TECHNISCHE
UNIVERSITÄT
WIEN

VIENNA
UNIVERSITY OF
TECHNOLOGY

DIPLOMARBEIT

Subgradient Optimization Based Lagrangian Relaxation and Relax-and-Cut Approaches for the Bounded-Diameter Minimum Spanning Tree Problem

Ausgeführt am
Institut für Computergraphik und Algorithmen
der Technischen Universität Wien

unter der Anleitung von
Univ.-Prof. Dipl.-Ing. Dr.techn. Günther Raidl
und
Univ.Ass. Dipl.-Ing. Martin Gruber

durch
Peter Putz
Neudeggergasse 10/11, 1080 Wien
Matrikelnr. 9726571
p.putz@yahoo.de

Wien, Oktober 2007

Abstract

The Bounded-Diameter Minimum Spanning Tree (BDMST) problem is a hard combinatorial optimization problem with applications in network design. In this thesis, I present two Lagrangian relaxation approaches for the BDMST problem with even diameter bound in order to obtain lower bounds as well as heuristic solutions. The first Lagrangian relaxation approach is based on the so called Predecessor-Depth model. In this model a solution is formulated by predecessor variables and depth variables. The relaxed constraints of this model can be listed explicitly. To solve the Lagrangian duals, Subgradient Optimization is employed. The second Lagrangian relaxation approach is based on the so called Predecessor-Jump model. In this model a solution is formulated by predecessor variables and jump constraints. There are exponentially many relaxed constraints in this model. Therefore they cannot be listed explicitly but are separated dynamically. Two different strategies to separate jump constraints are presented. To solve the Lagrangian duals a Relax-and-Cut approach is developed and Subgradient Optimization is employed.

A set of benchmark instances used in the literature serve as input for computational experiments. The Lagrangian relaxation approach based on the Predecessor-Jump model produces significantly better lower bounds than the approach based on the Predecessor-Depth model. Subsequently, I compare the computed lower bounds to results from Gruber 2006. The lower bounds produced with the Lagrangian relaxation approach on the Predecessor-Jump model are, with one exception, always better than the values of the LP relaxation with cuts from Gruber 2006 but require substantially more time to compute. For two of the instances the optimal objective value is reached.

Zusammenfassung

Das Problem des minimalen Spannbaums mit beschränktem Durchmesser (BDMST) ist ein schweres kombinatorisches Optimierungsproblem mit Anwendungen in der Netzwerkplanung. In der vorliegenden Diplomarbeit präsentiere ich zwei Lagrange Relaxierungsansätze für das BDMST Problem mit geradem Durchmesser, um untere Schranken und heuristische Lösungen zu finden. Der erste Lagrange Relaxierungsansatz basiert auf dem sogenannten Predecessor-Depth Modell. In diesem Modell wird eine Lösung mittels Predecessor Variablen und Depth Variablen formuliert. Die relaxierten Nebenbedingungen können explizit aufgelistet werden. Um das Lagrange duale Problem zu lösen, wird das Subgradientenverfahren eingesetzt. Der zweite Lagrange Relaxierungsansatz basiert auf dem sogenannten Predecessor-Jump Modell. In diesem Modell wird eine Lösung mittels Predecessor Variablen und Jump Nebenbedingungen formuliert. Da es exponentiell viele Jump-Nebenbedingungen gibt, können sie nicht explizit aufgelistet werden sondern werden dynamisch separiert. Zwei verschiedene Strategien zur Separierung von Jump-Nebenbedingungen werden präsentiert. Um das Lagrange duale Problem zu lösen wird ein Relax-and-Cut Ansatz entwickelt und das Subgradientenverfahren eingesetzt.

Die entwickelten Ansätze wurden mit, aus der Literatur bekannten, Instanzen getestet. Der auf dem Predecessor-Jump Modell basierende Lagrange Relaxierungsansatz liefert signifikant bessere untere Schranken verglichen mit dem Lagrange Relaxierungsansatz, der auf dem Predecessor-Depth Modell basiert. Weiters vergleiche ich die berechneten unteren Schranken mit Ergebnissen aus Gruber 2006. Die unteren Schranken, die mittels des Predecessor-Jump Modells erzielt wurden, sind mit einer Ausnahme immer besser als die LP Relaxierungswerte mit diversen Cuts aus Gruber 2006, brauchen allerdings deutlich mehr Berechnungszeit. Für zwei dieser Instanzen wird der optimale Zielfunktionswert erreicht.

Contents

Chapter 1

Introduction

Consider the design of a computer network. The locations of the computers are fixed and there is a number of potential links connecting the computers. The cost to build each link are known. A typical network design problem is to choose a set of links, such that all computers are connected and that the cost to build the network is as low as possible. This is more formally expressed in the following definition.

Definition 1 (Minimum Spanning Tree)
We are given an undirected connected graph $G = (V, E)$ and positive edge cost $c_{v,w} \in \mathbb{R}_{\geq 0}\ \forall (v, w) \in E$. The *Minimum Spanning Tree* problem (MST) asks for an acyclic spanning subgraph $T = (V, E_T)$, $E_T \subseteq E$ with minimal total edge cost $\sum_{(v,w)\in E_T} c_{v,w}$.

The MST problem is a well studied combinatorial optimization problem. It can be solved in polynomial time by either Kruskal's or Prim's algorithm (see [CLR00, p. 498 ff.]). Assume now that we want the network to satisfy an additional constraint. The maximum number of routers between two computers in the network shall not exceed a certain limit.[1] Formally this is expressed by the following definition.

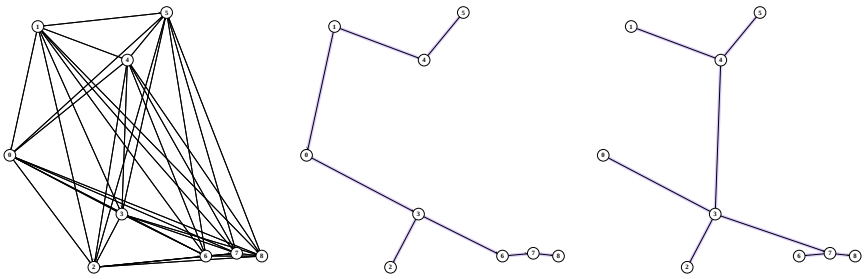
Definition 2 (Bounded-Diameter Minimum Spanning Tree)
We are given an undirected connected graph $G = (V, E)$, positive edge cost $c_{v,w} \in \mathbb{R}_{\geq 0}\ \forall (v, w) \in E$, and a positive integer D. The *Bounded-Diameter Minimum Spanning Tree* problem (BDMST) asks for a minimum spanning tree $T = (V, E_T)$, $E_T \subseteq E$, such that the path between two arbitrary nodes $v, w \in V$ in T does not consist of more than D edges.

Note that a BDMST has a "centre". If D is odd, the centre is an edge. This means there is an edge (v, w) such that the path from one of the nodes

[1] Note that limiting the number of routers by k is the same as limiting the number of links by $k + 1$.

v or w to any other node does not consist of more than $\lfloor \frac{D}{2} \rfloor$ edges. If D is even, the centre is a node. This means there is a node r such that the path from r to any other node does not consist of more than $\frac{D}{2}$ edges. This node r can be seen as the root of the BDMST.

The BDMST problem is also referred to as the *Diameter-Constrained Minimum Spanning Tree* (DCMST) problem (see [ADG00]). Figure 1.1 shows a graph along with its MST and BDMST.



(a) Instance (b) MST (c) BDMST for $D = 4$

Figure 1.1: (a) A complete graph with 9 nodes. The edge costs are proportional to the length of the edges. (b) A MST with a diameter of 7 and cost of 222. (c) A BDMST that satisfies the diameter bound of $D = 4$ and has cost of 240.

Another combinatorial optimization problem that is very similar to the BDMST problem is defined as follows.

Definition 3 (Hop-Constrained Minimum Spanning Tree)
We are given an undirected connected graph $G = (V, E)$, positive edge cost $c_{v,w} \in \mathbb{R}_{\geq 0}\ \forall (v, w) \in E$, a positive integer H, and a root node $r \in V$. The *Hop-Constrained Minimum Spanning Tree* problem (HMST) asks for a minimum spanning tree $T = (V, E_T)$, $E_T \subseteq E$, such that the path from r to an arbitrary node $v \in V$ in T does not consist of more than H edges (hops).

This means that a HMST is a tree rooted at r and with a height not greater than H. The HMST problem can be seen as a specialization of the BDMST problem with an even diameter bound and with a predefined root node. More specifically, asking for a BDMST with an even D and an additionally specified root node, is the same as asking for a HMST with $H = \frac{D}{2}$.

Let $n = |V|$ denote the number of nodes in the graph. The BDMST problem is NP-complete if the diameter bound is within the range $4 \leq D \leq (n - 2)$ and not all the edge costs are equal (see [GJ79, p. 206]).

In this work, I develop two Lagrangian relaxation approaches for the BDMST problem with an even diameter bound. These relaxations are used

to build two solvers. With these solvers I compute lower bounds for publicly available BDMST instances that have previously been investigated in the literature. I compare the results of my Lagrangian relaxation approaches to each other and subsequently to results from [Gru06]. For some of the instances from the literature the optimum objective values are known. Therefore, it can be analyzed how close the lower bounds are to the optimum values. For other instances no optimum objective values are known. However, there are heuristic solutions and lower bounds available. Consequently, it is possible to analyze, whether the new approaches can improve the known lower bounds.

The remainder of this thesis is structured as follows: Section 1.1 reviews real world problems related to the BDMST problem. Section 2 discusses the mathematical and algorithmic basis for the later sections. Section 3 presents an overview of some models and algorithms that have been published for the BDMST and HMST problems. Section 4 introduces my new Lagrangian relaxation approaches to compute lower bounds. Section 5 gives details on the implementation. Section 6 shows the computational results achieved with my implementation. Finally, section 7 summarizes my work and suggests directions for future research.

1.1 Applications

Woolston and Albin describe in [WA88] a heuristic for *The Design of Centralized Networks with Reliability and Availability Constraints*. That problem is about deciding upon the design of a computer network in which a central computing resource must be connected to several clients. The network shall minimize the cost of the communication links while satisfying additional constraints. *Availability* is the probability that a client can initiate a session on the server. One constraint demands that the availability is above a certain minimum level. *Reliability* is the probability that a session will not be interrupted by a failing transmission facility. One constraint demands that the reliability is above a certain minimum level. As starting solutions for their heuristic they used either star layouts or minimum spanning tree layouts. Their computational results on networks with 5 to 25 nodes showed significant differences in the cost of the final layout according to which initial layout was used. Availability and reliability are proportional to the number of nodes on the path from the client to the server. Therefore using an initial HMST layout might be a promising alternative.

Bala, Petropoulos and Stern discuss in [BPS93] the topic of *Multicasting in a Linear Lightwave Network*, which is a specific network design problem. They explicitly mention that a good network layout for their purposes would be built of trees with a small diameter. They propose a heuristic to find trees that should tend to smaller diameters. In opposition to this heuristic an algo-

rithm to solve the BDMST problem could guarantee a certain diameter and might therefore be an interesting approach for this network design problem.

Raymond presents an algorithm for distributed mutual exclusion in a computer network with tree topology in [Ray89]. A computer in this network that wants to enter its critical section has to send a request over the network and await a response. Raymond's algorithm accomplishes to limit the number of messages that have to be transmitted for a request and a response with two times the diameter of the network. In a network with BDMST topology this limit would be a constant.

Bookstein and Klein address the issue of data compression in [BK91]. The data is partitioned into packets and only differences between packets need to be stored. They map the problem of selecting which difference-relations to store, to the problem of finding a minimum spanning tree. The decompression of data packets is proportional to the diameter of this tree. They explicitly mention that "it would be desirable to create trees $[\cdots]$ constrained to have a small depth". An algorithm to solve BDMST could limit the cost of decompression.

Chapter 2

Methods

The subsequent discussion introduces the mathematical and algorithmic concepts relevant for the Lagrangian relaxation approaches. These concepts form the building blocks for the later sections.

2.1 Integer Linear Programming

A *Linear Program* (LP) is about finding an $x \in \mathbb{R}^n$ that satisfies a set of m linear constraints. It is usually expressed like this

$$\begin{aligned} \text{minimize} \quad & z = tx && (2.1)\\ \text{s.t.} \quad & Qx \geq g && (2.2)\\ & x \in \mathbb{R}^n && (2.3) \end{aligned}$$

where $t \in \mathbb{R}^n$, and Q is an $(m \times n)$ matrix over $\mathbb{R}$, and $g \in \mathbb{R}^m$.

Every $x \in \mathbb{R}^n$ is called a *solution* of the LP. If x satisfies the constraints (2.2) it is a *feasible solution*. The *objective value* of the LP for a given x is z. Among all feasible solutions an *optimal solution*[1] x^* exists that produces the minimal, i.e. the optimal objective value $z^* = tx^*$. The constraints (2.2) can also be written as follows:

$$\sum_{j=1}^{n} q_{i,j} x_j \geq g_i \; \forall 1 \leq i \leq m \tag{2.4}$$

Subsequently, I will use the notation from equation (2.2) when I want to refer to a "set" of constraints and the notation from equation (2.4) when I refer to single constraints.

If we replace the constraints (2.3) with the integrality constraints $x \in \mathbb{N}^n$, the problem is called an *Integer Linear Program* (ILP). If some of the variables are real valued and some are integers the problem is a *Mixed Integer*

[1] The optimal solution is not necessarily unique.

Program (MIP). A special variant of an ILP that will be used in this work is a 0-1 ILP which is defined as follows:

$$\begin{aligned} \text{minimize} \quad & z = tx && (2.5)\\ \text{s.t.} \quad & Qx \geq g && (2.6)\\ & x \in \{0,1\}^n && (2.7) \end{aligned}$$

LPs can be solved efficiently, for example, by the simplex method or the interior point method. ILPs and MIPs are more difficult to solve. General methods to solve ILPs and MIPs to optimality are *Branch and Bound* or *Branch and Cut* approaches (see [ES00]). These methods rely on the repeated division of the problem into smaller subproblems. However, not all of these subproblems need to be solved to optimality. If a lower bound on the optimal objective value of such a subproblem can be given and this bound shows that the optimal solution of this subproblem will not lead to an optimal solution for the original problem, then this subproblem needs not to be solved. Such lower bounds can also be used by other algorithms to improve their performance or solution quality, or serve as a basis to decide upon the quality of a heuristic solution.

One simple method to compute lower bounds is linear relaxation. Consider an ILP I. Dropping the integrality constraints of I results in a LP, called the *linear relaxation* I_L of I. Since the relaxation increases the set of feasible solutions, the optimal objective value of the linear relaxation $z^*_{I_L}$ cannot be greater[2] than the optimal objective value z^*_I of the original problem. Therefore $z^*_{I_L}$ is a lower bound for z^*_I. Another way to compute lower bounds is to use Lagrangian relaxation. This method will be described in the next section.

