arbitrarily select a *crossover-point* and to build a new offspring by using all genes left of the crossover-point from one parent, the genes on the right side from the other. Mutation can be performed by simply changing the value of one gene with some low probability. However, in general, these operators should be designed in a more problem specific way, such that profitable attributes are inherited to the offsprings in a meaningful way. To determine the parents for the creation of new offspring solutions *roulette-wheel* selection is frequently performed. Thereby individuals are selected with a probability proportional to their fitness values.

Several attempts to theoretically explain the functionality of GAs have been made. The *schema theorem* provides an explanation of how GAs operate by combining various small building blocks in a profitable way. Crucial part for the development of efficient GAs is to

find an appropriate encoding of the problem. The property of *locality* should be met, i.e. small changes in the genotype should correspond to small deviations in the phenotype and vice versa. This often implies the requirement to place strongly dependent genes close to each other, as they should not be detached by recombination.

Many extensions and improvements to the presented standard GA framework do exist. For instance, the use of *scaling* or *tournament selection* often facilitates a fine adjustment of *selection pressure*, i.e. the ratio at which high-quality solutions are preferred to average ones by the selection operator. In the case of irregular distributions of fitness values *rank-selection* may be applied, where instead of fitness values only the rank within the current population ordered decreasing w.r.t. the fitness values is accounted for selection probability. If invalid solution candidates can be produced by recombination, these individuals need either to be repaired or penalized accordingly, which is often a crucial part to achieve good convergence properties. In contrast to the above sketched generational model, it is also possible to use a *steady-state* approach. In this case just one individual is changed at a time.

Many further techniques exist, in order to prevent premature convergence to a global optimum. These include, but are not limited to the introduction of sub-populations ("niching") to support genetic diversity, or advanced population management techniques, ensuring a certain amount of diversity within the population. For a more detailed treatment of these issues the reader is referred to [54, 80, 3], which is however just a small selection of existing literature related to this topic.

Other variants of EAs are *evolution strategies* and *evolutionary programming*, both being similar to each other. In particular for continuous parameter optimization problems, ESs and EP are often considered to be more appropriate than GAs. Main differences are that parameters are directly encoded as real numbers, and mutation, based on the addition of (parametrized) Gaussian variables, is the primary operator. In particular ESs provide means for dynamic adaptation to local properties of the search space, by *self-adaptation of strategy parameters*. This property allows for faster convergence rates towards (local) optima. Furthermore, in contrast to GAs, no numeric issues w.r.t. the binary encoding of the parameters can occur. For an extensive treatment of the topic the reader is referred to [2].

Genetic programming is a popular machine learning technique with the goal of the derivation and optimization of formulae or computer programs to perform a specific task. Individuals are usually encoded by means of a tree structure corresponding to the syntax or parse tree of the corresponding program or formula. As a result of increasingly available computational power numerous successful applications of GP have been reported, e.g. in the areas of electronic circuit design, quantum computing, but also to create "inventions", cf. [68].

Many recent topics and achievements regarding evolutionary algorithms are covered in great detail in [3].

2.7 Hybrid Algorithms

Various exact and heuristic algorithms have largely different properties regarding their ability to find feasible or high-quality solutions fast, or to solve certain subproblems or even prove optimality for the considered problem. By exploiting individual advantages, a combination of auspicious methods may lead to *hybrid algorithms* with overall superior performance. During the last years, hybrid algorithms gained much attention amongst the scientific community, in particular regarding the application to combinatorial optimization problems. A comprehensive overview about the state-of-the art is provided in [10].

2.7.1 Hybrid Metaheuristics

Typically, different metaheuristics applied to a particular problem show different characteristics and may have different advantages. The motivation behind hybridizing metaheuristic primarily is to obtain a method with superior performance that exploits the individual advantages of the single heuristics. In particular for difficult large scale optimization problems hybrid approaches can more and more be found amongst the leading algorithms. A taxonomy for hybrid metaheuristics is proposed in [91] and pursued in [94]. Main properties are the types of algorithms being hybridized, level of hybridization, order of execution, and underlying control strategy.

A very successful type of combination is to use local improvement methods as subordinate method guided by another metaheuristic, which is itself primarily designed to find promising regions of the search space. Local search methods are often able to find a local optimum ("top of the hill") fast, but lack the ability to perform a global search covering many different promising regions of the search space. In a hybrid approach this particular task is referred to the guiding heuristic as well as the control of the global search strategy, i.e. intensification vs. diversification. Within the evolutionary computation community the term *memetic algorithm* is used for genetic algorithms guiding an additional subordinate heuristic.

For complicated large-scale real-world problems attempts of directly solving it by metaheuristic techniques are often impracticable due to high running times and poor average solution quality. Multi-stage approaches frequently yield significant improvements, if wise decompositions into subproblems can be found. Multi-level refinement strategies [109] attempt to obtain higher-level approximations to the original problem by coarsening, and then solve, iteratively refined problems, at each level.

Due to the increasingly available hardware facilities for parallel computing within the last years, the parallel approach becomes more and more important. One approach is to execute multiple metaheuristics in a collaborative way in order to improve overall convergence properties and success ratios by consecutive exchange of solutions and information. On the other hand, in particular subordinate methods may be subject to parallelization themselves, for instance by using graphics processing units for the efficient solution of subproblems, cf. [20].

Numerous applications and various examples of combinations of metaheuristics are presented in [55].

2.7.2 Hybridizing Exact and Heuristic Algorithms

According to [89], hybrid approaches combining exact and heuristic algorithms can be classified as follows:

- *Collaborative Combination:* Algorithms exchange information during their execution, which may be performed sequentially, intertwined or in parallel.
- *Integrative Combination:* In this case one algorithm is part of the other one. Either the exact as well as the heuristic algorithm can be used as subordinate method.

It is often the case that considered instance sizes are too large for the direct application of exact methods. Hence, metaheuristics are frequently used to derive a restricted search space, which is usually obtained by considering all elements being part of the best solutions created by the metaheuristic. Subsequent applications of exact methods are often able to obtain further improvements. It is however also possible to start the computation with an exact procedure like B&B, but stop the computation at a specified level of the B&B-tree. With the subsequent application of heuristics feasible solutions are then derived from these partial ones. Such approaches are particularly useful if the first decisions are very critical.

Parallel and intertwined collaborative approaches are less frequent, probably due to the higher complexity of such approaches. However, frameworks for cooperative construction and modification of solutions have been proposed, as for instance *asynchronous teams* (AT) [103], or *teams for cooperative heterogeneous search* (TECHS) [38]. Both approaches have in common that algorithms of very different complexity, including different running times and memory requirements, interact by constructing or modifying candidate solutions in a collaborative way. Within the AT approach these algorithms asynchronously operate on a pool of candidate solutions stored in a shared memory. In contrast, TECHS works by sequential execution of the contributing algorithms, which is however, periodically interrupted. At this time gathered information is passed to further contributing algorithms.

Turning our attention to integrative combinations now, we first consider the case where the exact method is used as subordinate algorithm. In many cases it turned out to be beneficial to solve certain subproblems with exact methods. Examples are recombination or mutation operators within EAs, as well as local improvement methods. Furthermore information provided by LP-relaxations can be exploited by metaheuristics, cf. [90]. A further application of exact algorithms is to explore *very large-scale neighborhoods* (VLSN) (cf. [1]), as for instance performed by a dynamic programming approach for the *dynasearch* neighborhood [27].

A further, frequently applied approach is to embed heuristics into exact algorithms, with the primary goal of speeding up the overall process. A popular approach is to use *primal*

heuristics within the B&B method, to obtain feasible solutions or better incumbent solutions fast. Starting point for such heuristics is usually the LP-relaxation of the B&B-nodes at which the heuristic is applied. A further beneficial way of embedding heuristics is to use them for cut-generation or for solving the pricing problem within column-generation. If exact polynomial-time separation algorithms for some considered cutting-planes are too slow for practical purposes, or have not been found at all, heuristic cut-separation may often be a sufficient mean to benefit from the application of these cutting-planes, though. Another approach called *local-branching* incorporates the idea of local search into the B&B framework [48]. If MIPs contain binary variables, decisions on these variables usually have a high impact on the objective values of corresponding solutions to the whole problem. Finding feasible incumbent solutions having cleverly devised values of the binary variables is thus of high importance for the overall performance of the B&B process. Having such an incumbent solution at hand, the idea of local branching is to incorporate local search around this particular solution into the B&B process. For this purpose, branching is performed in such a way, that the first branch contains all solutions with a Hamming distance smaller than a certain value, whereas the second branch contains all other solutions. Such constraints can be formulated in terms of linear constraints for the binary variables. After such a branching it is enforced that the first subtree is entirely solved in a classical way before the second branch is considered, with which is dealt with in the same way again. This closely corresponds to performing local search around the considered feasible solution and is thus likely to yield improved feasible solutions fast. Further potential of improvement exists regarding the node-selection strategies used within the B&B algorithm. In [67] the authors optimized this strategy by GP, based on the information gathered in the hitherto search process.

A comprehensive survey regarding the combination of mathematical programming methods with metaheuristics – often called *matheuristics* – is given in [93], numerous examples can be found in [78].

Chapter 3

Heuristic Methods

Intelligence is quickness to apprehend as distinct from ability, which is capacity to act wisely on the thing apprehended.

Dialogues of Alfred North Whitehead (1953)

Before we work on artificial intelligence why don't we do something about natural stupidity?

Steve Polyak

Metaheuristic methods have become rather popular in recent decades, as they are usually the only chance to handle medium to large scale data instances of difficult $\mathcal{NP}$-hard problems in practice. Most existing work related to the MLST problem can be found in the field of metaheuristic methods, and is reviewed in this chapter. Subsequently the application of metaheuristics like *greedy randomized adaptive search procedures* and *ant colony optimization* to the minimum label spanning tree problem is described.

3.1 Previous Work

The minimum label spanning tree (MLST) problem has been introduced by Chang and Leu [16] for the first time. In this work the authors showed the MLST problem to be $\mathcal{NP}$-complete, and proposed an exact and an approximative algorithm.

Here we present a proof with minor differences to the proof given in [16]. The proof proceeds by reduction of the *set-covering problem*, well known to be $\mathcal{NP}$-complete, to

MLST. The decision-variant of the problem required for this reduction asks for the existence of a feasible solution with $|L_T| < K$ for some integer K. The problem is clearly in $\mathcal{NP}$, as it is possible to check in polynomial time the feasibility of a given solution. Let $S = \{a_1, a_2, \ldots, a_n\}$ be the set of elements to be covered in the set covering problem, and $C_1 = \{a_{1_1}, a_{1_2}, \ldots, a_{1_{|C_1|}}\}, C_2 = \{a_{2_1}, a_{2_2}, \ldots, a_{2_{|C_2|}}\}, \ldots, C_m = \{a_{m_1}, a_{m_2}, \ldots, a_{m_{|C_1|}}\}$ be subsets of S. Let $C = \{C_1, \ldots, C_m\}$. The set covering problem asks for a minimum cardinality subset C' of C, such that $\bigcup_{C_i \in C'} C_i = S$. Figure 3.1 depicts graph G constructed from S and $C_i \in C$ having the property that there is a covering of S with K subsets of C if and only if G has a MLST with K labels. Graph $G = (V, E)$ is defined as follows: $V = \{a_1, \ldots, a_n, C_1, \ldots, C_m, s\}$, where s denotes an artificial root node; The edges are given by $E = \{(s, C_i \mid i = 1, \ldots, m\} \cup \{C_p, a_{p_l} \mid p = 1, \ldots, m,\ l = 1, \ldots, |C_p|\}$. All edges incident to s are labelled by l_s, all other edges being incident to C_p are labelled by l_p. The graph G fulfills the required property, i.e. G has an MLST with K labels if and only if a minimum covering of size $K - 1$ exists, and can be constructed within polynomial time.

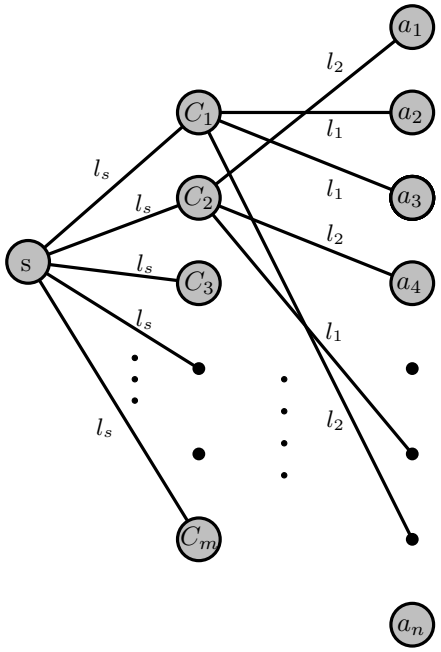

Figure 3.1: Construction of G for the proof of $\mathcal{NP}$-completeness of MLST. Image based on illustration in [16].

3.1.1 Construction and Approximation Algorithms

In [16], Chang and Leu propose a constructive method, called *Maximum Vertex Covering Algorithm* (MVCA). The basic idea is to start with a graph containing no edges, and then, iteratively select a label l and add the corresponding edges to the graph, until a connected subgraph is obtained. The greedy-criterion for the selection of the next label to consider is based on the number of currently uncovered vertices. However, this algorithm may

fail to yield connected graphs, as pointed out in [30]. Krumke and Wirth [69] proposed a modified construction algorithm (see Algorithm 7), which uses the reduction of the number of connected components achieved by the addition of label l to the current partial solution as greedy criterion. It is not specified which label to take in the case of more than one label yielding minimal number of components in line 4.

Algorithm 7: (modified) MVCA

```
1 H = (V, E_H = ∅)
2 L_H = L
3 while H is not connected do
4     l ← argmin_{l∈L\L_H} nr. of components (G = (V, E_H ∪ E(l)))
5     E_H = E_H ∪ E(l)
6     L_H = L_H\l
7 end
```

This algorithm is called MVCA or modified MVCA in literature though, despite the name not any longer accurately describing its behaviour. Within this thesis, we will follow this terminology and call the modified construction algorithm from [69] MVCA.

Further variations and enhancements of MVCA have been presented [115] and [30]. Many suggested improvements comprise a refined decision of which label to select in line 4 of Algorithm 7, but also hybridization with the pilot method. Such hybridizations, but also multiple executions of a randomized MVCA generally yield better results at the expense of higher running times. Simply considering the label frequency as discriminative factor in the case of ties for the decision in line 4 of Algorithm 7 turned out to be advantageous, as running times are barely reduced, but improvements regarding the solution quality have been obtained.

In [69] the authors present a performance guarantee of $2 \ln |V| + 1$ for the modified MVCA, and furthermore show, that no constant-factor approximation algorithm does exist for the MLST problem. An improved bound of $\ln(|V| - 1) + 1$ is obtained by Wan, Chen and Xu [110]. A tight bound has been derived by Xiong, Golden and Wasil [114]: if G contains no label occurring more than b times, no solution H_b times greater than the optimum will be constructed by MVCA, with

$$H_b = \sum_{i=1}^{b} \frac{1}{i} \tag{3.1}$$

being the b-th harmonic number. A further result according to approximability has been obtained by Brüggemann et al. [11]:

Theorem 7 *For $r \geq 3$ the problem* MLST_r *is APX-complete even if the input graph G is restricted to be bipartite and of maximum degree 3.*

3.1.2 Local-Search-Based Algorithms

Neighborhood structures for local search algorithms have been first analyzed by Brüggemann, Monnot and Woeginger [11]. In particular they consider the MLST_r problem, which is defined as having maximum label frequency r, and analyze the k-switch neighborhood for this problem. Therein, the k-switch neighborhood is defined as follows.

Definition 15 *Let $k \geq 1$ be an integer, and let L_1 and L_2 be two feasible label sets for some instance of MLST. Then the set L_2 is in the k-switch neighborhood of L_1 if and only if*

$$|L_1 - L_2| \leq k \quad \textit{and} \quad |L_2 - L_1| \leq k. \tag{3.2}$$

Hence L_2 can be obtained by first removing up to k labels from L_1, and then adding up to k labels to it. Then they prove the following theorem.

Theorem 8 *For any integer $r \geq 2$ and for any instance G of MLST_r the objective value of any local optimum with respect to the 2-switch neighborhood is at most a factor of $(r+1)/2$ above the optimal value.*

Furthermore it is shown that this bound is tight. For the k-switch neighborhood, the following result is presented in [11]:

Theorem 9 *For any integer $k \geq 2$, for any integer $r \geq 2$ and for any real $\epsilon > 0$, there exists an instance G of MLST_r and a spanning tree T for G that is a local optimum with respect to the k-switch neighborhood, such that the objective value of T is at least $r/2 + \epsilon$ above the optimal objective value.*

So far, many metaheuristics based on local search have been applied to the MLST problem. In [15] the authors propose Simulated Annealing, Reactive Tabu Search, build on top of the 1-switch neighborhood, the Pilot Method with the MVCA as construction method, and a VNS using k-switch neighborhoods.

Consoli et al. [29] furthermore applied GRASP and VNS to the MLST problem. Within the first two GRASP iterations the best label (w.r.t. reduction of the number of connected components) is added to the initially empty solution. In further iterations a random label is used for this purpose. The restricted candidate list is filled with labels that induce a minimum number of connected components. Subsequent local search only consists of removing redundant labels from the solution created in the construction phase.

3.1.3 Further Metaheuristic Algorithms

A genetic algorithm (GA) for the MLST problem has first been proposed in [114]. The algorithm operates on a list of labels, encoding a feasible solution. Crossover is performed by combining the label-lists of two candidate solutions, then reordering the labels l decreasing w.r.t. to $|E(l)|$, and finally taking as many labels till the corresponding graph is connected. Mutation is performed by first adding some label l to the solution, and then iteratively removing labels in increasing order of their label frequency as long as the solution remains valid. The algorithm is designed to require just one single parameter, the population size; selection and replacement is performed in a deterministic way, the number of iterations is set to the number of individuals in the initial population.

A modified GA has been presented in [115]. Within this variant, crossover is again performed by considering the combined list of labels of two individuals, but then applying MVCA on this particular set of labels. Although having longer running times, better solutions could be obtained, yielding less required generations. These GAs, as well as modified MVCA variants are also discussed in detail in the PhD-thesis of Xiong [113].

A further genetic algorithm has been proposed in [85]. Candidate solutions are encoded as permutations of all labels, decoding end evaluation is performed simultaneously by iterating through the permutation, adding as many labels as required to build a feasible solution, i.e. a connected subgraph. After evaluation, a redundancy check is performed, reordering redundant labels immediately after the feasible ones. Crossover is performed by alternatively taking labels from the parents, and then eliminating duplicate labels by always keeping the first occurrence. Mutation is performed by swapping the one value of the feasible part with another one of the infeasible part of the permutation.

3.2 Ant Colony Optimization

In this section we describe our new ant colony optimization approach (ACO) to the MLST proble, based on the generic ACO procedure listed in Algorithm 5 in Chapter 2. In step "solution construction" in line 3 of Algorithm 5 further labels are added iteratively, starting from an empty set of labels. The decision which label to take next is based on a probability distribution defined over all labels reducing the number of components implied by the labels (and corresponding edges) of the current partial solution, parametrized by pheromone values τ and local information η, to be defined in detail subsequently. In the following we describe the algorithm in detail, starting with the description of underlying pheromone models.

3.2.1 Pheromone Models

The most natural or obvious formulation is to introduce a pheromone value for each label $l \in L$, i.e. $\tau_l,\ l \in L$ (model P-I). This representation has the advantage of being very compact, but however, mutual dependencies are not represented very accurately. This

might be a problem as within this simple pheromone model the algorithm cannot gather information of particular labels being of high potential importance w.r.t. some constructed subset $L' \subset L$, but being unimportant regarding other subsets $L'' \subset L$. Such information can be better reflected by the following larger pheromone model (P-II). Let τ_{ij} denote the entries of a $|L| \times |L|$ matrix. Let us again assume, we already have constructed label set L' and are about to decide which label to take next. The desire to add label l is then given by

$$\tau(L', l) := \sum_{i \in L'} \tau_{il}. \tag{3.3}$$

A third alternative arises, when considering the main purpose of pheromones as biasing the solution process of the MVCA-heuristic (P-III). Hence we can introduce a $|L^{\text{mvca}}| \times |L|$ matrix, were L^{mvca} denotes the set of labels constructed by the MVCA-heuristic. Each row i of this matrix then provides a probability distribution for all labels to be used in the i-th construction step of each ant.

3.2.2 Solution Construction

In each iteration M ants construct solutions based on pheromones and local information. Let $\tau(L', l) := \tau_l$ for pheromone model (P-I), and $\tau(L', l) := \tau_{|L'|,l}$ for (P-III). Having constructed labels L' the probability for ant m of adding label l is given by

$$p^m_{L',l} = \begin{cases} p(L', l) & \text{if } c(L' \cup l) < c(L'), \\ 0 & \text{otherwise,} \end{cases} \tag{3.4}$$

with

$$p(L', l) = \frac{\tau(L', l)^\alpha \cdot \eta(L', l)^\beta}{\sum_{l' \in \{l \in L | c(L' \cup l) < c(L')\}} \tau(L', l')^\alpha \cdot \eta(L', l')^\beta}, \tag{3.5}$$

and $c(L')$ denoting the number of connected components of graph $G = (V, E')$ where E' denotes the set of edges with a label from set L'. The balance between pheromone information $\tau(L', l)$ and local information $\eta(L', l')$ is controlled by parameters α and β. Let $E(L') \subset E$ denote the subset of edges having associated to a label $l \in L'$, and let $c(L')$ denote the number of connected components of the subgraph $G' = (V, E(L'))$. Local information is then defined by expression

$$\eta(L', l) = c(L') - c(L' \cup l). \tag{3.6}$$

Hence, if setting $\alpha = 0$ we obtain a randomized greedy heuristic, essentially following the greedy criterion from the MVCA-heuristic. In general, parameters α and β balance between following the already gathered global information provided by pheromone trails and the local information resulting from direct improvements of the label w.r.t. the current partial solution. Using simpler models for the local information, like simply considering the number of edges associated to the particular label turned out not to work well in preliminary tests.

The most obvious way to perform the construction process is to add further labels to an initially empty set L' until a feasible solution is obtained (C-I). However, solutions having more labels than the best-so-far solution likely do not contain any useful information with regard to finding further improvements and should therefore deposit no pheromone values. Even solutions having the same number of labels as the best-so-far solution may not contain such information, as possible lower cardinality solutions might have significant differences or even be completely disjoint to many or all higher cardinality solutions (without redundant labels). This observation suggests an alternative solution construction process (C-II) which limits the number of constructed labels to the size of the best-so-far solution decreased by one. This approach intends to minimize the number of connected components and at the same time maximize the number of additional arcs being represented within these fixed cardinality label sets. This strategy aims to increase the likelihood of ending up with connected feasible solutions after the addition of the last label. The overall construction process can also be performed in a combined way, i.e. one half of the ants constructing feasible, the other one infeasible ones (C-III).

To counteract stagnation we optionally perform pheromone smoothing, similar to the approach presented in [102]. Hence, we replace each $\tau(L', l)$ in Equation (3.5) by

$$\tau(L', l) + \lambda' \cdot \left(\max_{l \in L \setminus L'} \tau(L', l) - \min_{l \in L \setminus L'} \tau(L', l) \right), \tag{3.7}$$

where λ' is controlling the amount of smoothing. It is initially set to zero, and in case of stagnation, indicated by no improvement within it_λ iterations, successively increased in steps of λ / it_λ until the maximum value λ is reached. If no improvement occurs for $2 \cdot it_\lambda$ iterations, we reinitialize pheromones entirely.

3.2.3 Pheromone Update

Pheromone update, which takes place after each iteration basically consists of two components: evaporation and deposition. Whereas pheromone evaporation provides a mean for escaping local optima and enables to direct the search towards other regions of the solution space, pheromone deposition is the main mean to guide the search process towards regions appearing attractive as a result of solutions created so far.

Pheromone evaporation is governed by parameter ρ, the evaporation rate. The update rule is given by

$$\tau_i \leftarrow (1 - \rho') \cdot \tau_i, \tag{3.8}$$

where index i refers to all elements of the pheromone vector or matrix respectively. For (P-I) and (P-III) we directly use parameter ρ for ρ', in the case of (P-II) we set $\rho' = \rho / |L|$ in order to obtain a comparable evaporation for each particular label.

Afterwards, pheromone deposition is performed based upon certain solutions. Various strategies do exist regarding to which solutions are selected for this purpose, which we leave unspecified for the moment. However, all update rules have in common, that an

amount $\Delta L'$ is added to the pheromone values corresponding to the labels of solution L'. Regarding (P-I) the update is straightforwardly performed by

$$\tau_l \leftarrow \tau_l + \Delta L', \text{ for all } l \in L'. \tag{3.9}$$

As pheromone model (P-III) also accounts for the position at which a certain label has been added we have to consider L' as an ordered set and refer to the label constructed in step i by $L'[i]$. The pheromone update is then performed by

$$\tau_{il} \leftarrow \tau_{il} + \Delta L', \text{ for each } L'[i]. \tag{3.10}$$

For model (P-II) the update is performed in the following way:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta L', \text{ for all } i,\ j \in L'. \tag{3.11}$$

Following the approach of $\mathcal{MAX} - \mathcal{MIN}$ Ant System we introduce upper and lower bounds for the pheromone values $\tau^{\max}$ and $\tau^{\min}$. Pheromone values are initialized by the arithmetic mean of these two values.

The value $\Delta L'$ is calculated differently for the two proposed construction methods. If only feasible solutions are constructed, we only deposit pheromone for solutions having no more labels then the best-so-far solution.

In order to evaluate constructed candidate solutions we need to develop a function $f(L')$ which discriminates between solutions with equal $|L'|$. An evaluation function $f(L')$ can be built by considering function

$$h(L') = 1 - \frac{|E(L')|}{|E|}, \tag{3.12}$$

which accounts for additional edges being represented by labels L'. Feasible solutions can thus be evaluated by $f(L') = |L| + h(L')$.

If, on the other hand, infeasible solutions are constructed, we primarily have to account for the number of connected components induced by L'. We therefore use $f(L') = c(L') + h(L')$ in this case. In addition to the best-so-far feasible solution we also globally store the best-so-far created infeasible solution.

The pheromone deposition is performed by the best-so-far as well as the iteration-best ant in each iteration. If mixed construction (C-III) is performed, a total of four ants deposit pheromone. For both construction mechanisms we use $\Delta L' = 1$ when pheromone models (P-I) or (P-III) are used, and $\Delta L' = 1/|L'|$ in the case of model (P-III), which compensates the fact that for each label totally $|L'|$ pheromone values are increased. A further differentiation regarding the amount of pheromones to be deposited seems not to be reasonable within this context. In order to not implicitly limit pheromone values, we perform the evaporation step after the pheromone deposition.

3.2.4 Local improvement

Typically used neighborhoods for the MLST problem consist of the replacement of k labels within the current solution. Using $f(L') = |L'| + h(L')$ again allows for a better discrimination of solutions of equal cardinality. Regarding the solution construction process restricted to the cardinality of the incumbent solution minus one (see Section 3.2.2), we might also consider local search procedures working on infeasible solutions. Within such a process, solutions are evaluated by $f(L') = c(L') + h(L')$. Besides the specification of the size k of the neighborhood, we may consider various reinsertion strategies after having removed k labels. Completely traversing the whole neighborhood w.r.t. to some solution with already having k labels removed, is impracticable, in particular for larger k. Hence we follow the strategy to consider all extension candidates for the first k' places, and add the remaining $k - k'$ labels following the greedy MVCA strategy, similar to the approach used in [29].

A further local improvement method consists of simply checking the labels for redundancy, by removing each label of the solution, and then test for connectivity. In particular for solutions which labels induce many additional edges, this method is more likely to be successful.

3.2.5 Implementation Aspects

In order to compute the number of components induced by a certain set of labels L' we use a *disjoint set* (also called *union find*) datastructure (see for instance [33]), as also suggested in [69]. If we are considering edge set E', all the operations on the union find data structure can be carried out in a total time of $O(|E'| \cdot \alpha(|E'|, |V|))$, where $\alpha(|E'|, |V|)$ denotes the inverse *Ackermann function*. For all reasonably occurring $|E'|$ and $|V|$ it holds, that $\alpha(|E'|, |V|)) \leq 4$. In comparison, a depth-first search (DFS) procedure would take time $O(|V| + |E'|)$. Within local search algorithms but also the MVCA-heuristic and therefore also the computation of the local information within the ACO algorithm, we often face the situation of tentatively adding some label for evaluation, and then removing it immediately. Hence we can further benefit from disjoint set datastructures supporting a rollback mechanism. The major benefit of this approach is that we can evaluate all further labels w.r.t. some considered partial solution in quasi-constant time. Let us further consider the situation where we want to remove some label l' from a partial solution L', which also occurs within the local search procedure. In this case we need to rebuild the disjoint set datastructure which takes $O(|E(L' \backslash l')| \cdot \alpha(|E(L' \backslash l')|, |V|))$ time. In this situation DFS allows for an incremental computation by first removing $|E(l')|$ edges from the graph, and then running DFS ($O(|V| + |E(L' \backslash l')|)$). However, compared to the time required to rebuild the disjoint set datastructure $O(|E(L' \backslash l')| \cdot \alpha(|E(L' \backslash l')|, |V|))$, this is no real drawback. Consequently we consider the disjoint set datastructure to be overall more appropriate for the given task.

3.3 Unified Constructive Framework – GRASP

Furthermore we apply *greedy randomized adaptive search procedures* (GRASP) to the MLST problem. Our approach is different to the one proposed in [29] as we again use the more discriminative evaluation function $f(L') = c(L') + h(L')$ and describe the underlying randomized construction heuristic in a framework, which also consists of the classic MVCA algorithm as a special case, but also certain variations as for instance proposed in [30].

The parameters for Algorithm 8 are summarized in Table 3.1 and are discussed in detail after the description of the algorithm. The outer loop (line 3) in Algorithm 8 is executed as long as the solution set L' does not imply a connected graph. The next loop (line 5) then iterates over all labels that are not yet part of the partial solution L'. The labels can optionally be considered in decreasing order of the label frequency $|E(l)|$ which is advantageous for the variants of Algorithm 8 that for performance reasons explicitly prevent to iterate over all labels in each iteration. Within this loop the *restricted candidate lists* RCL_i are created. Each time a new improvement is found w.r.t. the number of connected components, index i is increased (line 7), and a new RCL_i is created for this new index value. The previous RCLs are kept, as they may optionally be used to enhance the final RCL (labelled with $\mathrm{RCL}_{\mathrm{tot}}$) to facilitate a more diverse construction process. This behaviour is controlled by parameter *threshold* and performed in the loop of line 25. The current RCL is then extended by the considered label if the condition of line 13 holds, i.e. the number of connected components induced by the current label is equal to the best number of connected components found so far. In line 16 it is then checked if the size of the current RCL is greater or equal to the size specified by rcl_{size}. If the maximal number of improvements is exceeded, the iteration over all labels in $L \backslash L'$ is quit, otherwise f^* is rounded down in order to allow only labels implying a further reduced number of connected components to be added to the RCL. The label to be added to L' is finally selected in line 28. The decision is usually performed in a random way, but may be also be performed based on $f(L' \cup \{l\})$, which is discussed in more detail subsequently.

Algorithm 8 describes several variations of MVCA and corresponding randomized variations in a unique framework. Table 3.1 gives an overview of possible parameter configurations. Setting $rcl_{\mathrm{size}} = \infty$, $imp_{\max} = \infty$ and $threshold = 0$ implies the classic MVCA. If $rcl_{\mathrm{size}} = \infty$ and the label selection in line 28 which always selects the label with minimal $f(L' \cup \{l\})$, i.e. the label which in addition to the best reduction of the number of connected components also covers most further edges in the graph, we obtain an enhanced MVCA variant. In Table 3.1 this variant is labelled with "impr. MVCA". Any parameter settings are, however, possible to be used as a randomized construction heuristic to be used for GRASP, as listed in column "GRASP" of Table 3.1. Smaller values of $imp_{\max}$ may be used to speed-up the construction process, which is particularly meaningful for instances containing a huge number of labels. In this case, the iteration over possible extensions to L' is stopped, as soon as $imp_{\max}$ improvements w.r.t. the number of components are obtained. The final RCL usually contains only the best labels (regarding their reduction of the number of connected components). However, when setting $threshold > 0$ also further labels can be used for this purpose to achieve a stronger diversification of the constructed

Algorithm 8: randomized MVCA

```
1  L' ← ∅    // currently used labels
2  f* ← ∞    // to store best objective
3  while c(L') ≠ 1 do
4      i ← 0
5      for all l ∈ L\L' (optionally in decreasing order of |E(l)|) do
6          if ⌊f(L' ∪ {l})⌋ < ⌊f*⌋ then
7              i ← i + 1
8              RCL_i = ∅
9          end
10         if f(L' ∪ {l}) ≤ f* then
11             f* ← f(L' ∪ {l})
12         end
13         if ⌊f(L' ∪ {l})⌋ = ⌊f*⌋ then
14             RCL_i = RCL_i ∪ {l}
15         end
16         if |RCL_i| ≥ rcl_size then
17             if imp_max ≠ 0 ∧ i ≥ imp_max then
18                 break (exit for loop)
19             else
20                 f* ← ⌊f*⌋
21             end
22         end
23     end
24     RCL_tot = RCL_i
25     for j = 1, ..., threshold do
26         RCL_tot = RCL_tot ∪ RCL_{i−j}
27     end
28     l ← element from RCL_tot
29     T ← T ∪ {l}
30     L' ← L'\{l}
31 end
```

solutions. In the right part of Table 3.1 we list two particular configurations, to be used for our computational experiments with GRASP, which are presented in the following section.

3.4 Computational Results

In this section we present computational results for the presented ACO and GRASP approaches. For our computational experiments we considered the instance set also used

Table 3.1: Parameters of the unified construction framework.

Parameter	Description	MVCA	impr. MVCA	GRASP	Var. I	Var. II
rcl_{size}	size of RCL	∞	∞	any	20	20
$imp_{\max}$	limit for improvements in each construction step	∞	∞	any	3	∞
threshold	threshold for labels considered in each construction step	0	0	any	0	0
label selection	line 28 in algorithm 8	random	best $f(L' \cup \{l\})$	random	random	random
it	iterations	n/a	n/a	n/a	30	30

in [30, 29, 28, 15]. All tests have been performed on a Intel Nehalem E5540 (2,53 GHz) CPU, under Linux with Kernel 2.6.31.

3.4.1 Ant Colony Optimization Results

Within comprehensive preliminary testing we determined a generally well-working configuration of the presented algorithmic components of the ACO algorithm. Table 3.2 gives a summary of the determined parameter settings used for the subsequently presented computational results. Relatively high values of β are required, to give the labels mostly reducing the number of connected components a reasonable high chance of being selected. On average, in particular in the first construction steps, almost all labels will provide comparable reductions in the number of connected components, but higher reductions provide significant information that should be exploited.

Table 3.2: Parameter settings

Description	Parameter	Value
minimum pheromone value	$\tau^{\min}$	10^{-3}
maximum pheromone value	$\tau^{\max}$	10
pheromone-contribution	α	2
local-information-contribution	β	12
evaporation rate	ρ	0.1
pheromone-smoothing parameter	λ	0.2
iterations for pheromone smoothing	it_λ	20

Table 3.3 shows our results obtained for the instances with $|V| = 200$ and $|V| = 500$. Results have been computed with the parameter settings listed in Table 3.2 and $it = 100$ iterations and $M = 20$ ants. Columns VNS contain the results of the variable neighborhood search presented in [29], being the best method therein. For a certain number of nodes, groups with various ratios of number of labels compared to the number of nodes as well

Table 3.3: ACO results for instances with $|V| = 200$ and $|V| = 500$.

Parameters			VNS		ACO					
$\|V\|$	$\|L\|$	d	obj.	$t_{\mathrm{b}}[s]$	obj.	$\bar{\sigma}_{\mathrm{obj}}$	$t_{\mathrm{avg}}[s]$	$\bar{\sigma}_{\mathrm{t}}$	$t_{\mathrm{b}}[s]$	$\bar{\sigma}_{\mathrm{tb}}$
200	50	0.8	2.0	0.0	2.00	0.00	136.09	23.51	0.00	0.00
		0.5	2.2	0.03	2.20	0.00	15.21	6.59	0.07	0.26
		0.2	5.2	0.23	2.20	0.00	15.21	6.59	0.07	0.26
200	100	0.8	2.6	0.14	2.60	0.00	107.10	83.67	1.16	2.66
		0.5	3.4	0.16	3.40	0.00	23.48	7.93	2.41	5.07
		0.2	7.9	2.9	8.09	0.16	15.52	2.23	2.09	3.71
200	200	0.8	4.0	0.08	4.00	0.00	39.54	6.26	0.00	0.00
		0.5	5.4	0.88	5.40	0.00	33.56	5.10	7.58	8.25
		0.2	12.0	33.7	12.35	0.13	26.38	3.64	3.70	5.21
200	250	0.8	4.0	1.5	4.02	0.04	42.56	7.17	9.32	13.20
		0.5	6.3	2.3	6.37	0.05	35.87	5.34	4.53	6.48
		0.2	13.9	1.5	13.98	0.11	34.12	4.80	4.96	6.32
500	125	0.8	2	0.05	2.00	0.00	81.70	6.18	0.00	0.00
		0.5	2.6	0.56	2.61	0.01	109.32	18.48	6.10	20.25
		0.2	6.2	3.7	6.25	0.07	103.22	17.26	14.93	24.58
500	250	0.8	3	0.49	3.00	0.00	188.46	4.40	0.00	0.00
		0.5	4.1	26.9	4.26	0.08	196.18	57.65	7.52	32.97
		0.2	9.9	10.2	10.16	0.18	160.47	19.12	22.96	41.19
500	500	0.8	4.7	8.6	5.0	0.00	403.84	27.57	6.26	51.01
		0.5	6.5	110.2	7.30	0.20	365.20	47.80	20.02	70.72
		0.2	15.8	50.3	16.63	0.30	356.08	55.81	55.06	90.68
500	625	0.8	5.1	0.97	5.56	0.12	487.07	29.38	5.67	57.24
		0.5	7.9	33.9	8.39	0.08	629.85	62.70	41.04	101.34
		0.2	18.3	60.0	19.37	0.33	593.66	71.82	60.10	133.56

as various graph densities $|E| = d \cdot \frac{|V| \cdot (|V|-1)}{2}$ exist. For each group ten different instances do exist. Reported objective values (column "obj.") and running times t_b at which the best solution was found, are average values over these ten instances. In columns ACO we report our results for 30 independent runs. Objective function values are listed in column "obj.", corresponding standard-deviations in column $\bar{\sigma}_{\mathrm{obj}}$. By t_{avg} we denote the average total running times, and by t_{b} the average times at which the best solutions of the individual runs have been obtained. Corresponding standard-deviations are listed in columns $\bar{\sigma}_{\mathrm{t}}$ and $\bar{\sigma}_{\mathrm{tb}}$. With this particular parameter settings it is possible to obtain good solutions relatively fast, but however, in particular for the low density instances the average objective function values are generally higher than the ones obtained by VNS.

In Table 3.4 we report the results for various configurations of the ACO algorithm for a selected subset of low density graphs, i.e. the hardest instances within the sample. Again, we performed 30 independent runs for each instance and used the parameter settings from Table 3.2, but 200 iterations and $M = 50$ ants. If local search is applied (indicated in

column "LS"), we set $it = 100$ and $M = 30$ to compensate for the longer computational time required for each local improvement. The best objective value obtained for each group of instances is highlighted in the table. Unfortunately it is not possible to draw a clear conclusion which pheromone model is overall superior. Model (P-II) yields the best results for instances with $|V| = 200$, $|L| \in \{100, 250\}$, but is generally worse for larger instances. For these instances (P-I) and (P-III) yield comparable results. The construction method (C-I) generally shows the worst performance on these instances, (C-II) and (C-III) show a similar average performance.

Various configurations regarding the subordinate local search method have been evaluated in preliminary experiments. However, due to longer running times, no configuration could outperform ACO without local improvement. To limit the time requirements for the local search, it turned out to be advantageous only to apply it for the best solutions regarding number of labels and number of components for feasible and infeasible solutions respectively. The neighborhood size was set to $k = 2$, as smaller neighborhoods did not yield sufficient improvements, and traversing larger neighborhoods turned out to be too time-consuming.

Although the running times of the configurations reported in Table 3.4 are higher than the ones with less iterations and ants reported in Table 3.3, they are still reasonable for many purposes. With these configurations improved average solution values compared to [29] could be obtained for some groups of instances.

3.4.2 GRASP Results

Table 3.5 shows the results obtained with the presented GRASP using the parameter settings listed in Table 3.1. Again, the results of 30 independent runs, each one consisting of $it = 30$ iterations, are reported. For the local search we use $k = k' = 2$. Within preliminary tests this configuration was found to be superior over $k = 1$, which corresponds to a simple redundancy check for each label of the current solution. Running times for local search have been limited to 30 seconds for each individual run after the construction phase. This timelimit is, however, only reached for the largest instances reported in Table 3.5.

The results show that the particular value of parameter $imp_{\max}$ does not affect the obtained solution qualities very much. It may, however, yield a significant speedup for instances with a huge amount of labels and thus be beneficial for instances as occurring in the context of the data compression application presented in Chapter 5. The results reported in Table 3.5 are superior to the results of ACO in the fast setup reported in Table 3.3. But, however, no parameter configuration has been found that yields better results with slightly increased running times. This is due to the dramatic increase of running times for larger k or $k' < k$.

Table 3.4: Comparison of different pheromone models and construction mechanisms for instances with $|V| = 200$ and $|V| = 500$; $M = 50$, $it = 200$.

$\|V\|$	$\|L\|$	d	Pheromones	Construction	LS	obj.	$\bar{\sigma}_{\text{obj}}$	$t_{\text{avg}}[s]$	$\bar{\sigma}_{\text{t}}$	$t_{\text{b}}[s]$	$\bar{\sigma}_{\text{tb}}$
			P-I	C-I	–	8.17	0.07	80.49	12.43	3.68	11.98
			P-I	C-II	–	8.00	0.14	173.93	142.17	28.99	77.59
			P-I	C-III	–	8.03	0.15	75.30	14.18	7.56	17.24
			P-II	C-I	–	8.02	0.14	96.09	14.82	9.62	18.07
200	100	0.2	P-II	C-II	–	**7.91**	0.03	49.75	5.15	5.14	10.29
			P-II	C-III	–	7.93	0.06	75.03	9.95	8.80	17.01
			P-III	C-I	–	8.16	0.12	92.19	14.40	5.46	14.03
			P-III	C-II	–	7.95	0.10	108.51	76.15	17.86	40.03
			P-III	C-III	–	8.10	0.16	102.63	34.08	8.66	24.93
			P-III	C-I	✓	8.09	0.18	165.99	123.16	26.10	60.07
			P-I	C-I	–	12.27	0.07	116.88	14.70	6.62	13.35
			P-I	C-II	–	12.14	0.10	205.14	153.83	19.96	53.90
			P-I	C-III	–	12.19	0.13	103.74	27.96	8.63	18.18
			P-II	C-I	–	12.30	0.14	177.24	26.89	18.86	33.27
200	200	0.2	P-II	C-II	–	12.14	0.16	89.54	11.64	11.96	21.39
			P-II	C-III	–	12.24	0.18	136.87	18.49	11.70	24.25
			P-III	C-I	–	12.28	0.12	155.74	21.49	12.41	18.89
			P-III	C-II	–	**12.10**	0.10	113.32	55.58	11.01	22.86
			P-III	C-III	–	12.17	0.12	131.43	34.58	12.34	24.48
			P-III	C-I	✓	12.25	0.15	199.75	176.59	50.82	53.48
			P-I	C-I	–	13.90	0.02	132.10	20.27	6.08	8.82
			P-I	C-II	–	**13.89**	0.03	191.20	110.09	3.78	17.94
			P-I	C-III	–	13.90	0.02	114.24	17.90	3.74	7.65
			P-II	C-I	–	13.93	0.07	213.18	29.64	25.45	35.18
200	250	0.2	P-II	C-II	–	**13.89**	0.05	118.40	15.71	5.64	10.17
			P-II	C-III	–	**13.89**	0.03	166.75	22.85	10.68	19.93
			P-III	C-I	–	13.90	0.02	178.46	23.16	14.24	16.11
			P-III	C-II	–	13.90	0.00	133.80	39.43	3.96	3.07
			P-III	C-III	–	13.90	0.00	150.73	26.62	6.16	7.72
			P-III	C-I	✓	13.91	0.05	161.63	92.77	44.68	48.69
			P-I	C-I	–	6.23	0.05	491.23	81.82	14.91	27.67
			P-I	C-II	–	6.20	0.00	542.53	91.33	16.08	31.82
			P-I	C-III	–	6.20	0.00	466.21	74.85	18.61	37.56
			P-II	C-I	–	6.20	0.00	538.08	89.72	24.36	44.55
500	125	0.2	P-II	C-II	–	6.20	0.00	342.43	50.45	19.82	36.25
			P-II	C-III	–	6.20	0.00	487.29	86.41	26.15	50.40
			P-III	C-I	–	6.25	0.07	589.74	112.51	22.20	53.69
			P-III	C-II	–	6.21	0.03	537.69	256.29	29.22	82.07
			P-III	C-III	–	6.20	0.00	543.99	98.21	23.05	48.80
			P-III	C-I	✓	6.21	0.03	551.19	247.50	121.28	223.48
			P-I	C-I	–	9.93	0.12	494.51	77.73	53.44	91.76
			P-I	C-II	–	**9.87**	0.10	393.68	70.28	49.23	84.44
			P-I	C-III	–	9.89	0.10	445.86	67.57	59.85	102.39
			P-II	C-I	–	10.05	0.14	587.90	98.54	79.25	135.14
500	250	0.2	P-II	C-II	–	10.03	0.18	460.34	86.31	65.70	111.33
			P-II	C-III	–	10.05	0.19	491.29	82.24	73.31	126.40
			P-III	C-I	–	9.89	0.11	537.09	87.52	75.80	108.99
			P-III	C-II	–	9.91	0.10	384.20	68.47	47.06	70.65
			P-III	C-I	✓	10.03	0.18	612.77	289.21	190.16	267.34

Table 3.5: GRASP results for instances with $|V| = 200$ and $|V| = 500$.

