

A branch-and-cut algorithm with betweenness variables for the Linear Arrangement Problem

Diplomarbeit von
Robert Schwarz

Betreuer:
Prof. Dr. Gerhard Reinelt

Universität Heidelberg
Fakultät für Mathematik und Informatik

10. März 2010
(überarbeitet am 7. Juli 2010)

Preface

The subject of this thesis is the Minimum Linear Arrangement Problem, a classical problem in combinatorial optimization. It consists in finding a positioning p of the vertices V of some graph G on a line, that minimizes the weighted sum of the resulting edge lengths:

$$\min_p \sum_{ij \in E} c_{ij} |p(i) - p(j)|$$

Some work has been done to find good feasible solutions via heuristics. Lower bounds have been computed using combinatorial ideas or linear programs, but the gap usually remains quite large. The most successful exact solving algorithm so far is based on dynamic programming and limited to very small graphs.

We present here a novel approach to solving the Minimum Linear Arrangement Problem with a branch-and-cut algorithm using so called betweenness variables. The lower bounds are improved and many instances could be solved to proven optimality.

The thesis is structured as follows: The first chapter introduces necessary notation and elementary results from different fields, including graph theory, linear and integer programming, and polyhedral combinatorics.

Next we highlight some of the previous work on the problem. In particular one integer programming formulation we can borrow many ideas from. In some sense our variables act as a refinement of the variables used there.

The third chapter is dedicated to the new betweenness model. We present some classes of valid inequalities and discuss their separation. We also include results about the polytopes associated with the problem.

SCIP is used as a framework for the branch-and-cut algorithm. We show how our code is integrated with this framework and explain implementation details in the fourth chapter.

We then use a set of benchmark instances to illustrate how our new ideas contribute to the improvement of the algorithm. The fifth chapter is dedicated to these results and compares them to other approaches.

Finally, in the sixth chapter, an alternative integer programming formulation is presented. It uses even more variables, but unfortunately does not turn out to be computationally efficient.

Contents

1 Preliminaries	7
1.1 Graph Theory	7
1.2 Polyhedral Theory	8
1.3 Linear Programming	8
1.4 Integer Programming	9
1.4.1 Branch-and-Bound	10
1.4.2 Cutting Planes	12
2 Minimum Linear Arrangement Problem	15
2.1 Definition	15
2.2 Applications	16
2.3 Complexity	17
2.3.1 General Case	17
2.3.2 Special Classes of Graphs	17
2.4 Previous Work	17
2.4.1 Lower Bounds	18
2.4.2 Upper Bounds	20
3 Betweenness Approach	23
3.1 Betweenness Variables	23
3.2 Polytope	24
3.3 Small Instances	27

3.4	Rank Inequalities	29
3.5	Completion	32
3.6	Relation to the Cut Polytope	34
3.7	Feasibility	36
4	Implementation	39
4.1	SCIP with SoPlex	39
4.1.1	Contribution to Program Code	39
4.1.2	ReaderLAP	40
4.1.3	ConshdlrPQtree	41
4.1.4	ConshdlrComplete	42
4.1.5	HeurMLS	43
4.1.6	General Cutting Planes and Primal Heuristics	44
4.1.7	Presolving	46
4.2	ABACUS with Clp	46
5	Computational Results	47
5.1	Problem Instances	47
5.2	New Results	49
5.3	Comparisons	49
6	x_{ijkl} Approach	53
6.1	Variables	53
6.2	Constraints	54
6.3	Implementation	56
7	Outlook	57
8	Appendix	59

Chapter 1

Preliminaries

In this chapter we summarize definitions and basic results that will be used throughout this work. The main reference is [21]. Some graph theory is needed since graphs are the objects that define our problem instances. Then we review some polyhedral combinatorics that are used to derive some properties about the set of solutions and the constraints that define it. Finally the methods of linear and integer programming are briefly introduced since our algorithm heavily depends on them.

1.1 Graph Theory

An (undirected) **graph** $G = (V, E)$ is defined by its finite set of vertices $V = \{v_1, \dots, v_n\}$ and the set of edges $E = \{e_1, \dots, e_m\}$ that connect the vertices. That is, every edge $e \in E$ is a 2-subset of V . An edge $\{u, v\}$ is often denoted by uv for short. From now on we will always use $n = |V|$ and $m = |E|$. A graph $G' = (V', E')$ is called a **subgraph** of G if $V' \subseteq V$ and $E' \subseteq E$.