2.2 Lagrangian Relaxation

The *Lagrangian Relaxation* (LR) is a method to compute lower bounds for LPs[3], ILPs or MIPs. In this work we will be facing ILPs and therefore describe LR in this context, but the principle is the same also with LPs or MIPs.

The idea of a LR is to relax some of the constraints and "move" them into the objective function. Assume we are given an ILP

$$\begin{aligned} \text{minimize} \quad & z_{\text{ILP}} = tx && (2.8)\\ \text{s.t.} \quad & Qx \geq g && (2.9)\\ & Sx \geq b && (2.10)\\ & x \in \mathbb{N}^n && (2.11) \end{aligned}$$

[2] Note that we are dealing with minimization problems.

[3] Although LPs can be solved efficiently in general, LR might be interesting on certain LPs that are extremely large or otherwise too complex to be solved directly.

with two "sets" of constraints $Q \in \mathbb{R}^{m_Q \times n}$ and $S \in \mathbb{R}^{m_S \times n}$. The LR resulting from relaxing the constraints (2.10) would be

$$\begin{aligned} \text{minimize} \quad & z_{\text{LLBP}} = tx + \lambda(b - Sx) && (2.12)\\ \text{s.t.} \quad & Qx \geq g && (2.13)\\ & x \in \mathbb{N}^n && (2.14) \end{aligned}$$

Here $\lambda \in \mathbb{R}^{m_S}_{\geq 0}$ is any positive vector. Its components are called *Lagrangian multipliers*. The program (2.12)–(2.14) is called *Lagrangian Lower Bound Program* (LLBP). Intuitively, λ can be seen to impose a penalty on violated constraints. The optimal objective value of a LLBP is a lower bound for the optimal objective value of the ILP. Beasley demonstrates this fact in [Bea93] as follows.

The optimal objective value z^*_{ILP} of an ILP as in equations (2.8)–(2.11) is not smaller than the optimal objective value of

$$\begin{aligned} \text{minimize} \quad & z = tx + \lambda(b - Sx) && (2.15)\\ \text{s.t.} \quad & Qx \geq g && (2.16)\\ & Sx \geq b && (2.17)\\ & x \in \mathbb{N}^n && (2.18) \end{aligned}$$

since $\lambda(b - Sx) \leq 0$. This in turn is not smaller than the optimal objective value of

$$\begin{aligned} \text{minimize} \quad & z_{\text{LLBP}} = tx + \lambda(b - Sx) && (2.19)\\ \text{s.t.} \quad & Qx \geq g && (2.20)\\ & x \in \mathbb{N}^n && (2.21) \end{aligned}$$

since dropping constraints in a minimization problem can only lead to a smaller optimal objective value.

The key for a useful LR is to relax constraints of an ILP such that the resulting problem is easier to solve than the original one. The next step is to find the vector λ^*, that produces the best, i.e. greatest lower bound. This is called the *Lagrangian Dual* (LD) problem. One heuristic algorithm to find good values for the Lagrangian multipliers and to approach λ^* is the Subgradient Optimization described in the next section.

2.3 Subgradient Optimization

The *Subgradient Optimization* (SG) is a method to heuristically solve a Lagrangian dual problem. It iteratively adjusts the Lagrangian multipliers to find values that produce the best or nearly the best lower bound. It relies on a solver for the LLBP and on an upper bound z_{ub} for the optimal objective

value of the original problem. An upper bound could, for example, be calculated by finding a feasible solution with a heuristic for the original problem. Assume we are given an ILP as in equations (2.8)–(2.11) and we want to relax the constraints (2.10). This results in a LLBP as in equations (2.12)–(2.14). The Subgradient Optimization, as described in [Bea93], is depicted in Algorithm 1.

Input: `LLBP()` ; *// Lagrangian lower bound program solver*
Input: z_{ub} ; *// upper bound value for original problem*
1 $\pi = \pi_{\text{init}}$; *// subgradient agility, Beasley sugg. $\pi_{init} = 2$*
2 $\lambda_i = 0 \; \forall 1 \leq i \leq m_S$; *// Lagrangian multipliers*
3 $z_{\max} = -\infty$; *// best lower bound so far*
4 **repeat**
5 $x^*_{\text{LLBP}} =$`LLBP`(λ) ; *// solve the LLBP to optimality*
6 $z^*_{\text{LLBP}} = t \cdot x^*_{\text{LLBP}} + \lambda(b - S \cdot x^*_{\text{LLBP}})$; *// corresp. objective value*
7 $\delta = b - S \cdot x^*_{\text{LLBP}}$; *// compute subgradients $\delta \in \mathbb{R}^{m_S}$*
8 $\Delta = \frac{\pi(z_{\text{ub}} - z^*_{\text{LLBP}})}{\sum_{1 \leq i \leq m_S} \delta_i^2}$; *// compute step size $\Delta \in \mathbb{R}$*
9 $\lambda_i =$`max`$(0, \lambda_i + \Delta \cdot \delta_i) \; \forall 1 \leq i \leq m_S$; *// update Lagrangian mult.*
10 **if** $z^*_{LLBP} > z_{max}$ **then**
11 $z_{\max} = z^*_{\text{LLBP}}$; *// remember best lower bound*
12 **end**
13 **if** `no_improvement()` **then**
14 $\pi = \frac{\pi}{2}$; *// reduce agility*
15 **end**
16 **until** `terminate()` ;

Algorithm 1: Subgradient Optimization algorithm. The function `no_improvement()` is true if $z_{\max}$ did not improve in a specified number of recent iterations (Beasley suggests 30). And `terminate()` is true if the optimum has been found (i.e. $z_{\text{ub}} = z^*_{\text{LLBP}}$) or π becomes smaller than a specified limit $\pi_{\min}$ (Beasley suggests 0.005) [Bea93].

2.4 Relax and Cut

The approach described in the previous section relies on a fixed set of constraints. Sometimes it might, however, not be desirable to list all constraints of a certain type explicitly. For example there could be an exponential number of these constraints or it might be too costly to compute each of these constraints. This is the case with the LR approach presented in section 4.1.4.

Lucena describes the *Relax-and-Cut* algorithm in [Luc05]. In the beginning not all relaxed constraints are known. Relax-and-Cut starts with

an initial set of relaxed constraints. Subsequently it iterates between solving the Lagrangian Dual and separation of new constraints. The Lagrangian Dual can, for example, be solved with Subgradient Optimization as described in the previous section. Lucena differentiates between *Delayed Relax-and-Cut* and *Non Delayed Relax-and-Cut*. With Delayed Relax-and-Cut new constraints are used after the Lagrangian Dual is solved, while with Non Delayed Relax-and-Cut new constraints are separated and used after every solution of the Lagrangian Lower Bound Program.

I implemented a Delayed Relax-and-Cut approach that is based on Subgradient Optimization. It is schematically described in the following paragraph.

1. Initially consider an empty set of relaxed constraints. Formally this corresponds to all Lagrangian multipliers being 0.
2. Solve the LLBP. Note that without any known relaxed constraints the objective function becomes $z_{\text{LLBP}} = tx$.
3. Separate initial constraints based on the first solution to the LLBP.
4. Perform SG up to some $\pi_{\min}$.
5. Separate new constraints that are violated at the end of SG.
6. Reset π and continue at step 4 as long as this improves the best lower bound for the original problem.

I implement this Relax-and-Cut approach as a modification to SG as shown in Algorithm 1. It is described in detail in Algorithm 2.

The lines 17-19 of Algorithm 2 clarify what is informally described above by "at the end of SG". The obvious strategy would be to separate new constraints in the last iteration of SG, i.e. right before it is restarted. To achieve this, the call to `separate()` should occur between lines 20 and 21. To be able to add more constraints at once when calling `add_constraints()`, I decided not only to separate new constraints in the last iteration. Instead, separation is done in a series of iterations before SG will be restarted. Specifically, separation is done from the second-last reduction $\frac{\pi}{2} \leq \pi_{\min}$ until the last reduction, i.e. the actual restart.

2.5 Extensions to the Subgradient Optimization

This section describes extensions and modifications, that can be applied to the Subgradient Optimization.

```
   Input: LLBP() ;              // Lagrangian lower bound program solver
   Input: z_ub ;                                        // upper bound value
   Input: separate() ;                                  // separate new constraints
 1 π = π_init ;                 // subgradient agility, Beasley sugg. π_init = 2
 2 S, b ;                       // relaxed constraints, initially empty, i.e. m_S = 0
 3 λ ;                                                // Lagrangian multipliers
 4 z_max = −∞ ;                                       // best lower bound so far
 5 repeat
 6     x*_LLBP = LLBP(λ) ;                            // solve the LLBP to optimality
 7     z*_LLBP = t · x*_LLBP + λ(b − S · x*_LLBP) ;   // corresp. objective value
 8     δ = b − S · x*_LLBP ;                          // compute subgradients δ ∈ R^{m_S}
       Δ = π(z_ub − z*_LLBP) / Σ_{1≤i≤m_S} δ_i^2 ;
 9                                                   // compute step size Δ ∈ R
10     λ_i = max(0, λ_i + Δ · δ_i) ∀1 ≤ i ≤ m_S ;     // update Lagrangian mult.
11     if z*_LLBP > z_max then
12         z_max = z*_LLBP ;                        // remember best lower bound
13     end
14     if no_improvement() then
15         π = π/2 ;                                // reduce agility
16     end
17     if π/2 ≤ π_min then
18         separate() ;                             // separate new constraints
19     end
20     if π ≤ π_min then
21         add_constraints() ;          // add recently separated constraints
22         π = π_init ;                             // reset agility, i.e. restart SG
23     end
24 until terminate() ;
```

Algorithm 2: Relax-and-Cut based on Subgradient Optimization. Here `no_improvement()` is true if $z_{\max}$ has not improved in a specified number of recent iterations (Beasley suggests 30). The function `terminate()` is true if the optimum has been found (i.e. $z_{\text{ub}} = z^*_{\text{LLBP}}$) or if $z_{\max}$ has not improved in a specified number of recent *restarts* (line 22). The function `add_constraints()` adjusts S, b and m_S and sets $\lambda_k = 0$ for all new constraints. For $\pi_{\min}$, Beasley suggests 0.005 [Bea93].

Adjustment of the Subgradient

The computation of the step size in line 8 of Algorithm 1 is:

$$\Delta = \frac{\pi(z_{\text{ub}} - z^*_{\text{LLBP}})}{\sum_{1 \leq i \leq m_S} \delta_i^2} \tag{2.22}$$

The denominator includes the squares of all subgradients, even if the corresponding Lagrangian multiplier will not be updated anyway. Precisely, if $\lambda_i = 0$ and the corresponding subgradient $\delta_i < 0$, then λ_i will not be changed in line 9. Beasley suggests in [Bea93] that in this case the subgradient could be set to $\delta_i = 0$. The effect will be that the step size Δ is not reduced due to the greater denominator.

Exceed Upper Bound

If the upper bound z_{ub} and the value of the Lagrangian lower bound program z^*_{LLBP} are close, i.e. they are near the optimal value, the step size can become very small. The effect could be that the Subgradient Optimization gets slower when approaching the optimum objective value. Beasley suggests to include a kind of excess-factor in the computation of the step size in line 8 of Algorithm 1:

$$\Delta = \frac{\pi(1.05 z_{\text{ub}} - z^*_{\text{LLBP}})}{\sum_{1 \leq i \leq m_S} \delta_i^2} \tag{2.23}$$

Direction Vector

Crainic et al. describe in [CFG01] the following scheme to update the Lagrangian multiplier vector λ not only with respect to the current value of the subgradient vector δ, but also based on previous modifications. Therefore we define a direction vector κ with respect to the current subgradient vector δ, the previous direction vector κ_{prev}, and a weight Θ for the previous direction.

$$\kappa = \delta + \Theta \kappa_{\text{prev}} \tag{2.24}$$

The Lagrangian multipliers are now updated by using κ instead of δ. The computation of the weight Θ can be done according to the *modified Camerini-Fratta-Maffioli rule*:

$$\Theta = \begin{cases} \|\delta\| / \|\kappa_{\text{prev}}\|, & \text{if } \delta \cdot \kappa_{\text{prev}} < 0, \\ 0, & \text{otherwise.} \end{cases} \tag{2.25}$$

The computation of the stepsize is also modified:

$$\Delta = \frac{\pi(z_{\text{ub}} - z^*_{\text{LLBP}})}{\delta \cdot \kappa} \tag{2.26}$$

Algorithm 3 shows the corresponding modification of the Subgradient Optimization algorithm from section 2.3. These modifications can similarly be applied to the Relax-and-Cut approach based on Subgradient Optimization, described in section 2.4.

Input: `LLBP()` ; *// Lagrangian lower bound program solver*
Input: z_{ub} ; *// upper bound value for original problem*
1 $\pi = \pi_{\text{init}}$; *// subgradient agility, Beasley sugg. $\pi_{init} = 2$*
2 $\lambda_i = 0 \; \forall 1 \leq i \leq m_S$; *// Lagrangian multipliers*
3 $z_{\max} = -\infty$; *// best lower bound so far*
4 $\kappa_{\text{prev}} = 0$; *// previous direction $\kappa \in \mathbb{R}^{m_S}$*
5 **repeat**
6 $x^*_{\text{LLBP}} =$`LLBP`(λ) ; *// solve the LLBP to optimality*
7 $z^*_{\text{LLBP}} = t \cdot x^*_{\text{LLBP}} + \lambda(b - S \cdot x^*_{\text{LLBP}})$; *// corresp. objective value*
8 $\delta = b - S \cdot x^*_{\text{LLBP}}$; *// compute subgradients $\delta \in \mathbb{R}^{m_S}$*
9 $\Theta = \begin{cases} \|\delta\| / \|\kappa_{\text{prev}}\| & \text{if } \delta \cdot \kappa_{\text{prev}} < 0 \\ 0 & \text{otherwise} \end{cases}$; *// prev-dir-weight $\Theta \in \mathbb{R}$*
10 $\kappa = \delta + \Theta \cdot \kappa_{\text{prev}}$; *// compute direction $\kappa \in \mathbb{R}^{m_S}$*
11 $\Delta = \dfrac{\pi(z_{\text{ub}} - z^*_{\text{LLBP}})}{\delta \cdot \kappa}$; *// compute step size $\Delta \in \mathbb{R}$*
12 $\lambda_i =$`max`$(0, \lambda_i + \Delta \cdot \kappa_i) \; \forall 1 \leq i \leq m_S$; *// update Lagrangian mult.*
13 **if** $z^*_{LLBP} > z_{max}$ **then**
14 $z_{\max} = z^*_{\text{LLBP}}$; *// remember best lower bound*
15 **end**
16 **if** `no_improvement()` **then**
17 $\pi = \frac{\pi}{2}$; *// reduce agility*
18 **end**
19 $\kappa_{\text{prev}} = \kappa$;
20 **until** `terminate()` ;

Algorithm 3: Subgradient Optimization algorithm with direction vector κ. The functions `no_improvement()` and `terminate()` are defined as in Algorithm 1.