Parameters			GRASP (Var. I)						GRASP (Var. II)					
$\|V\|$	$\|L\|$	d	obj.	$\bar{\sigma}_{\text{obj}}$	$t_{\text{avg}}[s]$	$\bar{\sigma}_{\text{t}}$	$t_{\text{b}}[s]$	$\bar{\sigma}_{\text{tb}}$	obj.	$\bar{\sigma}_{\text{obj}}$	$t_{\text{avg}}[s]$	$\bar{\sigma}_{\text{t}}$	$t_{\text{b}}[s]$	$\bar{\sigma}_{\text{tb}}$
200	50	0.8	2.00	0.00	1.37	0.22	0.05	0.01	2.00	0.00	1.65	0.18	0.06	0.01
		0.5	2.20	0.00	1.57	1.47	0.05	0.06	2.20	0.00	1.67	1.47	0.06	0.10
		0.2	5.21	0.00	9.03	3.51	0.77	1.49	5.21	0.00	9.66	3.91	0.69	1.35
200	100	0.8	2.61	0.00	7.59	2.51	0.68	1.23	2.60	0.00	7.82	2.62	0.69	1.26
		0.5	3.53	0.00	9.79	1.77	1.29	2.17	3.52	0.00	9.97	1.78	1.31	2.23
		0.2	8.30	0.00	44.36	15.18	4.24	8.86	8.30	0.00	44.34	15.73	4.06	7.13
200	200	0.8	4.00	0.00	19.98	2.94	0.66	0.17	4.00	0.00	20.48	3.50	0.68	0.19
		0.5	5.43	0.00	50.53	16.46	5.63	6.70	5.43	0.00	51.02	16.66	6.73	9.85
		0.2	12.31	0.00	160.34	48.72	26.68	38.65	12.33	0.00	160.53	48.53	24.90	35.91
200	250	0.8	4.11	0.00	29.17	13.23	2.53	4.57	4.12	0.00	29.38	13.18	3.17	5.71
		0.5	6.44	0.00	70.82	20.36	7.51	11.16	6.45	0.00	71.02	19.67	6.75	9.89
		0.2	14.03	0.00	221.21	46.96	26.88	33.13	14.02	0.00	219.96	48.81	28.81	40.66
500	125	0.8	2.00	0.00	12.71	1.32	0.42	0.05	2.00	0.00	13.06	1.23	0.43	0.05
		0.5	2.65	0.00	32.63	23.76	2.46	5.23	2.64	0.00	31.28	21.89	2.57	5.37
		0.2	6.30	0.00	183.69	74.45	16.30	26.47	6.30	0.00	178.46	74.23	14.61	21.03
500	250	0.8	3.00	0.00	63.50	8.58	2.06	0.51	3.00	0.00	62.86	8.68	2.13	0.54
		0.5	4.30	0.00	111.31	55.04	3.81	2.14	4.30	0.00	111.60	56.10	3.76	2.21
		0.2	10.15	0.00	309.09	9.65	33.34	42.66	10.17	0.00	308.30	10.79	29.57	39.81
500	500	0.8	4.80	0.00	306.00	18.12	34.84	64.67	4.79	0.00	299.75	23.18	32.23	57.39
		0.5	6.97	0.00	316.24	12.74	29.57	44.62	6.98	0.00	315.58	15.23	28.53	48.45
		0.2	16.35	0.00	333.92	5.61	49.95	63.05	16.34	0.00	333.33	5.33	50.86	61.67
500	625	0.8	5.39	0.00	293.11	56.29	48.67	78.92	5.39	0.00	290.50	58.24	46.04	73.66
		0.5	8.40	0.00	327.70	3.74	13.01	13.69	8.40	0.00	327.49	3.67	12.99	8.63
		0.2	18.96	0.00	344.65	7.18	66.97	65.10	18.99	0.00	343.87	7.34	63.80	60.88

3.5 Conclusive Remarks

This chapter provided a comprehensive review of existing work regarding construction and approximation algorithms and metaheuristics for the MLST problem. In the main part of this chapter we have then proposed an ant colony optimization (ACO) approach – a metaheuristic that has not yet been applied to the MLST problem. Different pheromone models and construction mechanisms have been discussed and analyzed. Here, it turned out to be advantageous to also construct incomplete rather than feasible solutions. Using a parameter configuration that keeps running times relatively short good results could be obtained, in particular for medium and high density instances. With a higher number of iterations and more artificial ants the algorithm was able to find new better solutions for certain instances of a benchmark set. Furthermore greedy randomized adaptive search procedures (GRASP) have been considered. Several variations of the underlying randomized construction method based on MVCA have been described in a unified framework. GRASP was able to produce good solutions in relatively short running times. However, parameter settings yielding significantly better results imply drastically increased running times. Hence, if computational times are not crucial, but high solution qualities are inevitable, ACO should be preferred over GRASP. In particular for very large scale, dense graph instances with huge numbers of labels, as we are faced with in Chapter 5, GRASP may though be an interesting way to obtain better results compared to simply using MVCA.

Chapter 4

Exact Methods

Nil satis nisi optimum.
Nothing but the best is good enough.

In this chapter we propose a branch-and-cut(-and-price) (BCP) framework for the solution of moderately sized problem instances. We present a polyhedral and computational comparison of an underlying flow-formulation to a (usually stronger) formulation based on directed connection cuts. For the latter one we show how the cut-separation can be performed more efficiently than for many other spanning tree problems. New inequalities are introduced to strengthen the formulations. Optionally also cycle-elimination cuts are separated. Furthermore we show how to use odd hole inequalities to strengthen the formulation by cutting off fractional values of the label variables. We furthermore consider branch-and-cut-and-price, i.e. instead of starting the algorithm with a full model, we use a restricted model and then generate new (label) variables on demand. In order to obtain valid integral solutions in each node of the branch-and-bound (B&B) tree fast, we apply primal heuristics based on the well known MVCA-heuristic [16, 69]. After reviewing related previous work in Section 4.1, a detailed description of the formulations and algorithmic building blocks is presented in Section 4.2. In Section 4.3 we finally present a comparison of the described formulations and algorithmic components based on computational experiments.

4.1 Previous Work

In [16] an exact algorithm based on A^*-search has been proposed. The A^*-algorithm, introduced in [58], is a general technique for traversing graphs in order to reach a particular "goal node" on a minimum cost path. It can be seen as a generalization of the Dijkstra algorithm, which incorporates heuristic information into the search. Let the costs of any partial solution ("path") x be given by function $g(x)$, and let further $h(x)$ denote a

heuristic estimate for the additional costs to obtain a feasible solution, i.e. to reach the goal node. The A^* algorithm maintains a list of open problems. Then, in each step the problem with minimum $f(x) = g(x) + h(x)$ is extended. This is done by first calculating $f(x')$ for all possible extensions x' and then putting them on the list of open problems. If $h(x)$ is an underestimate, the first feasible solution found by A^* will be optimal.

For the MLST problem, the computation of $h(L')$, with L' denoting a partial solution, is performed by considering the remaining labels in decreasing order w.r.t. to their label frequency, and setting it to the number of labels required to represent the number of edges that is required to obtain a feasible solution. The algorithm maintains two data structures O and C to store open partial and already expanded (closed) partial solutions. To each such element also an associated function value is stored. The algorithm works as specified in Algorithm 9.

Algorithm 9: A^*-algorithm [16]

```
1  O = {∅}
2  C = {}
3  while L' ← element L'' ∈ O with minimum f(L'') do
4      if c(L') = 1 then
5          determine spanning tree for label set L'
6          exit
7      else
8          for each l ∈ L\L' do
9              L''' ← L ∪ {l}
10             if L''' ∉ O ∧ L''' ∉ C then
11                 calculate f(L''') = g(L''') + h(L''')
12                    where g(L''') = g(L') + 1 and g(∅) = 0
13                 add L''' to O
14             end
15         end
16     end
17 end
```

Line 4 in Algorithm 9 makes use of the previously introduced notation $c(L')$ to indicate the number of connected components induced by the arcs having assigned to a label from set L'.

A similar algorithm has been described in [29], however without guidance function $f(L')$. The recursive Algorithm 10 is initially called with an empty set, the global set C^* stores the smallest feasible solution found so far.

So far, only two publications are dealing with exact methods for the MLST problem based on mathematical programming techniques. The first MIP formulation proposed by Chen et al. [17] is based on Miller-Tucker-Zemlin inequalities which ensure that the decision variables for the edges induce a connected subgraph covering all nodes of the initial graph. In a recent work of Captivo et al. [13], the authors propose a MIP formulation based on

Algorithm 10: Test(C) [29]

```
1  if |C| < |C*| then
2      if c(C) = 1 then
3          C* ← C
4      else
5          if |C| < |C*| − 1 then
6              for each c ∈ L\C do
7                  Test(C ∪ {c})
8              end
9          end
10     end
11 end
```

single commodity flows, a frequently used modelling technique for spanning trees. Both approaches will be closer reviewed in the following section.

4.2 Mixed Integer Programming Framework

In this section we first give a formulation of the MLST as mixed integer program (MIP). For the spanning-tree property we present two alternative variants: 1) based on a flow-formulation and 2) a formulation based on directed connectivity cuts. Both formulations as well as additional inequalities to strengthen the formulation and methods for cutting-plane separation and dynamic variable generation are described within one generic framework, as they can be used in different combinations.

We use the following variables: variables $z_l \in \{0,1\}, \text{forall } l \in L$ indicate if label l is part of the solution; The edge variables $x_e, \text{forall } e \in E$ denote if edge e is used for the final spanning tree; Variables $y_{i,j}, \text{forall } i,j \in V$ denote directed arc variables used for the cut-based formulation, where we introduce for each edge $e = \{i,j\} \in E$ two arcs (i,j) and $(j,i) \in A$. For the flow formulation we analogously introduce two directed flow variables $f_{ij}, f_{ji} \in [0, n-1]$. Let further $L(e)$ denote the set of labels associated to edge e.

4.2.1 Mixed integer formulation

The basic formulation is given by the following integer linear program:

$$\text{min.} \quad \sum_{l \in L} z_l \tag{4.1a}$$

$$\text{s.t.} \quad \sum_{l \in L(e)} z_l \geq x_e \qquad \text{for all } e \in E \tag{4.1b}$$

$$x \equiv \text{“spanning tree”} \tag{4.1c}$$

The objective function (4.1a) minimizes the number of required labels, Inequalities (4.1b) ensure that for each selected edge (at least) one label is selected. For the abstract condition (4.1c) we will subsequently introduce two alternative formulations: a flow formulation and a cut-based formulation.

Fixing the number of selected edge according to a valid spanning tree

$$\sum_{e \in E} x_e = |V| - 1, \tag{4.2}$$

is a prerequisite for the subsequently presented flow formulation, but may be omitted when connectivity-based formulations are used, as our primary goal is to find a feasible set of labels from which the derivation of a a corresponding spanning tree is an easy task.

A single-commodity flow formulation, also considered in [13], is given as follows:

$$\sum_{(0,i) \in A} f_{0i} = |V| - 1 \tag{4.3a}$$

$$\sum_{(i,t) \in A} f_{it} - \sum_{(t,j) \in A} f_{tj} = 1 \qquad \text{for all } t \in V \backslash \{0\} \tag{4.3b}$$

$$f_{ij} \leq (|V| - 1) \cdot x_e \qquad \text{for all } \{i,j\} \in E \text{ and } e = \{i,j\} \tag{4.3c}$$

Equation (4.3a) ensures the correct quantity of flow leaving the (arbitrary) root node with index 0. For all other nodes flow consumption (4.3b) must hold, i.e. one unit of flow is consumed at each node. Inequalities (4.3c) finally ensure that only edges with a sufficient amount of flow may be selected. Flow formulations have the big advantage that they permit to formulate a spanning tree by a polynomial number of variables and therefore permit to provide a compact model to the MIP solver. It shortcomings are, however, that it provides a relatively poor LP-relaxation [76]. This is particularly due to the weak coupling of f to x-variables in Inequalities (4.3c), the linking constraints. This drawback can be circumvented by the introduction of multiple commodities k for each node $v \in V$. Again, all flows of commodity k originate from node 0 and must be delivered to node k. The formulation is given by the following equalities:

$$\sum_{(i,t) \in A} f^k_{it} - \sum_{(t,j) \in A} f^k_{tj} = \begin{cases} -1 & t = 0, \\ 0 & t \neq 0 \wedge t \neq k, \\ 1 & t = k, \end{cases} \qquad \text{for all } k \in V \backslash \{0\}. \tag{4.4}$$

Linkage of flow to edge variables is then given by

$$x_e \leq f^k_{ij} \qquad \text{for all } k \in V \backslash \{0\},\ e = \{i,j\} \in E \tag{4.5}$$

This formulation, however, has the drawback of having more variables than the single commodity flow formulation, i.e. $O(|V| \cdot |E|)$ flow variables in contrast to $O(|E|)$.

An alternative formulation is given by directed connection inequalities, stating that to each node a valid (directed) path must exist. In contrast to the flow model this formulation

consists of an exponential number of inequalities and therefore cannot be initially passed to an LP-solver for larger instances. However, this formulation provides a better LP-relaxation to many spanning tree problems, as it exactly describes the convex hull of the minimum spanning tree polytope. The corresponding inequalities are given by (4.6a), linkage to the edge variables is given by (4.6b).

$$\sum_{(i,j)\in\delta^-(S)} y_{ij} \geq 1 \qquad \text{for all } S \subseteq V,\ 0 \notin S \tag{4.6a}$$

$$x_e \geq y_{ij} \qquad \text{for all } \{i,j\} \in E \text{ and } e = \{i,j\} \tag{4.6b}$$

Here $\delta^-(S)$ denotes the set of ingoing arcs to some node set $S \subset V$. Instead of Inequalities (4.6b) we could also directly link the labels to the directed arcs. However, we proceed with Inequalities (4.6b) for sake of a unified notation. Moreover, additional $|E|$ constraints with only two non-zero coefficients do not significantly introduce further complexity. The separation of these directed-connection inequalities is discussed in Section 4.2.2.

It is well known to be practically advantageous to initially add the inequalities

$$y_{ij} + y_{ji} \leq 1, \qquad \text{for all } \{i,j\} \in E \tag{4.7}$$

and

$$\sum_{(i,j)\in\delta^-(j)} y_{ij} \geq 1, \qquad \text{for all } j \in V\backslash\{0\} \tag{4.8}$$

to directed (cut-based) formulations. Inequalities (4.7) limit the sum of the arcs corresponding to one edge to 1, Inequalities (4.8) assure that each node has one incoming arc. By $\delta^-(i)$ we denote the set of incoming arcs to node i.

We can also ensure feasibility for integer solutions by cycle-elimination inequalities. These inequalities enforce the resulting graph not to contain any cycles, which is together with the enforced number of arcs also a sufficient condition for spanning trees, and are given by the following Inequalities (4.9):

$$\sum_{e\in C} x_e \leq |C| - 1, \qquad \text{for all cycles } C \in G, |C| > 2. \tag{4.9}$$

A further way for prohibiting cycles are models based on the well known Miller-Tucker-Zemlin inequalities [81]. Such a model for the MLST problem has been proposed in [17], however with some differences. Let $u_i \in \mathbb{R}$ for all $i \in V$ denote variables assigning numeric values to teach node. By inequalities

$$u_i - u_j + |V| \cdot y_{ij} \leq |V| - 1 \qquad \text{for all } (i,j) \in A \tag{4.10a}$$

$$u_i \leq |V| \qquad \text{for all } i \in V \tag{4.10b}$$

cycles can be inhibited by just using a polynomial number of variables, however with the drawback, that a large multiplicative factor appears, usually leading to bad LP-relaxations. Main difference to the formulation proposed in [17] is the meaning of the variables. Whereas we use distinct variables for labels and edges ($O(|E| + |L|)$ variables), and link them by Inequalities (4.3c) which are totally $O(|E|)$ constraints, they introduce $O(|E| \cdot |L|)$ variables x_{ijk} with i, j corresponding to edges i, j and index k corresponding to labels.

In [13] the authors pointed out an important property of the flow formulation: They showed that the edge variables are not required to be integer in order to obtain the correct (optimal) objective function value. Furthermore it is easy to derive a valid MLST solution based on the set of labels provided by the MIP solution. Based on this reasoning, we can establish the following theorem, which extends this result to further MLST formulations, and also immediately provides an improved cut formulation with a fast separation method.

Theorem 10 For any MIP formulation given by equations 4.1a, 4.1b and 4.2, $z_l \in \{0, 1\}$, for all $l \in L$ any set of labels corresponding to an optimal solution to this formulation, and additionally meeting the following inequalities ("epsilon-connectivity")

$$\sum_{e \in \delta(S)} x_e \geq \epsilon \qquad \text{for all } S \subset V, S \neq \emptyset \tag{4.11}$$

implies a valid MLST. Here, $\epsilon > 0$ denotes some arbitrary small real number.

Proof The number of edges is fixed by (4.2), but a solution may still contain fractional edges. However, as the label variables z are integer and required to be greater than the value of the corresponding edge variables by inequalities (4.1b), they are always one if the corresponding edge variable has a value greater than ϵ. Consequently, fractional edge variables will only appear in the final solution if they do not raise the objective function value (by requiring additional labels). Due to Inequalities (4.11) the labels obtained from the MIP solution facilitate paths between all pairs of nodes. □

Given a label set of an optimal MIP solution, a feasible spanning tree can easily be derived in polynomial time, by determining an arbitrary spanning tree on the edges induced by the label set, as described in [13]. As a direct consequence of Theorem 10 the domain of the variables x and y need not be restricted to Boolean values, restricting them to non-negative values by inequalities

$$x_e \geq 0, \qquad \text{for all } e \in E, \tag{4.12}$$

and

$$y_{i,j} \geq 0, \qquad \text{for all } \{i, j\} \in E, \tag{4.13a}$$
$$y_{j,i} \geq 0, \qquad \text{for all } \{i, j\} \in E, \tag{4.13b}$$

is already sufficient.

Theorem 10 also suggests a further formulation to the MLST problem. Although not explicitly containing any constraints describing a valid spanning tree, equations (4.1a), (4.1b), (4.2) and (4.11) already provide a valid description to the MLST problem, and could be further strengthened by (4.16) and

$$\sum_{e \in \delta(i)} x_e \geq 1, \qquad \text{for all } i \in V. \tag{4.14}$$

Inequalities (4.11) will again be separated on demand, which can however be performed more efficiently then the separation for the directed connection cuts, which will be discussed in detail in Section 4.2.2.

However, epsilon-connectivity as defined by Theorem 10 is not guaranteed if cycle-elimination inequalities (4.9) are used exclusively to describe a valid spanning tree. A fractional LP-solution not containing a cycle may still contain a subtour, i.e. a subgraph where the sum over corresponding edges is larger than the size of its nodes minus one. Such a situation is depicted in Figure 4.1. As a consequence, the domain of the x-variables must be restricted to Boolean values if only cycle-elimination inequalities are used to describe a valid spanning tree. The same is true for the Miller-Tucker-Zemlin formulation given by Inequalities (4.10a).

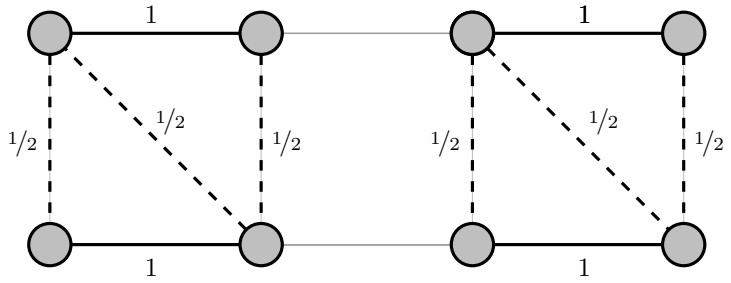

Figure 4.1: LP-solution that does not contain a cycle w.r.t. Inequalities (4.9), but still violates subtour elimination constraints. Corresponding (integer) label solutions are not necessarily feasible.

In the beginning of this section, fixing the number of edges by Equality (4.2) was stated to be a requirement for most flow and cut based formulations, but may however be omitted regarding the "ϵ-connectivity" formulation given by (4.1a), (4.1c), and (4.11).

We now draw our attention to the special case of having only one single label assigned to each edge. If we have not fixed the number of edges we can impose further equalities

$$\sum_{l \in L(e)} z_l = x_e \quad \text{for all } e \in E, \tag{4.15}$$

instead of Inequalities (4.1b), which provide a more direct link between labels and their corresponding edges. This approach emphasizes the search for a feasible label set of minimal cardinality rather then the search for a feasible spanning tree.

4.2.2 Cutting-Plane Separation

The directed connection Inequalities (4.6a) can be separated by computing the maximum flow from the root node r to each nodes i as target node. This provides a minimum (r, i)-cut. We have found a violated inequality if the value of the corresponding arcs according to the sum of the LP-values is less than 1. Our separation procedure utilizes Cherkassky and Goldberg's implementation of the push-relabel method for the maximum flow problem [18] to perform the required minimum cut computations.

The cycle-elimination cuts (4.9) can be easily separated by shortest path computations with Dijkstra's algorithm. Hereby we use $1 - y_{ij}^{\text{LP}}$ as the arc weights with y_{ij}^{LP} denoting the current value of the LP-relaxation for (i, j) in the current node of the B&B-tree. We obtain cycles by iteratively considering each arc $(i, j) \in A$ and searching for the shortest path from j to i. If the value of a shortest path plus y_{ij}^{LP} is less than 1, we have found a cycle violating Inequalities (4.9). We add this inequality to the LP and resolve it. In each node of the B&B-tree we perform these cutting plane separations until no further cuts can be found.

Theorem 10 suggested a formulation not requiring any auxiliary variables (like flow or arc variables), where validity of the labels is obtained by Inequalities (4.11) exclusively. Instead of using the minimum cut based separation routine (which would also be valid), we can perform a faster separation by a simple depth first search (DFS). Given an LP-solution, we first select an arbitrary start node for which we call the DFS procedure. Within this procedure we only consider edges e with $x_e \geq \epsilon$, the other ones are ignored. Within the DFS we keep track of all visited nodes, if there are unvisited nodes at the end of the DFS, we have found a valid cut. The DFS can be carried out in $O(|V| + |E|)$ time, which is clearly superior to the time of the maximum flow algorithm running in $O(|V| \cdot |E| + |V|^{2+\varepsilon})$.

4.2.3 Strengthening the Formulations

As each node must be connected to the spanning tree by one of its incident edges, we can further impose additional inequalities to strengthen the formulation w.r.t. the label variables:

$$\sum_{l \in L(v)} z_l \geq 1, \qquad \text{for all } v \in V. \tag{4.16}$$

Here, $L(v)$, $v \in V$ denotes the set of labels being associated to the edges incident to node v. We will subsequently refer to this set of $|V|$ inequalities as node-label-inequalities. Figure 4.2 gives a simple example of an LP solution where the node is sufficiently connected according to the sum of the LP-values of the ingoing arcs and therefore its incident edges, but the corresponding sum over the labels associated to these edges is clearly infeasible w.r.t. Inequalities (4.16). Therefore Inequalities (4.16) strengthen the presented formulations w.r.t. their LP-relaxation. In Section 4.2.6 we formally prove this property with respect to the particular proposed MIP-formulations for the MLST.

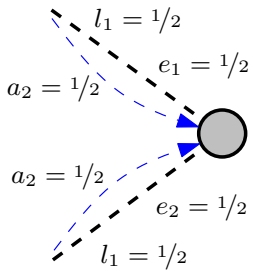

Figure 4.2: Example of node that is feasible connected w.r.t. its incoming arcs, but not w.r.t. inequalities (4.16). Edges $e_1 = e_2 = 1/2$ in the current LP-solutions, but as both edges have assigned to the same label $l_1 = 1/2$ the sum over the set of all labels assigned to incident edges of the considered node is also $1/2$. Such situations are forbidden by Inequalities (4.16).

This basic idea used in Inequalities (4.16) can be pursued by considering sets of two nodes, say v_1 and v_2. Let e_{12} denote the edge joining v_1 and v_2. Let further $L(e_{12})$ denote the set of labels associated to this edge. For set $L(v_1) \cup L(v_2)$ we can observe, that at least two labels are required to feasibly connect the nodes v_1 and v_2, if $L(v_1) \cap L(v_2) = \emptyset$. However, if $L(v_1) \cap L(v_2) = L(e_{12})$ we still require two labels from $L(v_1) \cup L(v_2)$. We therefore obtain the following valid inequalities,

$$\sum_{l \in L(v_1) \cup L(v_2)} z_l \geq 2, \qquad \text{for all } v_1, v_2 \in V \text{ with } L(v_1) \cap L(v_2) = L(e_{12}), \tag{4.17}$$

which are not directly implied by Inequalities (4.16). Figure 4.3 shows an example where Inequalities (4.17) dominate Inequalities (4.16).

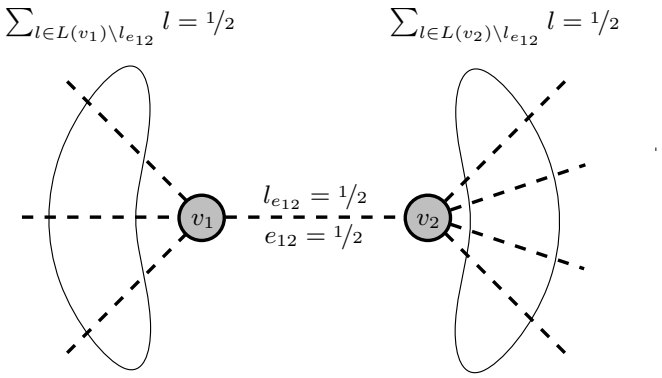

Figure 4.3: Example of node-label-constraints for sets of two nodes (4.17) dominating Inequalities (4.16), i.e. the node-label constraints for single nodes. For both nodes v_i, $i = 1, 2$ it holds that $\sum_{l \in L(v_i)} \geq 1$. Corresponding Inequality (4.17) is however violated, as $\sum_{l \in L(v_1) \cup L(v_2)} l = 3/2$.

As we can expect a lot of branching on the label variables, in particular for GMLST

instances, further cutting-planes cutting of fractional label solutions may be helpful. In order to identify such valid inequalities, we consider situations where fractional label variables lower the objective value of LP solutions. Such a situation is depicted in Figure 4.4. If labels $a = b = c = 1/2$ in the LP solutions the corresponding arcs can be set to $1/2$ as well without violating any directed connectivity inequality. However, w.r.t. these arc set, at least two labels must be selected in an integer solution. Consequently adding the inequality $a+b+c \geq 2$ will cut-off this fractional solution, but is only valid if no additional arcs/edges are incident to these nodes.

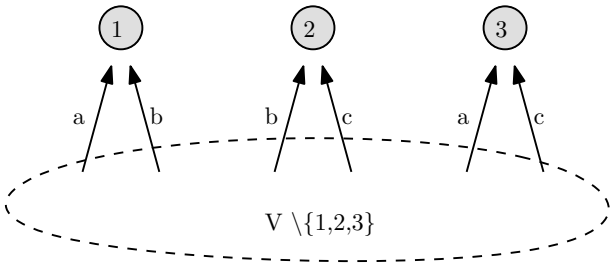

Figure 4.4: Example of fractional label solution

In the following we show how to apply odd hole inequalities to cut-off such and more general situations. These inequalities are well known from studies of the set-covering polytope, their application becomes evident by the observation that the MLST problem can be seen as a set covering problem where each node v needs to be covered by a label from the set $L(v)$ and the corresponding edges fulfilling further constraints (i.e. forming a valid spanning tree). In particular we use a MIP based heuristic to separate valid inequalities for the set-covering problem with coefficients $\{0, 1\}$, which have been proposed in [35].

Let $\boldsymbol{\Lambda}$ be a $|V| \times |L|$ matrix with $\lambda_{ij} = 1$ if node i is labeled with j, $\lambda_{ij} = 0$ otherwise. A $|V'| \times |L'|$ submatrix $\boldsymbol{\Lambda}'$ of $\boldsymbol{\Lambda}$ of odd order is called an odd hole if it contains exactly two ones per row and column. For the subproblem $\boldsymbol{\Lambda}'\boldsymbol{z}' \geq 1$ the inequality

$$\sum_{l \in L'} z_l \geq \frac{|L'| + 1}{2} \tag{4.18}$$

is valid. In [35] the authors showed, that this inequality even remains valid if $\boldsymbol{H} \leq \boldsymbol{\Lambda}' \leq \boldsymbol{H}^*$, where $\boldsymbol{H}$ is an odd hole, and $\boldsymbol{H}^*$ being a special matrix closely related to $\boldsymbol{H}$. Finding an odd hole $\boldsymbol{H}$ to a given matrix $\boldsymbol{\Lambda}'$ is $\mathcal{NP}$-complete, but if we have found such an odd hole, it is possible to decide in polynomial time whether $\boldsymbol{H} \leq \boldsymbol{\Lambda}' \leq \boldsymbol{H}^*$ and therefore (4.18) is valid [35].

Separation-Heuristic for the Odd Hole Inequalities

In order to cut-off fractional label solutions we consider the subset of nodes $V'' \subseteq V$ which labels are either fractional or zero in the current LP solution. Let $\tilde{\boldsymbol{\Lambda}}^{V''}$ denote the matrix

where each entry λ_{ij} represents the current LP value of label j associated to node i, or -1 if the label j is not associated to node i. Let further $\mathbf{\Lambda}^{V''}$ denote the corresponding matrix representing which labels are assigned to particular nodes, i.e. its elements $\lambda_{ij}^{V''}$ are one if label $j \in L(\delta(i))$, and zero otherwise. Our goal is to heuristically search for odd holes in $\mathbf{\Lambda}^{V''}$, based on the information provided by matrix $\tilde{\mathbf{\Lambda}}^{V''}$, and then transform the related inequality to a valid inequality for the initial problem by the according lifting steps. We are hence searching for an odd hole $\boldsymbol{H}$ with $\boldsymbol{H} \leq \mathbf{\Lambda}^{V',L'}$ with $V' \subseteq V'', L' \subseteq L''$ and $|V'| = |L'|$ being odd. By the procedure of [35] we can now decide if

$$\sum_{l \in L'' \setminus L'} \gamma_l \cdot z_l + \sum_{l \in L'} z_l \geq \frac{|L'| + 1}{2} \tag{4.19}$$

is valid for $\mathbf{\Lambda}^{V',L'}$. The term $\sum_{l \in L'' \setminus L'} \gamma_l \cdot z_l$ results from lifting all labels which are associated to a node $v \in V'$ but are not part of the odd hole induced by V' and L'. The lifting-coefficient is denoted by γ_l, the calculation of its value will be discussed later on. By the following MIP (4.20) we aim to find subsets V' and L' forming an odd hole and for which inequality (4.19) is violated according to the current LP solution. For this purpose we define a bipartite directed graph $\tilde{G} = (\tilde{V} = \tilde{V}_1 \cup \tilde{V}_2, \tilde{A}), \tilde{V}_1 = V'', \tilde{V}_2 = L'', \tilde{A} = \{(i,j) \mid i \in V'' \wedge j \in L''(V'')\}$. Each cycle with length $4 \cdot k + 2$ corresponds to an odd cycle w.r.t. the number labels, and is therefore a potential odd hole. Variables $x_{ij} \in \{0,1\}$ represent the arcs from node $i \in V''$ to label $j \in L''(V'')$ and are intended to finally describe a valid odd hole. Variables $a_{ij} \in [0,1]$ denote other arcs which connect nodes $i \in V''$ being part of the odd hole (described by the x variables) and other labels not being part of the odd hole. For each arc $a = (i,j)$ the coefficient c_a is the LP value of label j if $j \in L'$ and zero otherwise.

$$\max \quad k + 1 - \sum_{i \in \tilde{A}} x_i \cdot c_i - \sum_{i \in \tilde{A}} a_i \cdot c_i \tag{4.20a}$$

$$\text{s.t.} \quad k + 1 - \sum_{i \in \tilde{A}} x_i \cdot c_i - \sum_{i \in \tilde{A}} a_i \cdot c_i \geq 0 \tag{4.20b}$$

$$\sum_{i \in \tilde{A}} x_i = 4 \cdot k + 2 \tag{4.20c}$$

$$\sum_{(i,j) \in \delta^-(j)} x_{ij} \leq 1 \qquad \text{for all } j \in L'' \tag{4.20d}$$

$$\sum_{(i,j) \in \delta^+(i)} x_{ij} \leq 1 \qquad \text{for all } i \in V'' \tag{4.20e}$$

$$\sum_{(i,j) \in A} x_{ij} - \sum_{(j,k) \in A} x_{jk} = 0 \qquad \text{for all } i \in \tilde{V} \tag{4.20f}$$

$$y_i - y_j + 1 + |\tilde{V}| \cdot x_{ij} - |\tilde{V}| \cdot z_i \leq |\tilde{V}| \tag{4.20g}$$

$$\sum_{i \in V} z_i \leq 1 \tag{4.20h}$$

$$\sum_{(k,i) \in \delta^-(i)} x_{ki} - \sum_{(j,l) \in \delta^-(j)} x_{jl} \leq a_{ij} \qquad \text{for all } (i,j) \in \tilde{A} \tag{4.20i}$$

$$y_i \leq |\tilde{V}| \qquad \text{for all } i \in \tilde{V} \qquad (4.20j)$$
$$z_i \in \{0,1\} \qquad \text{for all } i \in \tilde{V} \qquad (4.20k)$$
$$x_i \in \{0,1\} \qquad \text{for all } i \in \tilde{A} \qquad (4.20l)$$
$$0 \leq a_i \leq 1 \qquad \text{for all } i \in \tilde{A} \qquad (4.20m)$$

From (4.20c) we can see that $\frac{|L'|+1}{2} = k+1$. As we prefer solutions where (4.19) is considerably violated we maximize the difference between $\frac{|L'|+1}{2}$ and $\sum_{i\in\tilde{A}} x_i \cdot c_i$. The term $\sum_{i\in\tilde{A}} a_i \cdot c_i$ gives a lower bound for the sum over all labels we need to lift w.r.t. some particular x-solution. The correct coefficient which is to be discussed later on, cannot be formulated by a linear expression. By Equation (4.20b) this particular expression is enforced to be larger than zero, as the resulting inequality to be added to the MLST-MIP would not be violated otherwise. As a consequence all feasible solutions to MIP (4.20) fulfill this property which is desirable for the heuristic separation procedure discussed subsequently. For each node on the cycle the numbers of ingoing and outgoing arcs are limited to one by equations (4.20d) and (4.20e) and flow-conservation is imposed for each node (4.20f). The integer variables y_i assign numeric values to the nodes $i \in V'' \cup L''$ and prevent multiple cycles in the solution in (4.20g) by enforcing for each arc on the cycle (except the one outgoing from the node i with $z_i = 1$ (4.20h)) to have an at least by one smaller source than target node. Such inequalities have been first proposed in [81], and are well known as Miller-Tucker-Zemlin-inequalities. By Inequalities (4.20i) all arcs connecting nodes $i \in V'$ which are part of the odd cycle to be determined (by x-variables) to nodes $j \in L''(i)$ not being part of this cycle. Finally, y_i, for all $i \in \tilde{V}$ are enforced to be smaller than $|\tilde{V}|$ (4.20j), and the node selection and arc variables are required to be Boolean (4.20k, 4.20l). The a-variables only need to be restricted to $0 \leq a_i \leq 1$, for all $i \in \tilde{A}$, as they are implicitly integer by Inequalities (4.20i). Figure 4.5 shows an example for a solution to the MIP. The arcs selected by x-variables are depicted in red color, the dashed ones do not contribute to the objective function. The blue arcs correspond to the "lifting-arcs", selected by a-variables.

Given a solution to the MIP (4.20), we still need to check, if (4.19) is valid for this particular solution. The z-variables are derived by taking all labels $j \in L'$ selected by x_{ij} in (4.20). For this purpose we use the criterions described in [35] – here we only provide a rough explanation. An arc connecting two nodes on the odd cycle determined by (4.20) which is not part of the cycle itself is called a chord. In order to (4.18), and therefore (4.19) after the lifting, to be valid, all chords of the odd cycle must be compatible. The chord set is called compatible, if 1) no chord induces even cycles (w.r.t. nodes $i \in V'$ on the cycle), and 2) every pair of crossing chords is compatible. Compatibility for crossing chords is defined on the basis of the mutual distances of their adjacent nodes on the cycle. Let $a_{ij} = (v_i, l_j),\ v_i \in V',\ l_j \in L'$ and $a_{hk} = (v_h, l_k),\ v_h \in V',\ l_k \in L'$ be two crossing chords. We now remove l_j and its two incident arcs from the odd hole. The chords are compatible, if the unique path from v_i to v_h has an even distance w.r.t. nodes in V' in this graph.

It remains to determine the lifting-coefficients γ_l. If a lifting-label only covers one node of the odd hole, the sum over all labels necessary to feasibly cover all nodes from the odd

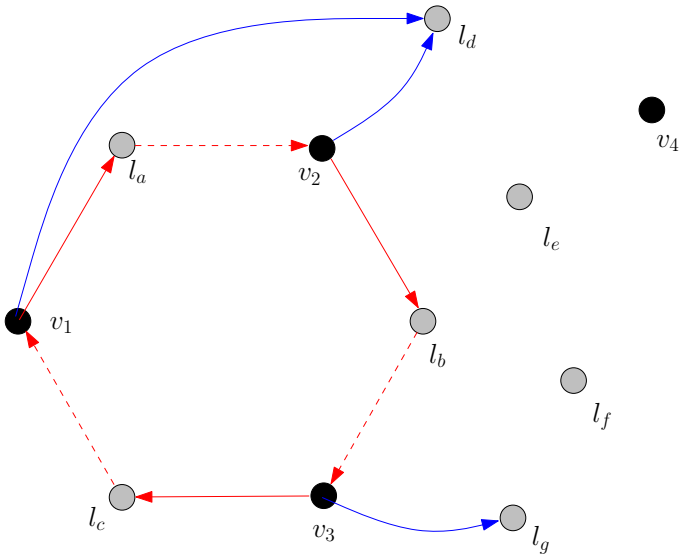

Figure 4.5: Example for a solution to 4.20. The cycle with red arcs constitutes the odd hole. The dashed red arcs do not contribute to the objective function, whereas the solid red arcs (which connect nodes to labels) contribute with the LP-value of the target-label as coefficient. Blue arcs provide a lower bound for the contribution of all labels that need to be lifted in order to obtain a valid inequality for the initial problem.

hole does not change. The label can however be used alternatively to one of the odd hole labels and therefore gets coefficient one. Otherwise, if one lifting-label covers all odd hole nodes, the coefficient must equal the right hand side of (4.19), i.e. $\gamma_l = \frac{|L'|+1}{2}$ in this case. Suppose some lifting-label l covers ν_l odd hole nodes, then the size of the remaining odd hole nodes is $\frac{|L'|+1}{2} = \lceil \frac{|L'|}{2} \rceil$. These remaining nodes are still adjacent to two labels in the odd hole, pairwise having one label in common. We can therefore derive the following value for the lifting coefficient

$$\gamma_l = \lceil \frac{|L'|}{2} \rceil - \lceil \frac{|L'| - \nu_l}{2} \rceil = \frac{|L'|+1}{2} - \left(\frac{|L'|+1}{2} - \lceil \frac{\nu_l}{2} \rceil \right) = \lceil \frac{\nu_l}{2} \rceil. \tag{4.21}$$

During the branch-and-bound MLST solution process the MIP (4.20) is solved with very tight runtime-limits. As soon as an incumbent integer solution has been found, this solution is checked for validity by the mentioned criterions. Obtained valid MLST-inequalities are added immediately. Then the incumbent integer solution is rejected to the MIP solver by which we enforce to search for further solutions. This process continues until the time limit is reached.

4.2.4 Heuristics

In order to improve the overall performance – in particular the ability to generate feasible integer solutions fast – we embed a primal heuristic into the framework. For this purpose we adopt the well known MVCA heuristic [16, 69, 30]. This heuristic can create feasible solutions itself, but also complete partial solutions $\tilde{L} \subset L$. Creating complete solutions is important for the acquisition of strong upper bounds to efficiently cut-off unprofitable branches of the B&B-tree from the beginning on, but also to obtain an initial solution for BCP (Section 4.2.5). On the other hand the MVCA heuristic can be used to obtain feasible integer solutions and therefore upper bounds in within each B&B-node based upon some variables already fixed to integer values.

We also consider the ant colony optimization(ACO) approach presented in Chapter 3 as a primal heuristic. Within this context ACO may be used to obtain good initial solutions, but also to complete partial solutions defined by the label variables which are equal to one in a particular B&B-node. For this purpose the pheromone structure may be initialized by setting pheromones for such labels to the maximum value ($\tau_{\max}$), whereas the other ones are set to the minimum value ($\tau_{\min}$).

Many further fast metaheuristic techniques do exist for this problem, which could also easily be integrated into this framework. This is however beyond the scope of this work, as we primarily focus on the mathematical programming methods for the MLST.

4.2.5 Pricing Problem

Problem formulations with a large (usually exponential) number of variables are frequently solved by column generation or branch-and-price algorithms. Such algorithms start with a restricted set of variables and add potentially improving variables during the solution process on demand, see Section 2.4.2. If these algorithms also include cutting-plane generation we obtain branch-and-cut-and-price (BCP), cf. Section 5.7. Although the presented MLST formulation only has a polynomial number of label variables, these particular variables lead to extensive branching on them, requiring a special treatment. Hence we based a solution approach on BCP.

We obtain the restricted master problem by replacing the complete set of labels L by a subset $L' \subseteq L$ in (4.1a). The set L' is required to imply a feasible solution, and is obtained by the MVCA heuristic. Then, new variables and therefore columns potentially improving the current objective function value in the simplex tableau are created during the B&B process. These new variables are obtained from the solution of the pricing problem which is based upon the dual variables. Let π_i denote the dual variables corresponding to constraints (4.1b), and μ_i the ones corresponding to (4.16). They therefore reflect a measure for the costs of some particular edge e w.r.t. the currently selected labels (π_e), and the costs of connecting some node v w.r.t. the currently selected labels (μ_v). The pricing problem is to find a variable with negative reduced costs

$$\bar{c}_l = 1 - \sum_{(i,j)\in A(l)} \pi_{ij} - \sum_{i\in V(l)} \mu_i, \tag{4.22}$$

within the set of all labels L. Here $A(l)$ denotes all arcs having label l, $V(l)$ denotes the set of nodes incident to arcs with label l. Finding such a variable or even the one with maximal reduced costs can be done by enumeration. Although only a polynomial number of labels is involved, we may benefit from the pricing scheme as we only need to solve smaller LPs within the B&B procedure.

4.2.6 Polyhedral Comparison

In this section we compare various formulations resulting from combining the equations and inequalities from Section 4.2 as listed in Table 4.1. The only formulation just requiring a polynomial number of constraints is the flow-formulation with roughly $O(|L| + 3 \cdot |E|)$ variables and $O(|L|+|V|+|E|)$ constraints. The directed cut-formulation requires $O(|L|+3\cdot|E|)$ variables and an exponential number of constraints. Also the modified "epsilon" cut-formulation requires exponentially many constraints, but only has $O(|L|+|E|)$ variables.

Table 4.1: MLST formulations resulting from combining the equations and inequalities from section 4.2. Further variants are given by the use of the components listed in the second part of the table, to be used as index for the formulation to be used with.

abbrevation	**involved equations and inequalities**
SCF	(4.1a), (4.1b), (4.3a) - (4.3c)
MCF	(4.1a), (4.1b), (4.2), (4.4), (4.3c)
DCut	(4.1a), (4.1b), (4.2), (4.6a), (4.6b), (4.7), (4.8)
EC	(4.1a), (4.1b), (4.2), (4.14), (4.11)
MTZ	(4.1a), (4.1b), (4.10a
CEF	(4.1a), (4.1b), (4.9)
n	node-label-constraints (4.16)
$\tilde{n}$	extended node-label-constraints (4.17)
t	tree search, i.e. fixed number of edges (4.2)
s	strong linkage (4.15)
c	cycle elimination inequalities (4.9)
o	odd-hole inequalities
p	variable pricing

In the following we use the graph depicted in Figure 4.6 to show the properties of the polyhedra defined by the formulations listed in Table 4.1.