A sequence of vertices (v_1, \dots, v_l) is called a **path** in G if for every $1 \leq i < l$, we have $v_i v_{i+1} \in E$ and no vertex occurs more than once. Given some edge weights $w_e, e \in E$, we can now look for the shortest path from some vertex s to another vertex t . I.e., a path with $v_1 = s$ and $v_l = t$ that minimizes $\sum_{i=1}^{l-1} w_{v_i v_{i+1}}$. Shortest paths can be computed in polynomial time, for example using Dijkstra's algorithm [11].

A **cut** of a graph is a partition of the vertices into two sets $V = U \cup W, U \cap W = \emptyset$. We are also interested in the edges **crossing** the cut: $d(U, W) = \{ij \in E \mid i \in U, j \in W\}$. For two given vertices $s, t \in V$, an s - t cut is a cut such that s and t lie in different subsets of the partition.

1.2 Polyhedral Theory

A set $S \subseteq \mathbb{R}^n$ is **convex** if for any two points $x, y \in S$ the line segment they span $\{ax + (1-a)y \mid 0 \leq a \leq 1\}$ is contained in S . The smallest convex set containing all points of a given set R is called the **convex hull**. It can be written as the set of **convex combinations** of all finite subsets of R :

$$\text{conv } R = \left\{ \sum_{i=1}^k a_i x_i \mid k \in \mathbb{N}, x_i \in R, a_i \geq 0, \sum_{i=1}^k a_i = 1 \right\}$$

The **Minkowski sum** of two sets $R, S \subseteq \mathbb{R}^n$ is given by

$$R + S = \{r + s \mid r \in R, s \in S\}.$$

Finitely generated convex cones C are the Minkowski sum of **rays** R_i

$$C = R_1 + \cdots + R_p, \quad R_i = \{cx_i \mid c \geq 0\}, \quad x_i \in \mathbb{R}^n.$$

A **polytope** P is the convex hull of some finite subset $R \subset \mathbb{R}^n$. A finite intersection of half-spaces $\{x \in \mathbb{R}^n \mid a^T x \leq b\}$ is called a **polyhedron**, given by its **outer description**. Bounded polyhedra are polytopes and unbounded polyhedra can be expressed as the Minkowski sum of a polytope with a finitely generated convex cone. The latter is called the **inner description**. With the help of Fourier-Motzkin elimination, one can compute the outer description from the inner description and vice versa. This procedure is implemented in the computer program PORTA¹ [5].

The **dimension** d of a polyhedron P is defined to be the size of a maximal set of affinely independent points in P , minus 1. The points $\{x_0, x_1, \dots, x_n\}$ are **affinely independent**, if $\{x_0 - x_1, \dots, x_0 - x_n\}$ are linearly independent. This way the definition of the dimension is invariant to translations of the polyhedron. P is **full-dimensional**, if $\dim P = n$; otherwise there exist linear equations that hold for all points in P .

A linear inequality $a^T x \leq b$ is **valid** for a polyhedron P if the inequality holds for all $x \in P$. This inequality yields a **face** $\{x \in P \mid a^T x = b\}$, again a polyhedron. Faces of dimension 0 are called **vertices** of P , faces of dimension $d-1$ are **facets**. Note that the empty set and P itself also form faces. Minimal (non-redundant) outer descriptions use facets only (apart from a minimal system of equations).

1.3 Linear Programming

In a **linear program** we have to minimize (or maximize) a linear objective function using real variables subject to linear constraints. Different normal forms

¹Polyhedron Representation Transformation Algorithm.

are used in the literature that can easily be transformed to one each other.

$$\begin{aligned} \max \quad & c^T x \\ \text{s. t.} \quad & Ax \leq b \\ & x \geq 0 \end{aligned}$$

Note that the linear constraints give the outer description of a polyhedron: every row of $Ax \leq b$ cuts off a half-space.

All points satisfying the constraints are called **feasible**. A linear program can either be

- infeasible, if no feasible points exist,
- unbounded, if the feasible set contains a ray, that is parallel to the objective
- or it can be solved to optimality (not necessarily with a unique optimal solution).

Theoretically, linear programs can be solved in polynomial time using the ellipsoid method. In practice, however, simplex methods are applied. These use the fact that one of the vertices of the polyhedron has an optimal value if one exists. They start from any vertex and move to neighboring vertices along edges of the polyhedron in a direction that improves the objective. The algorithm stops when one optimum solution is found, i.e., no improvement is possible. Note that this yields a global optimum since the feasible set $\{x \in \mathbb{R}^n \mid Ax \leq b, x \geq 0\}$ is convex.