2.6 Minimum Spanning Arborescences

At some point, my LR approaches depend on solving the *Minimum Spanning Arborescence* (MSA) problem, which is described below.

Definition 4 (Spanning Arborescence)
We are given a directed graph $G = (V, A)$. A *Spanning Arborescence* is a subgraph $T = (V, A_T)$, $A_T \subseteq A$ without cycles, such that there is a particular node r called the root, for which there is no arc $(v, r) \in A_T \; \forall v \in V$, and for

any node $w \neq r$ there is exactly one arc $(v, w) \in A_T \; \forall v \in V \setminus \{w\}$ directed towards it.

Given that $|V| = n$, it is obvious that a spanning arborescence has $n-1$ arcs. Based on this, we define the MSA problem.

Definition 5 (Minimum Spanning Arborescence)
We are given a directed graph $G = (V, A)$ and arc cost $c_{v,w} \in \mathbb{R} \; \forall (v, w) \in A$. The *Minimum Spanning Arborescence* problem asks for a spanning arborescence $T = (V, A_T)$ with minimal total arc cost $\sum_{(v,w) \in A_T} c_{v,w}$.

Both, the MSA problem as defined above as well as a variant, where the root node r is predefined, are relevant for my LR approaches. However, an algorithm for solving the unrooted MSA can also solve the rooted variant. A simple preprocessing step is needed: Delete all arcs directed towards the predefined root r. As a result every spanning arborescence in this reduced graph is rooted in r. Edmonds published in [Edm67] a polynomial time algorithm for the MSA problem.

2.7 Artificially Rooted HMST Problem

This section discusses how the close relation between the BDMST and the HMST problem can be used to transform a BDMST problem with an even diameter bound D into a HMST problem. A similar method was described in [GM03].

- Consider a BDMST problem on the graph $G(V, E)$ with edge cost $c_{v,w}$ and an even diameter bound D.
- Define a supergraph $G_{\text{art}}(V_{\text{art}}, E_{\text{art}})$ of G with an additional node r_{art} (artificial root):
  - $V_{\text{art}} = V \cup \{r_{\text{art}}\}$
  - $E_{\text{art}} = E \cup \{(r_{\text{art}}, v) \mid v \in V\}$
  - $c_{r_{\text{art}},v} = M \; \forall v \in V$, where M is a large constant.
- Consider the HMST problem defined on G_{art} with edge cost $c_{v,w}$, the root node r_{art}, and a hop bound of $H = \frac{D}{2} + 1$.
- Find an optimum solution $T^*_{\text{art}}(V_{\text{art}}, E_{T_{\text{art}}})$ for the HMST problem.
- Due to the large constant M, T^*_{art} contains exactly one arc (r_{art}, r).
- The graph $T^*(V, E_T)$ with $E_T = E_{T_{\text{art}}} \setminus \{(r_{\text{art}}, r)\}$ is an optimum solution for the BDMST problem. T^* is rooted at r.

Chapter 3

Related Work

Several algorithms dealing with BDMST and HMST problems have been published so far. Some of these solve the problem to optimality, while others produce heuristic solutions. This section gives a brief overview over some of these publications.

Abdalla, Deo, and Gupta present different heuristic algorithms for the BDMST problem in [ADG00]. Their *one-time-tree-construction* algorithm (OTTC) is a greedy construction heuristic. It is a modification of Prim's algorithm for the minimum spanning tree problem. It grows a spanning tree by subsequently adding a nearest neighbour. This node is connected with the cheapest edge that does not violate the diameter bound. In addition, they present a special heuristic for the BDMST problem with a diameter bound of $D = 4$ and two *Iterative Refinement Algorithms*.

Julstrom describes in [Jul04] two modifications of OTTC. *Center based tree construction* CBTC starts from a centre and subsequently connects nearest neighbours with cheapest edges that do not violate the tree depth constraint $\lfloor \frac{D}{2} \rfloor$. The centre is a single vertex if D is even, and an edge if D is odd. A randomized variant of this algorithm chooses the centre and the subsequent nodes at random. However, each of these nodes is still connected with the cheapest edge that does not violate the tree depth constraint $\lfloor \frac{D}{2} \rfloor$. This algorithm is called *randomized center-based tree construction* RTC.

Santos, Lucena, and Ribeiro describe in [dSLR04] a MIP formulation for the BDMST problem. The model contains 0-1 variables to define which arcs are in the solution. Additionally, there is an integral variable for every node. It denotes the number of arcs from the centre to the node. The so called Miller-Tucker-Zemlin inequalities establish a connection between the two types of variables. Informally, these inequalities express the fact that, if an arc (v, w) is in the solution, the path from the centre to w consists of one arc more than the path from the centre to v. They also present lifted Miller-Tucker-Zemlin inequalities which tighten the LP relaxation.

Raidl and Julstrom present an evolutionary algorithm (EA) and a ran-

domized greedy heuristic for the BDMST problem in [RJ03]. The heuristic is similar to RTC as described above. The EA encodes individuals as edge lists. The applied operators generate trees that are valid, i.e. satisfy the diameter restriction. The heuristic and the EA are compared on instances with up to 1000 nodes, where the EA produced substantially better solutions than the construction heuristic.

Julstrom and Raidl describe in [JR03] another evolutionary algorithm for the BDMST problem. Here, the individuals are encoded as permutations of the vertices of the graph. These permutations are transformed into trees by the centre-based greedy heuristic, which is similar to CBTC as described above. They compare the results of the permutation-coded EA and the edge-set-coded EA from [RJ03] on instances with up to 500 nodes. They observe that

> on the instances with 70 or more vertices, the permutation-coded EA consistently identified shorter bounded-diameter spanning trees in fewer iterations than did the edge-set-coded EA. However, because the heuristic that decodes permutations requires time that is $O(n^2)$, the permutation-coded EA is slower, and its disadvantage in time increases with the size of the problem instances.

Gouveia and Magnanti present in [GM03] ILP models for the BDMST problem and the Steiner Tree problem with diameter constraint. They present multicommodity flow models with hop constraints using different reformulations. They add an artificial root node r with zero-cost edges to all other nodes. The solution is the required to contain exactly one such edge (r, j) if D is even or exactly two edges (r, j) and (r, i) together with the central edge (i, j) if D is odd. Additionally they direct the problem by replacing every edge (i, j) from the original graph with two oppositely directed arcs. Finally they use hop-indexed variables that state that a specific arc (i, j) is the h^{th} arc in the path going from the root to a node k. Based on these reformulation techniques they present directed and undirected models as well as models including the hop-indexed variables or not including them. They compare the linear relaxations and the optimal solutions obtained via a Branch and Bound framework.

Dahl, Gouveia and Requejo survey and extend in [DGR06] different ILP formulations for the HMST problem. The MCF model (multi-commodity flow) has been studied in [Gou96] and [Gou98]. It formulates the HMST problem with design variables that select the arcs, multi-commodity flow variables, flow conservation constraints and coupling constraints. The coupling constraints establish a connection between the design variables and the flow variables. They consider the Lagrangian relaxation where the coupling constraints are relaxed. This results in a decomposition of the problem into a single inspection subproblem and a set of hop-constrained path subproblems.

Computational results show that the bounds derived from the Lagrangian relaxation are much better than the bounds derived from the linear relaxation of MCF for small values of H.

The Path model is another ILP formulation described in [DG04]. It is based on the sets P_k. P_k denotes the set of directed paths from r to k with not more than H arcs. The model contains 0-1 variables to decide which path is chosen for every node k. They prove that the value of the linear relaxation of the Path model is greater than the value of the linear relaxation of the MCF model and suggest a column generation based approach for the Path model.

The HopMCF model is a reformulation of the Path model presented in [Gou98]. It allows walks instead of paths[1]. To formulate the walk from r to k they define $H+1$ levels. Level 0 contains only the root node r, whereas level H contains only the node k. In all other levels the original nodes $V \setminus \{r\}$ are replicated. The nodes from all levels define the node set of an extended graph. The arc set of the original graph is replicated between every two consecutive levels. This defines the arc set of the extended graph. Every path from r (level 0) to k (level H) in this extended graph corresponds to a walk in the original graph with not more than H arcs. They prove that the linear relaxation of HopMCF is equal to the linear relaxation of the Path model.

Additionally they consider the Lagrangian relaxation approach based on the HopMCF model as described in [GR01]. The flow conservation constraints are relaxed, which leads to a relaxed problem that is decomposed into $|A|$ subproblems.

Their Jump formulation is the basis of one of my Lagrangian relaxation approaches, it is described in detail in section 4.1.2. Additionally, they present special ILP formulations for HMST with $H = 2$ and $H = 3$, respectively. They give computational results to compare the lower bounds achieved with the linear relaxation of their models and with the Lagrangian relaxations.

Gruber and Raidl present in [GR05a] a compact 0-1 ILP for the BDMST problem that is formulated with predecessor variables and depth variables. This model is described in detail in section 4.2.1. They use this model in a Branch and Cut framework with connection and cycle elimination cuts. Connection cuts ensure that the sets S and $V \setminus S$ must be connected, for every choice of $S \subset V$. Cycle elimination cuts state that, out of every cycle C of G, at most $|C|-1$ arcs may be in the solution. They compute linear relaxations and optimal objective values for a set of benchmark instances. I use their model as a basis for my Lagrangian relaxation approach in section 4.2.2.

Gruber extends the work from [GR05a] in [Gru06]. He introduces directed connection cuts and path cuts. Directed connection cuts are the

[1] In a walk the arcs may be repeated, whereas this is not allowed in a path.

directed version of the connection cuts mentioned above. Path cuts ensure that out of every path P of G, which has a length of $|P| = D + 1$, at most D arcs may be in the solution. I use the computational results in section 6 to compare the lower bounds achieved with my Lagrangian relaxation approaches to the lower bounds from Gruber. A simple construction heuristic is also described. Assume that an assignment of all nodes to levels $[0..H]$ is given. The *level construction heuristic* finds for every node a predecessor with lowest cost, among all nodes at a lower level. This heuristic is used in my approaches to create heuristic solutions from interim results computed by the Lagrangian relaxation approaches.

Gruber and Raidl present in [GR05b] a variable neighbourhood search approach for the BDMST problem. Four different types of neighbourhoods are described. Gruber, Hemert and Raidl [GHR06] continue and improve the work from [GR05b]. They integrate the neighbourhood searches into an evolutionary algorithm (EA), and an ant colony optimization (ACO) algorithm. The three different approaches: VNS, EA and ACO are compared computationally. Their computational experiments on BDMST instances with up to 1000 nodes lead to the observation that

> the EA and the ACO outperform the VNS on almost all used benchmark instances. Furthermore, the ACO yields most of the time better solutions than the EA in long-term runs, whereas the EA dominates when the computation time is strongly restricted.

Chapter 4

Lagrangian Relaxation Approaches

This section describes the two Lagrangian relaxations that form the basis of my LR approaches. Each of my LRs relies on a certain 0-1 ILP model. One of these models formulates the HMST problem, i.e. a rooted problem[1]. The other one directly formulates the BDMST problem with an even diameter bound, i.e. an unrooted problem.

Both models have in common that they are directed models. This means that they are built upon a bidirected interpretation $G_d(V, A)$ of the undirected graph $G(V, E)$ as defined by the HMST/BDMST problem(s). For every edge $(v, w) \in E$ in the original Graph G, the set of arcs A contains two directed arcs (v, w) and (w, v). Both arcs are associated the same cost $c(v, w) = c(w, v)$. The two models describe a subgraph $T(V, A_T)$ of the directed graph $G_d(V, A)$, $A_T \subset A$.

The next two sections give precise formulations of the ILP models, the applied Lagrangian relaxations, and algorithms to solve the resulting LLBPs. Both sections close with the descriptions of how these LLBP solvers are embedded in Subgradient Optimization or Relax-and-Cut to solve the Lagrangian duals.

4.1 Jump-Relaxation

4.1.1 Predecessor-Jump Model

This section describes a 0-1 ILP model for a HMST problem with a hop bound H in the bidirected graph $G_d(V, A)$ rooted at r. This model consists

[1] This work only investigates BDMST problems. The scheme described in section 2.7 will be applied to transform BDMST problems into HMST problems.

of predecessor variables $p_{v,w} \in \{0,1\}\ \forall (v,w) \in A$ defined as follows:

$$p_{v,w} = \begin{cases} 1, & \text{if } v \text{ is the predecessor of } w \text{ in the directed path} \\ & \text{from } r \text{ to } w \text{ in } T, \text{ i.e. } (v,w) \in A_T \\ 0, & \text{otherwise} \end{cases} \tag{4.1}$$

The HMST problem can be formulated as follows:

$$\text{minimize} \qquad z = \sum_{(v,w)\in A} p_{v,w} c_{v,w} \tag{4.2}$$

$$\text{s.t.} \qquad p \text{ forms a spanning arborescence rooted at } r \tag{4.3}$$

$$\sum_{(v,w)\in J} p_{v,w} \geq 1\ \forall J \tag{4.4}$$

Here the constraint (4.3) ensures that the set of arcs $A_T = \{(v,w) \in A \mid p_{v,w} = 1\}$ forms a spanning arborescence in G_d. There is no formal expression for this constraint as my solver satisfies it immediately. This is shown in section 4.1.4. The constraints (4.4) are the so called *jump constraints* as presented by [DGR06].

Jump constraints can be described as follows: Consider an arbitrary node $k \in V$ and a partition of the node set V into $H+2$ nonempty, disjoint sets V_i where $\bigcup_{i=0}^{H+1} V_i = V$, $V_0 = \{r\}$, and $V_{H+1} = \{k\}$. The set of *jump arcs* for this partition is now defined as $J = \{(v,w) \in A \mid v \in V_i, w \in V_j, j \geq i+2\}$, i.e. the set of arcs that "jump" over at least one partition (see Figure 4.1). J is called a *jump* and Γ denotes the set of all jumps.

Note that every tree that satisfies the hop constraints contains at least one arc out of every jump. Assume the contrary: Consider a node k, a node partition as described above, and the jump J defined according to that partition. Now assume that the tree T satisfies the hop constraints but does not contain any of the arcs of J. In T, there must be a directed path from r to k. This path does – by assumption – not contain a jump arc. Thereby it consists of at least $H+1$ arcs. This violates the hop constraints and contradicts the assumption.