Proposition 5

$$P^{\mathrm{SCF}_{tno}} \subsetneq P^{\mathrm{SCF}_{tn}} \subsetneq P^{\mathrm{SCF}_{t}} \subsetneq P^{\mathrm{SCF}} \tag{4.23}$$

Proof As $P^{\mathrm{SCF}_{tn}}$ contains the same equations and inequalities as $P^{\mathrm{SCF}_{\mathrm{t}}}$, but additionally the Inequalities (4.16), we have $P^{\mathrm{SCF}_{\mathrm{tn}}} \subseteq P^{\mathrm{SCF}_{\mathrm{t}}}$. Figure 4.7 shows a LP solution of $P^{\mathrm{SCF}_{\mathrm{t}}}$

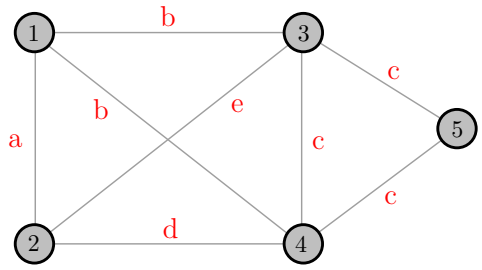

Figure 4.6: Example graph used in the following to show the properties of the formulations listed in Table 4.1. The set of labels is given by $L = \{a, b, c, d, e\}$, the optimal solution value is $f = 3$.

which is not contained in $P^{\mathrm{SCF_{tn}}}$, which implies $P^{\mathrm{SCF_{tn}}} \subsetneq P^{\mathrm{SCF_t}}$. Such an LP solution may still contain fractional labels due to odd holes, like shown in Figure 4.5, by which we obtain $P^{\mathrm{SCF_{tno}}} \subsetneq P^{\mathrm{SCF_{tn}}}$.

If the values of the edge and label variables in Figure 4.7 are decreased as much as possible for SCF, we obtain $l_a = 1/4$, $l_b = 3/8$ and $l_c = 1/8$ implying $f^{lp} = 3/4$. As $\mathrm{SCF_t}$ contains the additional Inequality (4.2), we can conclude that $P^{\mathrm{SCF_t}} \subsetneq P^{\mathrm{SCF}}$. □

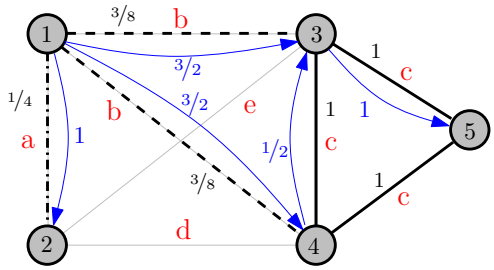

Figure 4.7: LP solution of $\mathrm{SCF_t}$ with objective value $f^{lp} = 1+5/8$ ($l_a = 1/4, l_b = 3/8, l_c = 1$). The blue arcs depict the flow variables with their according LP-values. This solution is not valid for $\mathrm{SCF_{tn}}$, as the sum over the set of labels adjacent to node v_2 is smaller than one.

Proposition 6

$$P^{\mathrm{DCut_{tno}}} \subsetneq P^{\mathrm{DCut_{tn}}} \subsetneq P^{\mathrm{DCut_t}} \subsetneq P^{\mathrm{DCut}} \tag{4.24}$$

Proof The proof of $P^{\mathrm{DCut_{tno}}} \subsetneq P^{\mathrm{DCut_{tn}}} \subsetneq P^{\mathrm{DCut_t}}$ follows by the same reasoning as for the proof of theorem 5. Figure 4.8 shows that $P^{\mathrm{DCut_{tn}}} \subsetneq P^{\mathrm{DCut_t}}$. However, the requirement that each directed cut must have a value greater than one already implies that $\sum_{e\in\delta(v)} x_e \geq 1$, for all $v \in V$. This implies $\sum_{e\in E} x_e \geq |V| - 1$. An LP-solution to DCut may contain more edges than an LP-solution to $\mathrm{DCut_t}$, which does, however, due to the minimality not affect the objective value of the LP-relaxation, i.e. $P_z^{\mathrm{DCut_t}} = P_z^{\mathrm{DCut}}$. □

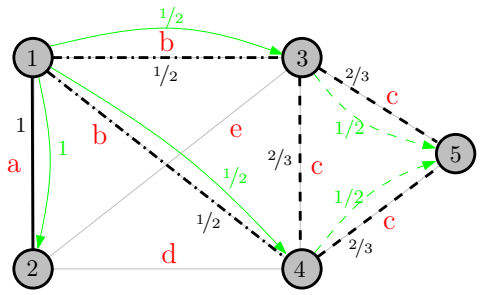

Figure 4.8: LP solution of $\mathrm{DCut_t}$ with objective value $f^{lp} = 2 + 1/6$ ($l_b = 1/2,\ l_c = 2/3,\ a + d + e \geq 1,$ w.l.o.g. $l_a = 1$). The green arcs depict the arc variables with their according LP values. The solution is not valid for $\mathrm{DCut_{tn}}$, as the sum over the set of labels adjacent to node v_5 is smaller than one.

Let P_S denote the projection of some polyhedron P to a subspace S.

Proposition 7

$$P_x^{\mathrm{ECtno}} \subsetneq P_x^{\mathrm{ECtn}} \subsetneq P_x^{\mathrm{ECt}} \subsetneq P_x^{\mathrm{EC}} \tag{4.25}$$

Proof By applying the same reasoning as for the proofs of the last two theorems, we can prove Proposition 7. Figure 4.9 gives an example for $P^{\mathrm{ECtn}} \subsetneq P^{\mathrm{ECt}}$. □

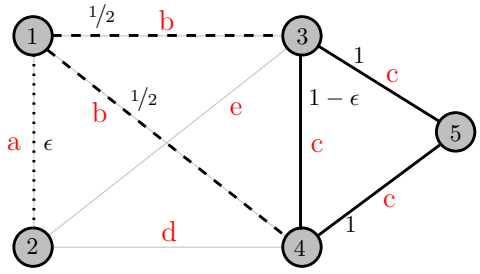

Figure 4.9: LP solution of $\mathrm{EC_t}$ with objective value $f^{lp} = 3/2 + \epsilon$ ($l_a = \epsilon, l_b = 1/2, l_c = 1$). The solution is not valid for $\mathrm{EC_{tn}}$, as the sum over the set of labels adjacent to nodes v_1 and v_2 are smaller than one.

In the following we will show the relations between the formulations $\mathrm{SCF_t}$, $\mathrm{DCut_t}$ and $\mathrm{EC_t}$.

Theorem 11

$$P_x^{\mathrm{DCut_t}} \subsetneq P_x^{\mathrm{SCF_t}} \subsetneq P_x^{\mathrm{EC_t}} \tag{4.26}$$

Proof Figures 4.8, 4.7 and 4.9 already showed, that the polyhedrons are not equal. To prove that $P_x^{\mathrm{DCut_t}} \subsetneq P_x^{\mathrm{SCF_t}}$ we show a procedure to transform all x-variables of any valid

LP-solution of $\mathrm{DCut_t}$ to a valid x-solution in $\mathrm{SCF_t}$. For all $i, j \in V$ there exists at least one path from i to j with all edges (k, l) having LP-values x^{lp}_{kl} greater than zero. If we consider a network with source i and target j, only containing edges e being part of one of these paths and having capacities x^{lp}_e there exists a flow of at least one unit from s to t. We now arbitrarily select a root node r (w.l.o.g. $r = 0$) and show how to construct a valid flow permitting the same x-configuration for $\mathrm{SCF_t}$ as in $\mathrm{DCut_t}$. For an edge e to have LP value x^{lp}_e a corresponding flow variable must be larger than $x^{\mathrm{lp}}_e/(n-1)$. We start by setting all flow variables to zero. Then for each node $t_i, i = 1, \ldots, n-1$ we construct all paths from r to t, considering all edges with $x^{\mathrm{lp}}_e > 0$. Summing up $x^{\mathrm{lp}}_e > 0$ for all edges e on these paths may not exceed $(n-1)$, as the number of edges is fixed by (4.2) when $i = 2$. However, this sum may usually be smaller than $(n-1)$, say λ_l, but integer. Now we backtrack all these paths and set their flow values to minimal values according to flow conservation (4.3b) and LP-values for the edges. Note that $\sum_{i \in \delta(r)} f_{ri} = \lambda_1$ after this first step. We then continue this procedure for all further $t_i, i = 2, \ldots, n-1$. According to (4.2) in step t_k at most $(n-1) - \sum_{l<k} \lambda_k$ not yet considered edges need to be added, possibly increasing $\sum_{i \in \delta(r)} f_{ri}$ by exactly this amount. We finally end up with all nodes being feasibly connected and $\sum_{i \in \delta(r)} f_{ri} = (n-1)$ fulfilling (4.3a) and flow conservation (4.3b) being fulfilled at each node.

It is trivial to see that the x-variables of a valid LP-solution of $\mathrm{SCF_t}$ is also valid for $\mathrm{EC_t}$. □

Theorem 12

$$P_x^{\mathrm{DCut_{tn}}} \subsetneq P_x^{\mathrm{SCF_{tn}}} \subsetneq P_x^{\mathrm{EC_{tn}}} \tag{4.27}$$

Proof In the proof of Theorem 11 we already showed how each projection of a solution of $\mathrm{DCut_t}$ to the subspace defined by the x-variables can be transformed into a solution of $\mathrm{SCF_t}$, and likewise $\mathrm{SCF_t}$ to $\mathrm{EC_t}$. The only difference of the polyhedrons considered in Theorem 12 are the constraints (4.16), which clearly do not affect this transformation. It needs to be shown, that the polyhedrons are not equal, which is done by the example in Figure 4.10. The depicted $\mathrm{EC_{tn}}$ solution is not valid for $\mathrm{SCF_{tn}}$ or $\mathrm{DCut_n}$ respectively, although the node-label constraints (4.16) are fulfilled. However, the value of edge $\{3, 4\}$ can be increased to $1/5$ (implying the need to decrease the values of edges $\{1, 4\}$ and $\{3, 6\}$ accordingly), which makes the solution feasible to $\mathrm{SCF_{tn}}$. Nevertheless, this solution remains infeasible to $\mathrm{DCut_{tn}}$, by which we have shown the theorem. □

4.3 Results

In this section we present a comprehensive computational comparison of the presented formulations and separation strategies, and compare our methods to other work. Three different datasets are used for our computational tests. We start by a description of the test instances used for our experiments and tests.

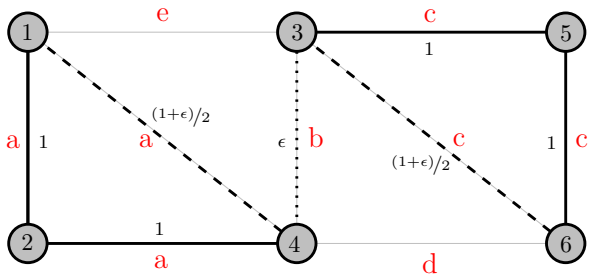

Figure 4.10: Valid LP-solution of $\mathrm{EC}_{\mathrm{tn}}$ with $f^{lp} = 2 + \epsilon$ ($l_a = 1, l_b = \epsilon, l_c = 1$) that is not valid for $\mathrm{SCF}_{\mathrm{tn}}$. It can however be transformed to such, by increasing $x_{3,4}$ to $1/5$, yielding $f^{lp} = 2 + 1/5$. It is easy to see, that this solution is still not valid for $\mathrm{DCut}_{\mathrm{tn}}$.

4.3.1 Test Instances

The first set is the publicly available benchmark set used in [30, 29, 28, 15]. We refer to this data set as **Set-I**. It consists of graphs with 100 to 500 nodes and various densities $d \in \{0.2, 0.5, 0.8\}$, defined by $|E| = d \cdot \frac{|V| \cdot (|V|-1)}{2}$, and different numbers of labels $|L| = l/4, l \in \{1, 2, 4, 5\}$. The instances are organized in groups of ten for each configuration of d and $|L|$ for each $|V|$. So far, primarily metaheuristics have been applied to this instance set, but also an exact algorithm based on A^*-search, as reported in [29].

The second test set **Set-II** is created following the specification of the instances used in [13], in order to obtain comparable results to the MIPs presented therein. This set is organized in four groups. In contrast to Set-I, the instances of the first two groups just contain very few labels, i.e. $|L| \in \{5, 10, 20\}$. The number of nodes ranges from 20 to 1000, network densities are set to $|E| = 4 \cdot |V|$ Moreover, this set contains various grid-graphs (group 3) of sizes 2×10, 4×5, 2×18, 3×12, and 6×6. The fourth group contains instances with $|V| \in \{20, 50\}$ and $|L| = |V|$ and various network densities $d \in \{0.2, 0.5, 0.8\}$.

In addition to Set-I and Set-II we created a further test set **Set-III** containing also instances with multiple labels assigned to the edges. The construction is performed by first creating a spanning tree and assigning labels from set $L^* \subseteq L$ to its edges. Usually $L^* = L$ if not stated otherwise, but $|L^*| \ll |L|$ is used to study the effect of having optimal solutions with significantly less labels than for completely random label assignment for the particular graph properties. Next further edges are added until a specified density $d \cdot \frac{n \cdot (n-1)}{2}, 0 < d \leq 1$ or specified number of edges $m := |E|$ is reached. Then we randomly assign all labels not used yet. In the final step we iterate over all edges and assign further labels by uniform random decision. Parameter a specifies how many labels can be assigned to each edge, if not stated otherwise $a = 1$. Instead of directly using $|L|$ as a parameter, we may also specify the size of the label set by parameter $r = \frac{|L|}{|E|}, 0 < r \leq 1$. In contrast to the other instances, the instances of Set-III have relatively high values of r, i.e. $r = 1/4$ and $r = 3/4$. Although such instances are less likely to occur within practical applications

regarding telecommunication network design, they may be relevant for other scenarios, as for instance the compression model presented in Chapter 5 of this thesis.

4.3.2 Test environment

The generic framework presented in Section 4.2 has been implemented in C++ (`gpp-4.3`) within the SCIP framework [101]. The standard-plugins have been used for all computational tests unless explicitly stated otherwise. In addition some branch-and-cut algorithms (not involving any pricing procedures) have been implemented within the ILOG CONCERT framework [61] for comparison purposes. As LP solver ILOG CPLEX (in version 12.0) [61] has been used for both frameworks.

All computational tests have been performed on an Intel Xeon E5540 processor operating at 2.53 GHz and having 24 GB for totally 8 cores. The operation system is Ubuntu 9.10 with Linux-kernel 2.6.31. All runs have been performed in single-threaded mode, running times have been limited to 7200 seconds, unless stated otherwise.

4.3.3 Comparison of Described Methods

In this section we present a comparison of the described formulations based on computational tests. Furthermore we analyze the impact of particular "components" to each of the formulations. These components consist of the node-label-inequalities (4.16), the extended node-label-inequalities (4.17), the strong linkage of the edges to the edges (4.15), which can however only be used if only one label is assigned to the edges and the number of edges is not fixed by Equation (4.2). Table 4.1 provides an overview of these components and corresponding notation. After the comprehensive analysis and comparison of the particular methods in this section, we compare the results of the newly proposed methods to previous work in Section 4.3.4.

MIP formulations

In this section we primarily focus on the comparison of formulations EC, DCut and SCF. However, particularities like node-label-constraints (4.16), or fixed number of edges (4.2), or the direct linkage of labels to edges (4.15), may significantly change the picture regarding the superiority of one method over another one. For this reason we present the results not only for three formulations, but rather four to five variants of each formulation. Recall, that directly linking the labels to edges by Equations (4.15) is only possible for instances with one label assigned to each edge (4.15), i.e. $a = 1$ and is generally not possible for flow-formulations. In order not to be biased towards some particular class of instances we report these results for each of the three instance sets.

Tables 4.2 and 4.3 show the results for instances of Set-I with $|V| = 100$ and $|V| = 200$. These instances include graphs with various densities $d \in \{0.2, 0.5, 0.8\}$, where $|E| = d \cdot \frac{|V| \cdot (|V|-1)}{2}$, and different numbers of labels, i.e. $|L| = 1/2 \cdot |V|$, $|L| = |V|$, and $|L| = 5/4 \cdot |V|$.

Table 4.2: Comparison of selected variants of formulations EC, DCut and SCF on the instances from SET-I with $|V| = 100$.

		$\|L\| = 50$						$\|L\| = 100$						$\|L\| = 125$					
d	alg	cnt	opt	obj	t	bbn	cuts	cnt	opt	obj	t	bbn	cuts	cnt	opt	obj	t	bbn	cuts
0.2	EC	10	10	6.7	7	234	65	10	10	9.7	62	3876	1072	9	9	11.2	123	9365	2843
	EC_t	10	10	6.7	8	347	196	10	10	9.7	25	2482	414	9	9	11.2	34	3769	871
	EC_{tn}	10	10	6.7	4	100	173	10	10	9.7	11	857	339	9	9	11.2	16	1401	402
	EC_n	**10**	**10**	**6.7**	**0**	**24**	**3**	10	10	9.7	9	1298	179	9	9	11.2	33	2648	594
	EC_{sn}	**10**	**10**	**6.7**	**0**	**31**	**3**	**10**	**10**	**9.7**	**6**	**1127**	**34**	**9**	**9**	**11.2**	**12**	**2570**	**89**
	DCut	10	10	6.7	80	1167	691	10	10	9.7	1013	14378	7798	9	8	11.2	1470	19646	11403
	$DCut_t$	10	10	6.7	125	1458	954	10	10	9.7	478	6676	3919	9	9	11.2	740	10273	6450
	$DCut_{tn}$	10	10	6.7	21	127	116	10	10	9.7	86	1021	729	9	9	11.2	115	1426	1017
	$DCut_n$	10	10	6.7	11	90	96	10	10	9.7	57	1157	599	9	9	11.2	176	3186	1868
	$DCut_{sn}$	10	10	6.7	12	94	89	10	10	9.7	51	1190	516	9	9	11.2	129	3086	1536
	SCF	10	2	6.8	7028	136930	-1	10	0	10.3	7200	19854	-1	9	0	12.4	7200	17296	-1
	SCF_t	10	1	6.7	7133	143226	-1	10	0	10.3	7200	101319	-1	9	0	11.7	7200	119683	-1
	SCF_{tn}	10	10	6.7	15	83	-1	10	10	9.7	67	673	-1	9	9	11.2	85	1033	-1
	SCF_n	10	10	6.7	15	83	-1	10	10	9.7	63	673	-1	9	9	11.2	84	1033	-1
0.5	EC	10	10	3.0	17	46	5	10	10	4.7	374	5584	282	10	10	5.2	446	5307	630
	EC_t	10	10	3.0	14	69	148	10	10	4.7	175	5039	330	10	10	5.2	129	3249	382
	EC_{tn}	10	10	3.0	9	7	136	10	10	4.7	34	524	165	10	10	5.2	39	452	277
	EC_n	**10**	**10**	**3.0**	**0**	**2**	**0**	**10**	**10**	**4.7**	**8**	**221**	**4**	**10**	**10**	**5.2**	**9**	**127**	**4**
	EC_{sn}	**10**	**10**	**3.0**	**0**	**2**	**0**	10	10	4.7	9	173	4	10	10	5.2	11	370	12
	DCut	10	10	3.0	217	488	272	10	8	4.8	3858	15810	7214	10	10	5.2	3150	9316	4187
	$DCut_t$	10	10	3.0	190	465	256	10	9	4.7	3471	11770	6246	10	10	5.2	1794	5366	3175
	$DCut_{tn}$	10	10	3.0	56	13	28	10	10	4.7	401	1133	450	10	10	5.2	305	633	342
	$DCut_n$	10	10	3.0	27	12	28	10	10	4.7	261	1597	518	10	10	5.2	250	1179	401
	$DCut_{sn}$	10	10	3.0	27	23	55	10	10	4.7	216	1539	390	10	10	5.2	225	1234	362
	SCF	10	10	3.0	850	1475	-1	10	0	5.0	7200	11453	-1	10	0	5.8	7200	7586	-1
	SCF_t	10	10	3.0	722	1597	-1	10	6	4.7	5319	23618	-1	10	0	5.5	7200	18169	-1
	SCF_{tn}	10	10	3.0	22	1	-1	10	10	4.7	211	617	-1	10	10	5.2	171	298	-1
	SCF_n	10	10	3.0	20	1	-1	10	10	4.7	203	617	-1	10	10	5.2	176	298	-1
0.8	EC	10	10	2.0	12	2	12	10	10	3.0	161	848	98	10	10	4.0	999	4310	20
	EC_t	10	10	2.0	11	7	196	10	10	3.0	36	142	179	10	10	4.0	135	2344	130
	EC_{tn}	10	10	2.0	14	4	102	10	10	3.0	24	11	187	10	10	4.0	44	394	83
	EC_n	**10**	**10**	**2.0**	**0**	**1**	**0**	**10**	**10**	**3.0**	**1**	**3**	**2**	**10**	**10**	**4.0**	**17**	**102**	**0**
	EC_{sn}	**10**	**10**	**2.0**	**0**	**1**	**0**	10	10	3.0	2	4	1	**10**	**10**	**4.0**	**17**	**51**	**0**
	DCut	10	10	2.0	255	97	65	10	9	3.1	2367	2440	1286	10	0	4.0	7200	6958	3525
	$DCut_t$	10	10	2.0	198	127	131	10	9	3.1	1997	2702	1635	10	9	4.0	5083	6363	3410
	$DCut_{tn}$	10	10	2.0	87	7	19	10	10	3.0	314	370	216	10	10	4.0	923	1193	408
	$DCut_n$	10	10	2.0	40	5	22	10	10	3.0	418	932	457	10	10	4.0	780	1853	546
	$DCut_{sn}$	10	10	2.0	33	3	29	10	10	3.0	128	161	93	10	10	4.0	740	2154	668
	SCF	10	10	2.0	274	103	-1	10	0	3.6	7200	2826	-1	10	0	4.0	7200	2752	-1
	SCF_t	10	10	2.0	33	5	-1	10	0	4.0	7200	3035	-1	10	0	4.0	7200	2849	-1
	SCF_{tn}	10	10	2.0	28	1	-1	10	10	3.0	29	2	-1	10	10	4.0	243	349	-1
	SCF_n	10	10	2.0	28	1	-1	10	10	3.0	29	2	-1	10	10	4.0	230	349	-1

In these tables, as well as in the following ones, we report the following entities for each method and group of instances: Columns "cnt" contain the number of instances within each group, which is 10 in most of the cases. The reason for less than ten instances reported is not being able to finish some instances with particular formulations due to high memory requirements. Columns "opt" report the number of instances that have been solved and proved to be optimal within the timelimit. In columns "obj" the average objective value for

Table 4.3: Comparison of selected variants of formulations EC, DCut and SCF on the instances from SET-I with $|V| = 200$.

		$\|L\| = 100$						$\|L\| = 200$						$\|L\| = 250$					
d	alg	cnt	opt	obj	t	bbn	cuts	cnt	opt	obj	t	bbn	cuts	cnt	opt	obj	t	bbn	cuts
0.2	EC	10	8	8.0	3770	52682	16621	10	0	12.9	7200	42289	22303	10	0	15.4	7200	32009	18669
	EC_t	10	7	8.0	4541	43756	7665	10	0	13.6	7200	43298	8007	10	0	16.0	7200	43792	7293
	EC_{tn}	10	10	7.9	1296	14254	539	10	2	13.3	6878	60643	1436	10	1	14.9	6761	46296	1658
	EC_n	10	10	7.9	201	8379	5	10	5	12.2	5146	189033	973	10	3	13.8	6387	144597	5207
	EC_{sn}	**10**	**10**	**7.9**	**191**	**8322**	**4**	**10**	**7**	**12.1**	**4331**	**182237**	**81**	**10**	**3**	**13.9**	**6153**	**250129**	**158**
	DCut	10	0	8.8	7200	10929	8971	10	0	14.5	7200	3850	3632	10	0	18.9	7200	3681	3548
	$DCut_t$	10	0	9.0	7200	4392	4036	10	0	14.5	7200	2896	2866	10	0	16.1	7200	2653	2622
	$DCut_{tn}$	10	7	8.1	5487	13056	8036	10	0	13.0	7200	5588	4763	10	0	15.7	7200	3982	3552
	$DCut_n$	10	9	8.0	3398	14653	8637	10	0	12.9	7200	15877	12200	10	0	14.9	7200	11207	9494
	$DCut_{sn}$	10	9	8.0	2996	14937	7658	10	0	12.7	7200	19280	13569	10	0	14.7	7200	13712	10837
	SCF	10	0	9.3	7200	1017	-1	10	0	14.5	7200	634	-1	10	0	16.8	7200	559	-1
	SCF_t	10	0	8.9	7200	3204	-1	10	0	13.5	7200	4733	-1	10	0	15.8	7200	5326	-1
	SCF_{tn}	10	8	8.0	3353	9362	-1	10	0	12.4	7200	6854	-1	10	0	14.1	7200	4343	-1
	SCF_n	10	8	8.0	3498	8308	-1	10	0	12.4	7200	6468	-1	10	0	14.1	7200	4099	-1
0.5	EC	10	10	3.4	769	2082	69	10	0	5.8	7200	8603	539	10	0	6.5	7200	5380	456
	EC_t	10	10	3.4	1452	2744	558	10	0	5.8	7200	10307	647	10	0	6.4	7200	9084	1024
	EC_{tn}	10	10	3.4	570	469	678	10	7	5.5	4249	6550	908	10	0	6.5	7200	15870	861
	EC_n	10	10	3.4	25	126	1	10	9	5.4	1291	14284	12	10	8	6.4	4323	57715	31
	EC_{sn}	**10**	**10**	**3.4**	**19**	**92**	**1**	**10**	**9**	**5.4**	**1176**	**14653**	**6**	**10**	**9**	**6.4**	**4049**	**62371**	**18**
	DCut	9	0	4.1	7200	1086	728	10	0	7.2	7200	323	265	10	0	7.9	7200	272	215
	$DCut_t$	9	0	4.3	7200	613	469	10	0	7.9	7200	298	290	10	0	8.2	7200	335	307
	$DCut_{tn}$	10	8	3.5	5135	954	481	10	0	6.6	7200	557	349	10	0	7.4	7200	420	282
	$DCut_n$	10	8	3.5	3132	1079	507	10	0	6.2	7200	1795	929	10	0	7.1	7200	1097	564
	$DCut_{sn}$	10	9	3.4	2054	979	412	10	0	6.0	7200	2072	912	10	0	6.7	7200	1432	671
	SCF	10	0	4.3	7200	124	-1	10	0	7.0	7200	69	-1	10	0	7.8	7200	54	-1
	SCF_t	10	0	3.9	7200	207	-1	10	0	6.5	7200	166	-1	10	0	7.3	7200	150	-1
	SCF_{tn}	10	10	3.4	1102	270	-1	10	0	5.7	7200	828	-1	10	0	6.5	7200	839	-1
	SCF_n	10	10	3.4	1204	270	-1	10	0	5.7	7200	749	-1	10	0	6.4	7200	728	-1
0.8	EC	10	10	2.6	2803	2968	16	10	0	4.0	7200	1132	16	10	0	5.0	7200	1640	48
	EC_t	10	10	2.6	3040	3650	505	10	0	4.0	7200	2146	656	10	2	4.4	7064	6763	757
	EC_{tn}	10	9	2.7	2739	103	613	10	10	4.0	5038	6819	634	10	9	4.1	2902	1331	776
	EC_n	10	10	2.6	76	2	73	10	10	4.0	1122	5975	1	10	10	4.0	609	3324	3
	EC_{sn}	**10**	**10**	**2.6**	**28**	**1**	**1**	**10**	**10**	**4.0**	**911**	**4845**	**1**	**10**	**10**	**4.0**	**301**	**777**	**1**
	DCut	4	0	3.0	7200	152	134	10	0	8.2	7201	105	110	10	0	7.1	7200	69	70
	$DCut_t$	10	0	3.9	7200	151	171	10	0	5.2	7202	110	112	10	0	6.8	7200	111	112
	$DCut_{tn}$	10	3	3.3	6344	142	67	10	0	4.6	7200	177	102	10	0	5.5	7200	120	98
	$DCut_n$	10	3	2.7	6528	1103	444	10	0	4.2	7200	329	170	10	0	4.9	7200	158	126
	$DCut_{sn}$	10	6	2.6	5135	799	292	10	0	4.0	7200	556	227	10	0	5.0	7200	226	140
	SCF	10	0	2.9	7200	69	-1	10	0	4.9	7200	35	-1	10	0	5.5	7200	33	-1
	SCF_t	10	2	2.8	5923	54	-1	10	0	4.3	7200	58	-1	10	0	5.1	7201	35	-1
	SCF_{tn}	10	9	2.6	4177	74	-1	10	1	4.0	6929	391	-1	10	3	4.7	6712	264	-1
	SCF_n	10	9	2.6	4183	79	-1	10	1	4.0	6973	375	-1	10	2	4.7	6871	229	-1

all instances in the group is reported. If not all instances have been solved to optimality, this value corresponds to the average value of feasible solutions that have been found within the timelimit. Average running times in seconds are then reported in columns "t". The average number of branch-and-bound nodes is listed in columns "bbn", the average number of generated cuts in column "cuts". Results of the fastest method(s) for each group are emphasized with bold letters.

From Tables 4.2 and 4.3 we can already observe that the difficulty of solving these instances is strongly correlated to the objective function values of the instances. Higher values, in particular those larger than ten, require significantly more B&B-nodes, and the separation of more cuts. This also implies longer average running times. This property holds for all of the considered formulations. The results in Tables 4.2 and 4.3 show that formulation EC_{sn} consistently gives the best results for these instances. The single commodity flow formulations show a slightly better performance than the directed cut formulations for most of the instances.

The strength of the node-label-inequalities (4.16) is also demonstrated by the results in Tables 4.2 and 4.3. Their addition to the plain formulations does not only yield a significant speedup, but also enables to solve more instances regarding the set with $|V| = 200$. The difference between Inequalities (4.16) and their extended form, given by Inequalities (4.17) is examined in Section 4.3.3. Regarding Equation (4.2) no clear conclusion can be drawn from these instances. If, however, combinations of these components are considered, the variants only using the node-label-constraints are superior in most of the cases. For formulations EC and DCut it is also possible to directly link the edges to the labels by Equations (4.15). In most of the cases, this yields the best results, when combined with the node-label-inequalities for both formulations, and in particular in combination with EC the overall best results.

Table 4.4 reports the results for the same formulations for the instances of Set-II. These instances have the major difference to contain only graphs of extremely low density d and just very few labels. Again, we can observe a clear superiority of formulations EC_{sn} and EC_{n}, which are able to solve all these instances with average running times of less than a half second.

In Tables 4.5, 4.6, and 4.7 results for the instances from Set-III are reported. Table 4.5 shows the results for instances with $|V| = 100$ and $a = 1$, i.e. one single label assigned to the edges. As already mentioned in Section 4.3.1, this instances differ from the previous ones in the way that they contain a higher number of labels, i.e. $r = 1/4$ and $r = 3/4$ with $r = \frac{|L|}{|E|}$. It can be observed that it is beneficial to limit the number of edges to $|V| - 1$ by Equation (4.2) in this case. Thus, the stronger LP-relaxation implied by this restriction is beneficial in the case of higher values of r. For instances with $r = 1/4$ formulation EC still shows the best performance, but DCut provides better results in the case of $r = 3/4$. Hence, the strong LP-relaxation becomes even more important if $|L|$ is in the same order of magnitude as $|E|$.

With a single exception the same effect can be observed for the instances with $a \in \{2, 5\}$ reported in Table 4.6. The effect of more than one label being assigned to the edges seems to make the problem easier to solve, but the effect is relatively small. It is important to note, that directly linking the labels to the edges, which was beneficial for the instances with $a = 1$, cannot be applied to instances with larger a.

Table 4.7 shows the result for grid-graphs with 100 and 400 nodes and $|L| \in \{30, 50, 80\}$. The average optimal objective value on these graphs is relatively high, which makes them

Table 4.4: Comparison of selected variants of formulations EC, DCut and SCF on the instances from SET-II.

		$\|L\| = 5$						$\|L\| = 10$						$\|L\| = 20$					
$\|V\|$, $\|E\|$	alg	cnt	opt	obj	t	bbn	cuts	cnt	opt	obj	t	bbn	cuts	cnt	opt	obj	t	bbn	cuts
200, 800	EC	10	10	3.0	0	1	0	10	10	5.0	0	3	0	10	10	7.9	0	13	2
	EC_t	10	10	3.0	0	2	236	10	10	5.0	0	8	315	10	10	7.9	2	47	582
	EC_{tn}	10	10	3.0	0	2	157	10	10	5.0	0	6	249	10	10	7.9	1	20	515
	EC_n	10	10	3.0	0	1	0	10	10	5.0	0	1	0	10	10	7.9	0	6	2
	EC_{sn}	10	10	3.0	0	1	0	10	10	5.0	0	1	0	10	10	7.9	0	6	1
	DCut	10	10	3.0	1	6	38	10	10	5.0	4	14	63	10	10	7.9	14	127	171
	$DCut_t$	10	10	3.0	3	6	26	10	10	5.0	7	18	37	10	10	7.9	18	161	173
	$DCut_{tn}$	10	10	3.0	0	4	29	10	10	5.0	2	7	36	10	10	7.9	8	21	63
	$DCut_n$	10	10	3.0	0	2	24	10	10	5.0	0	6	49	10	10	5.0	0	6	49
	$DCut_{sn}$	10	10	3.0	0	1	0	10	10	5.0	0	1	0	10	10	7.9	0	6	1
	SCF	10	10	3.0	2	2	-1	10	10	5.0	9	32	-1	10	10	7.9	69	800	-1
	SCF_t	10	10	3.0	3	2	-1	10	10	5.0	9	32	-1	10	10	7.9	71	800	-1
	SCF_{tn}	10	10	3.0	0	1	-1	10	10	5.0	1	3	-1	10	10	7.9	6	16	-1
	SCF_n	10	10	3.0	0	1	-1	10	10	5.0	1	3	-1	10	10	7.9	6	16	-1
500, 2000	EC	10	10	3.5	0	1	0	10	10	5.9	0	2	0	10	10	9.9	0	12	0
	EC_t	10	10	3.5	3	2	621	10	10	5.9	6	7	833	10	10	9.9	16	44	1129
	EC_{tn}	10	10	3.5	2	1	541	10	10	5.9	5	6	763	10	10	9.9	16	25	1385
	EC_n	10	10	3.5	0	1	0	10	10	5.9	0	1	0	10	10	9.9	0	8	0
	EC_{sn}	10	10	3.5	0	1	0	10	10	3.5	0	1	0	10	10	9.9	0	7	0
	DCut	10	10	3.5	5	5	34	10	10	5.9	14	14	76	10	10	9.9	48	152	184
	$DCut_t$	10	10	3.5	13	7	25	10	10	5.9	28	15	34	10	10	9.9	68	181	144
	$DCut_{tn}$	10	10	3.5	2	3	16	10	10	5.9	9	8	33	10	10	5.9	9	8	33
	$DCut_n$	10	10	3.5	1	2	46	10	10	5.9	3	6	67	10	10	9.9	20	20	139
	$DCut_{sn}$	10	10	3.5	1	2	82	10	10	5.9	3	6	60	10	10	9.9	20	19	129
	SCF	10	10	3.5	10	3	-1	10	10	5.9	28	18	-1	10	10	9.9	372	661	-1
	SCF_t	10	10	3.5	11	3	-1	10	10	5.9	29	18	-1	10	10	9.9	384	661	-1
	SCF_{tn}	10	10	3.5	0	1	-1	10	10	5.9	4	3	-1	10	10	9.9	20	20	-1
	SCF_n	10	10	3.5	0	1	-1	10	10	5.9	3	3	-1	10	10	9.9	18	20	-1
1000, 4000	EC	10	10	4.1	0	1	0	10	10	6.6	0	1	0	10	10	11.3	0	13	0
	EC_t	10	10	4.1	20	1	1182	10	10	6.6	46	6	1762	10	10	11.3	121	54	3514
	EC_{tn}	10	10	4.1	300	1	3823	10	10	6.6	234	6	2660	10	10	11.3	108	26	2909
	EC_n	10	10	4.1	0	1	0	10	10	6.6	0	1	0	10	10	11.3	0	7	0
	EC_{sn}	10	10	4.1	0	1	0	10	10	6.6	0	1	0	10	10	11.3	0	6	0
	DCut	10	10	4.1	16	5	40	10	10	6.6	47	13	259	10	10	11.3	144	191	275
	$DCut_t$	10	10	4.1	54	6	22	10	10	6.6	90	20	36	10	10	11.3	240	189	150
	$DCut_{tn}$	10	10	4.1	7	3	23	10	10	6.6	26	7	26	10	10	11.3	103	36	47
	$DCut_n$	10	10	4.1	12	1	184	10	10	6.6	13	5	195	10	10	11.3	64	26	355
	$DCut_{sn}$	10	10	4.1	11	1	178	10	10	6.6	47	6	495	10	10	11.3	57	24	253
	SCF	10	10	4.1	30	2	-1	10	10	6.6	99	14	-1	10	10	11.3	1243	416	-1
	SCF_t	10	10	4.1	31	2	-1	10	10	6.6	96	14	-1	10	10	11.3	1303	416	-1
	SCF_{tn}	10	10	4.1	1	1	-1	10	10	6.6	12	3	-1	10	10	11.3	52	31	-1
	SCF_n	10	10	4.1	1	1	-1	10	10	6.6	12	3	-1	10	10	11.3	48	31	-1

difficult to solve. However, all instances with $|L| \in \{30, 50\}$ could be solved to optimality by formulation EC_{sn}, which showed the overall best performance on this class of instances.

Having now analyzed the main variations of the discussed formulations we draw our attention to further approaches and enhancements that have been proposed in Section 4.2.

Table 4.5: Comparison of selected variants of formulations EC, DCut and SCF on the instances from SET-III with $|V| = 100,\ a = 1$.

		$\lvert L\rvert = 1/4 \cdot \lvert E\rvert$						$\lvert L\rvert = 3/4 \cdot \lvert E\rvert$					
d	alg	cnt	opt	obj	t	bbn	cuts	cnt	opt	obj	t	bbn	cuts
0.05	EC	10	10	19.6	2	1892	1722	10	0	61.6	7200	108393	124719
	EC_t	10	10	19.6	23	6887	6026	10	1	51.3	6480	178187	195559
	EC_{tn}	10	10	19.6	14	4944	4486	10	2	51.2	5760	163875	182801
	EC_n	10	10	19.6	2	1491	1246	10	0	62.5	7200	103413	119245
	EC_{sn}	**10**	**10**	**19.6**	**1**	**1313**	**966**	10	0	55.5	7200	133833	151889
	DCut	10	10	19.6	35	5214	3026	10	0	50.7	7200	367146	337417
	$DCut_t$	10	10	19.6	34	4174	2487	10	4	49.8	4381	76629	55867
	$DCut_{tn}$	10	10	19.6	12	1148	759	10	6	49.8	3195	87152	60318
	$DCut_n$	10	10	19.6	13	1437	941	10	0	50.4	7200	337882	315950
	$DCut_{sn}$	10	10	19.6	1	1313	966	10	0	50.4	7200	360801	343056
	SCF	10	3	19.6	5576	987720	-1	10	0	51.5	7200	1815207	-1
	SCF_t	10	9	19.6	1354	513872	-1	10	5	50.0	3823	3081605	-1
	SCF_{tn}	10	10	19.6	31	5160	-1	**10**	**8**	**49.9**	**1508**	**800971**	**-1**
	SCF_n	10	10	19.6	49	9415	-1	10	0	50.6	7200	894051	-1
0.2	EC	10	1	15.1	7099	208082	117362	10	0	45.1	7200	62300	63102
	EC_t	10	10	14.8	675	54742	23790	10	4	36.5	6326	138547	137549
	EC_{tn}	**10**	**10**	**14.8**	**344**	**36386**	**16745**	10	2	37.1	6450	120465	121687
	EC_n	10	2	15.3	6369	136657	94927	10	0	46.0	7200	40163	41557
	EC_{sn}	10	4	14.8	4894	231148	95062	10	0	39.2	7200	65167	63256
	DCut	10	0	16.3	7200	48073	35316	10	0	38.9	7200	119789	87861
	$DCut_t$	10	6	14.8	3196	55169	37144	10	6	35.8	3706	61762	47626
	$DCut_{tn}$	10	10	14.8	835	13677	8698	**10**	**7**	**35.8**	**2432**	**36099**	**27852**
	$DCut_n$	10	0	15.7	7200	39132	29700	10	0	38.5	7200	48576	40239
	$DCut_{sn}$	10	1	15.6	7134	78339	57546	10	0	38.0	7200	88645	73235
	SCF	10	0	17.0	7200	14435	-1	10	0	40.5	7200	39472	-1
	SCF_t	10	0	15.5	7200	173479	-1	10	0	37.8	7200	480980	-1
	SCF_{tn}	10	9	14.8	2401	31073	-1	10	1	36.1	6495	152260	-1
	SCF_n	10	0	15.2	7200	63078	-1	10	0	38.4	7200	29479	-1
0.5	EC	10	0	15.8	7200	69554	36260	10	0	38.8	7200	54153	55751
	EC_t	10	8	13.3	2570	62487	30425	10	5	30.7	4064	51442	50552
	EC_{tn}	**9**	**8**	**13.2**	**1400**	**34106**	**18769**	10	4	31.0	5038	59605	57375
	EC_n	10	0	16.2	7200	35944	18323	10	0	38.9	7200	28984	31127
	EC_{sn}	10	0	13.7	7200	296782	88962	10	0	33.8	7200	63430	64801
	DCut	10	0	15.4	7200	7645	4291	10	0	36.0	7200	31008	19324
	$DCut_t$	10	6	13.5	5152	9463	9036	10	6	30.3	4331	16003	13285
	$DCut_{tn}$	10	5	13.5	4557	9195	8543	**10**	**7**	**30.2**	**3552**	**12337**	**9780**
	$DCut_n$	10	0	15.4	7200	5807	3785	10	0	36.6	7200	6935	5131
	$DCut_{sn}$	10	0	14.4	7200	12834	8154	10	0	32.4	7200	14108	9913
	SCF	10	0	16.4	7200	1345	-1	10	0	34.5	7200	3160	-1
	SCF_t	10	0	14.6	7200	19313	-1	10	0	32.3	7200	35869	-1
	SCF_{tn}	10	5	13.5	4756	9842	-1	10	2	30.7	5842	16960	-1
	SCF_n	10	0	14.5	7200	6765	-1	10	0	33.1	7200	2327	-1

Further Methods

In Section 4.3.3 the node-label-inequalities (4.16) have been shown to be of utter importance for a strong formulation. In Section 4.2.3 we have also presented an extension of this idea, where two nodes are considered instead of just one. This led to the class of Inequal-

Table 4.6: Comparison of selected variants of formulations EC, DCut and SCF on the instances from SET-III with $|V| = 100$, $a \in \{2, 5\}$.