1.4 Integer Programming

We can further restrict the values of the variables in linear programs to be integer or even binary. These problems are called (mixed) integer programs. It is known that solving general integer programs is NP-hard.

The convex hull of a finite set of points with integer coordinates results in a polytope whose vertices are all integer. Thus, solving a linear program on this polytope as the feasible set will give integer solutions automatically if the linear programming solver returns a vertex as do, for example, simplex algorithms.

However, for most of time the enumeration of all feasible integer points and the computation of the outer description are computationally too expensive.

Instead a polyhedron is used such that the integer points of the polyhedron are feasible. Of course, solving the linear programming relaxation, i.e., ignoring the integer constraint might lead to solutions with fractional values that have to be dealt with.

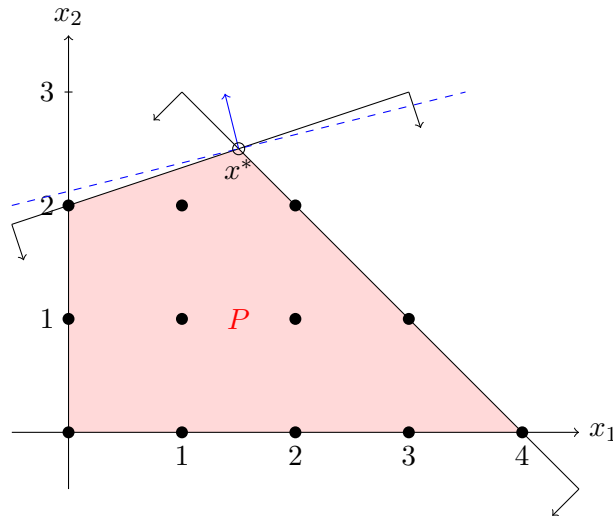


Figure 1.1: Example of an integer program.

An example of an integer program is visualized in figure 1.1. The black dots represent its feasible points and the feasible region of the linear programming relaxation is shown in the shaded area. This polyhedron has four facets defined by two inequalities and the sign constraints of the variables. The solution of the relaxation is denoted by x^* and drawn as a small circle together with the direction of the objective function. The problem is defined as follows:

$$\begin{aligned}
 \max \quad & -x_1 + 4x_2 \\
 \text{s. t.} \quad & -x_1 + 3x_2 \leq 6 \\
 & x_1 + x_2 \leq 4 \\
 & x \in \mathbb{N}^2
 \end{aligned}$$

1.4.1 Branch-and-Bound

The following ideas are based on the presentation in [1].

One general approach to solving integer linear programs is the **branch-and-bound** algorithm. The idea is to divide the master problem into smaller subproblems that are easier to solve. One optimal solution has to be among the solutions of the subproblems. Furthermore, many subproblems can be discarded using dual bounds from the solutions of relaxations and primal bounds from feasible (integer) solutions. In a minimization problem, the relaxation yields lower bounds for the subproblem while global upper bounds are given by feasible solutions.

In algorithm 1, the subproblems are represented in a set. Using the branching relationship, they naturally form a tree with the master problem as root node.

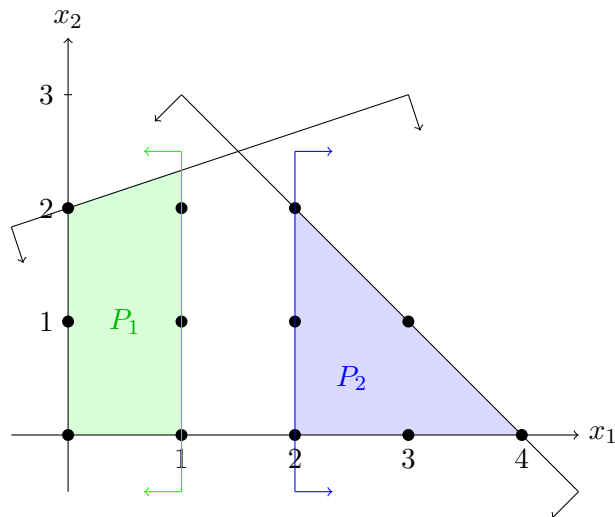


Figure 1.2: Branching on one variable.