It follows that every HMST must satisfy the constraints $\sum_{(v,w)\in J} p_{v,w} \geq 1$ for every possible jump J. Note that it is not necessary to enumerate every jump $J \in \Gamma$ explicitly. Instead, violated jumps will be computed dynamically in a Relax-and-Cut algorithm. Subsequently, the corresponding jump constraint will be added to the Lagrangian relaxation approach.

4.1.2 LR for the Predecessor-Jump Model

The jump constraints (4.4) are relaxed in the usual Lagrangian way. This results in a LLBP with the following objective function:

$$\sum_{(v,w)\in A} p_{v,w} c_{v,w} + \sum_{J\in\Gamma} \lambda_J (1 - \sum_{(v,w)\in J} p_{v,w}) \tag{4.5}$$

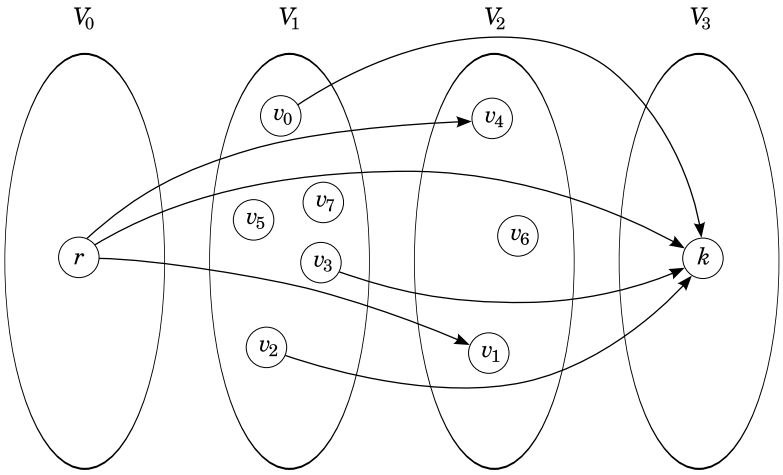


Figure 4.1: Assume a HMST problem with $|V| = 10$ nodes and a hop bound of $H = 2$. The figure shows one possible node partition into $H + 2 = 4$ disjoint sets. Depending on the arc set A, the jump arcs for this partition could be the depicted arcs.

The objective function (4.5) can be rewritten as follows:

$$\sum_{(v,w)\in A} p_{v,w}c_{v,w} + \sum_{J\in\Gamma} \lambda_J - \sum_{J\in\Gamma} \left(\sum_{(v,w)\in J} p_{v,w}\lambda_J \right) = \underbrace{\sum_{(v,w)\in A} p_{v,w} \left(c_{v,w} - \sum_{\substack{J\in\Gamma| \\ (v,w)\in J}} \lambda_J \right)}_{} + \underbrace{\sum_{J\in\Gamma} \lambda_J}_{} \tag{4.6}$$

The resulting objective function (4.6) consists of two parts. The predecessor variables $p_{v,w}$ occur only in the first part. The second part is a constant term for every given set of Lagrangian multipliers. To make this clearer, I define $\alpha_{v,w} = \left(c_{v,w} - \sum_{\substack{J\in\Gamma| \\ (v,w)\in J}} \lambda_J \right)$ and the constant term $L = \sum_{J\in\Gamma} \lambda_J$. The LLBP can now be rewritten as:

$$\text{minimize} \quad \left(\sum_{(v,w)\in A} p_{v,w}\alpha_{v,w} \right) + L \tag{4.7}$$

$$\text{s.t.} \quad p \text{ forms a spanning arborescence rooted at } r \tag{4.8}$$

Essentially the LLBP can be solved by solving a MSA problem. This MSA problem has arc cost α which are computed from the original arc cost c and the current values of the Lagrangian multipliers λ.

4.1.3 Jump Constraint Separation

This section describes the two strategies that I have implemented to separate jump constraints. As demonstrated in section 4.1.1 every node partition induces a jump constraint. It seems unpractical to enumerate every possible node partition to compute all jump constraints. Instead, the Relax-and-Cut approach as described in Algorithm 2 was utilized. Therefore, a method to separate jump constraints is required. I separated violated constraints in an optimal solution of the LLBP for some given λ. As shown in the previous section, the solution to the LLBP is a minimum spanning arborescence $T(V, A_T)$.

Both strategies start by calculating the depth $d(v)$ for every node v in T, i.e. the number of arcs on the unique path from r to v. Every node v with a depth of $d(v) = H + 1$ is the endpoint of a path with $H + 1$ arcs, i.e. a path that violates the hop constraints[2]. For each of these paths $((r, v_1), (v_1, v_2), \ldots, (v_H, v))$ we can easily create a partitioning $\bigcup_{i=0}^{H+1} V_i = V$, which induces a violated jump constraint. Practically, all the nodes of the violating path from r to v are assigned to the sets V_i according to their depth:

1. $r \in V_0$
2. $v_i \in V_i \; \forall 1 \leq i \leq H$
3. $v \in V_{H+1}$

The two partitioning strategies now differ in the way they distribute the other nodes: Put each node w that is not in the violating path into

$$\begin{cases} V_1 & \text{with strategy } [V_1], \text{or} \\ V_i, i = \min(H, d(w)) & \text{with strategy } [V_{\text{depth}}]. \end{cases} \tag{4.9}$$

These rules ensure that none of the arcs in the minimum spanning arborescence T is a jump arc, i.e. we have constructed a violated jump constraint. Adding this constraint will make it less likely that T will be the optimum solution to the LLBP with any λ in the further execution of the Subgradient Optimization.

4.1.4 LR Approach for the Predecessor-Jump Model

After presenting all necessary parts for the first Lagrangian relaxation approach, I discuss how these parts work together to compute lower bounds for the BDMST problem with an even diameter bound.

[2] If there is no such node, then T satisfies the hop constraints, i.e. T is a feasible solution for the original problem.

1. Transform the BDMST problem into a HMST problem with an artificial root node as described in section 2.7.
2. Transform the graph from the HMST problem into its bidirected interpretation as described in section 4.
3. Perform the Relax-and-Cut algorithm as described in Algorithm 2.
   - The LLBP is given as in section 4.1.2, i.e. equation (4.7) and equation (4.8).
   - The LLBP can be solved by Edmonds algorithm for the Minimum Spanning Arborescence problem, described in section 2.6.
   - Start with an empty set of jump constraints.
   - Separate jump constraints via one of the strategies described in section 4.1.3.

4.2 Predecessor-Depth-Relaxation

4.2.1 Predecessor-Depth Model

This 0-1 ILP model is taken from [GR05a]. It formulates a BDMST problem with an even[3] diameter bound D in the bidirected graph $G_d(V, A)$. As mentioned in section 1, there is a root node r in every BDMST, the centre. The depth of a node $d(v)$ is the number of arcs on the directed path from r to v.

The Predecessor-Depth model consists of predecessor variables $p_{v,w} \in \{0,1\}\ \forall (v,w) \in A$ and depth variables $u_{v,l} \in \{0,1\}\ \forall v \in V, 0 \leq l \leq H$ which are defined as follows:

$$p_{v,w} = \begin{cases} 1, & \text{if } v \text{ is the predecessor of } w \text{ in the directed path} \\ & \text{from } r \text{ to } w \text{ in } T, \text{ i.e. } (v,w) \in A_T \\ 0, & \text{otherwise} \end{cases} \tag{4.10}$$

$$u_{v,l} = \begin{cases} 1, & \text{if } v \text{ has a depth of } l, \text{ i.e. } d(v) = l \\ 0, & \text{otherwise} \end{cases} \tag{4.11}$$

[3] They also describe another similar model for BDMST with an odd D. The LR approach presented in this section could be extended to use that model to find lower bounds for problems with an odd D. This idea is discussed in section 7.

The BDMST problem can be formulated as follows:

$$\text{minimize} \qquad z = \sum_{(v,w)\in A} p_{v,w} c_{v,w} \tag{4.12}$$

$$\text{s.t.} \qquad \sum_{l=0}^{H} u_{v,l} = 1 \quad \forall v \in V \tag{4.13}$$

$$\sum_{v\in V} u_{v,0} = 1 \tag{4.14}$$

$$\sum_{v|(v,w)\in A} p_{v,w} = 1 - u_{w,0} \quad \forall w \in V \tag{4.15}$$

$$p_{v,w} \leq 1 - u_{w,l} + u_{v,l-1} \quad \forall (v,w) \in A, \forall 1 \leq l \leq H \tag{4.16}$$

Here the constraints (4.13) ensure that each node gets assigned to a unique depth. Constraint (4.14) forces exactly one node to depth 0, i.e. the root node of the BDMST. Constraints (4.15) ensure that each node has exactly one predecessor except the node at level 0. Finally, the constraints (4.16) establish a connection between the predecessor variables and the depth variables. If v is the predecessor of w then the depth of v must be 1 less than the depth of w.

4.2.2 LR for the Predecessor-Depth Model

The constraints (4.16) are relaxed in the usual Lagrangian way. This results in a LLBP with the following objective function:

$$\sum_{(v,w)\in A} p_{v,w} c_{v,w} + \sum_{l=1}^{H} \left(\sum_{(v,w)\in A} \lambda_{v,w,l} \left(p_{v,w} - 1 + u_{w,l} - u_{v,l-1} \right) \right) \tag{4.17}$$

Equation (4.17) can be rewritten as

$$\begin{aligned} \sum_{(v,w)\in A} p_{v,w} c_{v,w} &+ \sum_{l=1}^{H} \left(\sum_{(v,w)\in A} p_{v,w} \lambda_{v,w,l} \right) \\ &+ \sum_{l=1}^{H} \left(\sum_{(v,w)\in A} (u_{w,l} - u_{v,l-1}) \lambda_{v,w,l} \right) \\ &- \sum_{l=1}^{H} \left(\sum_{(v,w)\in A} \lambda_{v,w,l} \right) = \end{aligned}$$

which in turn can be expressed as follows:

$$\underbrace{\sum_{(v,w)\in A} p_{v,w}\left(c_{v,w}+\sum_{l=1}^{H}\lambda_{v,w,l}\right)}$$
$$+\underbrace{\sum_{l=1}^{H}\left(\sum_{(v,w)\in A}(u_{w,l}-u_{v,l-1})\lambda_{v,w,l}\right)}$$
$$-\underbrace{\sum_{l=1}^{H}\left(\sum_{(v,w)\in A}\lambda_{v,w,l}\right)} \tag{4.18}$$

The resulting objective function (4.18) consists of three parts. The third part is a constant term[4] $L = \sum_{l=1}^{H}\sum_{(v,w)\in A}\lambda_{v,w,l}$. The first part depends only on the predecessor variables $p_{v,w}$. The coefficients can be expressed as $\alpha_{v,w} = \left(c_{v,w} + \sum_{l=1}^{H}\lambda_{v,w,l}\right)$. The second part depends only on the depth variables u. This part contains a sum over all arcs but the depth variables are indexed with nodes. It is possible to reorder the summands and combine the coefficients of all occurrences of any depth variable $u_{v,l}$ into one coefficient $\beta_{v,l}$. This way we can write

$$\sum_{l=0}^{H}\left(\sum_{v\in V} u_{v,l}\beta_{v,l}\right) = \sum_{l=1}^{H}\left(\sum_{(v,w)\in A}(u_{w,l}-u_{v,l-1})\lambda_{v,w,l}\right) \tag{4.19}$$

where the β values are computed by the simple Algorithm 4.

```
   β_{v,l} = 0 ∀v ∈ V, 0 ≤ l ≤ H ;                    // initialize β
1  foreach l, 0 ≤ l ≤ H do
2      foreach (v, w) ∈ A do
3          β_{w,l}   = β_{w,l} + λ_{v,w,l} ;
4          β_{v,l-1} = β_{v,l-1} − λ_{v,w,l} ;
5      end
6  end
```

Algorithm 4: Compute coefficients β for the LLBP.

[4] Note that we are talking about the LLBP and hence the Lagrangian multipliers λ are constants in this context.

The LLBP can now be rewritten as:

$$\text{minimize} \quad \underbrace{\sum_{(v,w)\in A} p_{v,w}\alpha_{v,w}} + \underbrace{\sum_{l=0}^{H}\left(\sum_{v\in V} u_{v,l}\beta_{v,l}\right)} - \underbrace{L} \tag{4.20}$$

$$\text{s.t.} \quad \text{constraints (4.13)–(4.15)} \tag{4.21}$$

Only the constraint (4.15) contains both the predecessor variables $p_{v,w}$ and the depth variables $u_{v,l}$. That allows us to split the problem into two nearly independent problems.

Consider the first part of the objective function (4.20) with the constraints (4.13)–(4.15). This asks for a subgraph[5] with minimum cost where each node, except one, has exactly one predecessor. This is a minimum spanning arborescence problem without a predefined root node.

Next, consider the second part of the objective function (4.20) with constraints (4.13)–(4.15). This requires the assignment of a unique depth to every node[6]. The only additional restriction is that the root node from the MSA problem above must be assigned to depth 0. To all other nodes an arbitrary depth out of $1 \leq l \leq H$ can be assigned. The only connection between these problems is that both must use the same root node r. This means essentially that we can solve the LLBP by solving the following problem

$$z_{\text{LLBP}} = \min_{r\in V}\left(z_{\text{MSA}}(r) + z_{\text{MA}}(r) - L\right) \tag{4.22}$$

which consists of a minimum spanning arborescence problem

$$\text{minimize} \quad z_{\text{MSA}}(r) = \sum_{(v,w)\in A} p_{v,w}\alpha_{v,w} \tag{4.23}$$

$$\text{s.t.} \quad p \text{ forms a spanning arborescence rooted at } r \tag{4.24}$$

and a minimum assignment problem.

$$\text{minimize} \quad z_{\text{MA}}(r) = \sum_{l=0}^{H}\left(\sum_{v\in V} u_{v,l}\beta_{v,l}\right) \tag{4.25}$$

$$\text{s.t.} \quad \sum_{l=0}^{H} u_{v,l} = 1 \;\forall v \in V \tag{4.26}$$

$$\sum_{v\in V} u_{v,0} = 1 \tag{4.27}$$

$$u_{r,0} = 1 \tag{4.28}$$

[5] That is, assign 1 to some of the p variables.

[6] That is, assign 1 to the corresponding $u_{v,l}$ variable.