		$\lvert L\rvert = 1/4 \cdot \lvert E\rvert$, $a = 2$						$\lvert L\rvert = 3/4 \cdot \lvert E\rvert$, $a = 2$						$\lvert L\rvert = 1/4 \cdot \lvert E\rvert$, $a = 5$						$\lvert L\rvert = 3/4 \cdot \lvert E\rvert$, $a = 5$					
d	alg	cnt	opt	obj	t	bbn	cuts	cnt	opt	obj	t	bbn	cuts	cnt	opt	obj	t	bbn	cuts	cnt	opt	obj	t	bbn	cuts
0.05	EC	10	10	16.6	7	3222	3277	10	0	42.8	7200	112158	123363	10	10	10.5	0	152	452	10	0	21.4	7200	111275	106501
	EC_t	10	10	16.6	63	11657	9901	10	7	35.0	2558	113102	106934	10	10	10.5	0	362	400	10	4	20.8	4763	178547	160611
	EC_{tn}	10	10	16.6	12	4707	4080	10	7	35.0	2375	102179	93532	10	10	10.5	0	306	359	10	6	20.6	3467	143690	125295
	EC_n	**10**	**10**	**16.6**	**2**	**1883**	**1956**	10	0	42.4	7200	116472	124838	10	10	10.5	0	158	426	10	0	21.5	7200	107718	103093
	DCut	10	10	16.6	31	4173	2409	10	0	35.1	7200	374197	286064	10	10	10.5	7	247	238	10	0	20.5	7200	375009	236293
	$DCut_t$	10	10	16.6	36	3523	2114	10	10	34.7	112	6725	4842	10	10	10.5	10	350	301	10	9	20.5	1066	75587	46230
	$DCut_{tn}$	10	10	16.6	16	1162	787	**10**	**10**	**34.7**	**61**	**4661**	**3696**	10	10	10.5	5	115	134	**10**	**10**	**20.5**	**698**	**45870**	**25073**
	$DCut_n$	10	10	16.6	11	1026	694	10	0	35.0	7200	376389	268567	10	10	10.5	3	95	115	10	7	20.5	4126	215290	129837
	SCF	10	1	16.7	7153	896512	-1	10	0	37.1	7200	701281	-1	10	9	10.5	2318	507224	-1	10	0	21.8	7200	276624	-1
	SCF_t	10	7	16.6	4038	1339695	-1	10	1	35.0	7046	3581766	-1	10	10	10.5	1580	440888	-1	10	0	21.0	7200	1707957	-1
	SCF_{tn}	10	10	16.6	29	3894	-1	10	10	34.7	609	239014	-1	10	10	10.5	4	284	-1	10	9	20.5	2218	462195	-1
	SCF_n	10	10	16.6	67	11755	-1	10	0	35.8	7200	666248	-1	10	10	10.5	5	329	-1	10	0	20.9	7200	720521	-1
0.2	EC	10	3	12.3	6521	268240	117079	10	0	31.8	7200	72720	69345	10	10	7.8	3364	236585	34987	10	0	19.5	7200	159231	107082
	EC_t	10	8	12.1	1840	81598	40109	10	3	26.2	5845	141568	127229	10	10	7.8	265	39875	773	10	3	15.1	5184	171490	89976
	EC_{tn}	**10**	**10**	**11.9**	**629**	**42052**	**20637**	10	3	26.3	5242	119550	107358	**10**	**10**	**7.8**	**128**	**12441**	**651**	10	3	15.1	5140	139983	90176
	EC_n	10	7	12.1	4285	166054	79457	10	0	34.0	7200	49859	50940	10	10	7.8	1041	56147	8166	10	0	19.0	7200	115188	89434
	DCut	10	0	13.4	7200	43412	32497	10	0	27.3	7200	72779	48772	10	1	7.9	6892	70690	36507	10	0	16.6	7200	33350	23851
	$DCut_t$	10	10	11.9	1477	17841	13183	10	7	25.6	2957	55433	38121	10	5	7.8	5653	67754	42560	10	4	14.9	5423	55800	43216
	$DCut_{tn}$	10	10	11.9	681	5906	5180	**10**	**9**	**25.6**	**1489**	**24931**	**17663**	10	10	7.8	1628	15222	9552	**10**	**6**	**14.8**	**4406**	**44628**	**32312**
	$DCut_n$	10	0	13.2	7200	39599	30225	10	0	27.4	7200	31085	23420	10	7	7.8	4679	65415	25812	10	0	15.8	7200	15761	11375
	SCF	10	0	14.3	7200	13282	-1	10	0	30.0	7200	19389	-1	10	0	8.8	7200	22933	-1	10	0	18.1	7200	12399	-1
	SCF_t	10	0	12.9	7200	105555	-1	10	0	27.0	7200	320786	-1	10	0	8.1	7200	107916	-1	10	0	15.8	7200	109149	-1
	SCF_{tn}	10	8	12.1	1909	18528	-1	10	4	25.9	5338	87496	-1	10	10	7.8	1669	11756	-1	10	2	15.1	6327	44397	-1
	SCF_n	10	0	12.8	7200	47542	-1	10	0	27.6	7200	15439	-1	10	9	7.8	4151	59801	-1	10	0	16.0	7200	10494	-1
0.5	EC	10	0	11.5	7200	98257	38000	10	0	28.4	7200	93989	77494	10	1	6.9	6785	101172	7242	10	0	14.5	7200	50518	29040
	EC_t	10	10	10.9	632	42951	5268	10	1	23.4	6487	85485	71003	10	10	6.9	612	27936	785	9	5	13.1	3804	100666	38284
	EC_{tn}	**10**	**10**	**10.9**	**506**	**22467**	**6432**	10	3	26.3	5242	119550	107358	**10**	**10**	**6.9**	**255**	**6604**	**532**	**9**	**6**	**13.0**	**3472**	**72582**	**38165**
	EC_n	10	0	11.3	7200	37858	18365	10	0	29.0	7200	44406	39926	10	0	7.0	7200	29141	2143	10	0	16.2	7200	11751	6165
	DCut	10	0	11.8	7200	9769	5503	10	0	30.4	7200	15852	9516	10	0	7.1	7200	11487	5995	10	0	15.2	7200	5742	3186
	$DCut_t$	10	0	11.3	7200	13089	11560	**10**	**8**	**22.5**	**4745**	**9585**	**8851**	10	0	7.4	7200	7977	6213	10	0	13.8	7200	7827	7958
	$DCut_{tn}$	10	5	11.2	5664	11813	9786	10	5	22.8	4259	8850	8481	10	6	7.1	5089	5318	3529	10	4	13.1	5743	7934	7389
	$DCut_n$	10	0	11.9	7200	7129	4426	10	0	27.1	7200	4738	2996	10	1	7.1	6717	7337	3172	10	0	15.2	7200	2682	1652
	SCF	10	0	13.2	7200	1477	-1	10	0	26.8	7200	1628	-1	10	0	8.0	7200	2606	-1	10	0	16.2	7200	923	-1
	SCF_t	10	0	12.1	7200	13970	-1	10	0	24.3	7200	26439	-1	10	0	7.6	7200	6757	-1	10	0	14.3	7200	10634	-1
	SCF_{tn}	10	6	10.9	4841	6975	-1	10	0	23.4	7200	12529	-1	10	7	7.0	4238	3003	-1	10	2	13.3	5923	6203	-1
	SCF_n	10	0	11.8	7200	9232	-1	10	0	25.0	7200	3000	-1	10	0	7.0	7200	21470	-1	10	0	14.4	7200	4668	-1

Table 4.7: Comparison of selected variants of formulations EC, DCut and SCF on the grid graph instances from SET-III with $|E| \approx 4 \cdot |V|$.

		$\|V\| = 10 \times 10$						$\|V\| = 20 \times 20$					
$\|L\|$	alg	cnt	opt	obj	t	bbn	cuts	cnt	opt	obj	t	bbn	cuts
30	EC	10	10	9.2	4	2041	1540	10	10	11.5	61	7496	400
	EC_t	10	10	9.2	6	2641	1843	10	10	11.5	183	7547	5819
	EC_{tn}	10	10	9.2	4	1639	1544	10	10	11.5	91	3757	4231
	EC_n	**10**	**10**	**9.2**	**1**	**1369**	**656**	**10**	**10**	**11.5**	**17**	**3602**	**200**
	EC_{sn}	**10**	**10**	**9.2**	**1**	**1281**	**601**	**10**	**10**	**11.5**	**17**	**3558**	**175**
	DCut	10	10	9.2	61	4948	3240	2	2	11.0	5224	6233	37505
	$DCut_t$	10	10	9.2	93	6196	4135	10	10	11.5	3029	22956	19238
	$DCut_{tn}$	10	10	9.2	34	1892	1710	10	10	11.5	627	4058	10961
	$DCut_n$	10	10	9.2	18	1512	1364	1	0	12.0	7200	39	24766
	$DCut_{sn}$	10	10	9.2	19	1356	2314	1	0	13.0	7200	26	23113
	SCF	10	10	9.2	815	118270	-1	10	0	11.8	7200	22192	-1
	SCF_t	10	10	9.2	641	90448	-1	10	0	11.7	7200	22091	-1
	SCF_{tn}	10	10	9.2	17	1280	-1	10	10	11.5	251	3127	-1
	SCF_n	10	10	9.2	12	1008	-1	10	10	11.5	218	3076	-1
50	EC	9	8	13.0	1018	54653	41036	10	9	17.1	4341	464602	31132
	EC_t	9	7	13.1	1959	99397	73687	10	0	17.0	7200	301858	167539
	EC_{tn}	9	8	13.0	1578	83996	67834	10	10	17.0	3834	212265	107170
	EC_n	9	9	12.9	129	25299	16378	10	10	17.0	1347	208221	11217
	EC_{sn}	**9**	**9**	**12.9**	**66**	**22339**	**13131**	**10**	**10**	**17.0**	**1087**	**193213**	**8470**
	DCut	9	9	12.9	867	67360	37873	7	0	17.3	7200	29390	47832
	$DCut_t$	9	9	12.9	1032	74718	42088	10	0	17.2	7200	32248	33668
	$DCut_{tn}$	9	9	12.9	302	20596	12907	10	0	17.0	7200	86172	64236
	$DCut_n$	9	9	12.9	336	28974	17613	7	0	16.3	7200	3345	36785
	$DCut_{sn}$	9	9	12.9	207	20951	13323	7	0	15.7	7200	648	23682
	SCF	9	0	13.3	7200	368322	-1	10	0	18.5	7200	13103	-1
	SCF_t	9	0	13.2	7200	741954	-1	10	0	18.5	7200	19592	-1
	SCF_{tn}	9	9	12.9	215	21632	-1	10	0	17.1	7200	131296	-1
	SCF_n	9	9	12.9	299	34004	-1	10	8	17.1	5532	169723	-1
80	EC	10	0	19.5	7200	134608	118377	10	0	25.0	7200	199143	139246
	EC_t	10	0	19.9	7200	231789	208661	10	0	24.9	7200	143214	104486
	EC_{tn}	10	0	19.6	7200	229619	202787	10	0	24.8	7200	146616	112030
	EC_n	10	0	19.5	7200	162192	138904	10	0	24.6	7200	305748	138953
	EC_{sn}	10	0	18.9	7200	228167	176741	10	0	24.6	7200	299188	134344
	DCut	10	0	18.8	7200	252826	225273	10	0	25.5	7200	31756	32402
	$DCut_t$	10	0	18.9	7200	248123	216513	9	0	19.8	7200	24121	25391
	$DCut_{tn}$	9	0	18.7	7200	283625	220940	10	0	25.2	7200	43604	44755
	$DCut_n$	10	0	18.8	7200	239489	197073	8	0	24.6	7200	54226	62124
	$DCut_{sn}$	10	0	18.9	7200	248966	213157	7	0	25.1	7200	25262	46754
	SCF	10	0	19.7	7200	285092	-1	10	0	27.0	7200	9767	-1
	SCF_t	10	0	19.2	7200	948572	-1	10	0	28.0	7200	22180	-1
	SCF_{tn}	10	0	19.0	7200	689593	-1	10	0	25.3	7200	68435	-1
	SCF_n	10	0	18.8	7200	863039	-1	10	0	25.2	7200	99643	-1

ities (4.17). Table 4.8 shows a comparison of formulations EC_t and $DCut_t$ with on the one hand the node-label-inequalities (4.16) and on the other hand additional Inequalities (4.17). In particular for formulation EC_t these further inequalities turn out to be useful

in many cases. They do not only speedup the solution process, but moreover frequently enable to solve more instances to provable optimality. However, also the opposite is often the case. It is therefore not possible to decide which approach is superior over the other based on the available data. On grid-graphs Inequalities (4.17) have not been beneficial at all.

Further formulations, considered in Section 4.2, are based on the property, that a tree must not contain a cycle by definition. Formulation MTZ_{tn} requires just a polynomial number of variables, but contains constraints with infamous "Big-M" constants, as the SCF formulation does. On the contrary CEF contains an exponential number of Inequalities (4.9), which need to be separated as cutting-planes as for the DCut or EC formulation. Due to their fast separation by a simple shortest-path computation, also other formulations may benefit from additionally using cycle-elimination cuts. Corresponding results are reported in Table 4.9, column "cec" lists the average number of separated cycle-elimination cuts. Whereas MTZ_{tn} and CEF_{tn} show a relatively weak performance on the instances with $r = 1/4$, they provide good results in the case of $r = 3/4$. In particular for the low density graphs CEF_{tn} could solve all instances to optimality, which no other method was able to. For the dense graphs best results are obtained by DCut_{tnc} and EC_{tnc}.

Table 4.10 shows the results that have been obtained by including primal heuristics into the branch-and-bound algorithm. Formulations EC_{tn}, EC_{sn}, DCut_{tn}, and DCut_{sn} are considered for this purpose. Results are reported for the MVCA and ACO used as primal heuristic. The parameters for ACO are chosen with an emphasis on fast execution times. We used $M = 10$ ants and $it = 40$ iterations. The simple pheromone model (P-i) was choosen as well as the solution construction mechanism (C-iii). The other parameters have been set as described in Chapter 3. As indicated by preliminary experiments it turned out to be advantageous only to use the primal heuristics in the root node, as they were generally not able to find improved solutions based on the information provided by the LP-solution in other B&B-nodes. The results in Table 4.10 show, that the fast MVCA yields superior results over the ACO, when used as primal heuristic. Embedding MVCA in B&B has a positive effect w.r.t. the variants "tn" of formulations EC and DCut, but a negative impact concerning variants "ts".

Odd-Hole Inequalities

We now draw our attention to the odd-hole inequalities. Within preliminary tests we determined a tight timelimit of 10^{-3} seconds for the MIP (4.20) to show the overall best performance. Two algorithmic variants are considered for the results reported in Table 4.11. The first version (denoted with index o) simply adds the found valid cutting-planes to the MIP. Alternatively, the set of labels corresponding to the obtained odd-hole can also be used to deduce a branching rule. This was motivated by the observation that many lifted odd-hole cutting planes, found by MIP (4.20), were not strong enough to define facets w.r.t. the involved label variables. As a consequence, these variables remained fractional after the cutting-plane was added to the MIP. However, odd-holes provide an important information and references to situations where special configurations of label-variables

artificially reduce the LP-relaxation. Hence it is likely, that immediately branching over these variables may be beneficial. This is done by inserting all labels of the odd-hole into a global queue, and always branch over such a variable unless the queue is empty. Index *ob* denotes this approach in Table 4.11. Odd-hole cuts are separated with lowest priority amongst the user-defined cutting-planes, and are only separated in levels of the B&B-tree which are multiples of ten.

The results in Table 4.11 show that the odd-hole inequalities are beneficial in many cases, in particular when used to deduce branching-rules from the corresponding label-variables. For instances from Set-I and Set-II almost no odd-holes have been found with the described parameter settings. For dense graphs it is less likely to find odd-holes that are violated by the current LP-solution, as each node is incident to many edges. Hence $|L(v)|$ is in the same order of magnitude as $|V|$ in the expected case. This implies many non-zero lifting coefficients in Inequalities (4.19), reducing the chance of finding a valid inequality that is actually violated by the current LP-solution. Hence, the separation of odd-hole inequalities is most beneficial for sparse graphs. Also the number of labels compared to the number of edges has an impact on the efficiency of the odd-hole separation. If the number of labels is relatively low, the expected label frequency ν_l will be high. This implies high values for the lifting coefficients γ_l, which in turn reduces the chance of finding violated odd-hole inequalities. If, on the other hand, the number of edges is too high, odd-holes are generally less likely to occur, as the sets $L(v) \cap L(u),\ \text{for all } v, u \in V$ can be expected to be very small or even empty.

Branch-and-Cut-and-Price

Additionally using the column generation approach within the B&C framework, i.e. branch-and-cut-and-price (BCP) is only beneficial for a very special class of instances. For most of the instances almost all variables are priced in during the solution process. The computational overhead for solving the pricing problem and resolving the MIP implies significantly higher running times in this case. However, if the instances consist of a high number of labels, and have an optimal solution that is significantly lower than the average optimal solution value when assigning the labels to the edges randomly in the instance construction process, BCP shows a superior performance. To study this effect, special instances have been created, that contain single optima having a relatively low number of labels. The computational results for these instances are reported in Table 4.12. In particular for the larger instances a clear superiority of the BCP approach w.r.t. the corresponding B&C algorithm can be observed. For this special class of instances, the percentage of created label variables is always less than 30% of the total number of labels (reported in column "priced"). Although the importance of such instances may be quite limited for many purposes, the instances regarded in Chapter 5 exhibit comparable properties. For the data-compression application presented therein, the BCP approach is thus a valuable and important mean for exactly solving large instances.

Summary

In Table 4.13 we finally report the best method for each group of instances from the three instance sets. For this purpose, also variations including primal heuristic and using cycle-elimination cut separation are considered. In the case a variant including a primal heuristic yields the best performance, we additionally report the best method not using primal heuristics. Formulations EC_{sn} and EC_{snh} are the best formulations for almost all instances of Set-I, with the primal heuristic often yielding small improvements. The same is true for the instances of Set-II, where almost all variations of formulation EC are able to solve the considered instances in less than a second. For Set-III formulation DCut is superior for many instances with $|L| = 3/4 \cdot |E|$, whereas EC is better for instances with $|L| = 1/4 \cdot |E|$. In constrast to Set-I it is beneficial to restrict the number of edges to $|V| - 1$ as indicated with index "t". Additionally separating cycle-elimination cuts frequently yields the overall best method, in particular for instances with $|L| = 3/4 \cdot |E|$. Furthermore it can be observed that variants using separation of odd-hole inequalities are frequently the overall best methods for this group.

4.3.4 Comparison to Other Work

In this section we present direct comparisons to existing work, in particular [13]. Table 4.14 shows the results presented in [13], running times have been rounded to integers. Formulation "MLSTb" corresponds to formulation SCF of this work. Formulation "MLSTc" only uses a weaker coupling of labels to edges, given by the following inequalities

$$\sum_{(i,j)\in A} x_{ij} \leq \min\{|V| - 1, A(l)\} z_l, \quad \text{for all } l \in L. \tag{4.28}$$

Table 4.14 furthermore reports results for the implementation of the exact backtracking method from [16], labelled with "MLST-CL". Table 4.15 shows the running times of selected MIP variants in comparison to our reimplementation of the flow formulation "MLSTb" from [13] (SCF). Formulation EC_{tn} is clearly superior to the others, all instances have been solved in less than one second. Higher running times of SCF as opposed to "MLSTb" can be explained due to the fact that the SCIP framework [101] has been used for the implementation of SCF, whereas "MLSTb" has been implemented with the ILOG CONCERT framework [61].

Table 4.16 shows the results of selected MIP variants in comparison to the exact A^* backtracking-search procedure used in [29]. The A^*-algorithm is very effective for instances with small optimal objective value, but instances with larger objective values or large sets of labels cannot be solved. The time limit imposed by the authors of [29] was three hours. It is important to note that the running times listed in Table 4.16 are not directly comparable, as the authors of [29] list the computation time at which the best solution was obtained, and also different hardware has been used. For some groups, where A^* could not solve all instance (indicated by "NF"), the MIP method was able to do so. Furthermore, it is reported if the MIP method could solve some but not all instances within some group. In any case the average objective value for the ten instances of each group is reported

in column "avg($|L_T|$)", also considering the best feasible solutions that have been found within the time limit of two hours. If not all instances have been solved to optimality, this is indicated with "(*)" in this particular columns.

In general it can be observed that relatively small instances could be solved efficiently by the MIP approach, but for larger instances with $|V| = 400$ and $|V| = 500$ it generally fails to produce provable optimal solutions within the allowed time limit.

4.3.5 Summary

For all formulations the node-label-constraints (4.16) significantly improved running-times and reduced the number of branch-and-bound nodes. Despite its relatively poor LP-relaxation, formulation $\mathrm{EC}_{\mathrm{tn}}$ turned out to be superior to the other ones for a broad class of test instances, which is mainly to the fast cut separation and the low number of involved variables. Amongst the other considered formulations $\mathrm{DCut}_{\mathrm{tn}}$ is superior over $\mathrm{EC}_{\mathrm{tn}}$ for dense graphs with a huge number of labels.

The odd-hole cuts (4.19) significantly improved running-times and number of branch-and-bound nodes for some classes of instances, in particular when branching-rules are deduced from the label sets corresponding to the found odd-holes. Using BCP for dynamically adding new labels during the solution process turned out to be only beneficial in the case where the input instances significantly deviate from random label assignments, i.e. where the optimal solution is much lower than the expectation value of randomly assigned labels. However, such solutions may likely easily be found also by heuristic methods. Nevertheless, this could remain the only way to prove optimality for "easy" large-scale instances and is also of importance for the data-compression application presented in Chapter 5.

4.4 Conclusions

In this chapter we presented a branch-and-cut(-and-price) framework for solving MLST instances exactly. We gave a comparison of an underlying flow-formulation in comparison to the (better) directed cut-based formulations. Furthermore, a new connectivity formulation permitting a fast cutting-plane separation has been presented. We further presented the application of odd-hole inequalities to this problem for the first time. To separate cutting-planes based on these odd-hole inequalities, a separation heuristic based on a mixed integer program using Miller-Tucker-Zemlin inequalities has been proposed.

Moreover, a detailed comparison of the contribution of the presented algorithmic building blocks has been presented. Our results show that the presented framework is able to solve small to medium sized instances to optimality within a relatively short amount of time. Existing benchmark instances could be solved within a significantly shorter computation time than before and new (larger) instances could be solved to proven optimality for the first time.

Table 4.8: Comparison of formulations EC_t and $DCut_t$ with Inequalities (4.16), indicated with index n, and with additional Inequalities (4.17), indicated with index $\tilde{n}$.

	$EC_{tn}/DCut_{tn}$						$EC_{t\tilde{n}}/DCut_{t\tilde{n}}$					
$\|V\|, \|E\|, a, \|L\|$	cnt	opt	obj	t	bbn	cuts	cnt	opt	obj	t	bbn	cuts
$100, 247, 1, 61$	10	10	19.6	14	4944	4486	**10**	**10**	**19.6**	**0**	**499**	**492**
	10	**10**	**19.6**	**12**	**1148**	**759**	10	10	19.6	19	1577	1034
$100, 247, 1, 185$	10	2	51.2	5760	163875	182801	**10**	**7**	**50.0**	**2174**	**65781**	**68723**
	10	**6**	**49.8**	**3195**	**87152**	**60318**	10	5	49.8	4284	86769	62183
$100, 900, 1, 247$	10	10	14.8	344	36386	16745	**10**	**10**	**14.8**	**177**	**18352**	**6741**
	10	**10**	**14.8**	**835**	**13677**	**8698**	10	10	14.8	1949	26973	16494
$100, 900, 1, 742$	10	2	37.1	6450	120465	121687	**10**	**7**	**35.8**	**2403**	**48038**	**45398**
	10	7	35.8	2432	36099	27852	**10**	**8**	**35.7**	**1822**	**20522**	**15153**
$100, 2475, 1, 618$	**10**	**8**	**13.2**	**1400**	**34106**	**18769**	9	8	13.2	1801	46469	22363
	10	**5**	**13.5**	**4557**	**9195**	**8543**	10	4	13.6	5417	10773	9933
$100, 2475, 1, 1856$	10	4	31.0	5038	59605	57375	**10**	**7**	**30.3**	**2506**	**27241**	**24957**
	10	7	30.2	3552	12337	9780	**10**	**8**	**30.2**	**2907**	**3715**	**3689**
$100, 247, 2, 61$	10	10	16.6	12	4707	4080	**10**	**10**	**16.6**	**1**	**601**	**498**
	10	**10**	**16.6**	**16**	**1162**	**787**	10	10	16.6	21	1348	907
$100, 247, 2, 185$	10	7	35.0	2375	102179	93532	**10**	**10**	**34.7**	**9**	**3269**	**3274**
	10	10	34.7	61	4661	3696	**10**	**10**	**34.7**	**19**	**1425**	**1226**
$100, 900, 2, 247$	**10**	**10**	**11.9**	**629**	**42052**	**20637**	10	10	11.9	912	55106	20864
	10	**10**	**11.9**	**681**	**5906**	**5180**	10	10	11.9	1523	16435	12011
$100, 900, 2, 742$	10	3	26.3	5242	119550	107358	**10**	**7**	**25.8**	**3265**	**69685**	**53785**
	10	**9**	**25.6**	**1489**	**24931**	**17663**	10	9	25.6	1583	17076	12240
$100, 2475, 2, 618$	**10**	**10**	**10.9**	**506**	**22467**	**6432**	10	10	10.9	558	37153	3921
	10	**5**	**11.2**	**5664**	**11813**	**9786**	10	1	11.7	7050	12503	11444
$100, 2475, 2, 1856$	**10**	**4**	**23.2**	**5213**	**61908**	**53294**	10	3	23.2	5757	73145	57657
	10	5	22.8	4259	8850	8481	**10**	**8**	**22.5**	**3649**	**4185**	**4254**
$100, 247, 5, 61$	**10**	**10**	**10.5**	**0**	**306**	**359**	**10**	**10**	**10.5**	**0**	**248**	**306**
	10	**10**	**10.5**	**5**	**115**	**134**	10	10	10.5	11	316	321
$100, 247, 5, 185$	10	6	20.6	3467	143690	125295	**10**	**9**	**20.5**	**1202**	**60571**	**49983**
	10	10	20.5	698	45870	25073	**10**	**10**	**20.5**	**498**	**36774**	**20977**
$100, 900, 5, 247$	**10**	**10**	**7.8**	**128**	**12441**	**651**	10	10	7.8	288	39324	825
	10	**10**	**7.8**	**1628**	**15222**	**9552**	10	5	7.8	5675	66499	42125
$100, 900, 5, 742$	10	3	15.1	5140	139983	90176	**9**	**4**	**15.0**	**4344**	**155589**	**73198**
	10	**6**	**14.8**	**4406**	**44628**	**32312**	10	4	14.9	5171	50434	38692
$100, 2475, 5, 618$	**10**	**10**	**6.9**	**255**	**6604**	**532**	10	10	6.9	624	27936	785
	10	**6**	**7.1**	**5089**	**5318**	**3529**	10	0	7.4	7200	8303	6431
$100, 2475, 5, 1856$	**10**	**6**	**13.0**	**3472**	**72582**	**38165**	10	6	13.0	3848	103635	41651
	10	**4**	**13.1**	**5743**	**7934**	**7389**	10	1	13.6	7191	8419	8617
$10 \times 10, 360, 1, 30$	**10**	**10**	**9.2**	**4**	**1639**	**1544**	10	10	9.2	6	2641	1843
	10	**10**	**9.2**	**34**	**1892**	**1710**	10	10	9.2	90	6196	4135
$10 \times 10, 360, 1, 50$	**9**	**8**	**13.0**	**1578**	**83996**	**67834**	9	8	13.0	1877	102421	75692
	9	**9**	**12.9**	**302**	**20596**	**12907**	9	9	12.9	1034	74718	42088
$10 \times 10, 360, 1, 80$	**10**	**0**	**19.6**	**7200**	**229619**	**202787**	10	0	19.9	7200	251957	226941
	10	**0**	**18.7**	**7200**	**283625**	**220940**	9	0	18.8	7200	283288	241399
$20 \times 20, 1520, 1, 30$	**10**	**10**	**11.5**	**91**	**3757**	**4231**	10	10	11.5	176	7547	5819
	10	**10**	**11.5**	**627**	**4058**	**10961**	10	10	11.5	2866	22956	19238
$20 \times 20, 1520, 1, 50$	**10**	**10**	**17.0**	**3834**	**212265**	**107170**	10	1	17.0	7194	326051	178058
	10	**0**	**17.0**	**7200**	**86172**	**64236**	10	0	17.2	7200	34742	35779
$20 \times 20, 1520, 1, 80$	**10**	**0**	**24.8**	**7200**	**146616**	**112030**	10	0	24.9	7200	150529	110120
	10	0	25.2	7200	43604	44755	**10**	**0**	**20.6**	**7200**	**25718**	**26997**

Table 4.9: Comparison of various formulations based on cycle elimination, i.e. the Miller-Tucker-Zemlin formulation MTZ and the CEF on the instances from SET-III with $|V| = 100$, $a = 1$. Furthermore results for connectivity-based formulations (EC and DCut), enhanced by cycle elimination inequalities are reported.

		$\lvert L\rvert = 1/4 \cdot \lvert E\rvert$							$\lvert L\rvert = 3/4 \cdot \lvert E\rvert$						
d	alg	cnt	opt	obj	t	bbn	cuts	cec	cnt	opt	obj	t	bbn	cuts	cec
	MTZ_{tn}	10	5	19.7	3931	502063	-1	-1	10	7	49.9	3112	721279	-1	-1
0.05	CEF_{tn}	10	6	19.6	3000	135407	-1	16638	10	10	49.8	901	94205	-1	6389
	$CEF_{t\tilde{n}}$	10	5	19.6	3998	155632	-1	17642	10	7	49.9	2208	208113	-1	14321
	EC_{nc}	10	10	19.6	14	1353	121	59	10	0	50.3	7200	532836	170798	1783
	EC_{tnc}	10	10	19.6	12	915	153	96	10	7	49.8	2566	62656	44607	5502
	$DCut_{nc}$	10	10	19.6	13	1433	931	55	10	0	50.4	7200	376649	351288	1630
	$DCut_{tnc}$	10	10	19.6	13	1029	700	94	10	7	49.8	2291	36745	27748	2859
	MTZ_{tn}	10	7	15.0	4276	45276	-1	-1	10	5	35.8	4272	87003	-1	-1
0.2	CEF_{tn}	10	7	14.9	3217	36913	-1	3298	10	7	35.7	2313	91215	-1	5422
	$CEF_{t\tilde{n}}$	10	3	15.2	5307	61399	-1	5690	10	7	35.7	2668	40566	-1	2478
	EC_{nc}	10	0	15.6	7200	31835	143	118	10	0	37.8	7200	59337	1533	30
	EC_{tnc}	10	10	14.8	701	10687	196	670	10	8	35.7	1871	51079	15544	3426
	$DCut_{nc}$	10	0	15.9	7200	39225	29755	171	10	0	39.4	7200	47294	39206	3
	$DCut_{tnc}$	10	10	14.8	737	11214	7212	581	10	8	35.7	1537	21721	13795	1400
	MTZ_{tn}	10	5	13.5	5555	7818	-1	-1	10	3	30.9	5658	13851	-1	-1
0.5	CEF_{tn}	10	5	13.6	5038	8686	-1	763	10	7	30.1	4444	19156	-1	1653
	$CEF_{t\tilde{n}}$	10	3	13.6	5791	8399	-1	1063	10	5	30.5	5570	6646	-1	711
	EC_{nc}	10	0	14.1	7200	3463	26	35	10	0	32.7	7200	6497	118	25
	EC_{tnc}	10	7	13.5	3865	8772	116	913	10	9	30.1	2112	9120	1344	665
	$DCut_{nc}$	10	0	15.6	7200	5353	3395	24	10	0	38.2	7200	6964	5357	6
	$DCut_{tnc}$	10	8	13.5	3394	7452	6433	675	10	9	30.0	2427	7475	5918	576

Table 4.10: Comparison of best formulations used without and with primal heuristics, i.e. MVCA and ACO.

		$\lvert L\rvert = 1/4 \cdot \lvert E\rvert$						$\lvert L\rvert = 3/4 \cdot \lvert E\rvert$					
d	alg	cnt	opt	obj	t	bbn	cuts	cnt	opt	obj	t	bbn	cuts
0.05	EC_{tn}	10	10	19.6	14	4944	4486	10	2	51.2	5760	163875	182801
	EC_{sn}	10	10	19.6	1	1313	966	10	0	55.5	7200	133833	151889
	$DCut_{tn}$	10	10	19.6	12	1148	759	10	6	49.8	3195	87152	60318
	$DCut_{sn}$	10	10	19.6	10	1336	895	10	6	49.8	3195	87152	60318
	EC_{tn} + MVCA	10	10	19.6	12	5474	4866	10	2	51.3	5771	145352	166307
	EC_{sn} + MVCA	10	10	19.6	2	1635	1255	10	0	52.3	7200	137776	154933
	$DCut_{tn}$ + MVCA	10	10	19.6	11	1056	714	10	6	49.8	2902	57282	41009
	$DCut_{sn}$ + MVCA	10	10	19.6	12	1530	993	10	0	50.4	7200	371927	343330
	EC_{tn} + ACO	10	10	19.6	12	4425	3916	8	4	49.6	3600	107155	110775
	EC_{sn} + ACO	10	10	19.6	2	1339	978	10	0	49.9	7200	134930	151941
	$DCut_{tn}$ + ACO	10	10	19.6	10	937	627	9	4	49.8	4019	74506	52846
	$DCut_{sn}$ + ACO	10	10	19.6	11	1303	828	10	0	50.4	7200	371927	343330
0.2	EC_{tn}	10	10	14.8	344	36386	16745	10	2	37.1	6450	120465	121687
	EC_{sn}	10	4	14.8	4894	231148	95062	10	0	39.2	7200	65167	63256
	$DCut_{tn}$	10	10	14.8	835	13677	8698	10	7	35.8	2432	36099	27852
	$DCut_{sn}$	10	10	14.8	835	13677	8698	10	0	38.0	7200	88645	73235
	EC_{tn} + MVCA	10	10	14.8	767	52496	28531	10	2	36.9	6004	120042	120862
	EC_{sn} + MVCA	10	4	14.8	4904	212222	89873	10	0	38.7	7200	63308	60554
	$DCut_{tn}$ + MVCA	10	10	14.8	799	12503	8284	10	8	35.7	2169	34219	25835
	$DCut_{sn}$ + MVCA	10	1	15.5	7067	72658	54894	10	0	38.1	7200	82575	67712
	EC_{tn} + ACO	10	10	14.8	345	31157	14313	9	5	36.0	4135	67362	66292
	EC_{sn} + ACO	10	3	14.9	5323	228273	94410	10	0	36.1	7200	64340	60405
	$DCut_{tn}$ + ACO	10	10	14.8	640	11034	7149	9	6	35.8	2524	35560	26632
	$DCut_{sn}$ + ACO	10	2	15.0	6977	80725	54925	10	0	36.2	7200	83553	69016
0.5	EC_{tn}	9	8	13.2	1400	34106	18769	10	4	31.0	5038	59605	57375
	EC_{sn}	10	0	13.7	7200	296782	88962	10	0	33.8	7200	63430	64801
	$DCut_{tn}$	10	5	13.5	4557	9195	8543	10	7	30.2	3552	12337	9780
	$DCut_{sn}$	10	0	14.4	7200	12834	8154	10	7	30.2	3552	12337	9780
	EC_{tn} + MVCA	10	8	13.3	1991	40440	23995	10	5	30.5	4274	53202	50948
	EC_{sn} + MVCA	10	0	13.7	7200	279895	79633	10	0	32.4	7200	67026	66888
	$DCut_{tn}$ + MVCA	10	7	13.3	3321	6462	5852	10	9	30.0	2728	8479	7236
	$DCut_{sn}$ + MVCA	10	0	14.3	7200	11154	7224	10	0	32.2	7200	14878	10082
	EC_{tn} + ACO	9	7	13.3	2261	44962	25718	6	2	31.3	5419	61481	59297
	EC_{sn} + ACO	10	0	13.7	7200	345358	80054	10	0	31.1	7200	64469	63099
	$DCut_{tn}$ + ACO	10	7	13.4	3534	7659	6933	8	5	30.5	3374	10577	8304
	$DCut_{sn}$ + ACO	10	0	13.8	7200	12035	7319	10	0	31.2	7200	13919	9175

Table 4.11: Comparison of formulations EC_t and $DCut_t$ with and without using odd-hole inqualities. Index o denotes if odd-hole inequalities are separated, index b indicates that odd-hole inequalities have been used to induce branching over related label variables.

	$EC_{tn}/DCut_{tn}$						$EC_{tno}/DCut_{tno}$						$EC_{tnob}/DCut_{tnob}$					
$\|V\|, \|E\|, a, \|L\|$	cnt	opt	obj	t	bbn	cuts	opt	obj	t	bbn	cuts	oh	opt	obj	t	bbn	cuts	oh
100, 247, 1, 61	10	**10**	**19.6**	**14**	**4944**	**4486**	10	19.6	25	5692	5141	23	10	19.6	17	5226	4701	19
	10	**10**	**19.6**	**12**	**1148**	**759**	10	19.6	13	1163	772	1	10	19.6	13	1121	733	1
100, 247, 1, 185	10	**2**	**51.2**	**5760**	**163875**	**182801**	2	51.2	5760	144151	159964	194	2	51.1	5760	143562	159321	173
	10	**6**	**49.8**	**3195**	**87152**	**60318**	6	49.8	3214	83347	58322	11	6	49.8	3197	85997	60585	13
100, 900, 1, 247	10	10	14.8	344	36386	16745	10	14.8	417	37993	18284	134	**10**	**14.8**	**343**	**34639**	**15817**	**120**
	10	**10**	**14.8**	**835**	**13677**	**8698**	10	14.8	892	14659	9364	1	10	14.8	987	15816	10131	2
100, 900, 1, 742	10	2	37.1	6450	120465	121687	**3**	**36.8**	**6377**	**126819**	**128924**	**1251**	2	36.7	6007	119218	119667	1308
	10	7	35.8	2432	36099	27852	**7**	**35.8**	**2369**	**34717**	**26348**	**77**	7	35.8	2371	33757	26083	61
100, 2475, 1, 618	10	**8**	**13.2**	**1400**	**34106**	**18769**	8	13.3	2499	48599	28883	174	8	13.3	1820	35479	21775	138
	10	5	13.5	4557	9195	8543	**7**	**13.5**	**4261**	**8305**	**7615**	**9**	7	13.3	4318	8322	7769	8
100, 2475, 1, 1856	10	4	31.0	5038	59605	57375	**5**	**30.8**	**4133**	**53079**	**52191**	**709**	4	31.0	4499	54945	54254	614
	10	7	30.2	3552	12337	9780	9	30.0	2827	8780	6809	51	**9**	**30.0**	**2778**	**8061**	**6898**	**39**
100, 247, 2, 61	10	**10**	**16.6**	**12**	**4707**	**4080**	10	16.6	14	5148	4496	23	10	16.6	13	4987	4363	20
	10	**10**	**16.6**	**16**	**1162**	**787**	10	16.6	17	1161	788	0	10	16.6	16	1146	775	1
100, 247, 2, 185	10	**7**	**35.0**	**2375**	**102179**	**93532**	6	35.1	3485	137028	125999	272	7	35.0	2431	102595	90520	178
	10	10	34.7	61	4661	3696	10	34.7	69	5073	4081	8	**10**	**34.7**	**22**	**1758**	**1630**	**4**
100, 900, 2, 247	10	**10**	**11.9**	**629**	**42052**	**20637**	9	12.0	997	48954	22866	137	10	11.9	857	41933	21903	129
	10	**10**	**11.9**	**681**	**5906**	**5180**	10	11.9	891	8242	6868	6	10	11.9	1000	8936	7734	11
100, 900, 2, 742	10	3	26.3	5242	119550	107358	3	26.2	5395	118483	104341	1312	**4**	**26.3**	**4841**	**107072**	**94593**	**1220**
	10	9	25.6	1489	24931	17663	9	25.6	1603	24112	18031	82	**9**	**25.6**	**1443**	**23837**	**16801**	**67**
100, 2475, 2, 618	10	10	10.9	506	22467	6432	10	10.9	748	28917	9107	44	**10**	**10.9**	**388**	**17770**	**3684**	**14**
	10	5	11.2	5664	11813	9786	5	11.2	5736	10958	9467	7	**6**	**11.1**	**5727**	**11155**	**9316**	**8**
100, 2475, 2, 1856	10	**4**	**23.2**	**5213**	**61908**	**53294**	1	23.8	6490	74525	64822	822	2	23.5	6298	72256	62059	754
	10	5	22.8	4259	8850	8481	7	22.6	3642	7480	7073	22	**8**	**22.5**	**3479**	**7831**	**7173**	**20**
100, 247, 5, 61	10	**10**	**10.5**	**0**	**306**	**359**	10	10.5	0	306	359	1	10	10.5	1	267	338	1
	10	**10**	**10.5**	**5**	**115**	**134**	10	10.5	5	115	134	0	10	10.5	5	115	134	0
100, 247, 5, 185	10	6	20.6	3467	143690	125295	4	20.7	4461	157779	137051	937	**6**	**20.6**	**3047**	**116829**	**103068**	**673**
	10	**10**	**20.5**	**698**	**45870**	**25073**	9	20.5	787	43883	24921	12	9	20.5	783	45144	25575	10
100, 900, 5, 247	10	**10**	**7.8**	**128**	**12441**	**651**	10	7.8	134	12521	631	2	10	7.8	157	13710	862	3
	10	10	7.8	1628	15222	9552	**10**	**7.8**	**1513**	**15071**	**9222**	**1**	10	7.8	1540	15153	9365	0
100, 900, 5, 742	10	3	15.1	5140	139983	90176	4	15.0	4838	132756	83094	702	**4**	**15.0**	**4458**	**125665**	**78327**	**651**
	10	**6**	**14.8**	**4406**	**44628**	**32312**	5	14.9	4520	45818	33172	22	5	14.9	4392	44435	32267	18
100, 2475, 5, 618	10	**10**	**6.9**	**255**	**6604**	**532**	10	6.9	253	6581	519	0	10	6.9	262	6701	577	0
	10	**6**	**7.1**	**5089**	**5318**	**3529**	6	7.0	5092	5503	3685	0	6	7.0	5093	5447	3608	0
100, 2475, 5, 1856	10	6	13.0	3472	72582	38165	7	12.9	3132	62473	33275	255	**7**	**12.9**	**2343**	**50106**	**24087**	**158**
	10	**4**	**13.1**	**5743**	**7934**	**7389**	3	13.3	6505	8896	8420	9	3	13.1	6213	8259	7770	8
$10 \times 10, 360, 1, 30$	10	**10**	**9.2**	**4**	**1639**	**1544**	10	9.2	5	1427	1350	10	10	9.2	5	1427	1384	11
	10	10	9.2	34	1892	1710	**10**	**9.2**	**32**	**1685**	**1510**	5	10	9.2	34	1711	1576	6
$10 \times 10, 360, 1, 50$	9	8	13.0	1578	83996	67834	8	13.0	1550	81112	66089	310	**9**	**12.9**	**619**	**54900**	**42488**	**145**
	9	9	12.9	302	20596	12907	9	12.9	387	25142	16180	23	**9**	**12.9**	**298**	**19938**	**12261**	**8**
$10 \times 10, 360, 1, 80$	10	0	19.6	7200	229619	202787	0	19.8	7200	211903	186885	1086	**0**	**19.5**	**7200**	**211561**	**188812**	**1102**
	10	**0**	**18.7**	**7200**	**283625**	**220940**	0	18.8	7200	232626	190432	403	0	19.0	7200	223970	188832	381
$20 \times 20, 1520, 1, 30$	10	**10**	**11.5**	**91**	**3757**	**4231**	10	11.5	107	3749	4310	0	10	11.5	106	3799	4249	0
	10	**10**	**11.5**	**627**	**4058**	**10961**	10	11.5	753	4028	11567	0	10	11.5	705	4112	11261	0
$20 \times 20, 1520, 1, 50$	10	**10**	**17.0**	**3834**	**212265**	**107170**	9	17.0	4232	209923	105755	32	9	17.0	4324	210003	107385	39
	10	**0**	**17.0**	**7200**	**86172**	**64236**	0	17.0	7200	77221	60603	6	0	17.0	7200	80453	59244	4
$20 \times 20, 1520, 1, 80$	10	**0**	**24.8**	**7200**	**146616**	**112030**	0	24.8	7200	133250	102086	626	0	24.9	7200	131641	101479	577
	10	**0**	**25.2**	**7200**	**43604**	**44755**	0	25.2	7200	39262	41704	21	0	25.2	7200	39451	42952	19

Table 4.12: Branch-and-cut-and-price results for a special class of instances containing many labels and isolated optima with a relatively low number of labels.

$\|V\|, \|E\|, a, \|L\|$	method	cnt	opt	obj	t	bbn	cuts	priced
$100, 247, 2, 61$	EC_{tn}	10	10	5.0	0	1	32	-1
	$DCut_{tn}$	10	10	5.0	0	1	7	-1
	EC_{tnp}	10	10	5.0	0	1	64	14
	$DCut_{tnp}$	10	10	5.0	0	1	13	17
$100, 247, 2, 185$	EC_{tn}	10	10	10.0	0	1	1	-1
	$DCut_{tn}$	10	10	10.0	0	1	2	-1
	EC_{tnp}	10	10	10.0	0	1	2	11
	$DCut_{tnp}$	10	10	10.0	0	1	3	7
$100, 900, 2, 247$	EC_{tn}	10	10	5.0	0	1	30	-1
	$DCut_{tn}$	10	10	5.0	1	1	15	-1
	EC_{tnp}	10	10	5.0	0	1	72	29
	$DCut_{tnp}$	10	10	5.0	0	2	19	28
$100, 900, 2, 742$	EC_{tn}	10	10	10.0	0	14	42	-1
	$DCut_{tn}$	10	10	10.0	8	13	25	-1
	EC_{tnp}	10	10	10.0	2	497	328	30
	$DCut_{tnp}$	10	10	10.0	12	32	41	25
$100, 2475, 2, 618$	EC_{tn}	10	10	5.0	1	2	46	-1
	$DCut_{tn}$	10	10	5.0	19	4	15	-1
	EC_{tnp}	10	10	5.0	1	6	51	27
	$DCut_{tnp}$	10	10	5.0	11	4	19	26
$100, 2475, 2, 1856$	EC_{tn}	10	10	10.0	2	15	48	-1
	$DCut_{tn}$	10	10	10.0	40	11	23	-1
	EC_{tnp}	10	10	10.0	10	237	174	24
	$DCut_{tnp}$	10	10	10.0	36	23	26	16
$300, 22425, 2, 1856$	EC_{tn}	10	10	10.0	228	1	273	-1
	$DCut_{tn}$	10	10	10.0	617	1	6	-1
	EC_{tnp}	10	10	10.0	105	1	257	2
	$DCut_{tnp}$	10	10	10.0	459	1	13	2
$300, 35880, 2, 8970$	EC_{tn}	9	6	6.7	3846	1	600	-1
	$DCut_{tn}$	9	8	8.9	4113	1	17	-1
	EC_{tnp}	9	9	10.0	880	1	674	14
	$DCut_{tnp}$	9	9	10.0	1131	1	20	12
$300, 35880, 2, 26910$	EC_{tn}	10	10	10.0	627	1	254	-1
	$DCut_{tn}$	10	10	10.0	2735	1	10	-1
	EC_{tnp}	10	10	10.0	259	1	262	2
	$DCut_{tnp}$	10	10	10.0	1212	1	18	3

Table 4.13: Overview of all test instances from Set-I, Set-II and Set-III and corresponding best formulations.