Algorithm 1: Branch-and-bound

Input: Problem p (minimization)

Output: Optimal solution x^* with value c^* or, if p is infeasible, $c^* = \infty$

Initialize set of subproblems and upper bound: $P \leftarrow \{p\}$, $\bar{c} \leftarrow \infty$

while $P \neq \emptyset$ **do**

 Remove a subproblem $P \leftarrow P \setminus \{p'\}$

 Solve the relaxation of p' and set $\underline{x}, \underline{c}$ to the solution and value ($\underline{c} = \infty$ if infeasible).

if $\underline{c} < \bar{c}$ **then**

if \underline{x} is feasible in p **then**

 update upper bounds: $\bar{x} \leftarrow \underline{x}$, $\bar{c} \leftarrow \underline{c}$

else

 Create new subproblems $p' = p_1 \cup \dots \cup p_r$ and add them to set

$P \leftarrow P \cup \{p_1, \dots, p_r\}$

return \bar{x}, \bar{c}

Many different strategies are available for the choice of the next subproblem. For example, a depth-first search might be appropriate to get a feasible integer solution as fast as possible. The best-first search chooses the subproblem having the best objective in the relaxation in hope of increasing the (global) lower bound which will also reduce the number of subproblems.

The relaxation of the problem most commonly used is the linear programming relaxation, i.e., the variables no longer have to take integer values. Other relaxations are possible though, like Lagrangian relaxations, combinatorial relaxations or more general forms of convex optimization. The infeasibility of the relaxation implies the infeasibility of the subproblem, which can then be discarded.

If the lower bound of some subproblem is greater than or equal the global upper bound we need no longer consider it, for it might contain feasible solutions, but we are only interested in one optimal solution. When dividing one problem p' in subproblems p_1, \dots, p_r it is important that the feasible solutions of p' equal the union of feasible solutions of p_1, \dots, p_r . One way of subdividing is the **branching on variables** where we create two subproblems p_1, p_2 and bound one variable x by adding the constraints $x \leq \lfloor x^* \rfloor$ and $x \geq \lceil x^* \rceil$ for a fractional value of x^* occurring in the solution of the relaxation. Iterating this process will finally yield an integer solution when all variables are set to their (integer) bounds.

The branching on one variable is illustrated in figure 1.2. In that example the linear programming relaxation has a solution of $x^* = (\frac{3}{2}, \frac{5}{2})$ and two new subproblems are created, one with $x_1 \leq 1$, the other with $x_1 \geq 2$ added to the constraints. All integer feasible points are contained in either of the two subproblems, but the region of fractional points in between is no longer considered. In the worst case that no subproblems can be deleted prior to fixing all variables the algorithm would enumerate all possible feasible solutions. So the running time heavily depends on good lower and upper bounds.

1.4.2 Cutting Planes

Another approach tries to find inequalities that are satisfied by all integer feasible points but violated by the current solution of the linear programming relaxation. Such inequalities can then be added to the problem without changing the feasible set and thus the optimal solution but help get better bounds from the relaxation. Because they remove (or **cut**) the relaxation's solution from the feasible set, they are called **cutting planes**. Repeating this procedure will finally yield an integer solution as shown in [14] where the special class of Gomory Cutting Planes was introduced. Figure 1.3 shows the feasible region of the relaxation after the cut $x_2 \leq 2$ is added to the constraints. This inequality is easily seen to be valid, since all integer feasible points have $x_2 \in \{0, 1, 2\}$.

In many cases the number of constraints of the problem is so huge that solving

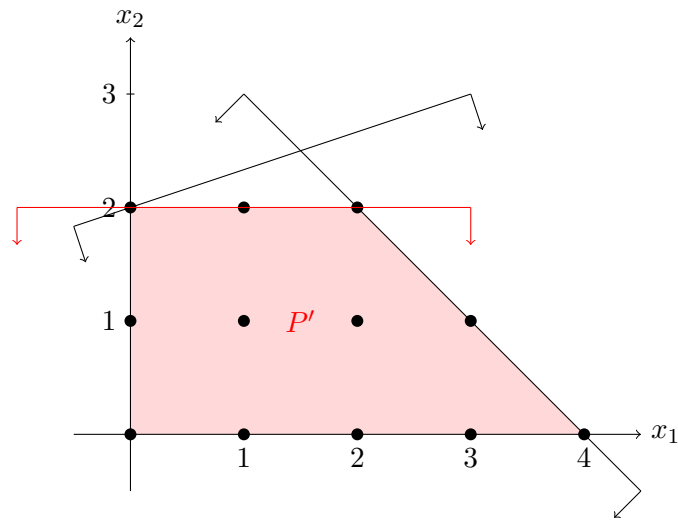


Figure 1.3: Applying Gomory Cutting Planes.

the linear relaxation is already too hard. One can then try to use only a subset of the inequalities first and check for violated constraints to be added later, when the relaxation's solution is available. This last problem is called **separation**.