The assignment problem can be simplified.

$$\text{minimize } z_{\text{MA}}(r) = \sum_{l=0}^{H} \left(\sum_{v \in V} u_{v,l} \beta_{v,l} \right) \tag{4.29}$$

$$\text{s.t.} \sum_{l=1}^{H} u_{v,l} = 1 \; \forall v \in V, v \neq r \tag{4.30}$$

$$u_{r,0} = 1 \tag{4.31}$$

$$u_{v,0} = 0 \; \forall v \in V, v \neq r \tag{4.32}$$

The optimal solution to the minimum assignment problem is simply:

$$z_{\text{MA}}(r) = \beta_{r,0} + \sum_{\substack{v \in V, \\ v \neq r}} \min_{1 \leq l \leq H} (\beta_{v,l}) \tag{4.33}$$

Algorithm 5 describes the algorithm to solve the LLBP as given in equations (4.20)–(4.21).

4.2.3 LR Approach for the Predecessor-Depth Model

Now I have presented all necessary parts for the second Lagrangian relaxation approach. The following shows how these parts work together to compute lower bounds for BDMST problems with an even diameter bound.

1. Transform the graph from the BDMST problem into its bidirected interpretation as described in section 4.
2. Perform the SG as described in Algorithm 1.
   - The LLBP is given as in section 4.2.2, i.e. equation (4.22).
   - The LLBP can be solved by Algorithm 5.

```
   Input: MSA() ;                                          // MSA solver
   Input: α_{v,w} ∀(v,w) ∈ A ;                             // arc cost
   Input: β_{u,l} ∀u ∈ V, 0 ≤ l ≤ H ;                      // assignment cost
   Input: L ;                                              // constant term
   /* these values will contain the optimal solution at the end        */
 1 z* = ∞ ;                                                // objective value
 2 r* ;                                                    // root node
 3 p* ;                                                    // predecessor variables
 4 u* ;                                                    // depth variables
 5 foreach r ∈ V do
       /* minimum spanning arborescence                                */
 6     p = MSA(r) ;                                        // solve the MSA problem
 7     z_MSA = Σ_{(v,w)∈A} p_{v,w} · α_{v,w} ;             // corresponding objective value
       /* minimum assignment                                           */
 8     u_{v,l} = 0 ∀v ∈ V, 0 ≤ l ≤ H ;                     // initialize
 9     u_{r,0} = 1 ;                                       // root node is at depth 0
10     foreach v ∈ V, v ≠ r do
11         i = minarg_{1≤l≤H}(β_{v,l}) ;                   // depth value with smallest cost
12         u_{v,i} = 1 ;
13     end
14     z_MA = Σ_{l=0}^{H} ( Σ_{v∈V} u_{v,l} · β_{v,l} ) ;
       /* remember best solution so far                                */
15     if (z_MSA + z_MA − L) < z* then
16         z* = z_MSA + z_MA − L ;
17         r* = r ;
18         p* = p ;
19         u* = u ;
20     end
21 end
```

Algorithm 5: Algorithm to solve a Lagrangian Lower Bound Program of the Predecessor-Depth relaxation for given values of α, β, and L.

Chapter 5

Implementation

This section describes my implementation of the LR-approaches. I developed a command line application (`lrbdmst`) in `C++` under Linux. The following section presents the class hierarchy, while section 5.2 lists the external packages, and finally section 5.3 gives brief usage instructions.

5.1 Class Hierarchy

The global structure consists of two parts. The first part provides the functionality of Subgradient Optimization and Relax-and-Cut and is independent of any actual optimization problem. The second part is the BDMST-related part. It can itself be divided into parts for instance representation, solution representation, and for each of the two Lagrangian relaxation approaches. The more important classes are depicted in Figure 5.1 and are subsequently described in more detail.

5.1.1 Subgradient Optimization

`Constraint`
: An abstract class representing a constraint.

`Subgradient_Solver`
: Provides the functionality described in Algorithm 1, Subgradient Optimization, and Algorithm 2, Relax-and-Cut. It has a reference to a `LLBP_Solver` object.

`LLBP_Solver`
: An abstract class representing a solver for a given Lagrangian lower bound program. It holds a set of `Constraint` objects and a corresponding set of Lagrangian multipliers.

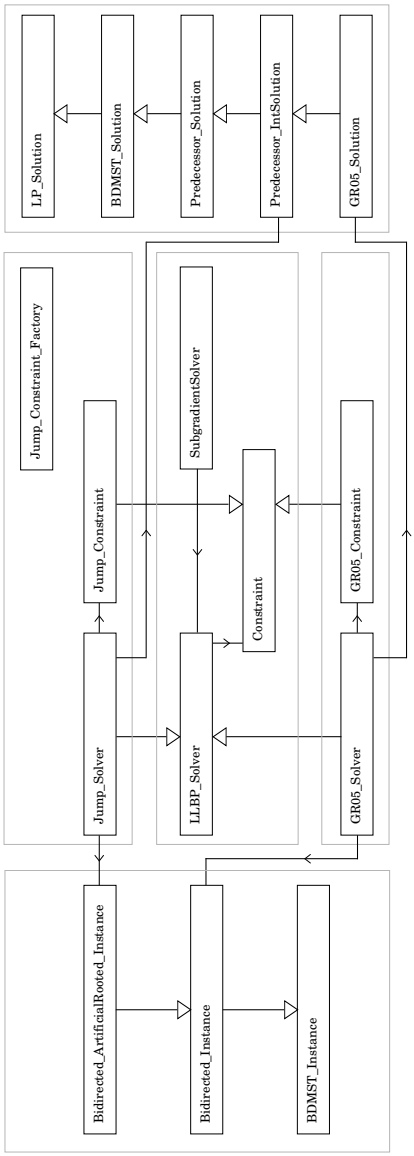


Figure 5.1: Class hierarchy of the more important classes of the `lrbdmst` program.

5.1.2 Instance Representation

`BDMST_Instance`
: Represents an instance of the BDMST problem. Essentially this is a graph with edge weights and a diameter bound D.

`Bidirected_Instance`
: A subclass of `BDMST_Instance`. This class provides the mapping from a BDMST instance to its bidirected interpretation as described in section 4.

`Bidirected_ArtificialRooted_Instance`
: A subclass of `Bidirected_Instance`. This class provides the mapping from a bidirected interpretation of a BDMST instance to an instance with an artificial root node as described in section 2.7.

5.1.3 Solution Representation

`LP_Solution`
: An abstract class that specifies a very simple interface to a LP solution.

`BDMST_Solution`
: An abstract subclass of `LP_Solution` that specifies a very simple interface to a BDMST solution.

`Predecessor_Solution`
: An abstract subclass of `BDMST_Solution` that specifies the interface of a predecessor solution for a BDMST problem. Essentially this specifies predecessor variables $p_{v,w} \in [0, 1]$.

`Predecessor_IntSolution`
: A subclass of `Predecessor_Solution` that actually implements the predecessor variables as integers, i.e. $p_{v,w} \in \{0, 1\}$.

`GRO5_IntSolution`
: A subclass of `Predecessor_IntSolution` that provides additional depth variables $u_{v,l} \in \{0, 1\}$.

5.1.4 LLBP Solver for the Predecessor-Jump Approach

`Jump_Solver`
: A subclass of `LLBP_Solver`. It solves the LLBP described in section 4.1.2, i.e. it computes the new arc costs α and solves the resulting minimum spanning arborescence problem. It works on `Bidirected_ArtificialRooted_Instance` and `Predecessor_IntSolution`.

`Jump_Constraint`
: A subclass of `Constraint` representing a jump as described in section 4.1.1. Essentially this is a list of arcs.

`Jump_Constraint_Factory`
: A static factory that creates new `Jump_Constraint` objects. It implements the two jump constraint separation strategies described in section 4.1.3.

5.1.5 LLBP Solver for the Predecessor-Depth Approach

`GR05_Solver`[1]
: A subclass of `LLBP_Solver`. It solves the LLBP described in section 4.2.2, i.e. it implements Algorithm 5. It works on `Bidirected_Instance` and `GR05_Solution`.

`GR05_Constraint`
: A subclass of `Constraint` representing a constraint as given in equation 4.16. Essentially it consists of references to one predecessor variable and two depth variables.

5.1.6 `main()` Method

The main method is implemented in the file `lrbdmst.cpp`. Basically, it is responsible for the following:

1. Parse the command line.
2. Read the BDMST instance from a file.
3. Calculate an upper bound for the optimal objective value.
4. Create a `LLBP_Solver`, i.e. either
   - a `Jump_Solver`, or
   - a `GR05_Solver`.
5. Create a `Subgradient_Solver`.
6. Initiate the computation with a call to `Subgradient_Solver::run()`.

[1] The name "GR05" is a reference to the paper in which the corresponding ILP-Model was published, i.e. Gruber and Raidl 2005.

5.2 External Packages

The following libraries and external modules were used:

`LEDA`, version 5.1.1
: *Library Of Efficient Data Types And Algorithms*, mainly used for the representation of graphs (see [LED06]).

`GOBLIN`, version 2.7.2
: *A Graph Object Library for Network Programming Problems.* It provides an implementation of Edmonds algorithm to solve the minimum spanning arborescence problem. This implementation can solve the rooted and the unrooted variants (see [FPSE06]).

`ilp`, as of September 18^{th}, 2007
: A set of classes from the `ilp` program by Martin Gruber.

  - Parse BDMST instances from files in various formats.
  - Initially compute a heuristic solution for the BDMST problem by means of the `CBTC` and `RTC` heuristics to provide an upper bound for the optimal objective value.
  - Compute heuristic solutions for the BDMST problem by means of the *level construction heuristic* during the execution of the Subgradient Optimization.

`log4cpp`, version 0.3.5rc3
: A "library of C++ classes for flexible logging" [Bak05].

5.3 Usage

This section includes the usage message of my program and describes the connection between some of the parameters to the corresponding part of this thesis.

```
Usage: ./lrbdmst --instance instance_file [options]

Compute a lower bound for the objective value of the
Bounded Diameter Minimum Spanning Tree problem (BDMST)
by means of Lagrangian Relaxation and Subgradient Optimization.
Author: Peter Putz

General Options:
  -h, --help: prints this usage message
  -H, --version: prints current version

Instance Selection Options:
  -I, --instance_type: the following instance types can be used
       'gnuplot' or 'gp' [see option '-g'],
```

```
       'santos' or 's',
       'gouveia' or 'g',
       'ea',
       'rand_ea' or 'r'
  -i, --instance: instance filename
  -d, --diameter: the diameter of the BDMST to be computed
  -g, --gp_lines: in case of a GNUPLOT instance this option can be used to
    specify the name of the file holding the line information
  -G, --gouveia_edges: number of the edges that will be used in a Gouveia
    instance ('-1' for the complete, fully connected graph)

Langrange Relaxation Options:
  -l, --lagrange_relaxation_method: which Lagrange Relaxation Method shall
    be used:
      'gr05' Based on the model described by Gruber and Raidl 2005
          (variables: predecessor p_{v,w} and height u_{v,l}),
          with relaxation of their equation (5), resulting in a
          decomposition into a minimum arborescence and an assignment
          problem.
      'jump' Based on the predecessor model, with relaxation of
          the jump constraints as desc. by Dahl, Gouveia and Requejo 2006.
  -j, --jump_separation: which strategy shall be used for node partitioning
    when generating jump constraints: put a node v, which is not in the
    violating path
      '1'   into V_1
      'd'   into V_d, d = min(depth(v),H)
    Can also be combined '1d', i.e. generate two constraints: one according
    to V_1 and one according to V_d.

Subgradient Optimization Options:
  -m, --maxIterations: maximum number of iterations to perform for the SG
  -a, --SG_baseAgility_TerminationLevel: the minimal subgradient agility
    value (the default is 0.005 as suggested by Beasley 1993)
  -A, --SG_baseAgility_ReductionAfterNoImprove: reduce (i.e. halve) the
    agility after that many iterations without improvement
    (the default is 30 as suggested by Beasley 1993)
  -r, --SG_repetitionsLimit: the maximum number of repetitions of the SG
    (only makes sense if constraints are separated dynamically,
    i.e. -l jump)
  -R, --SG_repetitionsNoImproveLimit: the maximum number of consecutive
    repetitions of the subgradient optimization without improving the
    lower bound
    (only makes sense if constraints are separated dynamically,
    i.e. -l jump)

Miscellaneous Options:
  -u, --upper_bound_method: use this primal heuristic to compute an
    initial upper bound:
      'rtc'   Randomized Tree Construction,
      'cbtc'  Center Based Tree Construction.
  -U, --upper_bound_iterations: the selected heuristic for the upper bound
    [see option -u] will stop to build new starting solutions after '-U'
    iterations (new solutions) without further improvement
  -v, --vnd:
```

```
    neighbourhoods and their order within the VND; used to locally improve
    starting solutions created by one of the primal heuristics
    [see option -u]:
      'none'  no VND,
      'e'     edge exchange,
      's'     subtree optimization/node swap,
      'c'     level center exchange,
      'l'     level change
  -V, --lev_nh_switch: percentage of the nodes when to switch from a level
    to a neighbour list based predecessor search. This is used for the
    initial upper bound.

Output Options:
  -o, --outputPrefix: all created output files will have names starting with
    outputPrefix (defaults to "output/instancefilename")
  -w, --writeMinArbosGnuplot: (yes|no) write the minimum arborescence in
    each iteration in gnuplot format to a file
  -W, --writeMinArbosGoblin:(yes|no) write the minimum arborescence in each
    iteration in goblin format to a file
  -L, --logrc: log4cpp configuration file (log4cpp.properties)
```

The most relevant parameters and the corresponding parts in this thesis are given as follows:

- `--lagrange_relaxation_method`
  Switch to choose between the model described in section 4.2.1 and the corresponding Lagrangian relaxation approach from section 4.2.2 (`gr05`), and the model from section 4.1.1 with the Lagrangian relaxation approach given in section 4.1.2 (`jump`).

- `--jump_separation`
  In case the parameter `--lagrange_relaxation_method jump` is given, this parameter selects the jump separation strategy, where `1` corresponds to $[V_1]$ and `d` to $[V_{depth}]$ (see section 4.1.3).

- `--SG_baseAgility_TerminationLevel`
  This value corresponds to $\pi_{\min}$ in Algorithm 1 for which [Bea93] suggests 0.005.

- `--SG_baseAgility_ReductionAfterNoImprove`
  This value represents the *maximum number of recent iterations without improving* z_{max}, for which [Bea93] suggests 30 (see Algorithm 1).

- `--SG_repetitionsNoImproveLimit`
  This value represents the *maximum number of recent restarts without improving* z_{max} (see Algorithm 2).

5.4 Auxiliary Scripts

As noted above the program `lrbdmst` is a command line application. The default behaviour is to write a block of information for every iteration of the Subgradient Optimization to `stdout`. Usually this results in rather voluminous logfiles which are not perfectly suited to get an overview of the computation. Another aspect is that usually a single computation on one BDMST instance is not significant. To get meaningful results, a series of computations has to be executed. It is desirable to compare one series of computations to another one, e.g. two series perform SG on the same set of instances but with different parameters.