Set	$\|V\|$	$d/\|E\|$	$\|L\|$	a	Best Formulation
Set-I	100	0.2	50	1	EC_{sn}
			100		EC_{sn}, EC_{snh}
			125		EC_{sn}
		0.5	50		EC_{sn}, EC_{n}, EC_{snh}
			100		EC_{n}, EC_{snh}
			125		EC_{n}, EC_{snh}
		0.8	50		EC_{sn}, EC_{n}
			100		EC_{n}, EC_{snh}
			125		EC_{sn}, EC_{n}, EC_{snh}
	200	0.2	100		EC_{sn}, EC_{snh}
			200		EC_{sn}
			250		EC_{sn}, EC_{snh}
		0.5	100		EC_{sn}
			200		EC_{sn}
			250		EC_{sn}, EC_{snh}
		0.8	100		EC_{sn}
			200		EC_{sn}, EC_{snh}
			250		EC_{sn}, EC_{snh}
Set-II	1000	4000	5		EC_{*} (several variants having same performance)
			10		EC_{*} (several variants having same performance)
			20		EC_{*} (several variants having same performance)
Set-III	100	0.05	$1/4 \cdot \|E\|$		several methods having same performance
			$3/4 \cdot \|E\|$		CEF_{tn}
		0.2	$1/4 \cdot \|E\|$		$EC_{t\tilde{n}}$
			$3/4 \cdot \|E\|$		$DCut_{tnc}$
		0.5	$1/4 \cdot \|E\|$		$EC_{t\tilde{n}o}$
			$3/4 \cdot \|E\|$		EC_{tnc}
		0.05	$1/4 \cdot \|E\|$	2	EC_{*} (several variants having same performance)
			$3/4 \cdot \|E\|$		$EC_{t\tilde{n}ob}$, EC_{tnc}
		0.2	$1/4 \cdot \|E\|$		$EC_{t\tilde{n}ob}$, EC_{tnh}
			$3/4 \cdot \|E\|$		$DCut_{t\tilde{n}obc}$
		0.5	$1/4 \cdot \|E\|$		EC_{tnob}
			$3/4 \cdot \|E\|$		EC_{tnoc}
		0.05	$1/4 \cdot \|E\|$	5	several methods having $t \leq 0$
			$3/4 \cdot \|E\|$		$DCut_{t\tilde{n}c}$
		0.2	$1/4 \cdot \|E\|$		EC_{tn}
			$3/4 \cdot \|E\|$		$DCut_{tnco}$
		0.5	$1/4 \cdot \|E\|$		EC_{tn}, EC_{tnh}
			$3/4 \cdot \|E\|$		EC_{tnob}, EC_{tnh}
	10×10		30	1	EC_{*} (several variants having same performance)
			50		EC_{snob}
			80		$DCut_{s\tilde{n}ob}$ (best relaxation)
	20×20		30		EC_{sn}, EC_{n}
			50		EC_{sn}
			80		$DCut_{t\tilde{n}}$ (best relaxation)

Table 4.14: Running times in seconds reported in [13], rounded to integers.

l	5	10	20	5	10	20	5	10	20
n	20			50			100		
MLSTb	0	0	0	0	0	1	0	1	3
MLSTc	0	0	0	0	0	1	0	1	7
MLST-CL	0	0	0	0	0	0	0	0	1
l	5	10	20	5	10	20	5	10	20
n	200			500			1000		
MLSTb	0	3	15	1	9	136	2	43	621
MLSTc	1	6	34	4	38	371	5	132	1994
MLST-CL	0	0	6	0	0	71	0	0	360
l	5	10	20	5	10	20	5	10	20
n	20			50			-	-	-
MLSTb	0	0	0	10	9	8	-	-	-
MLSTc	0	0	0	6	9	4	-	-	-
MLST-CL	0	0	0	45	0	0	-	-	-

Table 4.15: Running times for instances that have been created according to specification from [13]. The first column lists the method for the corresponding row. In parenthesis the corresponding method from [13] is reported.

l	5	10	20	5	10	20	5	10	20
n	20			50			100		
avg($\|L_T\|$)	2.0	2.5	3.8	2.4	3.3	5.0	3.0	4.1	6.6
SCF (MLSTb)	0	0	0	0	0	0	0	0	19
SCF_{tn}	0	0	0	0	0	0	0	0	1
$DCut_{tn}$	0	0	0	0	0	0	0	0	1
EC_{tn}	0	0	0	0	0	0	0	0	0
EC_{sn}	0	0	0	0	0	0	0	0	0
A* (MLST-CL)	0	0	0	0	0	0	0	0	1
l	5	10	20	5	10	20	5	10	20
n	200			500			1000		
avg($\|L_T\|$)	3.0	5.0	7.9	3.5	5.9	9.9	4.1	6.6	11.3
SCF (MLSTb)	3	3	9	71	29	384	31	96	1303
SCF_{tn}	0	1	6	0	4	19	1	13	51
$DCut_{tn}$	0	0	4	1	3	21	12	13	67
EC_{tn}	0	0	0	0	0	0	0	0	0
EC_{sn}	0	0	0	0	0	0	0	0	0
A* (MLST-CL)	0	0	13	0	0	159	0	0	609
d	0.2	0.5	0.8	0.2	0.5	0.8	-	-	-
n	20			50			-	-	-
avg($\|L_T\|$)	7.1	3.5	2.2	3.0	3.9	7.6	-	-	-
SCF (MLSTb)	0	0	0	23	40	25	-	-	-
SCF_{tn}	0	0	0	3	0	1	-	-	-
$DCut_{tn}$	0	0	0	5	2	2	-	-	-
EC_{n}	0	0	0	0	0	0	-	-	-
EC_{sn}	0	0	0	0	0	0	-	-	-
A* (MLST-CL)	0	0	0	67	0	0	-	-	-

Table 4.16: Comparison to results reported in [29] for the A^*-algorithm. Columns $\text{MLST}^{\text{EC}_n}$ list the average total running times for each group of this particular MIP in seconds, columns A^* list the running times in seconds (rounded to integers) reported in [29], at which the best solution was found.

$\|V\|$	$\|L\|$	d	avg($\|L_T\|$)	A^*	$\text{MLST}^{\text{EC}_{\text{tn}}}$	opt	$\|V\|$	$\|L\|$	d	avg($\|L_T\|$)	A^*	$\text{MLST}^{\text{EC}_{\text{tn}}}$	opt
100	25	0.8	1.8	0	0	10	400	100	0.8	2.0	n/a	60	10
100	25	0.5	2.0	0	0	10	400	100	0.5	2.2	n/a	61	10
100	25	0.2	4.5	0	0	10	400	100	0.2	5.8 (*)	n/a	NF	8
100	50	0.8	2.0	0	0	10	400	200	0.8	3.0	n/a	817	10
100	50	0.5	3.0	0	0	10	400	200	0.5	NA	n/a	NA	NA
100	50	0.2	6.7	10	0	10	400	200	0.2	9.3 (*)	n/a	NF	0
100	100	0.8	3.0	0	2	10	400	400	0.8	-	n/a	NF	0
100	100	0.5	4.7	2	9	10	400	400	0.5	6.2 (*)	n/a	NF	0
100	100	0.2	9.7	NF	6	10	400	400	0.2	14.6 (*)	n/a	NF	0
100	125	0.8	4.0	0	17	10	400	500	0.8	-	n/a	NF	0
100	125	0.5	5.2	180	11	10	400	500	0.5	7.3 (*)	n/a	NF	0
100	125	0.2	11.0	NF	12	10	400	500	0.2	17.1 (*)	n/a	NF	0
200	50	0.8	2.0	0	3	10	500	125	0.8	2.0	0	157	10
200	50	0.5	2.2	0	2	10	500	125	0.5	2.6	0	196	10
200	50	0.2	5.2	5	10	10	500	125	0.2	6.3 (*)	NF	NF	2
200	100	0.8	2.6	0	28	10	500	250	0.8	3.0	5	2192	10
200	100	0.5	3.4	0	19	10	500	250	0.5	4.3 (*)	NF	NF	1
200	100	0.2	7.9	NF	191	10	500	250	0.2	10.3 (*)	NF	NF	0
200	200	0.8	4.0	23	911	10	500	500	0.8	4.8 (*)	NF	NF	0
200	200	0.5	-	NF	NF	9	500	500	0.5	6.9 (*)	NF	NF	0
200	200	0.2	-	NF	NF	7	500	500	0.2	16.4 (*)	NF	NF	0
200	250	0.8	4.0	21	301	10	500	625	0.8	5.1 (*)	NF	NF	0
200	250	0.5	-	NF	NF	9	500	625	0.5	8.4 (*)	NF	NF	0
200	250	0.2	-	NF	NF	3	500	625	0.2	19.0 (*)	NF	NF	0

Chapter 5

Application: Biometric Data Compression

To iterate is human, to recurse is devine.
L. Peter Deutsch

In this chapter we present an application of a particular variant of the minimum label spanning tree (MLST) problem. The objective is to develop a method capable of compressing relatively small sets of spatial points (coordinates). The application background is to compress fingerprint (minutiae) templates to enable their embedding in (e.g. passport) photographs by watermarking techniques as an additional security feature.

A particular variant and extension of the MLST serves as an encoding-model, compression is finally achieved by finding high-quality or optimal solutions for the particular input instances.

This chapter mainly rephrases [24] and finally reviews recent improvements to the there mentioned approach, being topic of two diploma-theses [104, 86] which have been supervised by Günther Raidl and myself. The diploma-thesis [41] is also related to the subject of this chapter, as well as the conference papers [23, 19, 92].

5.1 Introduction

In this work, we describe a new possibility for compressing relatively small unordered data sets. Our particular application background is to encode fingerprint template data by means of watermarking techniques (see [63]) e.g. in images of identification cards as an additional security feature. Since the amount of information that can be stored by means of watermarking is very limited, extraordinary compression mechanisms are required in

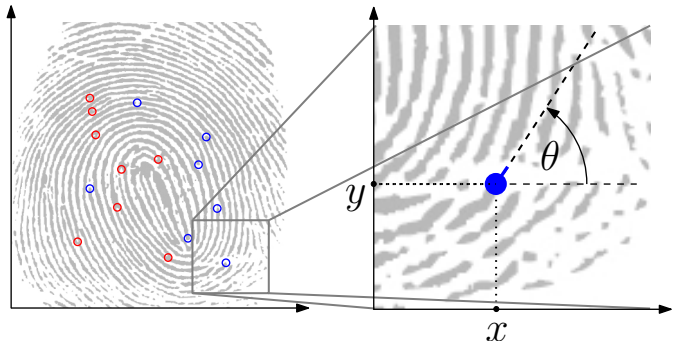

Figure 5.1: Minutiae points of a fingerprint image. The right figure shows the Cartesian coordinates x and y of a specific minutia point, as well as its orientation θ. Together with its type (e.g. ridge ending, bifurcation, etc.) one minutia point can be described as a 4-tuple (x, y, θ, t).

order to achieve reasonably small error rates when finally checking fingerprints against the encoded templates.

Having a scanned fingerprint image, traditional image processing techniques are applied for determining its minutiae, which are points of interest such as bifurcations, crossover, and ridge endings. Fingerprint matching algorithms are usually based on these minutiae data [77]. Typically, 15 to 80 minutiae are extracted from a single fingerprint, and for each we obtain as attributes its type t (e.g. ridge ending, bifurcation, etc.), x and y coordinates, and an angle θ, the tangent to the corresponding ridge line (see Fig. 5.1). A minutia can thus be interpreted as a four-dimensional vector. The task we focus on here is the selection of an arbitrary subset of k minutiae in combination with its lossless encoding in a highly compact way, with k being a prespecified number. The remaining nodes are simply ignored, as it is usually sufficient to perform a reliable matching of a fingerprint to a template by just considering a subset of its minutiae. Obviously k must not be chosen too small in order to preserve reliability of the matchings (see Section 5.10.3). Furthermore, any kind of bias w.r.t. the selection of the k points, as for instance systematically selecting points with low mutual distances, should be avoided.

For this purpose we formulate the problem as a combinatorial optimization problem, in particular a variant of the Minimum Label Spanning Tree (MLST) Problem, which is the main topic of this thesis. By finding optimal or near-optimal solutions to this problem, we can represent the minutiae data by means of a directed tree spanning k nodes, where each edge is encoded by a reference to a small set of template arcs and a small correction vector.

The paper is organized as follows: After a review of related work in Section 5.2 we give a detailed and more formal problem description in Sections 5.3 and 5.4. Section 5.5 describes the preprocessing which actually computes the labels from the input data. Section 5.6 presents a branch-and-cut algorithm to solve the MLST problem variant to optimality. To achieve shorter running times for practical purposes metaheuristics are applied to solve the

problem approximately. A greedy randomized adaptive search procedure and a memetic algorithm are described in detail in Section 5.8. In Section 5.9 we explicitly describe how to encode a solution on a binary level. Finally, we present the results from our computational experiments in Section 5.10, and conclusions are drawn in Section 5.11.

5.2 Previous Work

In general, data compression creates a compact representation of some input data by exhibiting certain structures and redundancies within these data. Various well established techniques exist for diverse fields of applications like text-, image-, and audio-compression.

For instance entropy based methods like Huffman coding or arithmetic coding are well approved in the field of lossless text compression. Alternatively, dictionary coders like the well known LZ77 and LZ78 [116, 117] try to account for (repeating) structures in the text by means of a dictionary. The idea of a dictionary can also be found in other compression techniques specialized for image-, audio- or video-compression. For example, consider the lossy vector quantization method for image compression. Hereby the input data is grouped in blocks of length L, and the respective elements of such a block correspond to the components of a vector of size L. The image is represented by subsequent references to the most similar vector in the codebook, a list of typical and frequently occurring vectors. For a comprehensive review of data compression techniques see [98].

Our approach follows this general idea of using a dictionary or codebook, but its determination as well as its usage is very different from the existing methods. Before going into details we point out the limitations and peculiarities of our approach.

If k is equal to the number of input data points our approach encodes the input data in a lossless way; for lower values of k the method can be considered a special form of lossy compression.

As we are not interested in the respective order of the minutiae, the initial sequence need not to be preserved. In this case a theoretical bound for the encoding length of $O(\log k)$, opposed to $O(k)$ exists [106]. As our encoding as directed k-node spanning tree does not preserve the relative order of the minutiae it can be interpreted as an attempt to benefit from the absence of the requirement to preserve the order.

In [23] the considered compression model and a GRASP algorithm to solve the associated optimization problem heuristically was outlined for the first time. An exact branch-and-cut approach was presented in [19, 92]. The topic has then been extensively presented in [24], which is primary basis of this chapter.

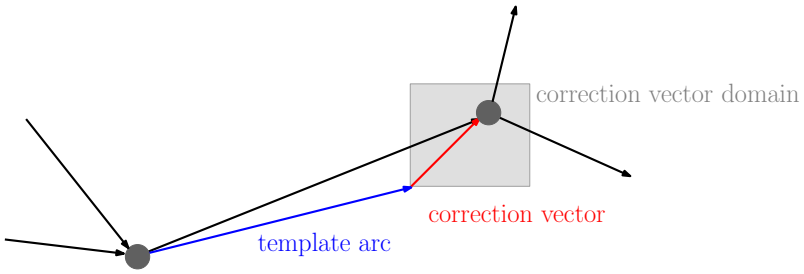

Figure 5.2: Encoding of some tree arc by means of a template arc and a (small) correction vector

5.3 Tree-Based Compression Model

More formally, we consider as raw data n d-dimensional points (vectors) $V = \{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n\}$ from a discrete domain $\mathbb{D} = \{0, \ldots, \tilde{v}^1 - 1\} \times \ldots \times \{0, \ldots, \tilde{v}^d - 1\}$, $\mathbb{D} \subseteq \mathbb{N}^d$ corresponding to our minutiae data ($d = 4$ in the above described application scenario). The domain limits $\tilde{v}^1, \ldots, \tilde{v}^d \in \mathbb{N}$ represent the individual sizes and resolutions of the d dimensions.

Our aim is to select k of these n points and to connect them by an outgoing arborescence, i.e. a directed k-node spanning tree. For this we start with a complete directed graph $G = (V, A)$ with $A = \{(\boldsymbol{u}, \boldsymbol{v}) \mid \boldsymbol{u}, \boldsymbol{v} \in V, \boldsymbol{u} \neq \boldsymbol{v}\}$ on which we search for the optimal arborescence by optimization. Each node in this complete graph corresponds exactly to one of the n points (vectors) and is therefore denoted by the same label $\boldsymbol{v}_i$, $i \in [0, n]$. Consequently, each arc of the arborescence represents the relative geometric position of its end point w.r.t. its starting point.

In addition, we use a small set of specially chosen template arcs. Instead of storing for each tree arc its length in any of the d dimensions, we encode it more tightly by a reference to the most similar template arc and a so called correction vector from a small domain (see Fig. 5.2). Thus, the set of template arcs acts as a codebook (Fig. 5.3). If many arcs of the initial graph have similar geometric properties the proposed encoding of each arc by a reference to a template arc and an additional correction vector is likely to yield a smaller representation (i.e. requires less bits) than the trivial one.

In order to achieve a high compression rate, we optimize the selection of the k encoded points, the tree structure, and the used template arcs simultaneously. The domain for the correction vectors is prespecified, while the number of template arcs is the objective to be minimized. Another possibility would be to prespecify the number of template arcs and minimize the correction vector domain. This approach, however, is not part of this work and could be a topic of further research.

Having solved this optimization problem, we finally store as compressed information the template arc set and the tree. The latter is encoded by traversing it with depth-first search; at each step we write one bit indicating whether a new arc has been traversed to reach a yet unvisited node or backtracking along one arc took place. When following a

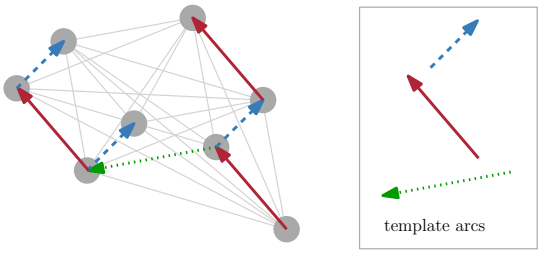

Figure 5.3: Illustration to the encoding of points via a directed spanning tree using a codebook of template arcs; correction vectors are omitted

new arc, a reference to its template arc plus the (small) correction vector are additionally written.

More formally, a solution to our problem consists of

1. a vector of template arcs $T = (\boldsymbol{t}_1, \ldots, \boldsymbol{t}_m) \in \mathbb{D}^m$ of arbitrary size m representing the codebook, and
2. a rooted, outgoing tree $G_T = (V_T, A_T)$ with $V_T \subseteq V$ and $A_T \subseteq A$ connecting precisely $|V_T| = k$ nodes, in which each tree arc $(i, j) \in A_T$ has associated
   - a template arc index $\kappa_{i,j} \in \{1, \ldots, m\}$ and
   - a correction vector $\boldsymbol{\delta}_{i,j} \in \mathbb{D}'$ from a prespecified, small domain $\mathbb{D}' \subseteq \mathbb{D}$ with $\mathbb{D}' = \{0, \ldots, \tilde{\delta}^1 - 1\} \times \ldots \times \{0, \ldots, \tilde{\delta}^d - 1\}$.

For any two points $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ connected by a tree arc $(i, j) \in A_T$ the relation

$$\boldsymbol{v}_j = (\boldsymbol{v}_i + \boldsymbol{t}_{\kappa_{i,j}} + \boldsymbol{\delta}_{i,j}) \bmod \tilde{\boldsymbol{v}}, \quad \forall (i, j) \in A_T, \tag{5.1}$$

must hold; i.e. $\boldsymbol{v}_j$ can be derived from $\boldsymbol{v}_i$ by adding the corresponding template and correction vectors. The modulo-calculation is performed in order to always stay within a finite ring, so there is no need for negative values and we do not have to explicitly consider domain boundaries. Fingerprint minutiae data is usually given by a set of four-dimensional data ($d = 4$). However, the fourth dimension usually consists of only a very few different values. It may thus be beneficial only to consider a smaller number of dimensions for compression, cf. Section 5.10. However, for the sake of a simpler notation we defer this issue to Section 5.9. Here, we assume that all dimensions are considered for compression.

Note that Equation (5.1) ensures, that the k selected points can be reconstructed exactly. As the correction $\boldsymbol{\delta}_{i,j}$ is always choosen in a way that the original arc can be reconstructed exactly, the relative geometric positions of the selected k nodes in the resulting tree precisely corresponds to their relative geometric positions in the initial graph.

Our main objective is now to find a feasible solution with a smallest possible codebook size, i.e. which requires a minimal number m of template arcs.

Regarding the considered application of fingerprint template verification, only parameter k has an impact on the resulting accuracy. This is discussed in more detail in Section 5.10.3.

The encoding of the resulting data is described in detail in Section 5.9. This data can then be directly embedded into images by watermarking techniques, as proposed by [63]. The verification process itself requires the aquisition of the fingerprint of the respective persion in a first step. Given this fingerprint image, minutiae need to be extracted by means of any minutiae extraction algorithm, cf. [77]. This data is then verified against the decoded template. It is important to note, that having just k nodes in the template does not induce any further complications. There are already various imprecisions in the aquisition process, e.g. due to different fingerprint scanners or different positions of the finger on the scanner. Moreover the result of minutiae extraction is highly dependent on the particular image, aquired in the previous step. Hence, matching algorithms are already able to cope with minutiae sets of different cardinality [77].

5.4 Reformulation as a Minimum Label k-Node Subtree Problem

We approach the problem of finding a smallest possible codebook of template arcs together with a feasible tree as follows: First we derive a large set T^c of candidate template arcs (see Section 5.5); then we assign to each arc $(i, j) \in A$ all template arcs $T^c(\boldsymbol{a}_{ij}) \subseteq T^c$ that are able to represent it w.r.t. equation (5.1). Secondly we optimize the codebook by selecting a minimal subset $T \subseteq T^c$ allowing a feasible tree encoding.

The remaining problem in the second part of this approach is related to the Minimum Label Spanning Tree (MLST) Problem. In our problem the candidate template arcs T^c correspond to the labels. Major differences are, however, that we have to consider complete directed graphs, multiple labels may be assigned to an arc, and the labels come up with certain geometric properties.

Although we do not have a proof yet, there are strong hints that the problem remains $\mathcal{NP}$-hard in this version. Consider the situation, where an arborescence is prespecified and its optimal labeling should be found by optimization. Due to the geometric properties of the arcs and the labels this problem is equivalent to the rectangle covering problem, which is known to be $\mathcal{NP}$-complete [59].

Another major extension to the MLST problem is the fact that not all nodes but only an arbitrary subset of size k shall be connected in general. We call the resulting version of the MLST problem k-node Minimum Label Spanning Arborescence (k-MLSA) problem.

5.5 Preprocessing

The preprocessing step is to derive a set of candidate template arcs from which the codebook will be chosen as a subset. This set of candidate template arcs has to be sufficiently large to allow an overall optimal solution, i.e. a minimal codebook.

In the following we will use the terms arc and vector equivalently, as each arc (i, j) in our graph represents the geometric information of the vector $(\boldsymbol{v}_j - \boldsymbol{v}_i) \bmod \tilde{\boldsymbol{v}}$. To describe the preprocessing in more detail we have to introduce further notation:

- $B = \{\boldsymbol{v}_{ij} = (\boldsymbol{v}_j - \boldsymbol{v}_i) \bmod \tilde{\boldsymbol{v}} \mid (i, j) \in A\} = \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{|B|}\}$, the set of different vectors we eventually have to represent.
- $D(\boldsymbol{t}) \subseteq \mathbb{D}$, the subspace of all vectors a particular template arc $\boldsymbol{t} \in \mathbb{D}$ is able to represent when considering the restricted domain $\mathbb{D}'$ for the correction vectors, i.e.

$$D(\boldsymbol{t}) = \{t^1, \ldots, (t^1 + \tilde{\delta}^1 - 1) \bmod \tilde{v}^1\} \times \ldots \times \{t^d, \ldots, (t^d + \tilde{\delta}^d - 1) \bmod \tilde{v}^d\}. \quad (5.2)$$

- $B(\boldsymbol{t}) \subseteq B$, $\boldsymbol{t} \in \mathbb{D}$, the subset of vectors from B that a particular template arc t is able to represent, i.e. $B(\boldsymbol{t}) = \{\boldsymbol{b} \in B \mid \boldsymbol{b} \in D(\boldsymbol{t})\}$.

Furthermore, let $B' \subseteq B$, $B' \neq \emptyset$, be some subset of vectors from B which can be represented by a single template arc. For each dimension $l = 1, \ldots, d$ assume the l-th elements (coordinates) of the vectors in B' are labeled by indices in a non-decreasing way, i.e. $b^l_1 \leq b^l_2 \leq \ldots \leq b^l_{|B'|}$. Let $b^l_0 = b^l_{|B'|} - \tilde{v}^l$ for convenience. (Note that b^l_0 can be negative.)

For such a B', we define the standard template arc to be a vector $\boldsymbol{\tau}(B')$ being able to represent all vectors $\boldsymbol{b} \in B'$ and the property of having maximal possible coordinate values in each dimension. In most cases it will be sufficient to take the minimum coordinate values b^l_1, $l = 1, \ldots, d$ for this purpose. However it might also occur, that the elements from B' can only be represented with template arcs $\boldsymbol{t}$ having $D(\boldsymbol{t})$ crossing the domain boarder w.r.t. at least one dimension. For this reason we need to define the standard template arc in the following way:

Definition 16 (Standard Template Arc)

$$\boldsymbol{\tau}(B') = (\tau^1(B'), \ldots, \tau^d(B')) \quad (5.3)$$

where

$$\tau^l(B') = b^l_{i^*_l} \text{ with } i^*_l = \text{argmax}_{i=1,\ldots,|B'|} b^l_i - b^l_{i-1} \quad \forall l = 1, \ldots, d. \quad (5.4)$$

Figure 5.4 shows an example of such a standard template arc. For the elements $b_1, \ldots, b_6$ we obtain the ordering $b^1_1, b^1_6, b^1_2, b^1_3, b^1_4, b^1_5$ for the first dimension, and $b^2_2, b^2_5, b^2_6, b^2_4, b^2_1, b^2_3$ for the second dimension.

The subspace $BB(B') = \{b^1_{i^*_1}, \ldots, b^1_{i^*_1 - 1} \bmod \tilde{v}^1\} \times \ldots \times \{b^d_{i^*_d}, \ldots, b^d_{i^*_d - 1} \bmod \tilde{v}^d\}$ is the smallest bounding box including all vectors from B' with respect to the ring structure.

To denote the limits of the bounding box $BB(B')$ in a simpler way, we further define $\hat{\boldsymbol{\tau}}(B') = (b^1_{i^*_1 - 1}, \ldots, b^d_{i^*_d - 1})$, i.e. $\hat{\boldsymbol{\tau}}(B')$ represents the corner point of the bounding box opposite to $\boldsymbol{\tau}(B')$.

These definitions lead to the following lemma.

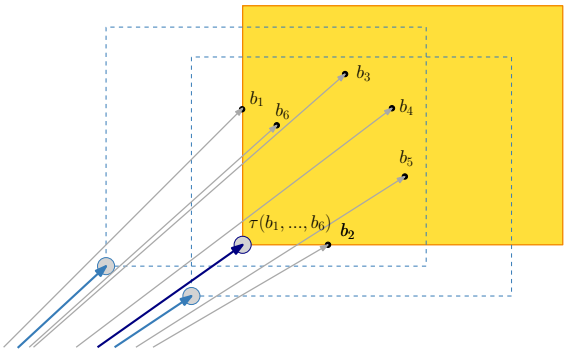

Figure 5.4: The big gray dots are three of the possible representants for the tree arcs $b_1, \ldots, b_6$, but the standard template arc τ is the lower left point of the shaded rectangle. The rectangles depict the $\tilde{\boldsymbol{\delta}}$-domain.

Lemma 1 If a subset $B' \subseteq B$ of vectors can be represented by a single template arc, then the standard template arc $\tau(B')$ always is such a template arc.

Proof This directly follows from the definition of $\boldsymbol{\tau}(B')$, since this is the corner point with the smallest coordinates of the smallest bounding box of all vectors in B'. □

We therefore can restrict all further considerations to the set of standard template arcs induced by all nonempty subsets of vectors that can be represented by a single template arc, i.e.

$$T = \{\tau(B') \mid B' \subseteq B, B' \neq \emptyset \wedge B' \subseteq D(\tau(B'))\}. \tag{5.5}$$

Lemma 2 A set $B' \subseteq B$ can be represented by a single template arc, thus in particular by $\boldsymbol{\tau}(B')$, if

$$\tilde{v}^l - (b^l_{i^*_l} - b^l_{i^*_l - 1}) < \tilde{\delta}^l, \quad \forall l = 1, \ldots, d. \tag{5.6}$$

Proof Case 1: $i^* = 1$. In this case we have $(\tilde{v}^l - (b^l_1 - (v^l_{|B'|} - \tilde{v}^l))) = v^l_{|B'|} - b^l_1 < \tilde{\delta}^l, \forall l = 1, \ldots, d$. Case 2: $i^* > 1$. In this case the bounding box associated to $\boldsymbol{\tau}(B')$ goes across the domain border and the condition is $(\tilde{v}^l - (b^l_{i^*} - v^l_{i^*-1})) < \tilde{\delta}^l,\ \forall l = 1, \ldots, d$. □

Definition 17 (Domination of template arcs) Let $\boldsymbol{t'} = \boldsymbol{\tau}(B')$ and $\boldsymbol{t''} = \boldsymbol{\tau}(B'')$, $B' \subseteq B, B'' \subseteq B$. Standard template arc $\boldsymbol{t'}$ dominates $\boldsymbol{t''}$ if and only if $B'' \subset B'$.

From the set T we only need to keep the non-dominated template arcs for our purpose, and call the resulting set T^c (candidate template arcs).

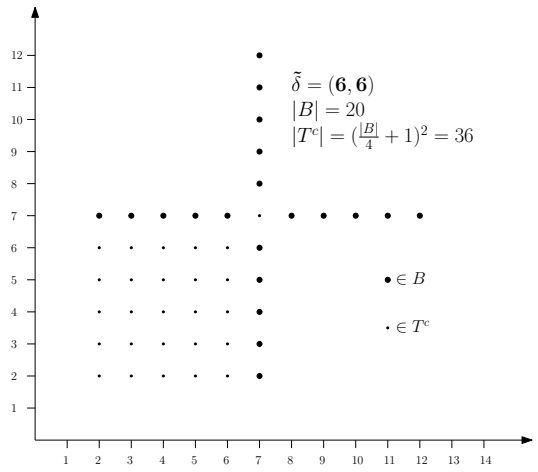

Figure 5.5: Example for $|T^c| = \Theta(|B|^d)$ with $d = 2$.

5.5.1 Bounds for the Number of Candidate Template Arcs

A lower bound on $|T^c|$ obviously is 1: in the best case, one template arc is able to represent all $\boldsymbol{b} \in B$.

An upper bound is given by $O(|B|^d)$: Each standard template arc $t \in T^c$ is composed of d coordinates that are adopted from up to d vectors from B. This bound is tight as the worst-case example in Fig. 5.5 shows for $d = 2$. Bold dots represent the vector set B, small dots the non-dominated standard template arcs T^c. Obviously, $|T^c| = (|B|/4 + 1)^2 = \Theta(|B|^2)$. The example can be extended to higher dimensions d and larger $|B|$ in a straight-forward way. In practice, however, we expect $|T^c| \ll \Theta(|B|^d)$.

5.5.2 An Algorithm for Determining T^c

We determine the set of candidate template arcs T^c by performing a restricted enumeration of all subsets $B' \subseteq B$, $B' \neq \emptyset$ that can be represented by their standard template arc $\boldsymbol{\tau}(B')$. The algorithm maintains three disjoint index sets $C,\ E,\ \Omega \subseteq \{1, \ldots, |B|\}$ that represent at all time a partitioning of B, i.e. $B = B(C) \cup B(E) \cup B(\Omega)$, $C \cap E = \emptyset,\ E \cap \Omega = \emptyset,\ C \cap \Omega = \emptyset$. Hereby $B(S)$ is considered to be (arbitrarily) ordered and denotes the vectors in B referenced by the indices in S. Set C contains the indices of the vectors which are covered by a current bounding box represented by vectors $\boldsymbol{t}$ and $\hat{\boldsymbol{t}}$, set E refers to the vectors that have been actively excluded and must not be covered, and Ω refers to the remaining, still "open" vectors. Table 5.1 summarizes these and a few local data structures.

The candidate template arc determination is started with the procedure `determine-`T^c (Algorithm 11), which performs the initialization of the global data structures and

Table 5.1: Basic data structures of the preprocessing algorithm.

Symbol	Purpose
C	Covered vectors (by current bounding box $(\boldsymbol{t}, \hat{\boldsymbol{t}})$)
E	Actively excluded vectors (must not be covered)
Ω	Open vectors
N	Newly covered vectors
F	Feasibly addable vectors

then calls the recursive procedure `recursive-`T^c`(var` C`,` $\boldsymbol{t}$`,` $\hat{\boldsymbol{t}}$`, var` E`, var` Ω`)` (Algorithm 12). The keyword `var` denotes that the respective variables are passed by call-by-reference. The overall procedure follows the subsequent principle.

1. find further vectors to be added to the current partial solution

2. - there are no further addable vectors $\Rightarrow$ add current vector $\boldsymbol{t}$ to T^c
   - otherwise: recursive calls for all possible extensions of current partial solution

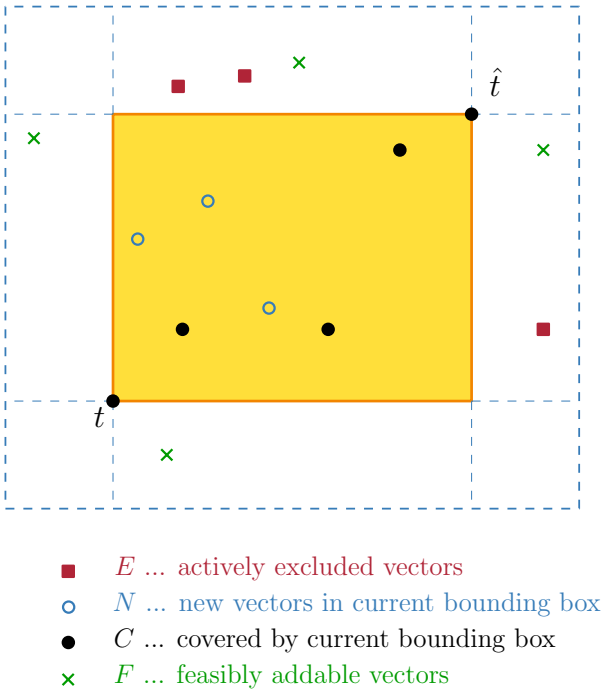

Figure 5.6: Partitioning of the nodes during the execution of Algorithm 12 w.r.t. the current bounding box, defined by $(\boldsymbol{t}, \hat{\boldsymbol{t}})$.

Algorithm 11: `determine-`T^c`()`

1 $T^c \leftarrow \emptyset$; $\Xi \leftarrow \emptyset$; $C \leftarrow \emptyset$; $E \leftarrow \emptyset$; $\Omega \leftarrow \{1, \ldots, |B|\}$
2 recursive-$T^c(C, 0, 0, E, \Omega)$
3 **return** T^c

Vectors $\boldsymbol{t}$ and $\hat{\boldsymbol{t}}$ are assumed to represent the bounding box for the vectors referenced by C. In the first step, a further set N of references to vectors in B is determined, which are now covered by the bounding box, but which are still contained in Ω. These vectors are then directly moved from Ω to S, and no branching will occur on these vectors (actively excluding them would not make sense). Furthermore, the set F of open variables (contained in Ω) which can be feasibly added to C (B'), hereby increasing the current bounding box up to size $\tilde{\boldsymbol{\delta}}$, is determined. Within the loop at line 15, the algorithm considers each element in F and adds it as next element to C. Procedure `update-BB(`$\boldsymbol{t}$`, `$\hat{\boldsymbol{t}}$`, `$\boldsymbol{b_i}$`)` (Algorithm 13) updates the bounding box $(\boldsymbol{t}, \hat{\boldsymbol{t}})$ accordingly. The addition of further vectors is handled via recursion. Before the loop continues with the next vector from F, the current vector is moved from C to E, i.e. it is actively excluded from further consideration and must not be considered in subsequent recursive calls. At the end of the procedure, sets C, E, and Ω are restored to their initial states. Figure 5.5.2 illustrates the partitioning of the nodes during the execution of Algorithm 12 w.r.t. the current bounding box.

When F becomes empty, no further vectors are available for addition and the recursion terminates. We then check if a previously created template arc exists that dominates the current template arc. This is efficiently done by just considering all actively excluded vectors referred to by E. If one of them can be added to C, then C is not maximal and another template arc dominating the current one must have already been previously found.

According to the ring structure of the domain, terms of the form $b \in (l, \ldots, u)$, e.g. in line 8 or 13 in `update-BB(`$\boldsymbol{t}$`, `$\hat{\boldsymbol{t}}$`, `$\boldsymbol{b_i}$`)` (Algorithm 13), have the following meaning: if $\boldsymbol{l} \leq \boldsymbol{r}$ it simply denotes $\{\boldsymbol{x} \mid \boldsymbol{x} \geq \boldsymbol{l}, \boldsymbol{x} \leq \boldsymbol{r}\}$; otherwise the interval goes across the domain border and thus the term denotes the values $\{\boldsymbol{x} \mid \boldsymbol{x} \geq \boldsymbol{0}, \boldsymbol{x} \leq \boldsymbol{l}\} \cup \{\boldsymbol{x} \mid \boldsymbol{x} \geq \boldsymbol{r}, \boldsymbol{x} \leq \tilde{\boldsymbol{v}}\}$, where $\tilde{\boldsymbol{v}}$ again denotes the domain border. To find new vectors, that are now covered by a just extended bounding box $(\boldsymbol{t}, \hat{\boldsymbol{t}})$ we use the procedure `find-new-vectors-in-BB(`$\boldsymbol{t}$`, `$\hat{\boldsymbol{t}}$`, var` Ω`)` (Algorithm 14). This procedure as well as `find-addable-vectors(`$\boldsymbol{t}$`, `$\hat{\boldsymbol{t}}$`, var` Ω`)` (Algorithm 15) run in time $O(|B| \cdot d)$.

Algorithm 12: recursive-T^c(**var** C, $\boldsymbol{t}$, $\hat{\boldsymbol{t}}$, **var** E, **var** Ω)

if $C = \emptyset$ **then**
  $N \leftarrow \emptyset$;
  $F \leftarrow \{1, \ldots, |B|\}$
else
  $N \leftarrow$ find-new-vectors-in-BB($\boldsymbol{t}$, $\hat{\boldsymbol{t}}$,Ω)
  $C \leftarrow C \cup N$; $\Omega \leftarrow \Omega \setminus N$
  $F \leftarrow$ find-addable-vectors($\boldsymbol{t}$, $\hat{\boldsymbol{t}}$, Ω)
end
if $F = \emptyset$ **then**
  /* no further i (referencing vectors b_i) can be added to C; check if C is also maximal with respect to E, the actively excluded vectors */
  if $\nexists j \in E \mid b_j^l \in \{(\hat{t}^l - \tilde{\delta}^l + 1) \bmod \tilde{v}^l, \ldots, (t + \tilde{\delta}^l - 1) \bmod \tilde{v}^l\},\ \forall l = 1, \ldots, d$ **then**
    $T^c \leftarrow T^c \cup \{\boldsymbol{t}\}$
  end
else
  for $i \in F$ **do**
    // Vectors $B(C \cup \{i\})$ can be represented by their $\boldsymbol{\tau}(B(C \cup \{i\}))$
    $C \leftarrow C \cup \{i\}$; $\Omega \leftarrow \Omega \setminus \{i\}$
    $(t', \hat{t}') \leftarrow$ update-BB($\boldsymbol{t}, \hat{\boldsymbol{t}}, \boldsymbol{b_i}$)
    /* only perform further investigation if bounding box has not yet been considered */
    if $\nexists j \in E \mid b_j^l \in \{t^l, \ldots, \hat{t}^l\},\ \forall l = 1, \ldots, d$ **then**
      recursive-T^c(C, t', $\hat{t}'$, E, Ω)
    end
    /* in the next iteration of the loop, vector $\boldsymbol{b_i}$ is actively excluded */
    $C \leftarrow C \setminus \{i\}$
    $E \leftarrow E \cup \{i\}$
  end
end
$C \leftarrow C \setminus N$
$E \leftarrow E \setminus F$
$\Omega \leftarrow \Omega \cup N \cup E$

Algorithm 13: `update-BB(`$\boldsymbol{t}$`, `$\hat{\boldsymbol{t}}$`, `$\boldsymbol{b_i}$`)`

if $|C| = 0$ **then**
  `// `$\boldsymbol{b_i}$` is the first vector in `C
  $\boldsymbol{t'} \leftarrow \boldsymbol{b_i}$, $\hat{\boldsymbol{t}}' \leftarrow \boldsymbol{b_i}$
else
  `// calculate new `$(\boldsymbol{t'}, \hat{\boldsymbol{t}}')$` based on `$(\boldsymbol{t}, \hat{\boldsymbol{t}})$` and `$\boldsymbol{b_i}$
  for $l = 1, \ldots, d$ **do**
    `// we assume `$\forall l = 1, \ldots, d : \tilde{\delta}^l \leq \tilde{v}^l/2$
    if $b_i^l \in \{(\hat{t}^l - \tilde{\delta}^l + 1) \bmod \tilde{v}^l, \ldots, t^l\}$ **then**
      $t'^l \leftarrow b_i^l$
    else
      $t'^l \leftarrow t^l$
    end
    if $b_i^l \in \{\hat{t}^l, \ldots, (t^l + \tilde{\delta}^l - 1) \bmod \tilde{v}^l\}$ **then**
      $\hat{t}'^l \leftarrow b_i^l$
    else
      $\hat{t}'^l \leftarrow \hat{t}^l$
    end
  end
  return $(\boldsymbol{t'}, \hat{\boldsymbol{t}}')$
end

Algorithm 14: `find-new-vectors-in-BB(`$\boldsymbol{t}$`, `$\hat{\boldsymbol{t}}$`, var `Ω`)`

for $j \in \Omega$ **do**
  if $b_j^l \in \{t^l, \ldots, \hat{t}^l\},\ \forall l = 1, \ldots, d$ **then**
    $N \leftarrow N \cup \{j\}$
  end
end
return N

Algorithm 15: `find-addable-vectors(`t`, `$\hat{t}$`, var `Ω`)`

$F \leftarrow \emptyset$
for $j \in \Omega$ **do**
  if $b_j^l \in \{(\hat{t}^l - \tilde{\delta}^l + 1) \bmod \tilde{v}^l, \ldots, (t + \tilde{\delta}^l - 1) \bmod \tilde{v}^l\},\ \forall l = 1, \ldots, d$ **then**
    $F \leftarrow F \cup \{j\}$
  end
  return F
end

Sets C, E, F, and N can efficiently be implemented by using simple arrays and corresponding variables for indicating the number of currently contained elements. In this way,

the restoration of C and E at the end of `recursive-`T^c (Algorithm 12) can even be done by simply memorizing the array sizes at the beginning and finally resetting the counters.

In order to speed up the overall method, we use a k-d tree as data structure (see [8]) for maintaining Ω. In this way geometrical properties can be exploited, and not all vectors in Ω need to be explicitly considered each time.

Theorem 13 The overall time complexity for determine-T^c is bounded above by $O(d \cdot |B|^{3d})$.

Proof Let $\zeta = |B|^d$. As $\boldsymbol{t}, \hat{\boldsymbol{t}} \in T^c$ and $|T^c| = O(\zeta)$ there are $\frac{\zeta!}{2(\zeta-2)!}$ possible bounding boxes $(\boldsymbol{t}, \hat{\boldsymbol{t}})$, and therefore $O(\zeta^2)$ recursive calls in the worst case. As the worst case runtime of the first part of the algorithm is $O(d \cdot |B|)$, we get an overall worst case time complexity of $O(d \cdot |B|^{3d})$. □

Note that the running time $O(d \cdot |B|^{3d})$ to enumerate a set of maximal cardinality $O(|B|^d)$ results from the necessity to remove all dominated elements, as described above. For our application we assume $\tilde{\boldsymbol{\delta}}$ to be relatively small, which implies that $B(\boldsymbol{t})$ will be small as well, i.e. one template arc typically just represents a small number of arcs. Hence, the running times of the procedure are much lower in practice.

5.6 An Exact Branch-and-Cut Algorithm for Solving k-MLSA

In order to solve the k-MLSA problem to optimality, we consider a branch-and-cut algorithm for the following formulation as an integer linear program (ILP).

5.6.1 ILP Formulation

To be able to choose the root node of the arborescence by optimization we extend V to V^+ by adding an artificial root node 0. Further we extend A to A^+ by adding the arcs $(0, i),\ \forall i \in V$. We use the following variables for modeling the problem as an ILP:

- For each candidate template arc $\boldsymbol{t} \in T^c$, we define a variable $y_t \in \{0, 1\}$, indicating whether or not the arc is part of the dictionary T.
- Further we use variables $x_{ij} \in \{0, 1\},\ \forall (i, j) \in A^+$, indicating which arcs belong to the tree.
- To express which nodes are covered by the tree, we introduce variables $z_i \in \{0, 1\},\ \forall i \in V$.