Chapter 2

Minimum Linear Arrangement Problem

We can now formally define the optimization problem and review the state of research on it. This is by no means a complete survey. Rather, we highlight some important results that are useful for our approach. In particular, one integer programming formulation is presented and compared with our new modeling ideas.

2.1 Definition

A **(linear) arrangement** is a bijective map $p : V \rightarrow \{1, \dots, n\}$. The set of all arrangements of a given graph G is denoted by $A(G)$. Every arrangement corresponds to a permutation of the set $\{1, \dots, n\}$. Thus, like the symmetric group S_n , $A(G)$ has $n! = n(n-1) \cdots 1$ elements.

In the figure 2.1 you can see a graph G and one of its arrangements p . The vertex 4 is mapped to the first position, i.e., $p(4) = 1$.

The problem is now to find an arrangement $p \in A(G)$ that minimizes the distance

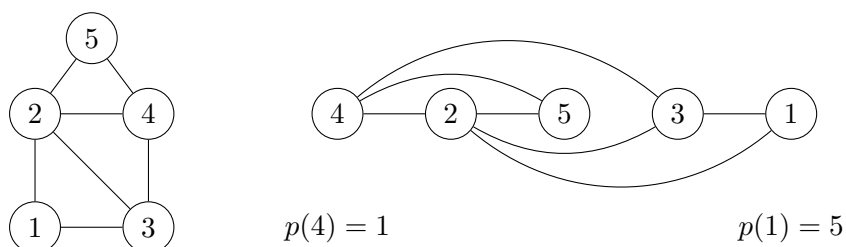


Figure 2.1: Example of a graph G with one possible arrangement p .

of all edges' end vertices in a weighted sum ($c_{ij} \geq 0$):

$$\min_{p \in A(G)} \sum_{ij \in E} c_{ij} |p(i) - p(j)| \quad (2.1)$$

For the graph above, with weights $c_{ij} = 1$, we would have a summed up length of 13 while the minimum arrangement is of length 10.

The Minimum Linear Arrangement Problem is part of a class of combinatorial optimization problems known as layout problems. Another important example is the **Bandwidth Minimization Problem**, where the sum is replaced by a maximum. [10] compiled a survey on all layout problems.

The arrangements are often also called labelings, layouts, orderings or permutations and other names for the Minimum Linear Arrangement Problem in the literature include (Optimal) Linear Arrangement Problem, Minimum Sum Problem and Edge Sum Problem.

To simplify the presentation we will only consider the unweighted case in the following, i.e., $c_{ij} = 1$, $ij \in E$. Of course, all the theory is still applicable to the weighted case.

For a graph G and an arrangement p let $\text{lap}(G, p)$ denote the value of this arrangement. We just write $\text{lap}(G)$ for the value of a minimum linear arrangement. Another name given to this value in the literature is $\text{rank}(G)$.

Note that if a graph G is disconnected, i.e., $V = U \cup W$ with $d(U, W) = \emptyset$, we can solve the linear arrangement problem independently for the individual components. The concatenation of the arrangements for U and W yields an optimal arrangement for V . For that reason we can assume all graphs to be connected from now on.

2.2 Applications

The problem was first proposed in [15] with an application in error-correcting codes. Namely, it consisted in assigning numbers to binary vectors so that the average error is minimized, assuming that only one-bit errors can be introduced during transmission. This is equivalent to the linear arrangement of hypercubes and interestingly, the obvious assignment of numbers to their binary representation is optimal in this respect.

Later it was used in the contexts of VLSI design [25] and graph drawing [4], among others.

2.3 Complexity

2.3.1 General Case

For general graphs, [13] showed that the (decision problem corresponding to the) Minimum Linear Arrangement Problem is NP-complete. This result is still true in the unweighted case and was shown by reduction from the Simple Max Cut Problem (that has all weights equal to 1). The problem remains NP-complete even when we restrict ourselves to only consider bipartite graphs [12]. It is also NP-complete for the class of interval graphs [8].