To tackle these problems I developed a set of `bash` and `perl` scripts. Note that none of them are required for the proper functioning of `lrbdmst`. They are intended to allow the scheduling of sequential experiments, to help with the reproducibility of computations, as well as to ease the analysis and comparison of results. The scripts provide usage descriptions, and the `README` file included with `lrbdmst` gives brief introductions and typical usage examples. For the sake of completeness the more important scripts are listed here. Those scripts that are meant to help with logfile analysis rely on ExpLab [HKPS03] which provides the data format `sus` together with tools that transform logfiles into the `sus` format, filter, sort or join multiple `sus`-files, and export the `sus` format into tables or plots.

`./bin/labrunner2.pl`
: Execute `lrbdmst` on a set of BDMST instances, all with the same parameters.

`./bin/log2sus.sh`
: Transform a logfile as written by `lrbdmst` into `sus`. One iteration of the Subgradient Optimization is represented as one table row. Suitable to analyze one execution of `lrbdmst`.

`./bin/lablogs2sus.sh`
: Transform a set of logfiles as written by `lrbdmst` into `sus`. One execution of `lrbdmst` is represented as one table row. Suitable to analyze a series of computations.

`./bin/comparesus.sh`
: Compare two `sus` files. Suitable to compare two series of computations with different parameters.

`./bin/sus2csv`, `./bin/csv2sus.sh`
: Transform `sus` to `csv` and vice versa. The format `csv` is suited for import into spread sheet applications.

Chapter 6

Computational Experiments

I tested my implementation on the same benchmark instances that have previously been investigated by [GR05a]. The tests were restricted to instances with an even diameter bound. One group of these instances was originally published by Santos et al. [dSLR04]. The other group is originally taken from Beasley's OR-Library [Bea05]. These instances were used by Gouveia and Magnanti [GM03] as BDMST instances. The tests were performed under the Linux operating system `2.6.8-12-amd64-k8-smp` on an AMD Opteron processor 270, 2000MHz.

I compare my results to so far unpublished data from Gruber [Gru06]. That data consists of LP relaxation values, LP relaxations strengthened with various cuts, optimal solution values and corresponding CPU times. Gruber's current research improves previously published results on the Predecessor Depth model in [GR05a].

The two Lagrangian relaxation approaches, together with the different separation strategies for jump constraints, lead to four interesting classes of experiments:

1. Predecessor-Depth approach (section 6.2), respectively

   Predecessor-Jump approach (section 6.3) ...

2. ... with separation strategy $[V_1]$

3. ... with separation strategy $[V_{\text{depth}}]$

4. ... with both[1] separation strategies $[V_1]$ and $[V_{\text{depth}}]$.

The following section discusses some general aspects of the computational experiments. Sections 6.2 and 6.3 present the results obtained with my LR approaches for the Predecessor-Depth model and the Predecessor-Jump model, respectively. The parameters that are applied and the achieved lower

[1] That is, two constraints are generated at once, one according to each strategy.

bounds are discussed. Finally, section 6.4 discusses first experiments on larger instances.

6.1 General Aspects of the Computational Experiments

The results for each of these classes of experiments are quite different. While the Predecessor-Depth approach was relatively insensitive to modifications of the parameters, the results produced by the Relax-and-Cut approach together with Subgradient Optimization for the Predecessor-Jump approach varied significantly under different settings. Usually, changing one parameter improved the results for some of the instances (i.e. running time or lower bound) but degraded the results for others. In general, it proved to be challenging to determine good parameters for the Subgradient Optimization.

Section 2.5 lists various modifications for the Subgradient Optimization. In order to avoid alternating between two vectors of Lagrangian multipliers, I utilized *Direction Vectors*. The *Exceed Upper Bound* modification was also applied. Preliminary experiments indicated that on average both modifications improve the results slightly. On the other hand, *Adjustment of the Subgradient* did not seem to produce any improvement. Therefore, this modification was not employed for the subsequently described experiments.

I present one run for each of the above classes of experiments. The parameters for the SG are the same within each of the classes, but are allowed to differ between the classes. The presented results are indicators of the quality of lower bounds, that could be achieved on average in each of the classes within reasonable time limits. For single instances the results could usually be improved by tweaking the parameters for that specific instance. However, my goal was to find parameters that work well for the whole set of instances. Besides the parameters for the SG, also the initial upper bounds had a substantial impact on the behaviour of the SG. Whether a good or an inferior upper bound was provided, sometimes influenced the performance of the SG significantly. Interestingly, this influence is hard to qualify, since a better upper bound would for some instances result in a better lower bound and in worse lower bounds for others.

The initial upper bound required by the Subgradient Optimization is computed by the RTC heuristic from [Jul04]. The best value produced after 100 attempts is taken. If RTC does not find any feasible solution[2], the initial upper bound is simply taken to be the sum of the edge costs of the $|V| - 1$ most expensive edges. Clearly no tree can cost more than this.

When the solution to a LLBP satisfies the diameter constraint, this corresponds to a feasible solution for the BDMST problem. Additionally, both LR

[2] This can happen on instances that are not complete.

approaches utilize the level construction heuristic from [Gru06] to find feasible BDMST solutions based on the solutions of the LLBPs. Both types of solutions denote upper bounds for the optimal objective value of the BDMST. To distinguish between the best feasible solution found directly by the LR approach, the best heuristic solution and the initial upper bound heuristic, all three values are provided in the results.

An optimality gap is given for every lower bound. Denote the optimal objective value that is known from [Gru06] with O and the achieved lower bound with L. The optimality gap is defined as follows:

$$g = \frac{O - L}{O} \tag{6.1}$$

Additionally, an alternative gap is also given. Note that the Subgradient Optimization starts with all $\lambda_i = 0$. This means that the value of the optimal solution to the LLBP in the first iteration is equal to the value of the minimum spanning arborescence, which – under these circumstances – is in turn equal to the value of the minimum spanning tree. This is true for both Lagrangian relaxation approaches. Therefore, both LR approaches compute lower bounds that are greater or equal to the MST objective value. Accordingly, I decided to take into account, whether the final lower bound improves this initial lower bound. Denote the objective value of the minimum spanning tree with M. The progress that can potentially be made is $(O-M)$, while the progress that is actually made is $(L-M)$. Now the gap G is defined as follows:

$$G = \frac{\text{potential progress} - \text{actual progress}}{\text{potential progress}} \tag{6.2}$$

$$= \frac{(O - M) - (L - M)}{O - M} \tag{6.3}$$

$$= \frac{O - L}{O - M} \tag{6.4}$$

This gap G was used as the primal criterion to decide upon the quality of a lower bound.

A value of 1 (or a value close to 1) indicates that no (or only a very small) improvement upon the objective value of the MST could be achieved. A value of 0 (or a value near 0) means that the lower bound is equal to (or very close to) the optimal objective value.

Note that in experiments where the optimal objective value is reached, SG stops as soon as the smallest integer greater or equal to the lower bound is equal to some upper bound[3]. The values listed in these situations are L and $G = \frac{O-L}{O-M} > 0$, indicating a non-optimal gap and lower bound, although the optimality of $\lceil L \rceil$ with a gap $G = \frac{O-\lceil L \rceil}{O-M} = 0$ has actually been proven.

[3] The initial upper bound, a feasible solution value, or a heuristic solution value.

Furthermore, if – for the same instance – the value of the LP relaxation with cuts is equal to the optimal objective value, this could mistakenly be interpreted as if the LP relaxation with cuts leads to a better result than the lower bound. In fact, both find the optimal objective value.

For each of the classes the following data is given:

Inst The type of the instance, i.e. one of
- `c` Complete instances from [dSLR04].
- `g` Sparse instances from [dSLR04].
- `TE` Euclidean instances from [GM03].
- `TR` Random instances from [GM03].

V The number of nodes.
E The number of edges.
D The diameter bound of the instance.
M The cost of the minimum spanning tree.
LP The LP relaxation from [Gru06].
LPC The LP relaxation strengthened with additional cuts from [Gru06].
Opt The optimal objective value from [Gru06].
LB The best lower bound computed by the corresponding LR approach.
F The best feasible solution found during the SG.
H The best heuristic solution found during the SG.
U The initial upper bound used by SG.
it The number of iterations of the SG.
r The number of repetitions of the SG.
C The number of constraints found by the SG.
A The number of arcs per constraint.
t The running time in seconds.
g The gap $g = \frac{O-L}{O}$.
G The alternative gap $G = \frac{O-L}{O-M}$.

The computation times for the LP relaxation and for the LP relaxation with cuts are not listed explicitly. According to [Gru06] the computation times for the LP relaxation with cuts are always less than 3 seconds for the instances with up to 40 nodes[4]. This is by far lower than the computation times for any of my LR approaches and therefore, a more detailed comparison of running times to the results from [Gru06] would not lead to further insights.

6.2 Results for the Predecessor-Depth Approach

This section discusses the results obtained from the LR approach for the Predecessor-Depth model from section 4.2.3. Table 6.1 compares my results to those from [Gru06].
The following observations can be made:

[4] On an AMD Opteron processor 270, 2000MHz.

Inst	V	E	D	M	LP	LPC	Opt	LB	F	H	U	C	it	t	g	G
TE	20	100	4	292	313.43	315.76	369	292.91	-	-	‡794	400	369	20.3	0.206	0.988
TE	20	100	6	292	286.67	303.00	322	292.00	-	-	354	600	272	20.1	0.093	1.000
TE	20	100	8	292	279.50	298.00	308	292.00	-	-	338	800	272	26.0	0.052	1.000
TE	30	200	4	396	444.12	447.31	599	403.30	-	-	599	800	411	63.3	0.327	0.964
TE	30	200	6	396	390.39	412.18	482	396.00	-	-	‡1256	1200	272	60.1	0.178	1.000
TE	30	200	8	396	364.07	404.67	437	396.00	-	-	482	1600	272	83.7	0.094	1.000
TR	20	100	4	145	180.27	183.55	233	149.00	-	308	‡971	400	472	25.0	0.361	0.955
TR	20	100	6	145	141.76	153.00	178	145.00	-	179	222	600	272	19.7	0.185	1.000
TR	20	100	8	145	136.00	151.00	154	145.00	-	155	185	800	272	25.3	0.058	1.000
TR	30	200	4	132	166.22	169.20	234	140.77	-	234	331	800	622	92.1	0.398	0.914
TR	30	200	6	132	126.83	137.67	157	132.12	-	189	209	1200	359	76.1	0.158	0.995
TR	30	200	8	132	115.00	135.00	135	132.00	-	135	170	1600	272	77.3	0.022	1.000
c	20	190	4	271	267.21	284.83	349	271.00	-	349	349	760	272	29.6	0.223	1.000
c	20	190	6	261	234.15	267.00	298	261.00	-	299	320	1140	272	40.3	0.124	1.000
c	20	190	10	313	239.38	317.50	324	313.00	-	336	359	1900	272	70.6	0.034	1.000
c	25	300	4	366	376.58	394.65	500	368.08	-	500	521	1200	572	118.9	0.264	0.984
c	25	300	6	342	315.23	349.25	378	342.00	-	387	424	1800	272	82.6	0.095	1.000
c	25	300	10	366	333.30	368.50	379	366.00	-	389	439	3000	272	156.8	0.034	1.000
g	20	50	4	332	393.88	395.69	442	336.11	-	446	‡751	200	422	13.1	0.240	0.963
g	20	50	6	300	237.79	304.50	329	300.00	-	329	366	300	272	11.5	0.088	1.000
g	20	50	10	357	240.89	359.00	359	357.00	-	360	377	500	272	17.1	0.006	1.000
g	40	100	4	500	602.85	623.67	755	508.27	-	757	‡1597	400	461	49.0	0.327	0.968
g	40	100	6	570	417.77	581.92	599	570.00	-	616	655	600	272	37.6	0.048	1.000
g	40	100	10	570	392.35	571.50	574	570.00	-	575	629	1000	272	55.6	0.007	1.000

Table 6.1: Computational results for the Predecessor-Depth approach. SG settings: `SG_baseAgility_TerminationLevel` 0.005 and `SG_baseAgility_ReductionAfterNoImprove` 30.

‡The upper bound is the sum of the $|\mathbf{V}| - 1$ most expensive edges.

1. On some instances the computed lower bound is greater than the LP relaxation. On these instances, also the value of the MST is greater than the LP relaxation.

2. The LP relaxation with cuts is in all cases greater than the lower bound computed by the LR approach.

3. On some instances the lower bound does improve upon the value of the MST. Disappointingly, this is only a minor improvement ($G \geq 0.914$) and on most instances no improvement was achieved at all.

4. No feasible solutions are found.

5. Heuristic solutions are found for all instances, except those of type `TE`. In general, they are quite close to the optimum. The optimum is even reached for some instances.

The results of this approach are relatively stable with respect to different values for the SG settings. With even extensively more relaxed parameters[5] the lower bound only improves marginally while consuming much more time.

The conclusion is that the implemented approach from section 4.2.3 produces significantly inferior lower bounds than the LP relaxation with cuts from [Gru06], while – at the same time – requiring multiple orders of magnitude more computation time. Moreover, it improves the value of the MST only in rare cases and only by some minimal amount. This indicates that the Lagrangian relaxation of the Predecessor-Depth model presented in section 4.2.2 does not qualify as a promising basis for further research.

6.3 Results for the Predecessor-Jump Approach

This section discusses the results obtained from the LR approach for the Predecessor-Jump model from section 4.1.4. The results herein are also compared to the values presented in [Gru06] [6].

As mentioned above the combination of Relax-and-Cut and Subgradient Optimization proved to be very sensitive to even minor changes in its parameters when applied to the Lagrangian relaxation for the Predecessor-Jump model. After extensive testing I decided to set `SG_baseAgility_TerminationLevel` to 0.05 and `SG_baseAgility_ReductionAfterNoImprove` to 10, since these values seemed to be a reasonable compromise between quality of the lower bound and total running time. More relaxed values improve the lower bounds slightly on average, but increase the total running time significantly.

[5] More relaxed values in this context means smaller values for `SG_baseAgility_TerminationLevel` and/or bigger values for `SG_baseAgility_ReductionAfterNoImprove`.

[6] Note that the values produced by [Gru06] are based on the Predecessor-Depth model and not on the Predecessor-Jump model.

The three different combinations of dynamic constraint separation strategies[7] turned out to differ in their success to improve the lower bound. Therefore, I decided to use different settings for `SG_repetitionsNoImproveLimit` for each of the three combinations. When both strategies are applied, the lower bounds can be improved early in the Subgradient Optimization and further improvements are achieved with some regularity. In opposition to this, the strategies $[V_1]$ and $[V_{\text{depth}}]$, applied for their own, lead to a faster convergence of the Subgradient Optimization. This happens because fewer improvements can be achieved. The consequence is that, compared to the configuration where both separation strategies are applied, SG terminates after fewer iterations and after a shorter computation time. To compensate this and to allow for a fairer comparison between the different combinations of separation strategies, I decided to set `SG_repetitionsNoImproveLimit` to `6` for the combined strategy, to `8` when only $[V_1]$ was applied, and to `10` when only $[V_{\text{depth}}]$ was applied. These increased values improved the found lower bounds to some degree, but clearly any of the strategies $[V_1]$ or $[V_{\text{depth}}]$ applied for their own produces inferior lower bounds compared to the combined strategy.