Let further $A(\boldsymbol{t}) \subset A$ denote the set of tree arcs a template arc $\boldsymbol{t} \in T^c$ is able to represent, and let $T(\boldsymbol{a})$ be the set of template arcs that can be used to represent an arc $\boldsymbol{a} \in A$, i.e. $T(\boldsymbol{a}) = \{\boldsymbol{t} \in T^c \mid \boldsymbol{a} \in A(\boldsymbol{t})\}$. We can now model the k-MLSA problem as follows:

$$\text{minimize} \quad m = \sum_{t \in T^c} y_t \tag{5.7a}$$

$$\text{s.t.} \sum_{t \in T(a)} y_t \geq x_a \qquad \forall a \in A \tag{5.7b}$$

$$\sum_{i \in V} z_i = k \tag{5.7c}$$

$$\sum_{a \in A} x_a = k - 1 \tag{5.7d}$$

$$\sum_{i \in V} x_{(0,i)} = 1 \tag{5.7e}$$

$$\sum_{(j,i) \in A^+} x_{ji} = z_i \qquad \forall i \in V \tag{5.7f}$$

$$x_{ij} \leq z_i \qquad \forall (i,j) \in A \tag{5.7g}$$

$$x_{ij} + x_{ji} \leq 1 \qquad \forall (i,j) \in A \tag{5.7h}$$

$$\sum_{a \in C} x_a \leq |C| - 1 \qquad \forall \text{ cycles } C \text{ in } G,\ |C| > 2 \tag{5.7i}$$

$$\sum_{a \in \delta^-(S)} x_a \geq z_i \qquad \forall i \in V,\ \forall S \subseteq V,\ i \in S,\ 0 \notin S \tag{5.7j}$$

Inequalities (5.7b) ensure that for each used tree arc $\boldsymbol{a} \in A$ at least one valid template arc $\boldsymbol{t}$ is selected. Equalities (5.7c) and (5.7d) enforce the required number of nodes and arcs to be selected. Equation (5.7e) requires exactly one arc from the artificial root to one of the tree nodes, which will be the actual root node of the outgoing arborescence.

Equations (5.7f) state that selected nodes must have in-degree 1. Inequalities (5.7g) ensure, that an arc may only be selected if its source node is selected as well. Inequalities (5.7h) forbid cycles of length 2, and finally inequalities (5.7i) forbid all further cycles ($|C| > 2$).

In order to strengthen the ILP we can additionally add (directed) connectivity-constraints, given by inequalities (5.7j), where $\delta^-(S)$ represents the ingoing cut of node set S. These constraints ensure the existence of a path from the root 0 to any node $i \in V$ for which $z_i = 1$, i.e. which is selected for connection. In principle, equations (5.7j) render (5.7f), (5.7g), (5.7h) and (5.7i) redundant [76], but using them jointly may be beneficial in practice.

5.6.2 Branch-and-Cut

As there are exponentially many cycle elimination and connectivity inequalities (5.7i) and (5.7j), directly solving the ILP would be only feasible for very small problem instances. Instead, we apply branch-and-cut [84], i.e. we just start with the constraints (5.7b) to

(5.7h) and add cycle elimination constraints and connectivity constraints only on demand during the optimization process.

The cycle elimination cuts (5.7i) can be easily separated by shortest path computations with Dijkstra's algorithm. Hereby we use $(1 - x_{ij}^{\mathrm{LP}})$ as the arc weights with x_{ij}^{LP} denoting the current value of the LP-relaxation for (i, j) in the current node of the branch-and-bound tree. We obtain cycles by iteratively considering each edge $(i, j) \in A$ and searching for the shortest path from j to i. If the value of a shortest path plus $(1 - x_{ij}^{\mathrm{LP}})$ is less than 1, we have found a cycle for which inequality (5.7i) is violated. We add this inequality to the LP and resolve it. In each node of the branch-and-bound tree we perform these cutting plane separations until no further cuts can be found.

The directed connection inequalities (5.7j) strengthen our formulation. Compared to the cycle elimination cuts they lead to better theoretical bounds, i.e. a tighter characterization of the spanning-tree polyhedron [76], but their separation usually is computationally more expensive. We separate them by computing the maximum flow (and therefore minimum $(0, i)$-cut) from the root node to each of the nodes with $z_i > 0$ as target node. If the value of this cut is less than z_i^{LP}, we have found an inequality that is violated by the current LP-solution. Our separation procedure utilizes Cherkassky and Goldberg's implementation of the push-relabel method for the maximum flow problem [18] to perform the required minimum cut computations.

The branch-and-cut algorithm has been implemented using C++ with CPLEX in version 11.0 [62].

5.7 Branch-and-Cut-and-Price

The exact branch-and-cut algorithm from Section 5.6.2 has two major shortcomings. First, the preprocessing method is relatively time-consuming, and second, the large amount of label-variables yields large LPs to be solved within every node of the branch-and-bound tree. These observations support the idea to develop a branch-and-cut-and-price (BCP) approach, where after starting with a small feasible set of labels, further labels are dynamically added on demand. In the diploma theses of Thöni [104] and Oberlechner [86] two different approaches following this general idea have been developed. After a formal introduction of the pricing problem in Section 5.7.1, the corresponding solution techniques presented in [104] are briefly summarized.

Following the idea of Inequalities (4.16) proposed in Section 4.2, we introduce inequalities

$$\sum_{t \in T(\Gamma^-(v_i))} y_t \geq z_i - x_{ri}, \quad \forall i \in V, \tag{5.8}$$

to provide (besides Inequalities (5.7b)) further information for the pricing step, but also to further strengthen the LP. Inequalities (5.8) state that the for each selected node except the artificial root node, the sum over the template-arc variables associated to the nodes incident (ingoing) arcs, must be greater or equal than one.

5.7.1 Pricing Problem

Let π_a denote the dual variables corresponding to Inequalities (5.7b), and further μ_j denote the dual variables corresponding to Inequalities (5.8). The reduced costs for each template arc t are then given by

$$\bar{c}_t = 1 - \left(\sum_{\boldsymbol{a} \in A(\boldsymbol{t})} \pi_a + \sum_{j \in \{\boldsymbol{v} | (\boldsymbol{u},\boldsymbol{v}) \in A(\boldsymbol{t})\}} \mu_j \right). \tag{5.9}$$

Any template arc with negative reduced costs $\bar{c}_t$ may potentially improve the current objective function value, if there are no such template arcs, the solution cannot be further improved. We define the pricing problem as finding the template arc with maximal negative reduced costs.

Definition 18 (Pricing Problem)

$$\boldsymbol{t}^* = \text{argmin}_{\boldsymbol{t} \in T} \left\{ 1 - \left(\sum_{\boldsymbol{a} \in A(\boldsymbol{t})} \pi_a + \sum_{j \in \{\boldsymbol{v} | (\boldsymbol{u},\boldsymbol{v}) \in A(\boldsymbol{t})\}} \mu_j \right) \right\}. \tag{5.10}$$

5.7.2 Solving the Pricing Problem

A nice geometrical interpretation for the pricing problem arises, when considering the two-dimensional case. Each tree arc corresponds to a point in $\mathbb{D}$ according to its associated geometric information. Furthermore, each point in $\mathbb{D}$ in the same way corresponds to a potential template arc. Hence, we will use the terms tree/template arc and their corresponding points interchangeably within this section. All template arcs potentially representing an arbitrary tree arc $\boldsymbol{b_i}$ must have their endpoint in the rectangle $D(\boldsymbol{b} - \tilde{\boldsymbol{\delta}} + \boldsymbol{\epsilon})$ with b_i corresponding to its upper right corner, and $\boldsymbol{\epsilon}$ denoting the d-dimensional vector with all components being 1. Let $T'(\boldsymbol{b})$ denote this area whose points correspond to the potential template arcs able to represent b. To each $T'(\boldsymbol{b})$ we now associate the value

$$\zeta_b = \sum_{i \in \{\boldsymbol{a} | \boldsymbol{a} \in A \wedge a = b\}} \pi_i + \sum_{j \in \{\boldsymbol{v} | (\boldsymbol{u},\boldsymbol{v}) \in A \wedge (\boldsymbol{u},\boldsymbol{v}) = b\}} \mu_j. \tag{5.11}$$

The first term on the right hand side in Equation (5.11) corresponds to the sum of all dual values associated to the constraints for the tree arcs corresponding to b, given by Inequalities (5.7b). The second term results from the dual values of all nodes according to Constraints (5.8) which are incident to a tree arc corresponding to $\boldsymbol{b}$. We can now imagine these rectangles $T'(\boldsymbol{b})$ as transparently shaded with a gray scale value ζ_b with higher values corresponding to a darker shades. See Fig. 5.7 for an example of two elements $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ and their corresponding regions $T'(\boldsymbol{b}_1)$ and $T'(\boldsymbol{b}_2)$ being drawn in the domain.

Let us now consider the situation of all $\boldsymbol{b} \in B$ and their corresponding $T'(\boldsymbol{b})$, shaded accordingly with ζ_b, being drawn in the area corresponding to the two-dimensional domain

$\mathbb{D}$. Due to the transparency of the rectangles, regions of overlapping rectangles will obtain darker colors. Formally we define for each region R a corresponding value

$$\zeta_R = \sum_{\boldsymbol{b} \in A(\boldsymbol{t}),\ \boldsymbol{t} \in R} \zeta_b \tag{5.12}$$

for some arbitrary $\boldsymbol{t}$ being located in region R. Figure 5.9 shows an example with two overlapping elements $\boldsymbol{b}_i$. We can now see that the pricing problem given by Definition 18 exactly corresponds to finding the darkest such area. This analogy remains valid even in the higher dimensional case if we use regions of corresponding dimensionality instead of areas with dimensionality two. The correspondence of the presented illustration to the pricing problem becomes evident by considering the correspondence of Equation (5.11) to the two sums in Equation (5.10). The only difference is that Equation (5.11) is formulated in terms of unique points $\boldsymbol{b}$ and Equation (5.10) in terms of template arcs $\boldsymbol{t}$, which we actually want to determine.

Based on this observation, we now outline an algorithm to solve the pricing problem. This algorithm was primary subject of a diploma thesis [104], for details according to the implementation of the algorithm, the reader is referred to this work. Here we proceed by describing the basic functionality and principles.

Underlying datastructure is a k-d tree which is used to partition the domain into the corresponding regions resulting from $T'(\boldsymbol{b})$, for all $\boldsymbol{b} \in B$ and resulting overlapping regions. Here k denotes the number of dimensions to be used within the tree, and should not be mixed up with the number of nodes to be connected to the arborescence. However, as the term k-d tree is commonly used for this datastructure, we refrain from referring to it as d-d tree. For convenience, we briefly review the principles of k-d trees. Primary field of application of k-d trees is to act as a search tree for k-dimensional points, being stored in the tree. However, the resulting tree implicitly defines a hierarchical partitioning of the underlying domain. Each node of the binary tree defines a division of the subspace in which it is located into exactly two subspaces. Within each level l coordinate $l \mod k$ is used to define this subdivision. At the root node the whole domain is subdivided according to some coordinate of the first dimension. Each child node then defines a subdivision according to a coordinate of the second dimension, and so forth. For our purpose we define each node to have either two children, or to be a leaf node. In our case, a leaf node may either correspond to a region that cannot contain a template arc, or otherwise, a region that contains all possible template arcs representing a unique subset of elements $\boldsymbol{b}_i \in \mathbb{B}$.

Figures 5.7 and 5.9 show examples of two and three elements being drawn in the domain respectively. These figures furthermore show how the domain is segmented into subregions according to $T'(\boldsymbol{b}_1)$ and $T'(\boldsymbol{b}_2)$ (and $T'(\boldsymbol{b}_7)$ in Figure 5.9). Corresponding k-d trees, which we will from now on call segmentation trees are depicted in Figures 5.8 and 5.10. As each node of the tree subdivides its subspace into two subspaces, it defines a hyperplane, which we will also call splitting-hyperplane. In the two-dimensional examples in Figures 5.7 and 5.9 these hyperplanes correspond to lines, which are depicted in the figures as well. In the example of Figure 5.7 the first split is performed according to the second

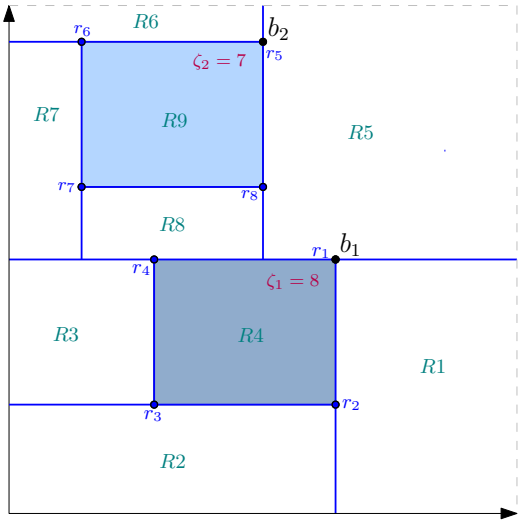

Figure 5.7: Example of two elements $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ and corresponding regions $T'(\boldsymbol{b}_1)$ and $T'(\boldsymbol{b}_2)$ drawn in the domain $\mathbb{D}$. (Image with minor modifications taken from [104])

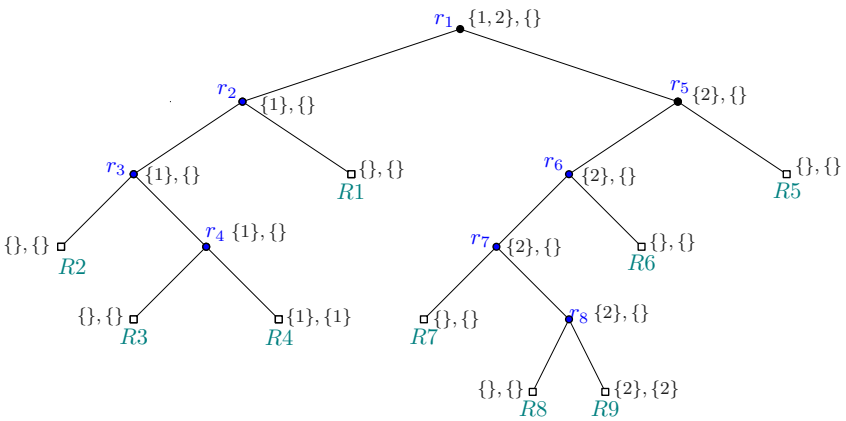

Figure 5.8: Segmentation tree corresponding to the example shown in Figure 5.7. (Image credits: [104])

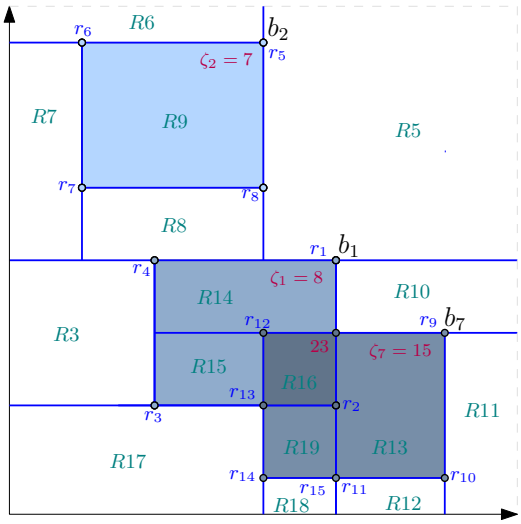

Figure 5.9: This illustration shows the situation after the insertion of a further element ($\boldsymbol{b}_7$) into the segmentation tree shown in Figures 5.7 and 5.8. (Image with minor modifications taken from [104])

coordinate at point $\boldsymbol{r}_1 = \boldsymbol{b}_1$. Nodes $\boldsymbol{r}_i$ denote the coordinates corresponding to each node i of the segmentation tree. The area $T'(\boldsymbol{b}_1)$ is finally defined by nodes $\boldsymbol{r}_2$, $\boldsymbol{r}_3$ and $\boldsymbol{r}_4$, area $T'(\boldsymbol{b}_2)$ by nodes $\boldsymbol{r}_5$, $\boldsymbol{r}_6$, $\boldsymbol{r}_7$ and $\boldsymbol{r}_8$. In the figure all points $\boldsymbol{r}_i$ correspond to the corner points of areas $T'(\boldsymbol{b})$ for all elements $\boldsymbol{b}$ in the tree, which is however an arbitrary decision for a better illustration. In fact, only one coordinate is required to define a hyperplane being orthogonal to the basis vector of the considered dimension, which is always the case in the segmentation tree. Besides the intermediate "splitting" nodes, the tree in Figure 5.8 also contains the leaf nodes, with corresponding regions depicted in Figure 5.7. The second example, given by Figures 5.9 and 5.10 shows the resulting tree after the insertion of element $\boldsymbol{b}_7$. Again, splitting nodes and leafs (corresponding to regions) are contained in the visualization of the tree, as well as in the corresponding illustration of the fragmented domain. To build up the whole tree, regions $T'(\boldsymbol{b})$ for all $\boldsymbol{b} \in B$ are iteratively inserted. For each such $T'(\boldsymbol{b})$ we need to find the correct position for inserting it into the tree. This is done by checking at each tree node $\boldsymbol{r}$ if $T'(\boldsymbol{b})$ is entirely located in one of the subspaces defined by $\boldsymbol{r}$. If a region is entirely contained in a region defined by a current leaf of the tree, this leaf is replaced with an according subtree corresponding to the splitting hyperplanes required to properly define $T'(\boldsymbol{b})$. However, if $T'(\boldsymbol{b})$ is part of both subspaces defined by current node $\boldsymbol{r}$, we need to split up $T'(\boldsymbol{b})$ accordingly, and insert the resulting subregions into both branches of $\boldsymbol{r}$. Having now described, how the segmentation tree can be created, we focus on how the tree can be used to efficiently search for the best template arc.

At this point we assume that the whole segmentation tree has been created in advance. As we will see later, this is not a real requirement. Our goal is to find the region R with maximum ζ_R, which is the solution to the pricing problem. As the pricing problem needs

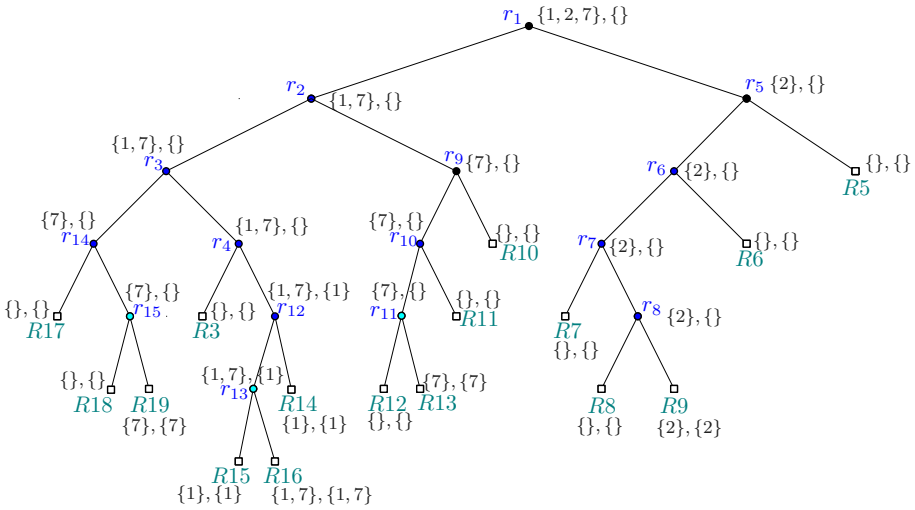

Figure 5.10: Segmentation tree corresponding to the example shown in Figure 5.9. (Image credits: [104])

to be solved many times, the search has to be efficient. In particular we want to avoid to assign ζ_B to all leafs (corresponding to regions) B in the tree according to the values ζ_b derived by the dual values. Therefore the search is based on upper and lower bounds used to prune branches at an early stage. Let $R(\boldsymbol{r})$ denote the subspace corresponding to node $\boldsymbol{r}$ of the segmentation tree. Upper and lower bounds for each node $\boldsymbol{r}$ of the tree can be derived based on the following definitions.

Definition 19 (Upper Bound Set) The upper bound set is given by all elements $\boldsymbol{b} \in B$ which can be represented by some potential template arc in the subspace corresponding to tree node $\boldsymbol{r}$.

$$UB(\boldsymbol{r}) = \{\boldsymbol{b} \in B \mid \exists \boldsymbol{t} \in R(\boldsymbol{r}) \wedge \boldsymbol{b} \in B(\boldsymbol{t})\}$$

Definition 20 (Lower Bound Set) The upper bound set is given by all elements $\boldsymbol{b} \in B$ which can be represented by all potential template arc in the subspace corresponding to tree node $\boldsymbol{r}$.

$$LB(\boldsymbol{r}) = \{\boldsymbol{b} \in B \mid \forall \boldsymbol{t} \in R(\boldsymbol{r}) \wedge \boldsymbol{b} \in B(\boldsymbol{t})\}$$

These bound sets are stored for each node of the search tree. In Figures 5.7 and 5.9 these sets are denoted in braces for each node.

Based on these sets, we can immediately derive numeric bounds, based on the dual values.

Definition 21 (Upper Bound)

$$ub(\boldsymbol{r}) = \sum_{\boldsymbol{b} \in UB(\boldsymbol{r})} \zeta_b$$

Definition 22 (Lower Bound)

$$lb(\boldsymbol{r}) = \max_{\boldsymbol{b} \in UB(\boldsymbol{r})} \zeta_b$$

The search process is performed based on these upper and lower bounds. Starting at the root node, the set B is divided into two not necessarily disjoint sets. These sets $UB(\boldsymbol{r})$ correspond to the nodes which are representable by some template arc of the subspaces introduced by the splitting-hyperplane defined by the current tree node $\boldsymbol{r}$. With $ub(\boldsymbol{r})$ we directly obtain a numeric value being the upper bound for this particular branch. A lower bound is given by $lb(\boldsymbol{r})$, i.e. the element with maximal ζ_b in this branch. For each node we check if $UB(\boldsymbol{r}) = LB(\boldsymbol{r})$ which implies that we have found a leaf node. A global lower bound lb^* is used to prune the search tree, as we do not have to follow branches with $ub(\boldsymbol{r}) < lb^*$. Initialization of the global lower bound can be performed with $lb^* = \max_{\boldsymbol{b} \in B} \zeta_b$. The search strategy to be used is best first search based on the upper bounds $ub(\boldsymbol{r})$.

Within the description of the algorithm, we have omitted many implementation issues. One important aspect to be considered is the fact that regions may cross the domain border. This needs to be checked in advance, and corresponding subregions must be inserted in this case. Furthermore a lot of design issues are involved in order to find the best way to implement the bounding procedure. Also the reconstruction of the coordinate values of the corner points of each region requires to take care of some special cases. For a detailed presentation and analysis of this issues the reader is referred to [104].

A further improvement of the overall process can be achieved, if the entire tree is not completely built in advance, but rather in a dynamic way during the search process. Each time the search is according to the bounds directed toward a certain branch of the tree, we check if this branch has already been created. If this is not the case, it is expanded as needed during the search process. Hence construction and traversing the tree is performed in an intertwined way. This has not only the advantage of the initial construction step to be omitted, but will also result in smaller trees to operate with. As certain regions of the domain will not contain any useful template arcs, corresponding branches are unlikely to be created during the whole BCP solution process, saving space and time.

Corresponding pseudocodes are omitted within this presentation, as they would require a more detailed formal description and notation. In the following Section 5.7.3 we show how this algorithmic framework for solving the pricing problem can be used within a branch-and-cut-and-price approach.

5.7.3 Branch-and-Cut-and-Price Algorithm

The first step of the entire branch-and-cut-and-price (BCP) algorithm is to determine a feasible starting solution. Any connected subgraph of k nodes is sufficient for this purpose. Hence, we determine a starting solution by connecting arbitrary k nodes by a star-shaped spanning tree, assign big values to the dual variables corresponding to this set of arcs, and use the pricing algorithm to determine a feasible starting solution.

The restricted master problem (RMP) is defined according to formulation (5.7) with additional Inequalities (5.8), however the entire set T^c is replaced by T^p denoting the set of label variables that have already been priced in. Within each node of the B&B-tree directed connection cuts and cycle elimination cuts are separated to obtain a feasible LP-relaxation. Afterwards new template arc variables are priced in as long as such variables with negative reduced costs according to Equation (5.10) can be found and no further cutting-planes can be added. It turned out to be advantageous to add all variables with negative reduced costs within each pricing iteration. The development of the BCP algorithm in combination with an alternative solution method for the pricing problem is subject of an ongoing diploma thesis [86]. The BCP results reported in Section 5.10.4 used the pricing solver based on the segmentation tree as described in Section 5.7.2 within this BCP framework.

5.8 Heuristic Methods

Practical results of the described branch-and-cut and branch-and-cut-and-price algorithms are presented in Section 5.10. They show that this approach is only applicable for small instances and requires relatively long running times. Therefore, we now describe a fast greedy construction heuristic and then focus on metaheuristics including a greedy randomized adaptive search procedure (GRASP) and a memetic algorithm (MA).

5.8.1 Greedy Construction Heuristic

Based on the greedy construction heuristic from [69], we developed a greedy algorithm for our k-MLSA problem. A solution is constructed by starting with an empty codebook T and graph $G' = (V', A')$ with $A' = \emptyset, V' = \emptyset$ and iteratively adding template arcs from T^c to T in a greedy fashion. In the following we will treat T as an ordered set, and refer to its elements by $T[i], i = 1, \ldots, |T|$. Each time a template arc $\boldsymbol{t}$ is added to T, all corresponding induced arcs $A(\boldsymbol{t}) \subset A$ are added to A'. For each arc (i, j) we also add the corresponding nodes i and j to V'. This is done until the resulting graph contains a feasible k-node arborescence. In contrast to the classical undirected MLST problem, the decision which template arc (label) to take next is significantly more difficult, as the impact of the addition of one template arc towards a final arborescence (with some specific root node) is not immediately obvious.

In the construction heuristic for MLST, a label that reduces the number of separated components the most is always chosen. The number of components of G' minus one corresponds to the number of edges that must be added at least to obtain a complete spanning tree, and this number of edges is an upper bound for the number of labels to be added.

In any case, a label which directly yields a spanning tree is an optimal choice and should be selected. A label which yields a G' to which only one more edge must be added is always the second best choice, since exactly one more label is necessary. Note that all

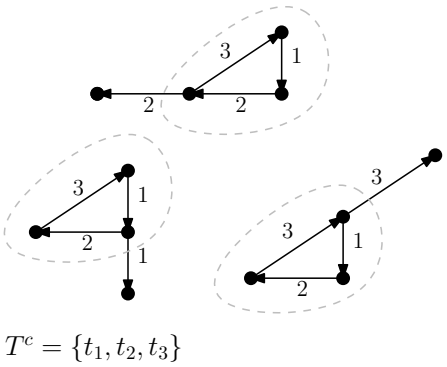

Figure 5.11: Suppose we want to connect all of the 12 nodes in the given graph G, i.e. $k = 12$. After adding the template arcs $\boldsymbol{t}_1$, $\boldsymbol{t}_2$ and $\boldsymbol{t}_3$ we can identify three nontrivial strongly connected components (SCCs), i.e. components consisting of more than one node. All nodes have incoming arcs, but these three SCCs do not. We need to add at least two more arcs as $\alpha(G') = 2$. Hereby G' denotes the graph where each SCC is contracted into one single node.

such situations are equally good. In general, the assumption is that a label yielding a G' to which a lower number of further edges must at least be added will usually lead to a better solution (requiring less labels) than a label yielding a G' having a higher lower bound of edges to be necessarily added.

While the notion of simple components does not make sense in the directed k-MLSA anymore, we can still follow the idea of determining the number of edges (arcs) that must at least be added to obtain a complete arborescence in order to decide upon the next label to be added.

Let $\alpha(G')$ denote the minimum number of arcs that need to be added, so that this augmented graph contains an arborescence. In principle, $\alpha(G')$ can be calculated efficiently as follows: Determine all maximal strongly connected components (SCCs) in G' and shrink them into corresponding representative single nodes. Arcs to or from a node in a strongly connected component are replaced by corresponding arcs to/from the representative node. Multiple arcs between two nodes are replaced by corresponding single arcs, and self-loops are simply deleted. By this transformation, we obtain a directed acyclic graph G_s. The problem is reduced, but the value $\alpha(G')$ will remain the same since within each strongly connected component, any node can be reached from each other and no further edges will therefore be necessary. It further does not matter to which particular node of a strongly connected component an ingoing or outgoing arc is connected. Let $Z \subseteq V$ be the set of nodes for which no ingoing arc exists in G_s. The minimum number of required additional arcs is now $\alpha(G') = k - (|V| - |Z| + 1)$, and the label that minimizes this number the most is considered the best choice. Figure 5.11 shows an example of the computation and usage of $\alpha(G')$. We do not need to explicitly shrink the SCCs each time if we keep track of all SCCs with in-degree zero. Algorithm 16 details the overall procedure.

Algorithm 16: k-MLSA-greedy(V, A, T^c)

```
1  G' = (V', A') with V' ← V, A ← ∅
2  T ← ∅   // currently used labels
3  while no k-arborescence exists do
4      t* ← 0   // best template arc of this iteration
5      for all t ∈ T^c do
6          z* ← ∞   //stores lowest found number of SCCs
7          A'' ← {a_ij ∈ A(t)}
8          compute SCCs of G' = (V', A' ∪ A'')
9          Z ← SCCs with in-degree zero
10         if |Z| < z* then
11             z* ← |Z|
12             t* ← t
13         end
14     end
15     A' ← A' ∪ {a_ij ∈ A(t*)}
16     T ← T ∪ {t*}
17     T^c ← T^c \ {t*}
18 end
19 remove redundant arcs and labels
```

Obviously, the algorithm frequently has to check if a partial solution already contains a feasible arborescence. This task can be achieved by performing depth first search (DFS) using each node as start node (time complexity $O(k^3)$). To achieve a speedup of this method we try to avoid or reduce the number of time consuming DFS calls. Let G' denote the graph containing just the edges and nodes induced by some template arc set T, i.e. if $(i, j) \in A$ is represented by template arc $t \in T$ we add the nodes i, j and the arc (i, j) to $G' = (V', A')$. Let further $\delta^-(v)$ denote the in-degree of a node v, i.e. the number of incoming arcs. Furthermore let $\delta_0^i(V')$ denote the subset of nodes from V' with $\delta^-(V') = 0$, and let us assume that the current partial solution consists of the template arcs (labels) T. First, we check the degree of each node to see if a sufficient number of nodes v with in-degree $\delta^-(v) > 0$ is available. If $|V'| - \delta_0^i(V') + 1 < k$ then G' cannot represent a valid solution, and we do not have to perform the DFS. If a solution is possible we distinguish the following two cases. In the first case, where $k = |V|$, there can be at most one node with in-degree zero. If there is such a node it has to be the root node and we perform the DFS starting from this node. Otherwise, if all nodes $v \in V'$ have $\delta^-(v) > 0$ we have no choice but to perform DFS starting from all nodes. In the more general second case $k < |V|$, if $|V'| - \delta_0^i(V') + 1 = k$, one of the nodes with in-degree zero has to be the root of the tree, otherwise the tree would not contain the required k nodes. So it is sufficient to perform the DFS starting at just these $\delta_0^i(V')$ nodes. Otherwise we again have to perform DFS starting from all nodes.

The final step is to remove redundant tree arcs and redundant template arcs. Because of mutual dependencies of these tasks, this is a non-trivial operation itself and hence we

apply a heuristic. As long as the solution remains valid we perform the following two steps: 1) try to remove redundant labels; 2) as long as $|A'| > k$ try to remove the leaves and intermediate arcs. By this procedure we finally obtain a valid k-node arborescence.

5.8.2 GRASP – Greedy Randomized Adaptive Search Procedure

The greedy heuristic is relatively fast but yields only moderate results. Significantly better solutions can be achieved by extending it to a greedy randomized adaptive search procedure (GRASP) [47]. The constructive heuristic is iterated and the template arc to be added is always selected at random from a restricted set of template arcs, the restricted candidate list (RCL). As soon as a valid solution exists, it is further improved by a local search procedure. In total, *it* iterations of these two steps are performed.

Function k-`MLSA-randomized-greedy`(V, A, T^c) (Algorithm 17) shows the randomized greedy construction of solutions in detail. One crucial part in designing an efficient GRASP is to define a meaningful RCL. The problem in our case is that there are many equally good template arcs that could be candidates to extend the current partial solution. On the other hand, finding the best template arcs, i.e. those template arcs reducing α of the current partial solution the most, can be very time consuming, as all candidate template arcs need to be considered. As GRASP also heavily relies on the subsequent local improvement, we do not necessarily have to find the best candidates to extend our partial solution. On the other hand, being too lazy with this decision might reduce the overall performance considerably. In the following we describe the parametrized procedure of building up the RCL in more detail.

Prior to each extension of the current partial solution (line 26), the RCL is built in the loop in lines 5 to 25. As soon as a further improving template arc is found (line 11) the RCL is cleared, and then successively filled with further template arcs of the same quality. This finally yields a list of equally good template arcs, which are after all the candidates for the next greedy decision. The size of this list is limited by $rcl_{\max}$ for performance reasons. There is one further, even more important parameter related to the issue of balancing the greedy solution quality versus run time efficiency of the process of building the RCL. The parameter $imp_{\max}$ limits the number of improvements according to line 11. In the special case of $imp_{\max} = 0$ the RCL finally simply consists of the first template arcs improving the current solution. In this case the major contribution to construct high quality solutions is passed to the subsequent local search. Setting $imp_{\max} = \infty$ implies a situation where the loop of line 6 iterates over all candidate template arcs each time. Due to the relatively large number of candidate template arcs this approach may be to time consuming for practice.

In each GRASP iteration, it_{ls} local search steps are performed after the randomized construction. The local search uses a template arc insertion neighborhood, where a new template arc is added to the solution, and then redundant template arcs are removed. The goal is to find template arcs that render at least two template arcs from the current solution redundant. Figure 5.12 shows such a situation. Another beneficial situation arises when further nodes are connected to the existing arborescence. In each iteration

Algorithm 17: k-MLSA-randomized-greedy(V,A, T^c)

```
1  G' = (V', A') with V' ← V, A ← ∅
2  T ← ∅   // currently used labels
3  while no k-arborescence exists do
4      i ← 0
5      for all t ∈ T^c do
6          z* ← ∞   //stores lowest found number of SCCs
7          A'' ← {a_ij ∈ A(t)}
8          compute SCCs of G' = (V', A' ∪ A'')
9          Z ← SCCs with in-degree zero
10         if |Z| < z* then
11             z* ← |Z|
12             RCL = ∅
13             i ← i + 1
14         end
15         if |Z| = z* then RCL = RCL ∪ t
16         if |RCL| ≥ rcl_max then
17             z* ← z* − 1
18             if i ≥ imp_max then
19                 break
20             end
21         end
22     end
23     t' ← random element from RCL
24     A' ← A' ∪ {a_ij ∈ A(t')}
25     T ← T ∪ {t'}
26     T^c ← T^c \ {t'}
27 end
28 remove redundant arcs and labels
```

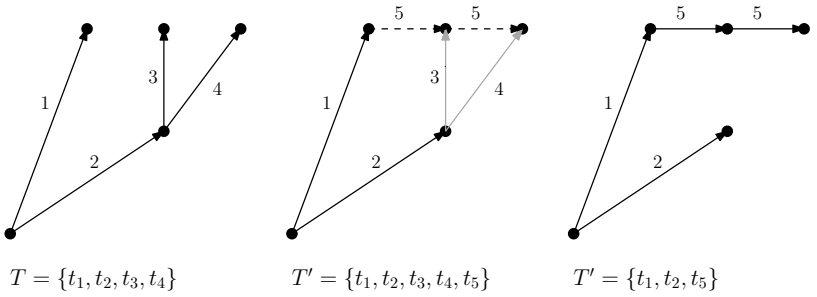

Figure 5.12: Template arc insertion neighborhood: after $\boldsymbol{t}_5$ is added to the solution $\boldsymbol{t}_3$ and $\boldsymbol{t}_4$ are redundant and thus can be removed from the solution

the template arcs are considered in decreasing order w.r.t. the number of tree arcs they represent. The neighborhood is searched with a first improvement strategy. Furthermore only a prespecified fraction of the template arcs is used, i.e. the template arcs representing most of the tree arcs.

5.8.3 Memetic Algorithm

As an alternative to GRASP we implemented a memetic algorithm (MA). It is based on a steady-state framework, where in each iteration a single offspring solution is derived and locally improved. It replaces a randomly chosen candidate solution from the population, to retain diversity. The algorithm uses tournament selection, and local improvement steps are performed for each new candidate solution after the application of the evolutionary operators, i.e. recombination and mutation. Algorithm 18 shows the overall framework.

Algorithm 18: k-`MLSA-MA()`

```
1 randomly create initial population
2 t ← 0
3 while t < t_max do
4     select parents T' and T'' by tournament selection
5     T ← crossover(T', T'')
6     mutation(T)
7     local improvement(T)
8     t ← t + 1
9 end
```

Following the ideas presented in [114] we encode a candidate solution as an ordered subset of labels. In our case the template arcs correspond to these labels and the chromosome of a candidate solution is therefore denoted by T, $T[i]$ denotes the i-th template arc of candidate solution T. If these template arcs induce a k-node arborescence we have a feasible solution, otherwise further template arcs need to be added to the candidate solution in order to make the solution feasible. Note however, that a feasible solution may

contain redundant template arcs, which are not necessarily part of an optimal solution induced by the other template arcs of the ordered set.

For candidate solutions of the initial population we ensure that they are feasible. To create a randomized candidate solution, all template arcs are shuffled and then added as long as the candidate solution remains infeasible.

The MA then tries to minimize the number of template arcs required for a feasible solution by iterative application of the genetic operators and local improvement. As many candidate solutions have the same number of template arcs, the total number of induced arcs is also considered in the fitness function $f(T)$, which is going to be minimized:

$$f(T) = |T| + \left(1 - \frac{|A'|}{|A|}\right). \tag{5.13}$$

Again, A' denotes the set of induced tree arcs. This accounts for the fact that candidate solutions whose template arcs cover many arcs are more likely to produce good offsprings and result in successful mutations.

Since the order of the template arcs does not need to be preserved, we use a crossover operator introduced in [85], which takes the template arcs for the child candidate solution alternatingly from the parents until a feasible solution is obtained. If a template arc reoccurs, it is not added to the offspring and the next template arc from the other parent is processed instead. Function `crossover(`T'`,` T''`)` (Algorithm 19) shows this procedure in detail. Again T denotes the (ordered) set of template arcs of an candidate solution, $T[i]$ denotes the i-th template arc.

Algorithm 19: `crossover(`T'`,` T''`)`

```
1  T ← ∅  // new offspring initialized with empty set
2  i ← 0, j ← 0  // counter variables
3  while T contains no k-MLSA do
4      if i mod 2 = 0 then
5          t ← T'[⌊i/2⌋]
6      else
7          t ← T''[⌊i/2⌋]
8      end
9      if t ∉ T then
10         T[j] ← t
11         j ← j + 1
12     end
13     i ← i + 1
14 end
15 return T
```

In addition to recombination we use two different types of mutation:

1. A randomly selected template arc $\boldsymbol{t} \notin T$ is appended. This increases the likelihood for the ability to remove some redundant template arc by a subsequent local improvement.
2. A randomly selected template arc $\boldsymbol{t} \notin T$, replaces either a random or the worst $\boldsymbol{t}' \in T$. The worst template arc is the one inducing the minimal number of arcs. If the solution is not feasible, further randomly selected template arcs are added until a feasible solution is reached.

The subsequent local improvement method `local-improvement(`T`)` (Algorithm 20), following the one presented in [85], uses the idea that a reordering of the template arcs could make some of them redundant. In contrast to the local improvement method used in the GRASP algorithm this method can only remove template arcs from a current solution if some of them are actually redundant. As the MA continuously modifies the candidate solutions from the population and also further template arcs are added to a candidate solution by mutation, there is no need to use a more expensive neighborhood search, which also considers currently unused template arcs.

The MA uses the same optimizations as the (randomized) greedy construction heuristics (described in Sections 5.8.1 and 5.8.2) to reduce the number of DFS calls. Furthermore it only checks for a k-node arborescence if the number of different nodes reaches the required size k, because a k-node arborescence is not theoretically possible before that.

Algorithm 20: `local-improvement(`T`)`

```
1  i ← 0  // counter variable
2  while i < |T| do
3      remove all arcs only labeled by T[i]
4      if T contains k-MLSA then
5          T ← T \ T[i]
6      else
7          restore respective arcs
8          i ← i + 1
9      end
10 end
```

Our computational experiments showed that the MA is able to produce high quality solutions very quickly and indeed finds an optimal solution in almost every case. Details on the results are given in Section 5.10.4.

5.9 Encoding of the Compressed Templates

In the following we describe how the compressed templates will be encoded on a binary level. The compression results from Section 5.10.2 are based on the definitions given in this section.

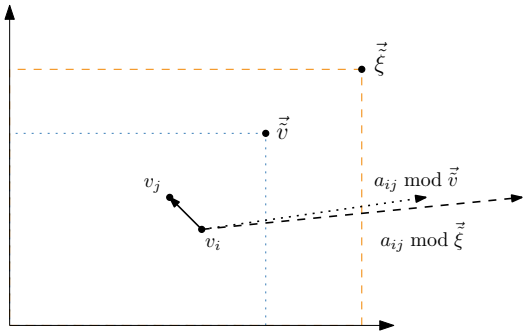

Figure 5.13: Representation of the arc $\boldsymbol{a}_{ij} = (\boldsymbol{v}_i, \boldsymbol{v}_j)$ using the domains $\tilde{\boldsymbol{v}}$ and $\tilde{\boldsymbol{\xi}}$ respectively.

We now need to extend the definition of the domain border given in Section 5.3. There, the domain border $\tilde{\boldsymbol{v}}$ was defined by $\tilde{v}^l = \max_{i=1,\ldots,n} \; v_i^l, \; l = 1, \ldots, d$. For the concrete specification of the original and the resulting compressed data structure we need to distinguish between the domain of one particular instance ($\tilde{\boldsymbol{v}}$), and a further entity, describing the size of the domain considering all possible input instances.

Definition 23 The overall domain border $\tilde{\boldsymbol{\xi}}$ is given by

$$\tilde{\xi}^l = \max_I \; \tilde{v}_I^l, \; l = 1, \ldots, d, \tag{5.14}$$

where the maximum goes over all input instances I.

To see the necessity to distinguish between $\tilde{\boldsymbol{\xi}}$ and $\tilde{\boldsymbol{v}}$, reconsider the preprocessing, i.e. the determination of the labels. For the further optimization it is obviously beneficial when single template arcs represent many tree arcs w.r.t. $\boldsymbol{\delta}$. Figure 5.13 shows the representation of the arc (i, j) using the domains $\tilde{\boldsymbol{v}}$ and $\tilde{\boldsymbol{\xi}}$ respectively. Unlike $(i, j) \bmod \tilde{\boldsymbol{\xi}}$, $(i, j) \bmod \tilde{\boldsymbol{v}}$ may be covered by a template arc in the domain $\tilde{\boldsymbol{v}}$ together with some other tree arcs in this domain. For arcs $a_{ij} > \tilde{\boldsymbol{\xi}} - \tilde{\boldsymbol{\delta}}$ it is not even possible to represent them together with arcs from the domain $\tilde{\boldsymbol{v}}$. Hence, an optimal solution to the k-MLSA problem may require more template arcs when using $\tilde{\boldsymbol{\xi}}$ instead of $\tilde{\boldsymbol{v}}$ as parameter for the preprocessing routine. As the values $\tilde{\boldsymbol{v}}$ often significantly deviate from $\tilde{\boldsymbol{\xi}}$ this effect is not negligible. Since m has a high impact on the compression ratio, we use $\tilde{\boldsymbol{v}}$ instead of $\tilde{\boldsymbol{\xi}}$ and accept the resulting disadvantage that we have to store $\tilde{\boldsymbol{v}}$ for each compressed template.

In our experimental evaluations (Section 5.10) we will compare our compressed data to the following size (in bits) of the original raw data

$$\lambda_{\text{raw}} = \mathit{size}(\text{ConstData}) + n \cdot \sum_{l=1}^{d} \lceil \text{ld } \tilde{\xi}^l \rceil, \tag{5.15}$$

which is the size of the constant data and n times the number of bits to store one particular point. The $\mathit{size}()$ operator denotes the number of bits needed to encode the data given as argument.

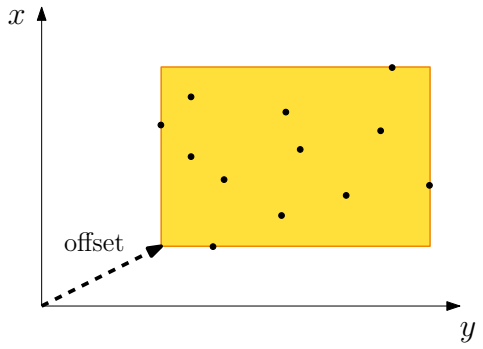

Figure 5.14: The points v_i are scattered in the shaded rectangle. Commonly used encodings of such data points indicate their common offset (dashed arrow) and the respective relative coordinate values of the points themselves. A direct encoding would require unnecessary many bits.

The variable ConstData denotes additional data of constant size that is related to the data points. E.g. if we have offset values (see Fig. 5.14) for each dimension we need this data to achieve a lossless compression of the data points themselves.