2.3.2 Special Classes of Graphs

However, there are many cases where (the values of) optimal arrangements are known or can be computed in polynomial time.

For example, the values of their optimal arrangements is known for clique graphs K_n (which is the same for all arrangements) [3]. It is given by

$$\text{lap}(K_n) = \binom{n+1}{3}.$$

Circle graphs C_n [3] have a value of

$$\text{lap}(C_n) = 2(n-1).$$

And the optimal arrangement of hypercubes [15] has a total length of

$$\text{lap}([0, 1]^n) = 2^{n-1}(2^n - 1).$$

The values for stars $K_{1,q}$ and other bicliques¹ $K_{p,q}$ will be given in section 3.4 on page 29. Some more classes of graphs, like paths, prisms and wheels, have known values. The optimal arrangements can be found in [20].

For some other classes of graphs polynomial time algorithms exist that compute the optimal value. These include trees [6], proper interval graphs [23] and more. See [10] for a comprehensive list.

2.4 Previous Work

Until recently, the best way to solve the Minimum Linear Arrangement Problem exactly was by dynamic programming. In [18], such an algorithm was implemented, with a runtime complexity of $O(2^n m)$, that can only be used with very small graphs.

¹also known as complete bipartite graphs.

2.4.1 Lower Bounds

We will now present some methods to get good lower bounds for the optimal value. In particular, one linear programming formulation that will be refined by our approach.

Combinatorial Lower Bounds Some lower bounds can be computed directly from simple data of the graph. For example, in [7] the degree lower bound was presented:

$$\text{lap}(G) \geq \frac{1}{2} \sum_{i \in V} \left\lfloor \frac{(\deg(i) + 1)^2}{4} \right\rfloor$$

This is equivalent to the summed up rank constraints for star graphs that will be introduced in section 3.4 on page 29.

Another bound was given in [16], the edge lower bound. In any arrangement there is a limit of edges having a certain distance. We can then sum up the ascending distances to get this bound.

A more comprehensive list of such bounds can be found in [16] and while many of them are tight for certain graphs the performance is rather weak for our benchmark instances.

Linear Programming Lower Bound The Minimum Linear Arrangement Problem can be formulated with the following integer program [20]:

$$\begin{aligned} \min \quad & \sum_{e \in E} c_e d_e \\ \text{s. t.} \quad & \sum_{j=1}^n x_{ij} = 1 && i \in V && (2.2a) \end{aligned}$$

$$\sum_{i \in V} x_{ij} = 1 \quad j \in \{1, \dots, n\} \quad (2.2b)$$

$$d_{ij} \geq |p - q|(x_{ip} + x_{jq} - 1) \quad ij \in E, p, q \in \{1, \dots, n\} \quad (2.2c)$$

$$x_{ij} \in \{0, 1\} \quad i \in V, j \in \{1, \dots, n\} \quad (2.2d)$$

$$d_{ij} \geq 0 \quad i \in V, j \in \{1, \dots, n\} \quad (2.2e)$$

The binary variables x model the position of the vertices, i.e.,

$$x_{ij} = \begin{cases} 1 & \text{if } p(i) = j \\ 0 & \text{else} \end{cases}$$

while the d variables represent the distance of an edge

$$d_{ij} = |p(i) - p(j)|.$$

The equations in (2.2a) and (2.2b) (with the binary condition (2.2d)) ensure that x actually describes an arrangement: every vertex has exactly one position and exactly one vertex is placed on every position. These kinds of variables and constraints have already been used to model the **Assignment Problem**.

Finally, (2.2c) combine the two types of variables. For all edges and all possible positions of their end vertices, bound the distance from below if the positions are actually assigned to in the arrangement.

Unfortunately, the formulation (2.2) is ineffective in typical branch-and-cut algorithms because the linear programming relaxation gives a very weak lower bound. It has the trivial solution:

$$\begin{aligned} x_{ij} &= \frac{1}{n} & i, j \in V \\ d_{ij} &= 0 & e \in E \end{aligned}$$

In [2] a linear programming formulation was used that uses only the d_e variables of (2.2) constrained by the so called **rank inequalities**:

$$\sum_{e \in H} d_e \geq \text{lap}(H) \quad H \subseteq G \quad (2.3)$$

Using (2.3) for all possible subgraphs $H \subseteq G$ would give an exact formulation, but also need the solution of the problem (for the case $H = G$).