Results for the Predecessor-Jump Approach with $[V_1]$

Table 6.2 shows the results of the Relax-and-Cut algorithm applied to the LR approach of the Predecessor-Jump model, together with jump separation according to the strategy $[V_1]$.
The following observations can be made:

1. The lower bounds are better than the values of the minimum spanning trees for all considered instances ($G \leq 0.759$).

2. In almost all cases, the lower bounds are better than the values of the LP relaxation with cuts from [Gru06].

3. Feasible solutions are found for three instances.

4. Heuristic solutions are found for all but one instance. In general, they are very close to the optimum. Actually, the optimum is reached for most of the instances.

5. Two instances can be solved to proven optimality.

Results for the Predecessor-Jump Approach with $[V_{\text{depth}}]$

Table 6.3 presents the results of the Relax-and-Cut algorithm applied to the LR approach of the Predecessor-Jump model, together with jump separation according to the strategy $[V_{\text{depth}}]$.
The following observations can be made:

[7] Only $[V_1]$ or only $[V_{\text{depth}}]$ or both.

Inst	V	E	D	M	LP	LPC	Opt	LB	F	H	U	C	A	it	rep	t	g	G
TE	20	100	4	292	313.43	315.76	369	321.53	-	370	‡794	110	21	4687	46	364.9	0.129	0.616
TE	20	100	6	292	286.67	303.00	322	305.25	-	332	354	89	32	4856	48	498.8	0.052	0.558
TE	20	100	8	292	279.50	298.00	308	301.96	-	308	338	101	40	4904	46	738.6	0.020	0.378
TE	30	200	4	396	444.12	447.31	599	456.51	-	-	599	215	30	9001	89	2152.2	0.238	0.702
TE	30	200	6	396	390.39	412.18	482	420.34	-	491	‡1256	141	43	5153	55	1358.4	0.128	0.717
TE	30	200	8	396	364.07	404.67	437	409.73	-	437	482	206	58	8270	72	4448.4	0.062	0.665
TR	20	100	4	145	180.27	183.55	233	192.79	-	233	‡971	118	20	6892	69	563.8	0.173	0.457
TR	20	100	6	145	141.76	153.00	178	164.77	-	178	222	144	28	10101	76	1626.9	0.074	0.401
TR	20	100	8	145	136.00	151.00	154	152.00	154	154	185	55	37	6921	61	654.4	0.013	0.222
TR	30	200	4	132	166.22	169.20	234	170.99	-	234	331	172	27	5355	55	1074.5	0.269	0.618
TR	30	200	6	132	126.83	137.67	157	139.63	-	157	209	114	40	4419	43	963.8	0.111	0.695
TR	30	200	8	132	115.00	135.00	135	*134.31	-	135	170	36	52	291	9	21.3	0.005	0.231
c	20	190	4	271	267.21	284.83	349	300.39	-	349	349	137	38	6336	74	1642.3	0.139	0.623
c	20	190	6	261	234.15	267.00	298	271.53	-	299	320	136	55	12807	123	4617.8	0.089	0.715
c	20	190	10	313	239.38	317.50	324	319.20	-	332	359	66	86	4057	45	1685.9	0.015	0.436
c	25	300	4	366	376.58	394.65	500	411.97	-	500	521	185	48	9619	88	4737.7	0.176	0.657
c	25	300	6	342	315.23	349.25	378	354.60	-	378	424	153	70	8351	79	5693.6	0.062	0.650
c	25	300	10	366	333.30	368.50	379	369.48	-	379	439	96	111	3026	34	2351.1	0.025	0.732
g	20	50	4	332	393.88	395.69	442	389.03	-	446	‡751	155	14	8703	84	406.5	0.120	0.482
g	20	50	6	300	237.79	304.50	329	307.00	-	329	366	79	18	6896	56	316.8	0.067	0.759
g	20	50	10	357	240.89	359.00	359	§358.30	359	359	377	7	24	96	4	0.8	0.002	0.352
g	40	100	4	500	602.85	623.67	755	595.54	-	755	‡1597	231	20	7255	78	768.8	0.211	0.625
g	40	100	6	570	417.77	581.92	599	584.67	-	599	655	72	19	5133	47	209.5	0.024	0.494
g	40	100	10	570	392.35	571.50	574	571.50	-	574	629	75	37	5019	55	367.7	0.004	0.625

Table 6.2: Computational results for the Predecessor-Jump approach with separation strategy $[V_1]$. SG settings: `SG_repetitionsNoImproveLimit` 8

*The computation stopped because the **LB** was greater than **H**−1, i.e. **H** is the optimum.
§The computation stopped because the **LB** was greater than **F**−1, i.e. **F** is the optimum.
‡The upper bound is the sum of the $|\mathbf{V}| - 1$ most expensive edges.

Inst	V	E	D	M	LP	LPC	Opt	LB	F	H	U	C	A	it	rep	t	g	G
TE	20	100	4	292	313.43	315.76	369	292.00	-	-	‡794	23	27	1753	13	72.2	0.209	1.000
TE	20	100	6	292	286.67	303.00	322	292.00	-	-	354	31	41	1679	13	122.8	0.093	1.000
TE	20	100	8	292	279.50	298.00	308	292.00	-	320	338	13	50	1946	12	99.4	0.052	1.000
TE	30	200	4	396	444.12	447.31	599	396.00	-	-	599	89	43	5637	30	1072.2	0.339	1.000
TE	30	200	6	396	390.39	412.18	482	396.00	-	482	‡1256	38	55	2861	14	522.1	0.178	1.000
TE	30	200	8	396	364.07	404.67	437	396.00	760	437	482	186	80	6415	42	6187.4	0.094	1.000
TR	20	100	4	145	180.27	183.55	233	145.00	-	316	‡971	34	28	3100	19	192.5	0.378	1.000
TR	20	100	6	145	141.76	153.00	178	145.00	380	179	222	36	38	2419	16	175.4	0.185	1.000
TR	20	100	8	145	136.00	151.00	154	145.00	289	171	185	19	49	2148	12	141.0	0.058	1.000
TR	30	200	4	132	166.22	169.20	234	133.00	-	234	331	188	41	16404	83	6929.4	0.432	0.990
TR	30	200	6	132	126.83	137.67	157	132.00	457	161	209	167	61	5017	30	3734.8	0.159	1.000
TR	30	200	8	132	115.00	135.00	135	132.00	255	135	170	106	76	3443	20	3669.9	0.022	1.000
c	20	190	4	271	267.21	284.83	349	271.75	629	349	349	152	38	13958	100	3094.6	0.221	0.990
c	20	190	6	261	234.15	267.00	298	261.00	591	299	320	66	67	4523	29	1164.4	0.124	1.000
c	20	190	10	313	239.38	317.50	324	313.50	344	327	359	101	117	12641	87	17186.6	0.032	0.955
c	25	300	4	366	376.58	394.65	500	367.00	1130	500	521	173	48	8309	60	3554.4	0.266	0.993
c	25	300	6	342	315.23	349.25	378	342.00	1241	387	424	30	79	1753	11	448.4	0.095	1.000
c	25	300	10	366	333.30	368.50	379	366.00	749	393	439	31	142	2496	12	742.3	0.034	1.000
g	20	50	4	332	393.88	395.69	442	335.00	-	446	‡751	117	23	13066	66	1294.9	0.242	0.973
g	20	50	6	300	237.79	304.50	329	300.00	-	329	366	52	30	3636	27	190.7	0.088	1.000
g	20	50	10	357	240.89	359.00	359	357.12	359	359	377	125	38	15631	103	2568.8	0.005	0.938
g	40	100	4	500	602.85	623.67	755	500.00	-	757	‡1597	33	41	2868	15	268.2	0.338	1.000
g	40	100	6	570	417.77	581.92	599	570.00	1443	619	655	100	52	4255	22	1352.0	0.048	1.000
g	40	100	10	570	392.35	571.50	574	570.00	790	580	629	65	78	3125	19	943.0	0.007	1.000

Table 6.3: Computational results for the Predecessor-Jump approach with separation strategy [V_{depth}]. SG settings: `SG_repetitionsNoImproveLimit` 10

‡The upper bound is the sum of the $|\mathbf{V}| - 1$ most expensive edges.

1. The lower bounds are not better than the values of the corresponding minimum spanning trees for most of the considered instances ($G \geq 0.938$).
2. More feasible solutions are found compared to the experiments with the separation strategy $[V_1]$.
3. Heuristic solutions are found for all but two instances. In general, they are very close to the optimum. The optimum is reached for some of the instances.

Results for the Predecessor-Jump Approach with $[V_1]$ and $[V_{\mathbf{depth}}]$

Table 6.4 shows the results of the Relax-and-Cut algorithm applied to the LR approach of the Predecessor-Jump model, together with jump separation according to the strategy $[V_1]$ and $[V_{\text{depth}}]$. In this table the first value in column **C** gives the number of constraints that were separated according to $[V_1]$, and the second value gives the number of separated constraints according to $[V_{\text{depth}}]$.
The following observations can be made:

1. The lower bounds are better than the values of the corresponding minimum spanning trees for all of the considered instances.
2. The lower bound is worse than the value of the corresponding LP relaxation with cuts for only one instance. For all other instances it is equally or even better.
3. Compared to the experiments with the separation strategy $[V_{\text{depth}}]$, even more feasible solutions are found.
4. Heuristic solutions are found for all but one instance. In general, they are very close to the optimum. Actually the optimum is reached for most of the instances.
5. Two instances can be solved to proven optimality.

To summarize, it can be said that combining the two separation strategies actually combines the strength's of both. The lower bounds are, with one exception, always equal or better than the values of the LP relaxation with cuts, and for most of the instances feasible solutions can be found. Unfortunately, the running times to compute the lower bounds are much longer than the ones of the LP relaxation with cuts. It seems that while the Lagrangian relaxation presented in section 4.1.2 does produce good lower bounds, the Relax-and-Cut approach based on Subgradient Optimization requires too many iterations. It might be promising to substitute the Subgradient Optimization with another scheme for solving Lagrangian duals.

Inst	V	E	D	M	LP	LPC	Opt	LB	F	H	U	C	A	it	rep	t	g	G
TE	20	100	4	292	313.43	315.76	369	325.49	-	369	‡794	137/374	28	5146	52	1546.5	0.118	0.565
TE	20	100	6	292	286.67	303.00	322	307.30	-	328	354	113/277	39	3203	35	1066.0	0.046	0.490
TE	20	100	8	292	279.50	298.00	308	301.99	466	308	338	63/130	50	1960	22	443.2	0.020	0.375
TE	30	200	4	396	444.12	447.31	599	449.37	-	-	599	156/326	40	4554	43	2123.0	0.250	0.737
TE	30	200	6	396	390.39	412.18	482	426.95	-	489	‡1256	168/562	64	4138	36	3637.0	0.114	0.640
TE	30	200	8	396	364.07	404.67	437	409.83	723	437	482	165/611	88	3748	37	4249.6	0.062	0.663
TR	20	100	4	145	180.27	183.55	233	200.80	-	233	‡971	149/572	26	5506	41	2292.9	0.138	0.366
TR	20	100	6	145	141.76	153.00	178	164.76	360	178	222	154/388	38	9255	94	4982.2	0.074	0.401
TR	20	100	8	145	136.00	151.00	154	152.00	154	154	185	49/117	51	3831	42	813.3	0.013	0.222
TR	30	200	4	132	166.22	169.20	234	176.75	-	234	331	255/1773	39	11668	100	19067.6	0.245	0.561
TR	30	200	6	132	126.83	137.67	157	143.63	489	157	209	137/849	67	8015	75	16796.8	0.085	0.535
TR	30	200	8	132	115.00	135.00	135	*134.66	283	135	170	70/69	68	672	11	131.8	0.003	0.113
c	20	190	4	271	267.21	284.83	349	301.24	1138	349	349	138/366	38	6761	71	3685.2	0.137	0.612
c	20	190	6	261	234.15	267.00	298	272.59	650	299	320	81/215	65	4043	43	2555.4	0.085	0.687
c	20	190	10	313	239.38	317.50	324	319.36	499	327	359	66/126	106	4047	43	3603.6	0.014	0.422
c	25	300	4	366	376.58	394.65	500	418.26	941	500	521	172/390	48	5982	58	4991.1	0.163	0.610
c	25	300	6	342	315.23	349.25	378	354.67	905	378	424	110/392	86	4738	47	7312.7	0.062	0.648
c	25	300	10	366	333.30	368.50	379	369.33	1186	379	439	85/170	146	5320	54	8746.6	0.026	0.744
g	20	50	4	332	393.88	395.69	442	399.63	693	442	‡751	163/401	22	11787	113	2484.3	0.096	0.385
g	20	50	6	300	237.79	304.50	329	306.50	474	329	366	69/98	24	3463	35	290.5	0.068	0.776
g	20	50	10	357	240.89	359.00	359	*358.73	469	359	377	4/4	33	12	3	0.1	0.001	0.135
g	40	100	4	500	602.85	623.67	755	590.40	-	757	‡1597	179/602	46	7333	79	4730.9	0.218	0.645
g	40	100	6	570	417.77	581.92	599	584.66	1456	616	655	43/301	68	2338	25	851.4	0.024	0.494
g	40	100	10	570	392.35	571.50	574	571.50	-	574	629	69/163	68	3346	35	911.9	0.004	0.625

Table 6.4: Computational results for the Predecessor-Jump approach with separation strategies $[V_1]$ and$[V_{\text{depth}}]$. SG settings: `SG_repetitionsNoImproveLimit` 6

*The computation stopped because the **LB** was greater than **H**−1, i.e. **H** is the optimum.

‡The upper bound is the sum of the $|\mathbf{V}| - 1$ most expensive edges.

6.4 Results for Large Instances

This section gives a rough indication of the performance of the Relax-and-Cut approach applied to the Lagrangian relaxation of the Predecessor-Jump model on larger instances. They are taken from the euclidean Steiner Tree problem instances from Beasley's OR-Library [Bea05]. In [GHR06] different algorithms to find heuristic solutions for these instances have been published. To the best of my knowledge, no optimal objective values are known for these instances. The combination of both separation strategies, $[V_1]$ and $[V_{\text{depth}}]$ is applied in the Lagrangian relaxation approach. The lower bounds computed with the LR approach are compared to the LP relaxation values and the values of the LP relaxation with cuts from [Gru06]. The large computation times prohibited extensive experiments. Nevertheless, the following results allow an estimation of the potential of the implemented LR approach.