As the algorithms may be applied to a subset of the dimensions, we need the following function to define the total encoding length:

$$\chi^l = \begin{cases} 1 & \text{dimension } l \text{ is considered by the compression method} \\ 0 & \text{otherwise.} \end{cases} \tag{5.16}$$

The total encoding length of the arborescence (achieved by the compression procedure) is given by the following formula:

$$\begin{aligned} \lambda(m, \boldsymbol{\delta}, k, \tilde{\boldsymbol{v}}, \tilde{\boldsymbol{\xi}}, \boldsymbol{\chi}) \;=\; & {}_{size}(\text{ConstData}') + \underbrace{2 \cdot 7}_{\text{values } k,m} + \underbrace{2 \cdot \sum_{l=1}^{d} \lceil \chi^l \text{ ld } \tilde{\xi}^l \rceil}_{\text{root node, domain } \tilde{\boldsymbol{v}}} \\ & + \underbrace{2 \cdot (k-1)}_{\text{encoding of tree structure}} + \underbrace{\lceil m \cdot \sum_{l=1}^{d} \chi^l \text{ ld } \tilde{v}^l \rceil}_{\text{template arcs}} \\ & + (k-1) \cdot \lceil \underbrace{\text{ld } m}_{\text{index to template arc}} + \underbrace{\sum_{l=1}^{d} \chi^l \text{ ld } \tilde{\delta}^l}_{\tilde{\delta}-\text{values}} + \underbrace{\sum_{l=1}^{\tilde{d}} (1-\chi^l) \text{ ld } \tilde{v}^l}_{\text{remaining dimensions}} \rceil \end{aligned} \tag{5.17}$$

Note that ConstData of the compressed data is not necessarily the same as the corresponding entity of the raw data. For instance this can be the case if the raw data does not contain explicit offset values, but however $\exists l \in \{1, \ldots, d\} \mid \min_{i=1,\ldots,n} v_i^l > 0$. We account

for this by drawing a distinction between the constant data of the raw data (ConstData) and that of the compressed data, namely ConstData′.

The second term in equation (5.17) constitutes 14 bits for the storage of k and m; the third term denotes the number of bits that are necessary to store the root node and the size of the domain $\tilde{\boldsymbol{v}}$. The next term represents the encoding of the tree structure. The encoding is based on the parenthesis theorem of the depth first search (DFS) algorithm [33]. If we traverse the resulting arborescence by DFS and write a "(" each time we traverse an arc forward, and write ")" each time we traverse an arc backward the resulting string is an representation of the structure of the DFS-tree. In our case we simply write "0"s and "1"s instead of "("s and ")"s, thus requiring $(k-1)\cdot 2$ bit in total. The next term in equation (5.17) constitutes the size of the m template arcs. We only need to store the components that are indicated by the characteristic function χ^l, as we do not consider the other dimensions for compression, but directly store their values instead. The term $\operatorname{ld} \tilde{v}^l$ denotes the number of bits that are necessary to store the respective component of the template arc. The last term in equation (5.17) describes the size of the encoding of the $(k-1)$ tree arcs. Their representation consists of an index to a template arc, the appropriate correction vectors and finally the components of the remaining dimensions. Note that it is sufficient to round up the whole last term in equation (5.17) (and not each individual logarithm in the respective sums), because it is always possible to find such an appropriate encoding. For this purpose consider the following example, where we want to encode values of the following domain: $\{0\dots4\}\times\{0\dots8\}\times\{0\dots17\}$. The number of bits necessary to encode each individual dimension are 3, 4 and 5 respectively, which yields 12 in total. Contrary a more tight encoding would use the representation $n = c_1 + c_2 \cdot 18 + c_3 \cdot (18 \cdot 9)$, which just requires 10 bit in total.

5.9.1 Encoding Example

Let $\tilde{\boldsymbol{\xi}}^\mathsf{T} = (512, 512, 512)$, $\tilde{\boldsymbol{\delta}}^\mathsf{T} = (5, 5)$ and $k = 9$. As $\tilde{\boldsymbol{\delta}}$ is only two-dimensional we do not consider the third dimension of the input data for compression. Instead we simply store the respective values for each tree arc. The input data is given by the following set:

$$\left\{ \begin{pmatrix} 200 \\ 200 \\ 21 \end{pmatrix}, \begin{pmatrix} 208 \\ 304 \\ 30 \end{pmatrix}, \begin{pmatrix} 211 \\ 386 \\ 97 \end{pmatrix}, \begin{pmatrix} 261 \\ 356 \\ 210 \end{pmatrix}, \begin{pmatrix} 313 \\ 330 \\ 293 \end{pmatrix}, \begin{pmatrix} 314 \\ 409 \\ 22 \end{pmatrix}, \begin{pmatrix} 503 \\ 252 \\ 268 \end{pmatrix}, \begin{pmatrix} 608 \\ 280 \\ 157 \end{pmatrix}, \begin{pmatrix} 414 \\ 356 \\ 77 \end{pmatrix}, \begin{pmatrix} 662 \\ 332 \\ 104 \end{pmatrix}, \begin{pmatrix} 702 \\ 676 \\ 78 \end{pmatrix} \right\}.$$

Thus the offsets are 200 for the first and second coordinate and the domain borders are $\tilde{\boldsymbol{v}}^\mathsf{T} = (503, 477, 294)$. Figure 5.15 shows a solution to the given problem. The trivial encoding requires 243 bit, whereas the encoded template has a size of 232 bit; ConstData' $= 14 + 27 + 27 + 16 = 84$. The resulting compression ratio is not very impressive, but in the example only two dimensions have been considered for compression and $\tilde{\boldsymbol{\delta}}$ is extremely small. Being able to reconstruct the original data points without loss would require additional $\operatorname{ld}(\xi_1) + \operatorname{ld}(\xi_2) = 18$ bits for the offsets.

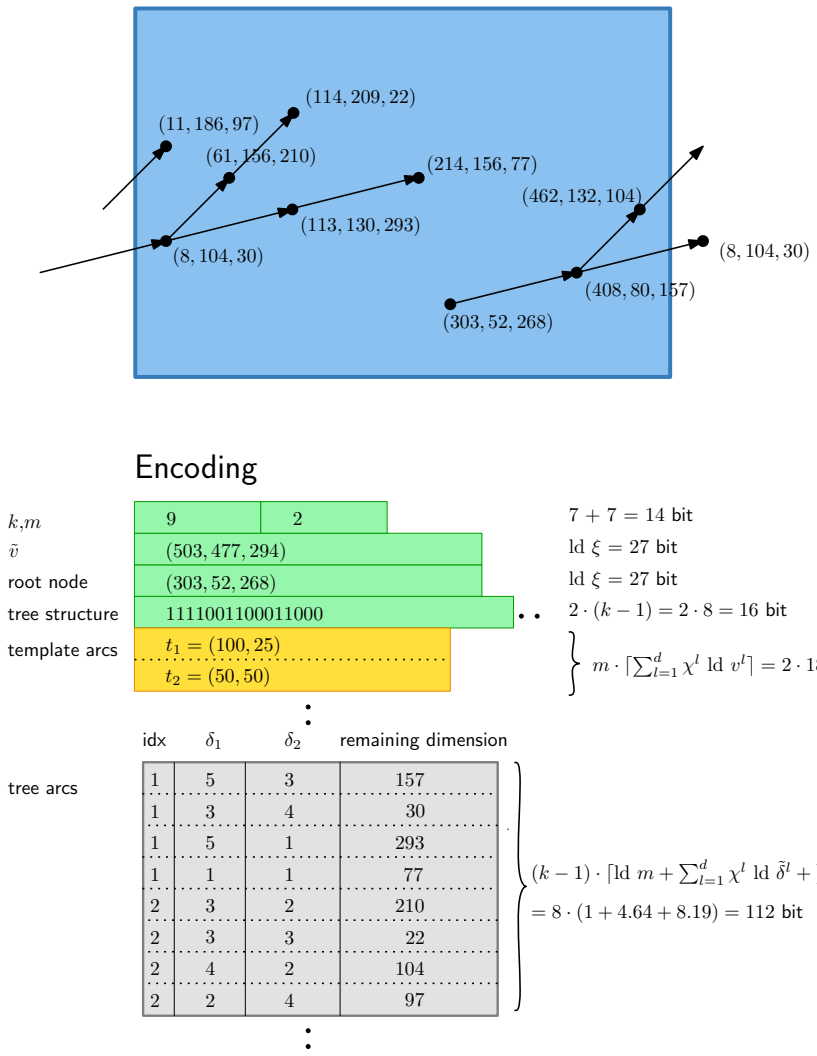

Figure 5.15: This figure shows a concrete encoding example. The first block basically contains information to be able to process the following blocks. It is followed by the list of the template arcs. This can be compared to a dictionary or codebook of traditional compression methods. The block on the bottom contains the actual tree information, i.e. a list of arcs encoded by an index to one of the template arcs, the respective correction vectors, and finally the values of the dimensions which are not considered for compression. The black dots indicate that the size of the respective (sub)blocks is not known in advance, because it depends on output values of the compression algorithm (like the number of template arcs m).

5.10 Experimental Results

This section starts by a description of the input data used for our computational experiments. Then we present the results of the exact method in order to analyze the compression ratios achievable by our methods. Furthermore we shortly evaluate the impact of the compression to the false non-match rate (FNMR). Finally we present the results of the metaheuristic techniques (greedy algorithm, GRASP, and the memetic algorithm) in comparison to the optimal results of branch-and-cut. All running times in this section refer to a 2.4 GHz Opteron processor with 4 GB RAM.

5.10.1 Test Instances

For our tests we use two different data sets. The first set of 20 templates was provided by the Fraunhofer Institute Berlin and is in the following referred to as Fraunhofer Templates. In addition we use a minutiae data set from the U.S. National Institute of Standards and Technology [50]. For the instances we use the prefix ft and nist, respectively.

The Fraunhofer data set contains 20 templates which are multiple scans of four different fingers from two persons. The encoding of the points is $\tilde{\boldsymbol{\xi}}^{\mathsf{T}} = (\xi_x, \xi_y, \xi_\theta, \xi_{\text{type}}) = (2^9, 2^9, 2^9, 2^1)$. The size of ConstData is 14 bit, i.e. 7 bit for the offset value for each spatial dimension respectively. The templates consist of 15 to 40 minutiae which corresponds to a typical amount of minutiae detected by an electronic fingerprint scanner. The templates are listed in the first part of Table 5.3, i.e. ft-01 to ft-20. The full name matches the pattern P[0-9999]_F[0-99]_R[0-99] where P abbreviates "Person", F "Finger", and R "Release". One can see, that various releases of the same finger involve significant differences to the number of minutiae that are detected. The second part of the table lists the templates from the NIST data set. From the large set of NIST Templates we selected a subset for our experiments, see Table 5.3. We chose five templates from each of the categories ugly, bad and good. The instance names reflect this classification. Furthermore for each fingerprint, there exists minutiae data to a latent and a corresponding tenprint image. Latent refers to fingerprints on e.g. crime scenes that are invisible to the eye and require some type of chemical processing or dusting to make them visible. Fingerprint images that are created by inking and rolling fingertips onto a paper or some scanning device and traditionally have been captured of all ten fingers are usually referred to as tenprints. Obviously the quality of tenprints is superior to the latents, which typically just consist of a few dozen minutiae (min=5, max=82, avg=20.55, std-dev=13.25). The tenprints have between 48 and 193 minutiae (avg = 106.3, std-dev=25), which is a significantly higher number than we can expect to get from an electronic fingerprint scanning device. None the less, in our experiments we only use the tenprint data, as the latents do not contain enough minutiae and the larger size of the tenprints enables us to test the performance of our method concerning higher numbers of data points. The encoding of the NIST points is $\tilde{\boldsymbol{\xi}}^{\mathsf{T}} = (\xi_x, \xi_y, \xi_\theta, \xi_{\text{type}}) = (2^{12}, 2^{12}, 2^9, 2^1)$. The size of ConstData is 33, which are the offsets for the dimensions x, y and θ.

Table 5.2: Characterization of the two template data sets

	Fraunhofer	NIST
avg($\|V\|$)	30.75	96.47
min($\|V\|$)	15	72
max($\|V\|$)	40	120
$\tilde{\boldsymbol{v}}_{\text{avg}}$	$(286, 383, 358, 2)^{\mathsf{T}}$	$(3993, 3368, 359, 2)^{\mathsf{T}}$
$\tilde{\boldsymbol{v}}_{\text{min}}$	$(129, 191, 252, 2)^{\mathsf{T}}$	$(2936, 2281, 359, 2)^{\mathsf{T}}$
$\tilde{\boldsymbol{v}}_{\text{max}}$	$(224, 287, 312, 2)^{\mathsf{T}}$	$(3293, 2788, 353, 2)^{\mathsf{T}}$

Column ρ^* shows the best compression ratios that could be achieved by our computational experiments, the column right to it shows the respective parameter values that yielded the result. The results for the Fraunhofer templates are exact ones, whereas the results for the NIST templates are results of metaheuristics. Table 5.2 gives an overview of the characteristics of the two data sets.

5.10.2 Compression Results

Table 5.3 gives an overview of the data instances used for our experimental evaluations, and the corresponding best results. Besides the number of nodes and resulting tree arcs we list the best compression ratios we could achieve by our method together with the respective parameter values. The compression ratios ρ are defined by

$$\rho\ [\%] = 100 - \frac{100 \cdot \lambda_{\text{raw}}}{\lambda_{\text{enc}}}, \tag{5.18}$$

where λ_{raw} refers to equation (5.15) of the k selected points and λ_{enc} to equation (5.17). As ρ does not reflect the size of the compressed template compared to the full template, but only to the trivial encoding of the k selected points, we will also refer to ρ as relative compression ratio. We denote the best found compression ratio by ρ^*. Though adequate for our application background, we do not set $\text{ConstData}' = 0$ for the subsequent experiments, in order to evaluate the compression ratios more objectively.

In Table 5.4 we present some data regarding the preprocessing – the determination of the candidate template arcs T^c. For larger values of $\tilde{\boldsymbol{\delta}}$ the running times become quite large, which is clearly not satisfactory. Obviously, the running times for $\tilde{\boldsymbol{\delta}}$ yielding the best compression ratios (e.g. $\tilde{\boldsymbol{\delta}}^{\mathsf{T}} = (25, 25)$ or $\tilde{\boldsymbol{\delta}}^{\mathsf{T}} = (30, 30, 30)$, see Tables 5.6 and 5.8) are of high importance. Fortunately the running times for these parameter values seem to be still reasonable for practical applications.

Tables 5.5, 5.6, 5.7, and 5.8 give an overview of the compression ratios of this approach. For the Fraunhofer data we used the parameter values $k \in \{10, 15, \ldots, 40\}$ and $\tilde{\boldsymbol{\delta}} \in \left\{ \binom{10}{10}, \binom{15}{15}, \ldots, \binom{45}{45}, \binom{50}{50} \right\}$ (i.e. applying the algorithm to two dimensions) as well

Table 5.3: Overview about the test instances used for our experiments and their best results

short name	**full name**	$\|V\|$	$\|B\|$	$\rho^*[\%]$	**parameters**
ft-01	P0001_F00_R00	31	930	9.9	$k = 25, \tilde{\boldsymbol{\delta}}^\mathsf{T} = (20, 20)$
ft-02	P0001_F00_R01	38	756	8.7	$k = 25, \tilde{\boldsymbol{\delta}}^\mathsf{T} = (40, 40)$
ft-03	P0001_F00_R02	35	1190	9.6	$k = 35, \tilde{\boldsymbol{\delta}}^\mathsf{T} = (35, 35, 35)$
ft-04	P0001_F00_R03	20	380	5.1	$k = 15, \tilde{\boldsymbol{\delta}}^\mathsf{T} = (35, 35, 35)$
ft-05	P0001_F00_R04	39	1482	11.5	$k = 30, \tilde{\boldsymbol{\delta}}^\mathsf{T} = (20, 20)$
ft-06	P0001_F01_R00	15	210	2.8	$k = 15, \tilde{\boldsymbol{\delta}}^\mathsf{T} = (40, 40)$
ft-07	P0001_F01_R01	28	756	7.2	$k = 25, \tilde{\boldsymbol{\delta}}^\mathsf{T} = (20, 20)$
ft-08	P0001_F01_R02	27	702	9.8	$k = 20, \tilde{\boldsymbol{\delta}}^\mathsf{T} = (35, 35)$
ft-09	P0001_F01_R03	27	702	8.6	$k = 25, \tilde{\boldsymbol{\delta}}^\mathsf{T} = (50, 50)$
ft-10	P0001_F01_R04	31	930	8.5	$k = 25, \tilde{\boldsymbol{\delta}}^\mathsf{T} = (45, 45)$
ft-11	P0001_F03_R00	38	1406	11.7	$k = 39, \tilde{\boldsymbol{\delta}}^\mathsf{T} = (45, 45)$
ft-12	P0001_F03_R01	28	756	12.0	$k = 25, \tilde{\boldsymbol{\delta}}^\mathsf{T} = (35, 35)$
ft-13	P0001_F03_R02	25	600	8.7	$k = 25, \tilde{\boldsymbol{\delta}}^\mathsf{T} = (50, 50)$
ft-14	P0001_F03_R03	33	1056	10.2	$k = 30, \tilde{\boldsymbol{\delta}}^\mathsf{T} = (45, 45)$
ft-15	P0001_F03_R04	29	812	9.9	$k = 29, \tilde{\boldsymbol{\delta}}^\mathsf{T} = (45, 45)$
ft-16	P0014_F00_R00	37	1332	10.6	$k = 25, \tilde{\boldsymbol{\delta}}^\mathsf{T} = (40, 40, 40)$
ft-17	P0014_F00_R01	31	930	8.6	$k = 25, \tilde{\boldsymbol{\delta}}^\mathsf{T} = (45, 45)$
ft-18	P0014_F00_R02	40	1560	13.5	$k = 30, \tilde{\boldsymbol{\delta}}^\mathsf{T} = (30, 30, 30)$
ft-19	P0014_F00_R03	35	1190	10.1	$k = 30, \tilde{\boldsymbol{\delta}}^\mathsf{T} = (45, 45)$
ft-20	P0014_F00_R04	28	756	7.1	$k = 20, \tilde{\boldsymbol{\delta}}^\mathsf{T} = (45, 45)$
nist-u-01-t	u201t6i	99	9702	18.9	$k = 80, \tilde{\boldsymbol{\delta}}^\mathsf{T} = (120, 120)$
nist-u-02-t	u202t8i	93	8556	18.9	$k = 80, \tilde{\boldsymbol{\delta}}^\mathsf{T} = (120, 120)$
nist-u-03-t	u204t2i	100	9900	18.9	$k = 80, \tilde{\boldsymbol{\delta}}^\mathsf{T} = (120, 120)$
nist-u-04-t	u205t4i	84	6972	13.8	$k = 80, \tilde{\boldsymbol{\delta}}^\mathsf{T} = (120, 120)$
nist-u-05-t	u206t3i	72	5256	18.9	$k = 80, \tilde{\boldsymbol{\delta}}^\mathsf{T} = (80, 80)$
nist-b-01-t	b101t9i	106	11130	18.9	$k = 80, \tilde{\boldsymbol{\delta}}^\mathsf{T} = (80, 80)$
nist-b-02-t	b102t0i	94	8742	13.4	$k = 80, \tilde{\boldsymbol{\delta}}^\mathsf{T} = (80, 80)$
nist-b-03-t	b104t8i	107	11342	18.9	$k = 80, \tilde{\boldsymbol{\delta}}^\mathsf{T} = (120, 120)$
nist-b-04-t	b105t2i	81	6480	18.6	$k = 80, \tilde{\boldsymbol{\delta}}^\mathsf{T} = (80, 80)$
nist-b-05-t	b106t8i	93	8556	13.8	$k = 80, \tilde{\boldsymbol{\delta}}^\mathsf{T} = (80, 80)$
nist-g-01-t	g001t2i	99	9702	13.4	$k = 80, \tilde{\boldsymbol{\delta}}^\mathsf{T} = (80, 80)$
nist-g-02-t	g002t3i	101	10100	13.8	$k = 80, \tilde{\boldsymbol{\delta}}^\mathsf{T} = (80, 80)$
nist-g-03-t	g003t8i	102	10302	18.9	$k = 80, \tilde{\boldsymbol{\delta}}^\mathsf{T} = (120, 120)$
nist-g-04-t	g004t8i	120	14280	13.8	$k = 80, \tilde{\boldsymbol{\delta}}^\mathsf{T} = (80, 80)$
nist-g-05-t	g005t8i	80	6320	13.8	$k = 73, \tilde{\boldsymbol{\delta}}^\mathsf{T} = (80, 80)$

Table 5.4: Number of candidate template arcs $|T^c|$ and running times of the preprocessing for the Fraunhofer data with some typical values of $\tilde{\boldsymbol{\delta}}$

$\tilde{\boldsymbol{\delta}}^\mathsf{T}$	$(10,10)$		$(20,20)$		$(30,30)$		$(40,40)$		$(10,10,10)$		$(20,20,20)$		$(30,30,30)$		$(40,40,40)$	
inst.	$\lvert T^c\rvert$	$t[s]$	$\lvert T^c\rvert$	$t[s]$	$\lvert T^c\rvert$	$t[s]$	$\lvert T^c\rvert$	$t[s]$	$\lvert T^c\rvert$	$t[s]$	$\lvert T^c\rvert$	$t[s]$	$\lvert T^c\rvert$	$t[s]$	$\lvert T^c\rvert$	$t[s]$
ft-01	797	7	1863	74	3747	526	5802	2810	826	3	860	6	1693	34	3897	272
ft-02	610	4	1443	30	2666	185	4374	911	664	2	715	4	1560	23	3353	241
ft-03	1002	12	2684	146	4644	902	7786	4799	1042	5	1152	11	2620	82	6464	1019
ft-04	296	1	510	4	1035	23	1582	101	356	1	352	1	695	8	1223	55
ft-05	1401	23	3398	297	7044	2052	11276	12144	1246	8	1486	20	3546	237	8669	2578
ft-06	145	1	248	1	354	3	574	11	202	1	156	1	231	1	439	2
ft-07	517	3	1019	19	2035	102	3280	426	680	2	614	3	1129	11	2237	63
ft-08	533	3	1030	19	1898	92	3402	463	630	2	570	3	1064	10	2164	57
ft-09	506	3	1022	17	1828	94	3429	382	626	2	578	3	1061	11	2091	58
ft-10	767	6	1721	46	3132	283	5365	1334	812	3	812	5	1548	28	2930	165
ft-11	1565	28	3751	377	7561	3261	11542	15580	1168	7	1677	25	4446	305	11497	3748
ft-12	699	5	1548	51	3268	356	5064	1836	632	2	752	5	1718	40	4224	341
ft-13	498	3	1010	19	2201	122	3980	617	517	1	531	3	1143	15	2573	143
ft-14	1002	10	2292	109	4458	991	7130	3468	842	4	1024	8	2286	58	5150	522
ft-15	742	6	1768	63	3487	442	5720	2549	656	2	786	5	1676	30	3980	333
ft-16	1144	14	2501	145	4984	919	7330	4431	1142	6	1457	21	3125	190	6095	2858
ft-17	800	5	1633	42	3317	301	5585	1365	814	3	878	6	1878	31	3847	273
ft-18	1288	22	2858	238	5632	1577	8950	6106	1298	10	1710	31	3963	270	8889	3435
ft-19	1017	10	2161	87	3857	531	6247	2515	1000	5	992	10	1944	49	4538	399
ft-20	622	3	1086	20	1649	107	3022	395	658	2	672	4	990	20	2021	104

Table 5.5: Average compression ratios for the Fraunhofer templates for $\tilde{\boldsymbol{\delta}}^{\mathsf{T}} = (30, 30, 30)$

k	$\rho_{\max}$	$\rho_{\min}$	ρ_{avg}	σ_ρ
10	-6.12	-6.80	-6.63	0.29
15	5.07	-9.22	1.15	3.67
20	6.62	-1.57	4.60	3.11
25	10.92	-0.14	5.05	3.68
30	13.47	1.29	6.31	3.05
35	9.66	4.63	7.56	2.26

as $\tilde{\boldsymbol{\delta}} \in \left\{ \begin{pmatrix} 10 \\ 10 \\ 10 \end{pmatrix}, \begin{pmatrix} 15 \\ 15 \\ 15 \end{pmatrix}, \begin{pmatrix} 20 \\ 20 \\ 20 \end{pmatrix}, \ldots, \begin{pmatrix} 45 \\ 45 \\ 45 \end{pmatrix}, \begin{pmatrix} 50 \\ 50 \\ 50 \end{pmatrix} \right\}$ (applying the algorithm to three dimensions) and all their combinations for our experiments. The results for the Fraunhofer data have been computed with the exact branch-and-cut method, and are therefore optimal values. Due to the long running time and memory requirements the branch-and-cut algorithm is not practicable for the NIST templates. These results have therefore been computed with the memetic algorithm. Tables 5.5, 5.6, 5.7, and 5.8 list average compression values, where the average goes over all instances and parameter settings listed in the table caption. These results are also illustrated in Figures 5.10.2, 5.10.2 and 5.10.2.

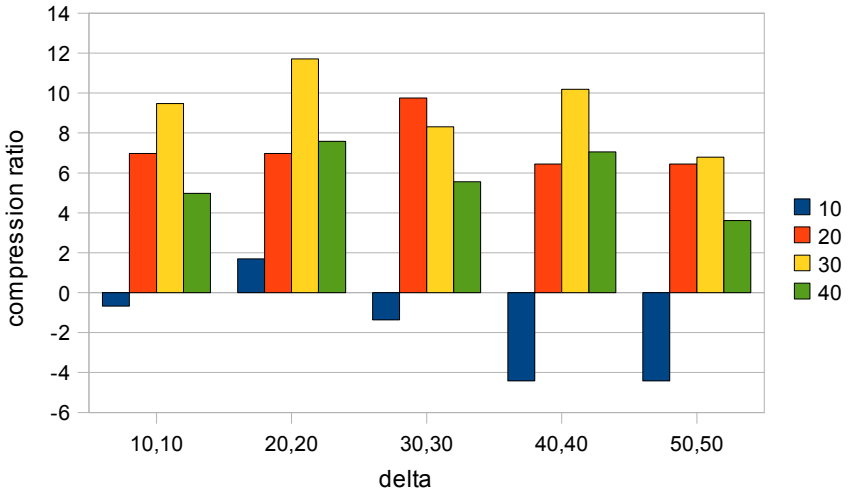

Figure 5.16: Average compression ratios for the Fraunhofer data for $k \in \{10, 20, 30, 40\}$ and 2-dimensional $\tilde{\boldsymbol{\delta}}$

At a first glance the compression ratios are not too high. However, when compared to other well established compression techniques, it turns out that all these methods consistently enlarge the templates. Table 5.9 shows the results of our application of other well known compression tools to our test data. Columns 2 to 7 list the results of some

Table 5.6: Average compression ratios for the Fraunhofer templates with $k \in \{20, 25, 30\}$

$\tilde{\boldsymbol{\delta}}^{\mathsf{T}}$	$\rho_{\max}$	$\rho_{\min}$	ρ_{avg}	σ_ρ
$(10, 10)$	9.48	-0.87	4.98	2.11
$(15, 15)$	9.93	1.39	5.44	2.06
$(20, 20)$	11.71	0.35	5.95	2.31
$(25, 25)$	9.80	0.87	5.43	1.89
$(30, 30)$	9.76	2.69	5.79	1.97
$(35, 35)$	12.04	0.00	4.52	2.87
$(40, 40)$	10.19	0.00	5.87	2.58
$(45, 45)$	10.19	0.35	6.03	2.56
$(50, 50)$	8.68	-2.96	4.54	2.31
$(10, 10, 10)$	0.84	-20.21	-5.82	4.95
$(15, 15, 15)$	6.72	-8.01	-0.30	3.73
$(20, 20, 20)$	11.01	-5.92	2.61	3.57
$(25, 25, 25)$	10.54	-3.36	3.90	2.83
$(30, 30, 30)$	13.47	-1.57	5.14	3.39
$(35, 35, 35)$	10.45	0.82	5.70	2.81
$(40, 40, 40)$	10.64	-0.52	4.29	2.93
$(45, 45, 45)$	9.60	0.35	4.99	1.92
$(50, 50, 50)$	9.60	0.42	4.84	2.51

Table 5.7: Average compression ratios for the NIST templates with $\tilde{\boldsymbol{\delta}}^{\mathsf{T}} = (80, 80, 80)$

k	$\rho_{\max}$	$\rho_{\min}$	ρ_{avg}	σ_ρ
20	17.35	4.85	9.47	2.87
40	18.90	11.03	15.42	2.97
60	18.97	10.00	16.08	2.20
80	19.30	10.70	15.49	3.26

Table 5.8: Average compression ratios for the NIST templates with $k \in \{40, 60, 80\}$

$\tilde{\boldsymbol{\delta}}^{\mathsf{T}}$	$\rho_{\max}$	$\rho_{\min}$	ρ_{avg}	σ_ρ
$(40, 40)$	19.02	12.40	15.09	1.77
$(80, 80)$	15.51	3.63	10.01	3.42
$(40, 40, 40)$	18.24	11.80	15.85	1.80
$(80, 80, 80)$	19.30	10.00	15.67	2.85

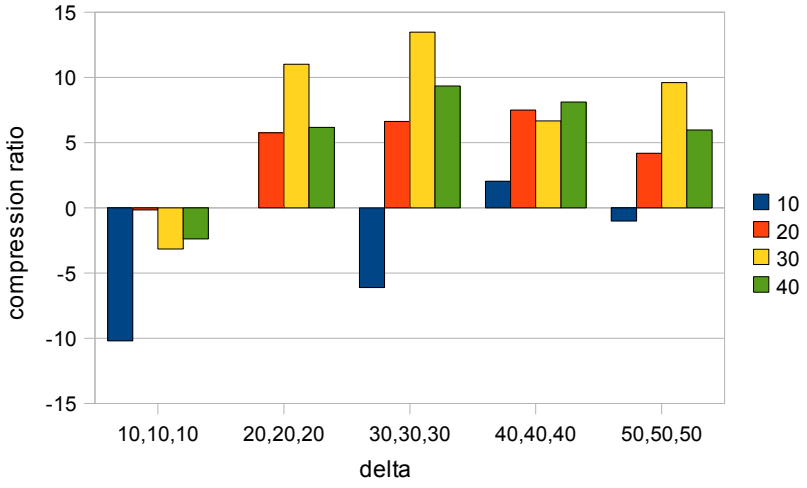

Figure 5.17: Average compression ratios for the Fraunhofer data for $k \in \{10, 20, 30, 40\}$ and 3-dimensional $\tilde{\boldsymbol{\delta}}$

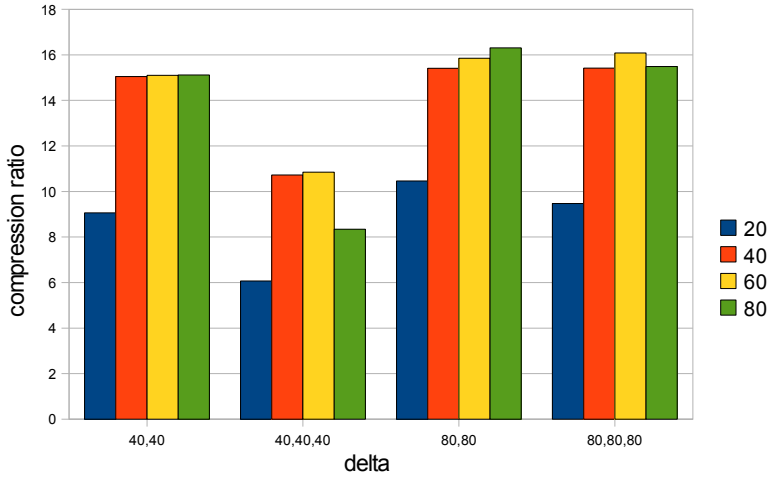

Figure 5.18: Average compression ratios for the NIST data for $k \in \{20, 40, 60, 80\}$

compression algorithms implemented in the LEDA C++ library [73] for various values of k. For this purpose we selected k points at random. The abbreviations in the first header row denote: Huff $\hat{=}$ Huffman Coding; BMA $\hat{=}$ Burrows-Wheeler Transform in combination with a Move-To-Front coder and an Adaptive Arithmetic Coder [108, 12, 83]; Columns 8 to 19 show the results for the application of some commonly used (Unix/Linux) compression tools, namely `zip`, `gzip`, `bzip2` and `compress` under Kubuntu Linux in version 8.04. Again, we selected k points at random and applied the compression tools to their binary encoding. Finally we subtracted the size of constant data of known size from the

compressed file size, in particular 22 byte for `zip`, 18 byte for `gzip` and 8 byte for `bzip2`.

The bad performance of these tools on our data set may be explained on the one hand by the fact that they are not specifically designed to compress very small data sets, but on the other hand also by the observation that the input data does not contain a significant amount of redundant information. Furthermore the underlying algorithms are not able to benefit from the special structure of the input data, i.e. spatial coordinates, but also cannot benefit from the fact that the relative order of the points need not be preserved.

5.10.3 Matching Results

The impact of the reduction of the number of minutiae, i.e. the parameter k to the reliability of the matching of different minutiae templates has been evaluated in [41]. Obviously k is the only parameter of our algorithm having an impact to matching quality. Other influences result from the concrete hardware (fingerprint scanner), the minutiae detection and extraction algorithm and also the concrete matching algorithm.

For this purpose, computational experiments have been performed with the Fraunhofer data together with a fingerprint minutiae matching algorithm, also provided by the Fraunhofer institute [41]. The false match rate (FMR) turned out not to be critical for our application scenario. For nearly all reasonable values of k, i.e. $k \geq 5$, it remained zero. For a verification application the false non-match rate (FNMR) is of higher importance. When mating different templates from the same finger the FNMR is about 5% on average. This non vanishing FNMR is due to some randomness in the template acquisition process (scanning, detection, extraction). Hence it is unlikely to achieve exactly the same minutiae sets by performing the acquisition process several times. Values of $k \leq 20$ yielded FNMRs of more than 30% on average, which definitely seems to be too high for our application scenario. Nevertheless, larger values, in particular $k \geq 25$ yielded FNMR of less than 20% on average. This seems to be reasonable as the acquisition process can be repeated several times in the case of failure of verification. Demanding $k \geq 25$ might indeed be very pessimistic, as 12-20 minutiae are often thought to be sufficient for a trusty matching [97].

Nevertheless the optimal value of k is highly dependent on the concrete implementation of this application, in particular the fingerprint scanner and the subsequent image processing algorithms. The exact requirements regarding the FNMR need to be specified for a concrete implementation; also the acceptable computation times have influence.

5.10.4 Algorithmic results

In this section we compare the presented k-MLSA algorithms regarding the running times and solution quality.

Although the running times of the B&C algorithm are sometimes very low, there are also many cases where the running times are clearly too high for practical purposes. Due to the larger size of the NIST templates the branch-and-cut approach is not practicable for

Table 5.9: Compression ratios in % of various well known compression techniques applied to our data sample

instance	Huff k=10	BMA k=10	Huff k=20	BMA k=20	Huff k=30	BMA k=30	zip k=10	gzip k=10	bzip2 k=10	compr k=10	zip k=20	gzip k=20	bzip2 k=20	compr k=20	zip k=30	gzip k=30	bzip2 k=30	compr k=30
ft-01	-91.7	-47.5	-90.9	-29.9	-89.9	-21.8	-331.2	-9.3	-165.6	-21.8	-155.8	-7.3	-79.4	-17.6	-101.9	-4.8	-66.3	-15.3
ft-02	-92.2	-47.5	-91.6	-30.6	-91.1	-23.8	-331.2	-9.3	-168.7	-21.8	-155.8	-7.3	-97.0	-17.6	n/a	n/a	n/a	n/a
ft-03	-91.9	-47.5	-91.1	-29.9	-90.7	-22.4	-331.2	-9.3	-159.3	-21.8	-155.8	-7.3	-86.7	-17.6	-101.9	-4.8	-72.1	-15.3
ft-04	-91.9	-47.5	-91.2	-29.9	-91.2	-29.9	-331.2	-9.3	-156.2	-21.8	-155.8	-7.3	-88.2	-17.6	n/a	n/a	n/a	n/a
ft-05	-92.2	-47.5	-91.5	-29.9	-90.8	-22.4	-331.2	-9.3	-175.0	-21.8	-155.8	-7.3	-100.0	-17.6	-101.9	-4.8	-68.2	-15.3
ft-06	-91.4	-47.5	-91.2	-35.8	-91.2	-35.8	-331.2	-9.3		-21.8	n./a	n./a	n./a	n./a	n/a	n/a	n/a	n/a
ft-07	-91.9	-47.5	-91.5	-30.6	-91.0	-23.8	-331.2	-9.3	-171.8	-21.8	-155.8	-7.3	-98.5	-17.6	n/a	n/a	n/a	n/a
ft-08	-91.7	-47.5	-90.9	-30.6	-90.6	-24.0	-331.2	-9.3	-153.1	-21.8	-155.8	-7.3	-83.8	-17.6	n/a	n/a	n/a	n/a
ft-09	-91.4	-47.5	-90.6	-30.6	-90.2	-24.6	-331.2	-9.3	-159.3	-21.8	-155.8	-7.3	-82.3	-17.6	n/a	n/a	n/a	n/a
ft-10	-91.7	-46.7	-91.1	-30.6	-90.5	-22.4	-331.2	-9.3	-175.0	-21.8	-155.8	-7.3	-92.6	-17.6	-101.9	-4.8	-75.0	-15.3
ft-11	-91.4	-47.5	-91.2	-29.9	-90.3	-22.4	-331.2	-9.3	-162.5	-21.8	-155.8	-7.3	-85.2	-17.6	-101.9	-4.8	-71.1	-15.3
ft-12	-91.4	-47.5	-91.4	-30.6	-90.7	-23.8	-331.2	-9.3	-153.1	-21.8	-155.8	-7.3	-89.7	-17.6	n/a	n/a	n/a	n/a
ft-13	-91.9	-47.5	-91.5	-29.9	-91.0	-25.7	-331.2	-9.3	-165.6	-21.8	-155.8	-7.3	-92.6	-17.6	n/a	n/a	n/a	n/a
ft-14	-91.9	-47.5	-90.9	-29.2	-90.2	-21.8	-331.2	-9.3	-162.5	-21.8	-155.8	-7.3	-89.7	-17.6	-101.9	-4.8	-69.2	-15.3
ft-15	-91.9	-47.5	-91.1	-30.6	-90.3	-22.5	-331.2	-9.3	-156.2	-21.8	-155.8	-7.3	-80.8	-17.6	n/a	n/a	n/a	n/a
ft-16	-91.1	-47.5	-90.5	-29.9	-89.8	-21.8	-331.2	-9.3	-156.2	-21.8	-155.8	-7.3	-89.7	-17.6	-101.9	-4.8	-71.1	-15.3
ft-17	-91.4	-47.5	-91.1	-29.2	-90.6	-23.0	-331.2	-9.3	-153.1	-21.8	-155.8	-7.3	-88.2	-17.6	-101.9	-4.8	-74.0	-15.3
ft-18	-91.9	-47.5	-91.2	-30.6	-90.8	-23.0	-331.2	-9.3	-181.2	-21.8	-155.8	-7.3	-94.1	-17.6	-101.9	-4.8	-67.3	-15.3
ft-19	-91.9	-47.5	-91.2	-29.9	-90.5	-22.4	-331.2	-9.3	-168.7	-21.8	-155.8	-7.3	-89.7	-17.6	-101.9	-4.8	-70.1	-15.3
ft-20	-91.7	-47.5	-91.4	-30.6	-90.7	-23.8	-331.2	-9.3	-156.2	-21.8	-155.8	-7.3	-89.7	-17.6	n/a	n/a	n/a	n/a
	k=20		k=40		k=80		k=20				k=40				k=80			
nist-u-01	-90.7	-24.6	-88.7	-14.5	-84.6	-8.2	-128.2	-5.4	-78.2	-16.3	-62.7	-2.6	-45.7	-14.3	-31.0	-1.3	-40.2	-16.0
nist-u-02	-90.2	-24.6	-87.8	-13.8	-84.2	-8.2	-128.2	-5.4	-76.0	-16.3	-62.7	-2.6	-34.5	-13.8	-31.0	-1.3	-39.9	-16.5
nist-u-03	-90.3	-24.6	-88.1	-14.2	-83.3	-7.5	-128.2	-5.4	-79.3	-16.3	-62.7	-2.6	-41.4	-14.3	-31.0	-1.3	-38.4	-16.0
nist-u-04	-90.3	-24.6	-88.3	-14.5	-84.6	-8.2	-128.2	-5.4	-72.8	-16.3	-62.7	-2.6	-39.3	-13.8	-31.0	-1.3	-39.9	-16.0
nist-u-05	-90.4	-24.6	-88.4	-14.5	-84.8	-8.5	-128.2	-5.4	-80.4	-16.3	-62.7	-2.6	-48.4	-14.3	-31.0	-1.3	-42.4	-16.8
nist-b-01	-90.5	-24.0	-88.2	-13.8	-84.8	-8.7	-128.2	-5.4	-79.3	-16.3	-62.7	-2.6	-42.0	-14.3	-31.0	-1.3	-39.9	-17.1
nist-b-02	-89.8	-24.6	-87.7	-14.5	-83.5	-8.0	-128.2	-5.4	-78.2	-16.3	-62.7	-2.6	-34.5	-13.8	-31.0	-1.3	-37.1	-16.8
nist-b-03	-89.9	-24.0	-88.5	-13.8	-84.5	-8.2	-128.2	-5.4	-75.0	-16.3	-62.7	-2.6	-44.6	-13.8	-31.0	-1.3	-38.1	-16.8
nist-b-04	-90.6	-24.6	-88.6	-14.5	-84.5	-8.0	-128.2	-5.4	-77.1	-16.3	-62.7	-2.6	-39.8	-13.8	-31.0	-1.3	-39.9	-16.5
nist-b-05	-90.3	-24.6	-88.3	-14.5	-84.4	-8.0	-128.2	-5.4	-75.0	-16.3	-62.7	-2.6	-44.6	-14.3	-31.0	-1.3	-38.4	-16.8
nist-g-01	-90.1	-24.0	-88.5	-13.8	-84.6	-8.4	-128.2	-5.4	-72.8	-16.3	-62.7	-2.6	-43.6	-13.2	-31.0	-1.3	-38.6	-15.5
nist-g-02	-90.9	-24.6	-88.8	-13.8	-84.2	-8.0	-128.2	-5.4	-80.4	-16.3	-62.7	-2.6	-46.2	-13.2	-31.0	-1.3	-39.4	-16.0
nist-g-03	-90.7	-24.6	-88.1	-13.8	-84.5	-8.4	-128.2	-5.4	-80.4	-16.3	-62.7	-2.6	-40.4	-14.3	-31.0	-1.3	-40.5	-17.1
nist-g-04	-90.6	-24.6	-88.3	-14.2	-84.9	-8.4	-128.2	-5.4	-78.2	-16.3	-62.7	-2.6	-47.3	-13.8	-31.0	-1.3	-78.2	-16.5
nist-g-05	-90.1	-24.0	-87.5	-14.2	-83.8	-7.1	-128.2	-5.4	-78.2	-16.3	-62.7	-2.6	-38.8	-14.3	-31.0	-1.3	-38.6	-16.8

them anymore. Computational experiments showed that using the cycle elimination cuts exclusively, i.e. without the directed connection cuts, yields lower running times than using both kinds of cuts. The exclusive use of the directed connection cuts turned out to be very slow, due to the computationally more expensive separation problem, i.e. the maximum flow problem. Most of the instances can be solved to provable optimality within a couple of seconds, but there are also many instances where the total running time lies between 10 and 30 minutes, or even more in some rare cases.

The BCP approach also allows to solve NIST instances with some parameter configurations which is a clear improvement to the B&C approach. Table 5.10 gives an overview of corresponding average running times and success ratios.

Table 5.10: Results of the Branch-and-Cut-and-Price algorithm. Average values for all solved instances in the particular group are reported.

Instances	Parameters	$t[s]$	pit	pvar	inst. solved
Fraunhofer	$\tilde{\boldsymbol{\delta}}^{\mathsf{T}} = (40, 40, 40), k = 20$	98	1436	753	19/19
Fraunhofer	$\tilde{\boldsymbol{\delta}}^{\mathsf{T}} = (40, 40, 40), k = 30$	132	1023	604	19/19
NIST	$\tilde{\boldsymbol{\delta}}^{\mathsf{T}} = (40, 40, 40), k = 40$	2002	1838	796	4/15
NIST	$\tilde{\boldsymbol{\delta}}^{\mathsf{T}} = (40, 40, 40), k = 80$	1270	2545	847	5/15
NIST	$\tilde{\boldsymbol{\delta}}^{\mathsf{T}} = (40, 40, 40), k = \lvert V \rvert$	2692	4636	1280	7/15

However, running times are still too high for practical applications, where a total running time of a few minutes will be acceptable. The metaheuristics are able to fulfill this goal.

The results regarding the running times and solution quality, i.e. the number of template arcs required for the resulting arborescence, are presented in Tables 5.11 and 5.12. The first three columns show the instance names and parameters k and $\boldsymbol{\delta}$. Then, in the first part of the table, we list the results from the exact branch-and-cut method, the second part contains the objective value of the currently best known solution. The column m_{best} shows the best result of the multiple runs of the algorithm, column m_{avg} shows the average value. By σ_x we denote the standard deviation of the entity x. The column $\#b.s.$ shows the percentage of runs, where the solution listed in m_{best} has been found. Average running times are listed in column t_{avg}. For each algorithm we compare two parameter settings yielding good compression ratios for the Fraunhofer and the NIST data respectively. Instead of using $k = 30$ which for the Fraunhofer data yielded the best compression results with $\tilde{\boldsymbol{\delta}} = (30, 30, 30)^{\mathsf{T}}$, we used $k = 20$ instead, as many Fraunhofer templates have $|V| < 30$.