Instead, only certain classes of subgraphs H are used whose optimal values $\text{lap}(H)$ are known or computable in polynomial time. This yields a valid lower bound for the problem.

This approach was developed further in [3]. Instead of just using the rank inequalities from subgraphs H occurring in G , the completion K_n of G was considered. Valid inequalities could then be projected back to the original problem keeping the number of variables at m . In section 3.5 on page 32 a similar method is used together with our betweenness variables.

As we will see in the next chapter, the d_e variables and the betweenness variables are in close relation, namely (3.1) on page 24. In this way the betweenness approach can be seen as a refinement of this formulation.

Integer Program Using Binary Distance Variables A variation of the above approach was presented in [24]. Here, **binary distance variables** d_{ijk} were used, for every $1 \leq i < j \leq n$ and $1 \leq k \leq n - 1$, with

$$d_{ijk} = \begin{cases} 1 & \text{if } |p(i) - p(j)| = k \\ 0 & \text{else.} \end{cases} \quad (2.4)$$

These variables can be related to the integer distance variables d_{ij} , via

$$d_{ij} = \sum_{k=1}^{n-1} k d_{ijk} \quad (2.5)$$

which allows formulating the objective and reusing the above constraints.

The following constraints constitute an integer programming formulation of the Minimum Linear Arrangement Problem:

$$\min \sum_{ij \in E} c_{ij} \sum_{k=1}^{n-1} k d_{ijk}$$

$$\text{s. t. } \sum_{k=1}^{n-1} d_{ijk} = 1 \quad \{i, j\} \subset V \quad (2.6a)$$

$$\sum_{i < j \in V} d_{ijk} = n - k \quad k \in \{1, \dots, n-1\} \quad (2.6b)$$

$$\sum_{j \neq i} d_{ijk} + d_{ij(n-k)} = 2 \quad i \in V, k < \left\lfloor \frac{n}{2} \right\rfloor \quad (2.6c)$$

$$\sum_{j \neq i} d_{ijk} \leq 1 \quad i \in V, k \in \left\{ \left\lfloor \frac{n-1}{2} \right\rfloor, \dots, n-1 \right\} \quad (2.6d)$$

$$d_{ijk} \in \{0, 1\} \quad i < j \in V, k \in \{1, \dots, n-1\} \quad (2.6e)$$

The first type of constraint, (2.6a), enforces that every edge has exactly one distance. On the other hand, (2.6b) counts how often a given distance can occur between pairs of vertices. Long distances are rare, while short distances occur more often. The next two types of constraints, (2.6c) and (2.6d), deal with distances that span less or more than half of the arrangement, respectively. Finally, in (2.6e), all variables are restricted to binaries.

There is also a sparse version of the formulation where the variable d_{ijk} is only used if there is an edge $ij \in E$.

Many more valid inequalities have been found and used in a branch-and-cut-and-price algorithm.

See table 5.2 on page 50 for lower bounds resulting from both of the two models compared with our approach.

2.4.2 Upper Bounds

There is an abundance of heuristical methods and approximation schemes for the Linear Arrangement Problem. We refer the reader to [10] for an overview.

To get a good upper bound from the beginning we will also integrate a heuristic in our branch-and-cut algorithm later. See section 4.1.5 on page 43.

Chapter 3

Betweenness Approach

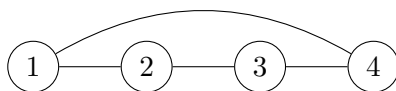
In this chapter, the betweenness approach is introduced. After the definition of the so called betweenness variables we analyze the polytope resulting from the convex hull of the feasible solutions. Some different classes of valid constraints are listed and their separation routines explained. Finally, an additional feasibility test is presented that takes a binary vector and finds all matching arrangements.

3.1 Betweenness Variables

For every edge $ik \in E$ and every vertex $j \in V \setminus \{i, k\}$, the binary variable x_{ijk} indicates whether j lies between i and k given a particular arrangement p . That is,

$$x_{ijk} = \begin{cases} 1 & \text{if } p(i) < p(j) < p(k) \text{ or } p(k) < p(j) < p(i) \\ 0 & \text{else.} \end{cases}$$

The example in figure 3.1 features a circle graph C_4 with four vertices. This particular arrangement places the vertices 1 and 4 at the extreme positions. Thus, this edge is very wide and the other two vertices 2 and 3 lie between them. All other edges are narrow and have no vertices between them. The resulting vector



$$x = \begin{matrix} & 132 & 142 & 124 & 134 & 213 & 243 & 314 & 324 \\ \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \end{matrix}$$

Figure 3.1: One arrangement of C_4 with corresponding betweenness variables.

of between variables is also shown below the graph.