Table 6.5 presents the results. In addition to the datasets in the previous sections, this table also lists computation times for the LP relaxation with and without cuts:

nr Number of the instance.
tLP Time to compute the LP relaxation **LP** in seconds.
tLPC Time to compute the LP relaxation with cuts **LPC** in seconds.

The following observations can be made:

1. The lower bound is better than the value of the minimum spanning tree for three of the considered instances.

2. The lower bound is better than the value of the LP relaxation for all of the instances.

3. The lower bound is better than the value of the LP relaxation with cuts for two instances with 100 nodes.

4. For the instance with 500 nodes, the LP relaxation with cuts could not be obtained with CPLEX. Even the LP relaxation without cuts shows a massive increase in computation time. This indicates that instances of this size are too large to be handled by the current LP approach. However, also the LR approach fails to produce a satisfying result on this instance. The lower bound is not greater than the value of the MST.

In summary, it can be seen that it is possible to produce lower bounds that are greater than the values of the LP relaxation with cuts from [Gru06]. As noted in the previous section, these results suggest that substituting the Subgradient Optimization with another method to solve the Lagrangian duals might be an interesting approach.

nr	V	E	D	LP	M	LPC	LB	F	H	C	ApC	it	rep	t	tLP	tLPC
1	100	4950	10	478876	660807	666613	663862.9	5756020	808469	294	744	1774	27	36513	1.3	687
2	100	4950	10	537463	683239	686828	683239.0	4659190	841594	62	1823	301	5	2199	1.4	646
3	100	4950	10	512200	676205	681403	681411.5	2884840	818515	786	916	1471	21	70586	1.4	406
4	100	4950	10	524526	679732	685047	679732.0	4699550	842575	14	606	301	5	517	1.4	1579
5	100	4950	10	528288	690283	697626	702988.8	4810400	844824	1040	895	3265	46	357666	1.8	821
6	100	4950	10	517299	669367	674742	669367.0	5167080	818567	113	1694	468	9	5084	1.4	2128
9	100	4950	10	578289	716480	721751	718490.3	3441480	867738	344	813	2056	33	74762	1.2	1713
1	500	124750	20	1148806	1482667	*-	1482667.0	27783400	1866800	77	34379	301	5	581472	962705	*-

Table 6.5: Computational results for the Predecessor-Jump approach with separation strategies $[V_1]$ and $[V_{\text{depth}}]$ for large instances. SG settings: `SG_baseAgility_TerminationLevel` 0.05, `SG_baseAgility_ReductionAfterNoImprove` 8, and `SG_repetitionsNoImproveLimit` 4.

*The LP relaxation with cuts could not be computed with CPLEX.

Chapter 7

Conclusion

This section summarizes the work and suggests directions for further research. I described two new Lagrangian relaxation approaches for the BDMST problem with an even diameter bound. Computational experiments were conducted on BDMST benchmark instances taken from the literature. Based on these experiments, the implementation of both approaches can be studied empirically.

One Lagrangian relaxation approach is based on the Predecessor-Depth model. This model consists of two types of variables: predecessor variables and depth variables. The set of constraints that couple the predecessor variables and the depth variables are relaxed in the usual Lagrangian way. This leads to a decomposition of the resulting problem into a minimum spanning arborescence problem and a minimum assignment problem. These two subproblems can be solved efficiently. The Lagrangian Dual is solved with Subgradient Optimization. The initial lower bound computed with this approach is the value of the minimum spanning tree. Unfortunately, the best lower bounds computed with this approach are only marginally better than their initial lower bounds.

The second approach is based on the Predecessor-Jump model. This model consists of predecessor variables and contains an exponential number of so called jump constraints. Jump constraints are responsible to limit the length of paths from the centre to any other node. These constraints are relaxed in the usual Lagrangian way. The resulting problem is a minimum spanning arborescence problem, which can be solved efficiently. This relaxation is embedded in a Relax-and-Cut approach. Subgradient optimization and subsequent jump constraint separation are performed iteratively. To separate new violated jump constraints, two different schemes were developed. As with the first approach, the initial lower bound is equal to the value of the minimum spanning tree. The relax and cut approach is able to produce lower bounds that are better than the value of the MST. The best lower bounds are obtained when jump constraints according to both

separation schemes are generated.

The computational results of the experiments with both approaches are compared to results from [Gru06]. This includes LP relaxation values and values of LP relaxation strengthened with additional cuts, based on the Predecessor-Depth model. Most of the lower bounds from [Gru06] are improved by the lower bounds computed with the jump constraints based approach. However, the computation of these lower bounds requires significantly more time than the computation of the LP relaxation with cuts. This observation holds for the smaller benchmark instances. On the largest tested instance with 500 nodes the LP relaxation with cuts could not be computed within on week of computation time. However, the Predecessor-Jump approach was unable to improve the initial MST lower bound for this instance.

The implemented combination of Relax-and-Cut with Subgradient Optimization exhibited two problems: First, it is very sensitive to changes in its parameters, which makes it difficult to find good parameters for a whole set of instances. Second, it is slow, i.e. it requires many iterations to converge. The sensitivity issue was especially troublesome with the jump constraints based approach. Several modifications to the Subgradient Optimization where tested. This includes adjustments to the subgradients as suggested by [Bea93] to avoid unnecessarily low step sizes. Additionally, an enhancement suggested by [CFG01] was implemented. Instead of the current value of the subgradient vector, a direction vector that takes also the previous direction into account was used to update the vector of the Lagrangian multipliers. However, none of the modifications did lead to any significant improvement regarding the performance of the Subgradient Optimization.

The described approaches were equipped with the so called level construction heuristic from [Gru06]. This heuristic takes the level information from the solution of the Lagrangian lower bound problem and creates – when possible – a feasible solution for the BDMST problem. These heuristic solutions provide quite good upper bounds for the optimal objective value of the BDMST problem.

7.1 Future Work

Future work should attempt to replace the Subgradient Optimization with another scheme to solve the Lagrangian duals. Any method that either helps with the sensitivity issue mentioned above, or that is able to improve the lower bounds faster than the Subgradient Optimization will provide a valuable improvement. Additionally, modifications to the Relax-and-Cut approach that lead to a greater number of constraints may also result in better lower bounds and faster convergence.

Investigating large instances, such as attempted in section 6.4, may be interesting. The LP relaxations with cuts can be computed relatively fast on

the instances taken from [dSLR04] and [GM03]. On the instances with 100 nodes from [Bea05] an increase in computation time can be noted. According to [Gru06], the LP relaxation with cuts could not be computed for an instance with 500 nodes within one week of computation time. Actually, even the LP relaxations without cuts took nearly 10^6 seconds to compute for this instance. Here the Lagrangian relaxation based approach may be able to compete with the LP relaxation; in terms of the quality of the lower bound, as well as in terms of processing time.

The LR approach based on the Predecessor-Jump model relies on a transformation of the BDMST problem with an even diameter bound into a HMST problem. This approach could be modified to be able to solve HMST problem instances. This would allow to compare the implemented Lagrangian relaxation approach with results published on HMST[1]. Another aspect one may consider, is to modify the approaches to handle BDMST problems with an odd diameter bound.

Regarding upper bounds and feasible solutions, future research may integrate other, more sophisticated heuristics. This may serve several purposes: First, it can be expected that this will further improve the upper bounds. Second, better upper bounds may in turn also enhance the performance of the Subgradient Optimization.

It may be promising to develop more jump separation schemes, different from the two that were presented here. This should help to increase the number of considered constraints which should in turn lead to faster and earlier improvements in the Subgradient Optimization. It may also improve the best lower bound that is found.

Last but not least it may be interesting to integrate the jump constraints into a Branch and Cut approach. Such an implementation is currently under development by Gruber [Gru06].

[1] For example, [DGR06] present computational results for the HMST problem.

Bibliography

[ADG00] Ayman Abdalla, Narsingh Deo, and Pankaj Gupta. Random-Tree Diameter and the Diameter-Constrained MST. *Congressus Numerantium*, 144:161–182, 2000.

[Bak05] Bastiaan Bakker. Log for C++, Version 0.3.5rc3, 2005. `http://log4cpp.sourceforge.net`.

[Bea93] John E. Beasley. Lagrangian Relaxation. In Colin R. Reeves, editor, *Modern heuristic techniques for combinatorial problems*, chapter 6, pages 243–303. John Wiley & Sons, Inc., New York, NY, USA, 1993.

[Bea05] John E. Beasley. Or-library. Last update: October 2005, 2005. `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/esteininfo.html`.

[BK91] A. Bookstein and S. T. Klein. Compression of Correlated Bit-Vectors. *Information Systems*, 16(4):387–400, 1991.

[BPS93] Krishna Bala, Konstantinos Petropoulos, and Thomas E. Stern. Multicasting in a Linear Lightwave Network. In *INFOCOM '93. Proceedings.Twelfth Annual Joint Conference of the IEEE Computer and Communications Societies. Networking: Foundation for the Future. IEEE*, volume 3, pages 1350–1358, 28 March-1 April 1993.

[CFG01] T. G. Crainic, A. Frangioni, and B. Gendron. Bundle-based relaxation methods for multicommodity capacitated fixed charge network design. *Discrete Applied Mathematics*, 112(1-3):73–99, 2001.

[CLR00] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, 2000.

[DG04] Geir Dahl and Luis Gouveia. On the directed hop-constrained shortest path problem. *Operations Research Letters*, 32:15–22, 2004.

[DGR06] Geir Dahl, Luis Gouveia, and Cristina Requejo. On Formulations and Methods for the HOP-Constrained Minimum Spanning Tree Problem. In Mauricio G. C. Resende and Panos M. Pardalos, editors, *Handbook of Optimization in Telecommunications*, chapter 19. Springer Science and Business Media, New York, NY, 2006.

[dSLR04] A. C. dos Santos, A. Lucena, and C. C . Ribeiro. Solving Diameter Constrained Minimum Spanning Tree Problems in Dense Graphs. In *Proceedings of the International Workshop on Experimental Algorithms*, volume 3059 of *LNCS*, pages 458–467. Springer, 2004.

[Edm67] Jack Edmonds. Optimum Branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240, 1967.

[ES00] H. A. Eiselt and C.-L. Sandblom. *Integer Programming and Network Models*. Springer, 2000.

[FPSE06] Christian Fremuth-Paeger, Bernhard Schmidt, and Birk Eisermann. GOBLIN, A Graph Object Library for Network Programming Problems, Version 2.7.2, 2006. `http://www.math.uni-augsburg.de/~fremuth/goblin.html`.

[GHR06] Martin Gruber, Jano van Hemert, and Günther R. Raidl. Neighbourhood Searches for the Bounded Diameter Minimum Spanning Tree Problem Embedded in a VNS, EA, and ACO. In Maarten Keijzer et al., editor, *Proceedings of the 8th annual conference on Genetic and Evolutionary Computation*, volume 2, pages 1187–1194, New York, NY, USA, 2006. ACM Press.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

[GM03] Luis Gouveia and Thomas L. Magnanti. Network Flow Models for Designing Diameter-Constrained Minimum-Spanning and Steiner Trees. *Networks*, 41(3):159–173, 2003.

[Gou96] Luis Gouveia. Multicommodity flow models for spanning trees with hop constraints. *European Journal of Operational Research*, 95(1):178–190, November 1996.

[Gou98] Luis Gouveia. Using Variable Redefinition for Computing Lower Bounds for Minimum Spanning and Steiner Trees with Hop Constraints. *INFORMS Journal on Computing*, 10:180–188, 1998.

[GR01] Luís Gouveia and Cristina Requejo. A new lagrangian relaxation approach for the hop-constrained minimum spanning tree problem. *European Journal of Operational Research*, 132:539–552, 2001.

[GR05a] Martin Gruber and Günther Raidl. A New 0-1 ILP Approach for the Bounded Diameter Minimum Spanning Tree Problem. In Luis Gouveia and C. Mourão, editors, *Proceedings of the 2nd International Network Optimization Conference*, volume 1, pages 178–185, Lisbon, Portugal, 2005.

[GR05b] Martin Gruber and Günther R. Raidl. Variable Neighborhood Search for the Bounded Diameter Minimum Spanning Tree Problem. In Pierre Hansen, Nenad Mladenovic, José A. Moreno Pérez, Belén Melián Batista, and J. Marcos Moreno-Vega, editors, *Proceedings of the 18th Mini Euro Conference on Variable Neighborhood Search*, Tenerife, Spain, 2005.

[Gru06] Martin Gruber. A Branch-and-Cut Approach for the Bounded-Diameter Minimum Spanning Tree Problem. Technical report, Vienna University of Technology, Institute of Computer Graphics and Algorithms, Algorithms and Data Structures Group, 2006.

[HKPS03] Susan Hert, Lutz Kettner, Tobias Polzin, and Guido Schäfer. ExpLab - A Tool Set for Computational Experiments Version 0.7, 2003. `http://explab.sourceforge.net`.

[JR03] Bryant A. Julstrom and Günther R. Raidl. A Permutation-Coded Evolutionary Algorithm for the Bounded-Diameter Minimum Spanning Tree Problem. In A. Barry, F. Rothlauf, D. Thierens, and et al., editors, *Genetic and Evolutionary Computation Conference's Workshops Proceedings, Workshop on Analysis and Design of Representations*, pages 2–7, 2003.

[Jul04] B. A. Julstrom. Greedy heuristics for the bounded-diameter minimum spanning tree problem. Technical report, St. Cloud State University, 2004. Submitted for publication in the ACM Journal of Experimental Algorithmics.

[LED06] LEDA. Library Of Efficient Data Types And Algorithms, Version 5.1.1. Algorithmic Solutions Software, 2006. `http://www.algorithmic-solutions.com/enleda.htm` .

[Luc05] Abilio Lucena. Non Delayed Relax-and-Cut Algorithms. *Annals of Operations Research*, 140(1):375–410, November 2005.

[Ray89] Kerry Raymond. A Tree-Based Algorithm for Distributed Mutual Exclusion. *ACM Transactions on Computer Systems*, 7(1):61–77, February 1989.

[RJ03] Günther R. Raidl and Bryant A. Julstrom. Greedy Heuristics and an Evolutionary Algorithm for the Bounded-Diameter Minimum Spanning Tree Problem. In *Proceedings of the 2003 ACM Symposium on Applied Computing*, pages 747–752, New York, NY, USA, 2003. ACM Press.

[WA88] Kathleen A. Woolston and Susan L. Albin. The Design of Centralized Networks with Reliability and Availability Constraints. *Computers and Operations Research*, 15(3):207–217, May 1988.