Table 5.11 shows the results of the GRASP algorithm for some parameters k and $\boldsymbol{\delta}$ in comparison to the best known solution. The presented results do not substantially differ from the ones for any other parameter settings of k and $\boldsymbol{\delta}$. The average running time to find reasonably good solutions w.r.t. our application background (i.e. to find the optimal solution in most of the cases) is roughly less than ten seconds for the Fraunhofer templates.

Due to their larger size it is much more expensive to solve the NIST data with relatively high reliability. In this case the running times range from less than one minute to slightly more than three minutes. Preceding experiments for finding a good parameter setup indicated that $imp_{\max}$ has little impact on the solution quality. Nevertheless setting $imp_{\max} = 0$ corresponds to a completely arbitrary decision in the randomized construction process. In such a case the quality of the GRASP solely relies on the subsequent local search. On the other hand, too high values of $imp_{\max}$ lead to unnecessary high running times. When comparing setups of $imp_{\max}$ and it_{ls} which yield approximately the same running times, it turned out that higher values of it_{ls} yield better average solutions. Setting $rcl_{\max} \approx 10$ turned out to be a good choice, but the exact value is uncritical. Of course, higher values imply more diversity, which may enable to find the global optimum of difficult instances more quickly.

The results of the memetic algorithm are presented in Table 5.12. We used a population size $size_{\text{pop}} \in \{100, 200\}$, and a group size of four for the tournament selection. The crossover and mutation probability is set to one, i.e. each offspring is created by crossover and subsequent mutation. In each iteration a randomly selected candidate solution from the population was replaced by the newly generated one. Local improvement is performed for each newly created candidate solution. As mutation type 2 produced better overall results than mutation type 1, the former was used to create the results listed in Table 5.12. Replacing a randomly selected $\boldsymbol{t} \in T$ turned out to be advantageous over replacing the worst one.

Table 5.12 shows the results of 30 runs with 10000 and 30000 iterations for the Fraunhofer templates (population size 100); for the NIST templates we list the results for 10000 and 60000 iterations with a population size of 100 and 200, respectively. Again, the presented results are not essentially different to the ones for any other parameter settings of k and $\tilde{\boldsymbol{\delta}}$. The Fraunhofer data can be compressed reliably within 10000 iterations, which takes an average running time of roughly 2 seconds. Due to the larger number of points, the compression of the NIST data is computationally more expensive. At least 60000 iterations must be used in order to be able to produce reliable results. The respective running times are roughly 100 seconds.

Due to the relatively high running time of the randomized greedy construction heuristic (Algorithm 17) it is impossible to find a parameter setup which does not degenerate the GRASP to some extend but still keeps the overall running times small. Such a setup would either start local search on nearly arbitrary random solutions (i.e. $imp_{\max} = 0$) or extremely limit the number of local search iterations, i.e. $it_{\text{ls}} < 5$. Allowing slightly higher running times enables to find the optimal solution in almost every case (Fraunhofer), except two difficult instances. For the NIST data, sufficiently good solutions can be produced with adequate reliability in less than four minutes. In the case of the Fraunhofer data the MA is clearly superior to the GRASP, as it produces better solutions in less time. In contrast to GRASP it is also possible to create reasonable solutions (though with moderate quality) in less than 20 seconds. However, when allowing higher running times of up to five minutes, GRASP clearly outperforms the MA.

Table 5.11: Results and running times of the GRASP

instance	k	$\tilde{\boldsymbol{\delta}}$	m	$t[s]$	m_{best}	m_{avg}	σ_m	#b.s. [%]	$t_{\text{avg}}[s]$	m_{best}	m_{avg}	σ_m	#b.s. [%]	$t_{\text{avg}}[s]$	$\rho_{\text{b.s.}}$[%]
			B&C results		GRASP $it = 10, it_{ls} = 5, rcl_{\max} = 5, imp_{\max} = 0$					GRASP $it = 10, it_{ls} = 20, rcl_{\max} = 10, imp_{\max} = 10$					
ft-01	20	$\begin{pmatrix}30\\30\\30\end{pmatrix}$	3	79	**4**	4.00	0.00	100	1.57	3	3.87	0.34	13	7.97	6.10
ft-02			3	0	**4**	4.00	0.00	100	1.00	3	3.67	0.47	33	6.20	6.10
ft-03			3	85	3	3.00	0.00	100	2.03	3	3.00	0.00	100	10.03	6.10
ft-04			4	10	4	4.70	0.46	30	0.15	4	4.30	0.46	70	2.03	-0.87
ft-05			3	327	3	3.00	0.00	100	4.82	3	3.00	0.00	100	16.90	6.10
ft-07			4	14	4	4.00	0.00	100	0.33	4	4.00	0.00	100	4.00	-1.57
ft-08			4	13	4	4.00	0.00	100	0.93	4	4.00	0.00	100	3.93	-1.57
ft-09			4	24	4	4.00	0.00	100	0.33	4	4.00	0.00	100	4.00	-1.57
ft-10			3	36	3	3.13	0.34	87	1.00	3	3.00	0.00	100	6.30	6.10
ft-11			3	853	3	3.00	0.00	100	4.27	3	3.00	0.00	100	23.07	6.10
ft-12			3	52	3	3.00	0.00	100	1.00	3	3.00	0.00	100	5.93	6.62
ft-13			3	21	3	3.00	0.00	100	0.20	3	3.00	0.00	100	2.97	6.62
ft-14			3	275	3	3.00	0.00	100	2.00	3	3.00	0.00	100	8.73	6.10
ft-15			3	15	3	3.00	0.00	100	1.00	3	3.00	0.00	100	5.93	6.62
ft-16			3	282	3	3.00	0.00	100	2.97	3	3.00	0.00	100	16.53	6.10
ft-17			3	235	3	3.00	0.00	100	1.03	3	3.00	0.00	100	5.97	6.10
ft-18			3	823	3	3.00	0.00	100	5.57	3	3.00	0.00	100	21.93	6.10
ft-19			3	97	3	3.00	0.00	100	1.90	3	3.00	0.00	100	8.10	6.10
ft-20			3	0	3	3.00	0.00	100	0.30	3	3.00	0.00	100	3.00	6.10
			best known solution		$it = 5, it_{ls} = 5, rcl_{\max} = 10, imp_{\max} = 0$					$it = 5, it_{ls} = 10, rcl_{\max} = 10, imp_{\max} = 5$					
nist-b-01	40	$\begin{pmatrix}80\\80\\80\end{pmatrix}$	4	n/a	4	4.83	0.37	17	91.60	4	4.50	0.50	50	189.97	18.90
nist-b-02			4	n/a	4	4.37	0.48	63	78.47	4	4.00	0.00	100	180.17	18.90
nist-b-03			4	n/a	4	4.43	0.50	57	89.57	4	4.03	0.18	97	190.67	18.90
nist-b-04			5	n/a	5	5.27	0.44	73	24.70	5	5.00	0.00	100	53.97	13.75
nist-b-05			4	n/a	4	4.93	0.25	7	73.37	4	4.77	0.42	23	157.47	18.90
nist-g-01			4	n/a	4	4.83	0.37	17	71.23	4	4.37	0.48	63	159.57	18.90
nist-g-02			5	n/a	5	5.73	0.44	27	67.50	5	5.40	0.49	60	151.13	13.38
nist-g-03			4	n/a	4	4.17	0.37	83	94.50	4	4.00	0.00	100	189.57	18.90
nist-g-04			4	n/a	**5**	5.00	0.00	0	85.77	4	4.97	0.18	3	201.70	18.60
nist-g-05			5	n/a	5	5.13	0.34	87	40.13	5	5.03	0.18	97	64.57	13.75
nist-u-01			5	n/a	**6**	6.03	0.18	0	68.90	5	5.90	0.30	10	156.70	18.38
nist-u-02			5	n/a	5	5.00	0.00	100	69.87	5	5.00	0.00	100	152.37	13.75
nist-u-03			4	n/a	4	4.47	0.50	53	82.20	4	4.13	0.34	87	193.13	18.90
nist-u-04			5	n/a	5	5.17	0.37	86	44.53	5	5.00	0.00	100	74.83	13.75
nist-u-05			5	n/a	5	5.93	0.25	6	20.90	5	5.70	0.46	30	42.77	13.75

Table 5.12: Results and running times of the memetic algorithm

instance	k	$\tilde{\boldsymbol{\delta}}$	m	$t[s]$	m_{best}	m_{avg}	σ_m	#b.s. [%]	$t_{\text{avg}}[s]$	m_{best}	m_{avg}	σ_m	#b.s. [%]	$t_{\text{avg}}[s]$	$\rho_{\text{b.s.}}[\%]$
			B&C results		MA ($it = 10000, size_{pop} = 100$)					MA ($it = 30000, size_{pop} = 100$)					
ft-01			3	79	3	3.70	0.47	30	1.84	3	3.23	0.42	77	5.58	6.10
ft-02			3	0	3	3.04	0.18	97	1.38	3	3.00	0.00	100	4.29	6.10
ft-03			3	85	3	3.00	0.00	100	2.45	3	3.00	0.00	100	7.51	6.10
ft-04			4	10	4	4.00	0.18	100	1.01	4	4.00	0.00	100	3.08	-0.87
ft-05			3	327	3	3.00	0.00	100	2.23	3	3.00	0.00	100	6.48	6.10
ft-07			4	14	4	4.00	0.00	100	1.49	4	4.00	0.00	100	4.62	-1.57
ft-08			4	13	4	4.00	0.00	100	1.50	4	4.00	0.00	100	4.08	-1.57
ft-09			4	24	4	4.00	0.00	100	1.35	4	4.00	0.00	100	4.54	-1.57
ft-10			3	36	3	3.43	0.50	57	1.53	3	3.17	0.38	84	5.79	6.10
ft-11	20	$\begin{pmatrix}30\\30\\30\end{pmatrix}$	3	853	3	3.00	0.00	100	2.01	3	3.00	0.00	100	6.50	6.10
ft-12			3	52	3	3.00	0.00	100	2.26	3	3.00	0.00	100	4.86	6.10
ft-13			3	21	3	3.03	0.18	97	1.59	3	3.00	0.00	100	3.60	6.10
ft-14			3	275	3	3.16	0.37	83	1.15	3	3.00	0.00	100	7.04	6.10
ft-15			3	15	3	3.00	0.00	100	2.33	3	3.00	0.00	100	4.88	6.10
ft-16			3	282	3	3.00	0.00	100	1.69	3	3.00	0.00	100	7.50	6.10
ft-17			3	235	3	3.46	0.48	67	1.82	3	3.17	0.38	84	5.36	6.10
ft-18			3	823	3	3.00	0.00	100	2.75	3	3.00	0.00	100	8.50	6.10
ft-19			3	97	3	3.06	0.18	97	2.69	3	3.00	0.00	100	8.05	6.10
ft-20			3	0	3	3.00	0.00	100	1.58	3	3.00	0.00	100	4.79	6.10
			best known solution		MA ($it = 10000, size_{pop} = 100$)					MA ($it = 60000, size_{pop} = 200$)					
nist-b-01			4	n/a	**5**	5.10	0.31	0	24.66	**5**	5.00	0.00	0	98.90	13.75
nist-b-02			4	n/a	4	4.74	0.44	27	15.16	4	4.20	0.40	80	89.95	18.90
nist-b-03			4	n/a	4	4.60	0.50	40	17.94	4	4.10	0.31	90	104.43	18.90
nist-b-04			5	n/a	5	4.94	0.37	10	16.32	5	5.60	0.50	40	91.75	13.75
nist-b-05			4	n/a	**5**	5.94	0.51	0	16.86	4	5.00	0.26	3	93.73	18.90
nist-g-01			4	n/a	4	4.87	0.35	14	17.57	4	4.64	0.49	33	101.82	18.90
nist-g-02			5	n/a	**6**	6.17	0.38	0	21.03	5	5.97	0.18	3	117.78	13.38
nist-g-03	40	$\begin{pmatrix}80\\80\\80\end{pmatrix}$	4	n/a	4	4.80	0.41	46	16.36	4	4.30	0.47	70	94.29	18.90
nist-g-04			4	n/a	**5**	5.38	0.49	0	21.38	**5**	5.00	0.00	0	115.99	13.38
nist-g-05			5	n/a	5	6.57	0.57	3	16.97	5	5.74	0.45	26	92.97	13.75
nist-u-01			5	n/a	**6**	6.90	0.31	0	21.17	**6**	6.10	0.31	0	117.19	11.03
nist-u-02			5	n/a	5	5.60	0.49	40	17.54	5	5.00	0.00	100	100.68	13.75
nist-u-03			4	n/a	**5**	5.04	0.18	0	17.41	4	4.50	0.50	50	98.18	18.90
nist-u-04			5	n/a	5	5.84	0.38	17	17.75	5	5.24	0.43	76	98.87	13.75
nist-u-05			5	n/a	**6**	6.37	0.49	0	15.30	**6**	6.04	0.18	0	83.56	11.47

Table 5.13: Average absolute compression ratios

instances	$\tilde{\boldsymbol{\delta}}^{\mathsf{T}}$	k	ρ_{avg}	σ_ρ	$\tilde{\rho}_{\text{avg}}$	$\sigma_{\tilde{\rho}}$
Fraunhofer	$(30, 30)$	20	4.81	2.13	34.50	12.08
	$(30, 30, 30)$	20	4.41	3.27	36.08	14.34
	$(30, 30)$	30	7.94	1.73	18.20	7.68
	$(30, 30, 30)$	30	5.98	3.23	17.68	10.11
NIST	$(80, 80)$	20	10.46	1.52	82.45	2.18
	$(80, 80, 80)$	20	9.47	2.87	82.25	2.21
	$(80, 80)$	40	15.41	2.26	65.92	4.15
	$(80, 80, 80)$	40	15.26	2.97	65.84	4.38
	$(80, 80)$	60	15.85	0.58	48.79	6.29
	$(80, 80, 80)$	60	16.08	2.20	48.79	6.27
	$(80, 80)$	80	16.31	1.99	30.49	8.25
	$(80, 80, 80)$	80	15.49	3.26	29.76	9.03

5.10.5 Absolute Compression Ratios

In Section 5.10.2 we presented the compression ratios achievable on our test data by our compression model in comparison to the trivial representation of the k selected points, for which the size is given by equation (5.15). With respect to our particular application background of fingerprint template verification not all entities considered in equation (5.17) need to be encoded. In the following we describe which information can be neglected.

It is obviously not possible that two scans of the same finger yield exactly the same minutiae. Distortions like rotations, scalings and shifts are always involved. Matching algorithms thus have to account for such distortions in order to be able to reliably match two minutiae sets from the same finger with e.g. different coordinate offset values. Consequently we do not need to store such offset values in our compressed template as they are of no importance for the matching algorithm. Therefore $\text{ConstData}' = 0$. Moreover, as we do not necessarily need all minutiae in order to perform a reliable matching (see Section 5.10.3), the absolute compression ratios are much better than the values given in Section 5.10.2, where the ratios are always related to the simple storage of k minutiae. By absolute compression ratio $\tilde{\rho}$ we mean the ratio of the simple encoding size of the full template, which is given by equation (5.15), to the compressed template with k points given by equation (5.17) with $\text{ConstData}' = 0$.

Whereas on the one hand the relative compression ratios are of more importance for evaluating the magnitude of our compression model, the absolute compression ratios are of higher importance for practical purposes on the other hand. Table 5.13 summarizes the absolute compression ratios for the Fraunhofer and the NIST data.

5.11 Conclusions and Further Work

We presented a new approach for compressing fingerprint templates, or more generally d-dimensional data points. A subset of k data points is encoded via a directed spanning tree, for which the arcs are represented by indices to a set of template arcs plus correction vectors from a small domain. The selection of stored data points (nodes), the tree structure, and the template arc dictionary are optimized at the same time with the objective to find a feasible encoding requiring the least number of template arcs.

The general idea of compressing data by solving a graph-based combinatorial optimization problem is completely novel to our knowledge. In our approach, we determine a (large) set of candidate template arcs during preprocessing and then solve an extended variant of the minimum label spanning tree problem. An exact branch-and-cut based algorithm as well as heuristic approaches are investigated for the solution of the latter.

The compression ratios presented in Section 5.10.2 are not really compelling, as the data instances do not contain many structural or redundant information, which would enable higher compression. Nevertheless our experiments showed that our approach outperforms several other well known compression techniques on these data sets. When considering reasonable large values of k (i.e. $k \geq 20$) that keep the false non-match rates reasonable small, average absolute compression ratios of more than 30% can be achieved. Hence the presented method can be suitable for compressing minutiae templates for embedding them into images by watermarking techniques.

The presented branch-and-cut algorithm finds provably optimal solutions in a couple of seconds in many cases. Unfortunately there are also instances for which the running times are much too high for practical applications. For this reason we developed faster metaheuristics and compared their running times and solution qualities. The MA turned out to be very fast, in particular the Fraunhofer instances can be solved to optimality in less than 10 seconds in almost every case. In contrast to the MA, GRASP is able to find the best known solutions with very high probability also for the larger NIST instances. Nevertheless, the running times are slightly higher and range from less than one minute to more than three minutes in this case.

Moreover, it is possible to extend the model in order to eventually achieve higher compression ratios. One such extension would be to permit tree arcs to be represented by a sequence of multiple template arcs and corresponding correction vectors. This may on the one hand reduce the number of template arcs, but also implies further storage overhead for a more complex encoding and enhanced algorithmic complexity.

Chapter 6

Conclusions

There are two rules to success in life: Rule #1. Don't tell people everything you know.

What is understood, need not be discussed.
Lauren Adam.

ontrary to the quite ironic introductory quotations of this final chapter, we will now reflect the most important achievements of this thesis. It is however in the nature of things that not all considered methods and ideas gained incorporation into this thesis. It is though important to mention potential links to further research and open questions, which will be done at the end of each section in this chapter.

6.1 Heuristic Methods

In Chapter 3 a comprehensive review of existing work related to the minimum label spanning tree (MLST) problem has been presented. The methods consist of approximation algorithms as well as metaheuristic techniques. Selected methods, like MVCA and GRASP have been reimplemented in a slightly modified or improved version. Then we shifted attention towards the *ant colony optimization* (ACO) metaheuristic. Several pheromone models and algorithmic variants have been proposed and evaluated. Furthermore, local search has been considered as a local method to improve promising solutions created by artificial ants. The method showed decent performance with certain parameter settings, i.e. results of good quality could be obtained relatively fast. With improved parameter settings, implying longer running times, even better than the currently best known solutions could be found for some of the largest instances of a frequently used MLST benchmark set.

6.2 Exact Methods

In Chapter 4 we then focused on exact methods, primarily based on mixed integer programming (MIP) techniques. Again, we started by reviewing existing exact approaches, which can be roughly divided into methods based on MIP techniques and on the other hand methods based on the A* algorithm. In the following several MIP formulations have been proposed, including reproductions and reformulations of existing models, as well as new formulations that have not yet been considered for the MLST problem. We have shown how these formulations could be strengthened by new additional classes of valid inequalities, and how cutting-planes could be separated efficiently within a branch-and-cut algorithm. Furthermore it has been shown how lifted odd-hole inequalities can be applied to the MLST problem, implying a strengthening of the model. All proposed models have then been compared regarding their polyhedral properties. Although one of the newly proposed formulations has a weaker LP-relaxation, it permits a fast cutting-plane separation. In particular it has been shown that feasibility for any model based on directed connection cuts can be achieved by a cutting-plane separation based on a fast depth first search algorithm. Moreover we have proposed a branch-and-cut-and-price algorithm, starting with a reduced set of labels and pricing in promising additional labels, potentially improving the objective value, on demand. For the separation of the odd-hole inequalities we used a heuristic approach based on a mixed integer formulation.

The chapter has then been closed with the presentation of the results of extensive computational tests. The performance of the considered formulations together with certain strengthening methods is strongly dependent on the particular class of input instances. However, for most instances the new formulation has shown by far the best performance. In the case of sparse graphs and a relatively low number of labels the single-commodity flow formulation has also produced acceptable results. The directed cut formulation is only beneficial for instances where the number of labels lies in the same order of magnitude as the number of edges in the input graph. The proposed classes of inequalities consistently improve all considered formulations from a theoretical and practical perspective. Together with these inequalities the new formulations are able to solve larger instances to optimality than previously proposed methods. The resulting branch-and-cut algorithm is thus considered to be the state-of-the-art exact mathematical programming algorithm for the MLST problem.

The branch-and-cut-and-price approach is only beneficial for graphs with huge amounts of labels, but isolated single optima of very low objective value. The separation of odd-hole inequalities is particularly beneficial for certain classes of instances of low to medium density graphs and many labels. However, in this case a significant speedup has been achieved for many groups of instances. Furthermore their application has made it possible to solve more instances to optimality than without their separation.

The development of improved separation heuristics may even render their application more beneficial. Again, ant colony optimization appears to be auspicious for this task as pheromones are able to retain information about found odd-hole candidates that have been shown to be invalid. Advances regarding such faster separation heuristics are not

only important for the MLST problem, but also for the well-known set covering problem. Regarding exact algorithms for the MLST problem it seems also promising to consider certain constraint programming techniques. This is, however beyond the scope of this thesis, and could be part of future research.

6.3 Application Scenario

Chapter 5 was finally devoted to the application of an extended version of the MLST problem to a particular application scenario. The considered situation is the compression of fingerprint minutiae data to facilitate their embedding into passport images by watermarking techniques. For this purpose a new compression model for unordered sets has been proposed. To our best knowledge, this is the first approach to data compression directly exploiting the property that the order of the given elements needs not to be preserved. A further difference to existing compression approaches is the underlying combinatorial model.

Compression is achieved by encoding a subset of the given points by means of a spanning tree, which should be uniform regarding to the geometric information reflecting the relative positions of the two points associated to the edges. The edges themselves are encoded by a reference into a small set of "template arcs" and a small correction vector. Compression will be achieved if the total number of required template arcs is sufficiently small.

More precisely the compression model is given by a directed version of the MLST problem defined on complete graphs. The labels themselves correspond to the above mentioned template arcs and are derived in a preprocessing step based on restricted enumeration. To solve the resulting optimization problem several heuristic algorithms like greedy randomized adaptive search procedures and genetic algorithms, but also exact methods like branch-and-cut have been applied. Furthermore an improved exact branch-and-cut-and-price method has been described, which does not require the preprocessing step anymore, but rather creates new label variables on demand during the solution process. The pricing problem, i.e. the determination of new labels to be added to the model is solved by an algorithm based on a k-d tree data structure.

The last part of this chapter is devoted to a comprehensive computational study of the proposed compression approach and corresponding solution methods. A comparison to other compression techniques shows the superiority of this approach for the considered data set. Computational experiments show the aptitude of the proposed methods regarding the considered application.

Future work could consist of various extensions and improvements to the proposed compression model. In this work we have focused on the derivation of a minimal codebook w.r.t. a prespecified correction vector domain. The complementary view would be to minimize the correction vector domain according to a prespecified number of template arcs. Optimizing over both parameters (the number of template arcs and the correction vector domain) would eventually yield the best compression ratios, but also bears the challenge of solving an even more difficult optimization problem. Further improvements could for

instance be the introduction of intermediate nodes, which need not to be connected to the tree, but may be used to obtain solutions requiring less template arcs, and therefore yield higher compression ratios. Such trees are well known as Steiner trees. However, from the current point of view it remains unclear, how to determine such a set of intermediate nodes. A further interesting idea would be to derive a model for a lossy compression, such that the decoded vectors are close to, but not identical to the original ones. These ideas might be interesting starting points for further research into this direction, but are beyond the scope of this thesis.

Bibliography

[1] R. K. Ahuja, Ö. Ergun, J. B. Orlin, and A. P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1-3):75 – 102, 2002.

[2] T. Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, 1996.

[3] T. Bäck, D. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation*. IOP Publishing Ltd., 1997.

[4] J. Bang-Jensen and G. Gutin. *Digraphs, Theory, Algorithms and Applications*. Springer, 2002.

[5] F. Barahona and R. Anbil. The volume algorithm: producing primal solutions with a subgradient method. *Mathematical Programming*, 87(3):385–399, 2000.

[6] R. Battiti and G. Tecchiolli. The reactive tabu search. *INFORMS Journal on Computing*, 6(2):126–140, 1994.

[7] J. E. Beasley. Lagrangean relaxation. In *Modern heuristic techniques for combinatorial problems*, pages 243–303, New York, NY, USA, 1993. Wiley.

[8] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

[9] D. Bertsimas and R. Weismantel. *Optimization Over Integers*. Dynamic Ideas, 2005.

[10] C. Blum, M. J. B. Aguilera, A. Roli, and M. Sampels, editors. *Hybrid Metaheuristics, An Emerging Approach to Optimization*, volume 114 of *Studies in Computational Intelligence*. Springer, 2008.

[11] T. Brüggemann, J. Monnot, and G. J. Woeginger. Local search for the minimum label spanning tree problem with bounded color classes. *Oper. Res. Lett.*, 31(3):195–201, 2003.

[12] M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, Digital SRC Research Report, 1994.

[13] M. Captivo, J. C. Clímaco, and M. M. Pascoal. A mixed integer linear formulation for the minimum label spanning tree problem. *Computers & Operations Research*, 36(11):3082 – 3085, 2009.

[14] A. Cayley. A theorem on trees. *Quarterly Journal of Pure and Applied Mathematics*, 23:376–378, 1889.

[15] R. Cerulli, A. Fink, M. Gentili, and S. Voß. Metaheuristics comparison for the minimum labelling spanning tree problem. In *Operations Research/Computer Science Interfaces Series*, volume 29, pages 93–106. Springer US, 2005.

[16] R.-S. Chang and S.-J. Leu. The minimum labeling spanning trees. *Information Processing Letters*, 63(5):277–282, 1997.

[17] Y. Chen, N. Cornick, A. O. Hall, R. Shajpal, J. Silberholz, I. Yahav, and B. L. Golden. *Operations Research/Computer Science Interfaces*, chapter Comparison of Heuristics for Solving the Gmlst Problem, pages 191–217. Springer, 2008.

[18] B. V. Cherkassky and A. V. Goldberg. On implementing the push-relabel method for the maximum flow problem. *Algorithmica*, 19(4):390–410, 1997.

[19] A. M. Chwatal and G. R. Raidl. Applying branch-and-cut for compressing fingerprint templates (short abstract). In *Proceedings of the European Conference on Operational Research (EURO) XXII*, Prague, Czech Republic, 2007.

[20] A. M. Chwatal and G. R. Raidl. Fitting multi-planet transit models to photometric time-data series by evolution strategies. In *GECCO '10: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, Portland, Oregon, 2010. ACM Press. (to appear).

[21] A. M. Chwatal and G. R. Raidl. Solving the minimum label spanning tree problem by ant colony optimization. In *Proceedings of GEM 2010*, Las Vegas, Nevada, 2010. (to appear).

[22] A. M. Chwatal and G. R. Raidl. Solving the minimum label spanning tree problem by mathematical programming techniques. Technical Report TR 186-1-10-03, Vienna University of Technology, Institute of Computer Graphics and Algorithms, June 2010.

[23] A. M. Chwatal, G. R. Raidl, and O. Dietzel. Compressing fingerprint templates by solving an extended minimum label spanning tree problem. In *Proceedings of the Seventh Metaheuristics International Conference (MIC)*, Montreal, Canada, 2007.

[24] A. M. Chwatal, G. R. Raidl, and K. Oberlechner. Solving a k-node minimum label spanning arborescence problem to compress fingerprint templates. *Journal of Mathematical Modelling and Algorithms*, 8:293–334, 2009.

[25] A. M. Chwatal, C. Thöni, K. Oberlechner, and G. R. Raidl. A branch-and-cut-and-price approach to the k-node minimum label spanning tree problem. Technical report, Vienna University of Technology, Institute of Computer Graphics and Algorithms, June 2010. (in final preparation).

[26] L. Clarke. On Cayley's formula for counting trees. *Journal of the London Mathematical Society*, s1-33(4):471–474, 1958.

[27] R. K. Congram. *Polynomially searchable exponential neighbourhoods for sequencing problems in combinatorial optimisation.* PhD thesis, University of Southampton, Department of Mathematics, 2000.

[28] S. Consoli, K. Darby-Dowman, N. Mladenovic, and J. Moreno-Pérez. Solving the minimum labelling spanning tree problem using hybrid local search. Technical report, n/a, 2007.

[29] S. Consoli, K. Darby-Dowman, N. Mladenovic, and J. Moreno-Pérez. Heuristics based on greedy randomized adaptive search and variable neighbourhood search for the minimum labelling spanning tree problem. *European Journal of Operational Research*, 196:440–449, 2009.

[30] S. Consoli, J. A. Moreno, N. Mladenovic, and K. Darby-Dowman. Constructive heuristics for the minimum labelling spanning tree problem: a preliminary comparison. Technical report, DEIOC Technical Report., 2006.

[31] S. A. Cook. The complexity of theorem-proving procedures. In *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, New York, NY, USA, 1971. ACM.

[32] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver. *Combinatorial Optimization.* Wiley-Interscience, 1997.

[33] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* The MIT Press, 2nd edition, 2001.

[34] G. Cornuéjols. Revival of the gomory cuts in the 1990's. *Annals OR*, 149(1):63–66, 2007.

[35] G. Cornuéjols and A. Sassano. On the 0, 1 facets of the set covering polytope. *Math. Program.*, 43(1):45–55, 1989.

[36] G. Dantzig. *Linear Programming and Extensions.* Princeton University Press, August 1998.

[37] G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8(1):101–111, 1960.

[38] J. Denzinger, F. Informatik, U. Kaiserslautern, and T. Offermann. On cooperation between evolutionary algorithms and other. In *Search Paradigms, Proc. CEC-99*, pages 2317–2324, 1999.

[39] G. Desaulniers, J. Desrosiers, and M. M. Solomon, editors. *Column Generation.* Springer, 2005.

[40] J. Desrosiers, F. Soumis, and M. Desrochers. Routing with time windows by column generation. *Networks*, 14(4):545–565, 1984.

[41] O. Dietzel. Combinatorial Optimization for the Compression of Biometric Templates. Master's thesis, Vienna University of Technology, Institute of Computer Graphics and Algorithms, May 2008. supervised by G. Raidl and A. Chwatal.

[42] M. Dorigo, V. Maniezzo, and A. Colorni. Positive feedback as a search strategy. Technical Report 91-016, Politecnico di Milano, Italy, 1991.

[43] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, 26:29–41, 1996.

[44] M. Dorigo and T. Stützle. *Ant Colony Optimization*. The MIT Press, 2004.

[45] L. Euler. Solutio problematis ad geometriam situs pertinentis. *Opera Omnia*, 7:128–140, 1736.

[46] J. D. Farmer, N. H. Packard, and A. S. Perelson. The immune system, adaptation, and machine learning. *Phys. D*, 2(1-3):187–204, 1986.

[47] T. Feo and M. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[48] M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, 2003.

[49] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman & Co Ltd, January 1979.

[50] Garris M. D. and McCabe R. M. Nist special database 27: Fingerprint minutiae from latent and matching tenprint images. Technical report, National Institute of Standards and Technology, 2000.

[51] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 9(6):849–859, 1961.

[52] F. Glover. Future paths for integer programming and links to artificial intelligence. *Comput. Oper. Res.*, 13(5):533–549, 1986.

[53] F. Glover. Tabu search – part i. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[54] D. Goldberg. *Genetic Algorithms in search, optimization and machine learning*. Addison Wesley, 1989.

[55] C. Grosan, C. Grosan, A. Abraham, and H. Ishibuchi. *Hybrid Evolutionary Algorithms*. Springer Publishing Company, Incorporated, 2007.

[56] P. Hansen and N. Mladenovic. Variable neighborhood search, 1997.

[57] J. M. Harris, J. L. Hirst, and M. J. Mossinghoff. *Combinatorics and graph theory*. Springer, 2000.

[58] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100 –107, july 1968.

[59] D. S. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and vlsi. *Journal of the ACM*, 32(1):130–136, 1985.

[60] J. Holland. *Adaption in natural and artificial systems*. The University of Michigan Press, 1975.

[61] ILOG Concert Technology, CPLEX. ILOG. http://www.ilog.com. Version 12.0.

[62] ILOG Concert Technology, CPLEX. ILOG. http://www.ilog.com. Version 11.0.

[63] A. Jain and U. Uludag. Hiding fingerprint minutiae in images. In *Proceedings of Third Workshop on Automatic Identification Advanced Technologies*, pages 97–102, 2002.

[64] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[65] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE Service Center, 1995.

[66] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[67] K. Kostikas and C. Fragakis. Genetic programming applied to mixed integer programming. In *Genetic Programming - EuroGP 2004, volume 3003 of LNCS*, pages 113–124. Springer, 2004.

[68] J. R. Koza. *Genetic Programming – On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[69] S. O. Krumke and H.-C. Wirth. On the minimum label spanning tree problem. *Information Processing Letters*, 66(2):81–85, 1998.

[70] J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, February 1956.

[71] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Dover Publications, 2001.

[72] J. Lee. *A First Course in Combinatorial Optimization*. Cambridge University Press, 2004.

[73] Library for Efficient Datastructures and Algorithms (LEDA). Algorithmics Solutions Software GmbH. http://www.algorithmic-solutions.com/. Version 5.1.

[74] M. E. Lübbecke and J. Desrosiers. Selected topics in column generation. *Operations Research*, 53:1007–1023, 2002.

[75] I. J. Lustig and J.-F. Puget. Program does not equal program: Constraint programming and its relationship to mathematical programming. *Interfaces*, 31(6):29–53, 2001.

[76] T. Magnanti and L. Wolsey. Optimal trees. *Handbook in Operations Research and Management Science*, Network Models:503–615, 1995.

[77] D. Maltoni, D. Maio, A. K. Jain, and S. Prabhakar. *Handbook of fingerprint recognition*. Springer, 2003.

[78] V. Maniezzo, T. Stützle, and S. Voß, editors. *Matheuristics – Hybridizing Metaheuristics and Mathematical Programming*, volume 10 of *Annals of Information Systems*. Springer, 2010.

[79] O. D. Merle, D. Villeneuve, J. Desrosiers, and P. Hansen. Stabilized column generation. *Discrete Math*, 194:229–237, 1997.

[80] Z. Michalewicz. *Genetic algorithms + data structures = evolution programs (2nd, extended ed.)*. Springer-Verlag New York, Inc., New York, NY, USA, 1994.

[81] C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer programming formulation of traveling salesman problems. *J. ACM*, 7(4):326–329, 1960.

[82] N. Mladenović and P. Hansen. Variable neighborhood search. *Comput. Oper. Res.*, 24(11):1097–1100, 1997.

[83] A. Moffat, R. M. Neal, and I. H. Witten. Arithmetic coding revisited. *ACM Transactions on Information Systems*, 16(3):256–294, 1998.

[84] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience, November 1999.

[85] J. Nummela and B. A. Julstrom. An effective genetic algorithm for the minimum-label spanning tree problem. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 553–558, New York, NY, USA, 2006. ACM.

[86] K. Oberlechner. Solving a k-node minimum label spanning arborescence problem with exact and heuristic methods. Master's thesis, Vienna University of Technology, Institute of Computer Graphics and Algorithms, 2010 (in final preparation). supervised by G. Raidl and A. Chwatal.

[87] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover, 1998.

[88] R. C. Prim. Shortest connection networks and some generalizations. *Bell System Technology Journal*, 36:1389–1401, 1957.

[89] J. Puchinger and G. R. Raidl. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In *IWINAC (2)*, pages 41–53, 2005.

[90] G. R. Raidl. An improved genetic algorithm for the multiconstrained 0-1 knapsack problem. In *Proceedings of the 5th IEEE International Converence on Evolutionary Computation*, pages 207–211. IEEE Press, 1998.

[91] G. R. Raidl. A unified view on hybrid metaheuristics. In F. Almeida et al., editors, *Proceedings of the Hybrid Metaheuristics Workshop*, volume 4030 of *LNCS*, pages 1–12. Springer, 2006.

[92] G. R. Raidl and A. Chwatal. Fingerprint template compression by solving a minimum label k-node subtree problem. In E. Simos, editor, *Numerical Analysis and Applied Mathematics*, volume 936 of *AIP Conference Proceedings*, pages 444–447. American Institute of Physics, 2007.

[93] G. R. Raidl and J. Puchinger. Combining (integer) linear programming techniques and metaheuristics for combinatorial optimization. In C. Blum, M. J. B. Augilera, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics – An Emerging Approach for Combinatorial Optimization*, volume 114 of *Studies in Computational Intelligence*, pages 31–62. Springer, 2008.

[94] G. R. Raidl, J. Puchinger, and C. Blum. *Handbook of Metaheuristics*, volume 2nd edition, chapter Metaheuristic hybrids. Springer, 2009 (accepted).

[95] M. G. C. Resende. Greedy randomized adaptive search procedures (grasp). *Journal of Global Optimization*, 6:109–133, 1999.

[96] F. Rossi, P. v. Beek, and T. Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006.

[97] A. Saleh and R. Adhami. Curvature-based matching approach for automatic fingerprint identification. In *Proceedings of the Southeastern Symposium on System Theory*, pages 171–175, 2001.

[98] K. Sayood. *Introduction to Data Compression*. Morgan Kaufmann Publishers Inc., USA, third edition, 2006.

[99] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1998.

[100] A. Schrijver. *Combinatorial Optimization*. Springer, 2003.

[101] SCIP – Solving Constraint Integer Programs. Konrad-Zuse-Zentrum für Informationstechnik Berlin. http://scip.zib.de/. Version 1.2.

[102] T. Stützle and H. H. Hoos. Max-min ant system. *Future Gener. Comput. Syst.*, 16(9):889–914, 2000.

[103] S. Talukdar, L. Baerentzen, A. Gove, and P. S. de Souza. Asynchronous teams: Cooperation schemes for autonomous agents. *J. Heuristics*, 4(4):295–321, 1998.

[104] C. Thöni. Compressing fingerprint templates by solving the k-node minimum label spanning arborescence problem by branch-and-price. Master's thesis, Vienna University of Technology, Institute of Computer Graphics and Algorithms, February 2010. supervised by G. Raidl and A. Chwatal.

[105] F. Vanderbeck and L. A. Wolsey. An exact algorithm for ip column generation. *Operations Research Letters*, 19(4):151 – 159, 1996.

[106] L. R. Varshney and V. K. Goyal. Benefiting from disorder: Source coding for unordered data. *arXiv*, abs/0708.2310, 2007.

[107] V. V. Vazirani. *Approximation Algorithms*. Springer, 2001.

[108] J. S. Vitter. Design and analysis of dynamic huffman codes. *Journal of the ACM*, 34(4):825–845, 1987.

[109] C. Walshaw. Multilevel refinement for combinatorial optimisation: Boosting metaheuristic performance. In C. Blum, M. J. B. Augilera, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics – An Emerging Approach for Combinatorial Optimization*, volume 114 of *Studies in Computational Intelligence*, pages 261–289. Springer, 2008.

[110] Y. Wan, G. Chert, and Y. Xu. A note on the minimum label spanning tree. *Information Processing Letters*, 84(2):99–101, 2002.

[111] P. Wolfe. The simplex method for quadratic programming. *Econometrica*, 27:382–398, 1959.

[112] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*, 1(1):67–82, 1997.

[113] Y. Xiong. *The Minimum Labeling Spanning Tree Problem and Some Variants*. PhD thesis, Robert H. Smith School of Business, 2005.

[114] Y. Xiong, B. Golden, and E. Wasil. A one-parameter genetic algorithm for the minimum labeling spanning tree problem. *IEEE Transactions on Evolutionary Computation*, 9(1):55–60, 2 2005.

[115] Y. Xiong, B. Golden, and E. Wasil. Improved heuristics for the minimum label spanning tree problem. In *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*, volume 10, December 2006.

[116] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.

[117] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978.

Appendix A

Curriculum Vitae

Personal Data

Name	Andreas M. Chwatal
Date of birth	11. May 1979 in Vienna, Austria
Citizenship	Austria

Education

Jan. 2008	Graduation to *Magister (Mag.rer.soc.oec.)* of Master Program *Informatikmanagement*
May 2006	Distinguished Young Alumnus Award from the Faculty of Informatics
since Feb. 2006	PhD in Computer Sciences in the field of Algorithms and Combinatorial Optimization
Feb. 2006	Graduation to *Diplomingenieur (Dipl.-Ing.)*, which is comparable to *Master of Science (MSc)*, of Master Program *Computational Intelligence*
2005	Finished first part of Astronomy studies
2003	Graduation to *Bachelor of Science (BSc)* of Bachelor Program *Software and Information Engineering*
1997 – 2008	Studies of *Computer Sciences* and *Astronomy* on Vienna University of Technology and University of Vienna
1989 – 1997	Gymnasium (Secondary School) "BRG XIV" in Vienna
1985 – 1989	Elementary school "Josefinum" in Vienna

Employment History

Feb. 2006 up to present	Research and Teaching Assistant at the Algorithms and Datastructures Group of the Institute of Computer Graphics and Algorithms, *Vienna University of Technology*
Jan. 2000 – Feb. 2006	Part-time employee at *Hewlett-Packard*, Department CRM EMEA/Austria. Responsibilities: Database Administration, Intranet Application Development, EMEA Project Management;
May 1998 – Sept. 1999	Civilian Service at *Arbeiter Samariter Bund* as paramedic. Subsequent full-time employment for four months.

Teaching Experience

Algorithms and Datastructures

Advanced Algorithms and Datastructures

Heuristic Optimization

Efficient Algorithms

Algorithms on Graphs

Seminar with Bachelor Thesis

Master and Bachelor Thesis supervision

Other activities

Aug. 2008	Member of the Local Organizing Committee, *Dark Sky 2008*, Kuffner-Observatory Vienna
May 2006	Member of the Local Organizing Committee, *Meeting on Asteroids and Comets in Europe 2006*, Kuffner-Observatory Vienna
Aug. 2002	Observations/telescope support with the Berlin Exoplanet Search Telescope, Karl Schwarzschild Observatory in Tautenburg (Germany).

Vienna, June 18, 2010

Appendix B

List of Publications

B.1 Refereed Conference Proceedings and Journal Articles

- A. M. Chwatal, K. Oberlechner, G. R. Raidl, C. Thöni. A Branch-and-Cut-and-Price Approach to the k-Node Minimum Label Spanning Arborescence Problem. *Technical Report (in final preparation)*.
- A. M. Chwatal, G. R. Raidl. Solving the Minimum Label Spanning Tree Problem by Mathematical Programming Techniques. Vienna University of Technology, Institute of Computer Graphics and Algorithms. June 2010. *Technical Report TR 186-1-10-03*.
- A. M. Chwatal, G. R. Raidl. Solving the Minimum Label Spanning Tree Problem by Ant Colony Optimization. *International Conference on Genetic and Evolutionary Methods – GEM 2010*, Las Vegas, Nevada, 2010
- A. M. Chwatal, G. R. Raidl. Fitting Multi-Planet Transit Models to Photometric Time-Data Series by Evolution Strategies. *Genetic and Evolutionary Computation Conference – GECCO 2010*, Portland, Oregon, 2010
- A. M. Chwatal, G. R. Raidl. Solving a k-node minimum label spanning arborescence problem to compress fingerprint templates. *Journal of Mathematical Modelling and Algorithms*, 8:293–334, 2009.
- A. M. Chwatal, N. Musil, and G. R. Raidl. Solving a multi-constrained network design problem by lagrangean decomposition and column generation. Pisa, Italy, April 2009. International Network Optimization Conference 2009.
- A. M. Chwatal and G. R. Raidl. Fitting rectangular signals to time series data by metaheuristic algorithms. In R. M.-D. et al., editor, *Computer Aided Systems Theory – EUROCAST 2009*, volume 5717 of *LNCS*, pages 649–656, 2009. Twelfth International Conference on Computer Aided Systems Theory

- A. Mayer, C. Nothegger, A. M. Chwatal, and G. R. Raidl. Solving the post enrolment course timetabling problem by ant colony optimization. In E. Burke, M. Gendreau, B. Gendron, and L.-M. Rousseau, editors, *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling*, Montreal, Canada, 2008.
- A. M. Chwatal, G. R. Raidl, and O. Dietzel. Compressing fingerprint templates by solving an extended minimum label spanning tree problem. In *Proceedings of the Seventh Metaheuristics International Conference (MIC)*, Montreal, Canada, 2007.
- G. R. Raidl and A. Chwatal. Fingerprint template compression by solving a minimum label k-node subtree problem. In E. Simos, editor, *Numerical Analysis and Applied Mathematics*, volume 936 of *AIP Conference Proceedings*, pages 444–447. American Institute of Physics, 2007.
- A. M. Chwatal and G. R. Raidl. Determining Orbital Elements of Extrasolar Planets by Evolution Strategies In R. M.-D. et al., editor, *Computer Aided Systems Theory – EUROCAST 2007*, volume 4739 of *LNCS*, pages 870–877, 2007. Eleventh International Conference on Computer Aided Systems Theory
- A. Chwatal. Bestimmung der Bahnelemente von extrasolaren Planeten aufgrund von Radialgeschwindigkeitsmessdaten mittels evolutionärer Algorithmen. Master's thesis, Vienna University of Technology, Institute of Computer Graphics and Algorithms, January 2006. supervised by G. Raidl.

B.2 Presentations and Posters

- A. M. Chwatal, G. Wuchterl, and G. R. Raidl. Fitting multi-planet transit models to corot time-data series by evolutionary algorithms. Poster presentation, First CoRoT International Symposium, 02 2009.
- A. M. Chwatal and G. R. Raidl. Applying branch-and-cut for compressing fingerprint templates (short abstract). In *Proceedings of the European Conference on Operational Research (EURO) XXII*, Prague, Czech Republic, 2007.