The total number of betweenness variables is $r = m(n-2)$. Using these variables one can easily express the distance of two end vertices of an edge ik

$$|p(i) - p(k)| = 1 + \sum_{j \in V \setminus \{i, k\}} x_{ijk}, \quad (3.1)$$

and the objective value

$$\text{lap}(G, p) = \sum_{ik \in E} |p(i) - p(k)| = m + \sum_{ik \in E} \sum_{j \in V \setminus \{i, k\}} x_{ijk}, \quad (3.2)$$

since the distance is determined by the number of intermediate vertices.

Using (3.1), all valid inequalities for the d_e variables can be transformed to valid inequalities in the betweenness variables simply by variable substitution. In section 3.4 on page 29 we will revisit this idea for the rank inequalities.

Note that the map taking an arrangement $p \in A(G)$ to the corresponding betweenness vector $x \in \{0, 1\}^r$ is neither injective nor surjective. On the one hand, several arrangements can lead to the same betweenness values. For example, if $p = (1, \dots, n)$, then the reversed arrangement $p' = (n, \dots, 1)$ will have the exact same betweenness vector. For disconnected graphs even more arrangements are equivalent in this respect. On the other hand, not all binary vectors x actually describe betweenness for some arrangement. A clique graph can not have empty betweenness, for instance. The latter can be tested by an efficient algorithm, as explained in section 3.7 on page 36.

3.2 Polytope

For a graph G , the betweenness polytope is defined as

$$P_{BTW}(G) = \text{conv}\{x^p \mid p \in A(G)\},$$

where x^p denotes the vector of betweenness variables for the arrangement $p \in A(G)$.

This polytope is not full-dimensional in general, since we have some valid equations:

Proposition 3.2.1. *For all triangles $\{i, j, k\} \subset V$, with $\{ij, ik, jk\} \subset E$, the following equation is valid for $P_{BTW}(G)$:*

$$x_{ijk} + x_{ikj} + x_{jik} = 1 \quad (3.3)$$

Proof. Clearly, in all possible arrangements of i, j and k exactly one of them lies between the other two. \square

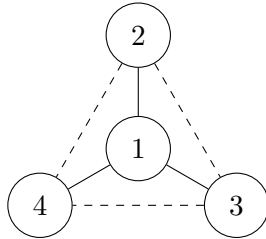


Figure 3.2: The 3-star H (solid edges) is a subgraph of K_4 .

If the value x_{ijk} is interpreted as the angle at j between the two rays ji and jk , scaled down by $\frac{1}{\pi}$, then we get through (3.3) the equation for the sum of angles in a triangle that is known from elementary geometry.

Since all the coefficients in the objective function are positive and we are minimizing, it is possible to add the cone $\mathbb{R}_{\geq 0}^r$ to the polytope $P_{BTW}(G)$ to get the dominant $P_{BTW}^D(G)$. Optimization over the full-dimensional and unbounded polyhedron $P_{BTW}^D(G)$ yields the same solutions. All valid inequalities for $P_{BTW}^D(G)$ are of the form $a^T x \geq b$ with $a \geq 0$, $b > 0$ which is an important feature used in section 3.5 on page 32.

The polytopes have a nice feature that allows us to use information from subgraphs for the main problem:

Proposition 3.2.2. *Given a graph G and a subgraph $H \subseteq G$, all valid inequalities $a^T x \leq b$ for $P_{BTW}(H)$ are also valid for $P_{BTW}(G)$.*

Proof. The coefficients for new variables x_{ijk} in $a^T x \leq b$ are set to 0. Thus, their value is not important for the inequality's validity. If only new edges are added to H , the set of arrangements $A(G) = A(H)$ does not change. On the other hand, if some vertices are added, their positions again do not matter. All these arrangements in $A(G)$ can be considered to have sub-arrangements in $A(H)$ after removing the additional vertices and closing the gaps. \square

Unfortunately, trivial lifting is not possible in general. For example, we will see in proposition 3.4.1 on page 30 that for the graph H shown in figure 3.2 we have the valid inequality

$$x_{123} + x_{124} + x_{132} + x_{134} + x_{142} + x_{143} \geq 1 \quad (3.4)$$

that is facet-defining for $P_{BTW}(H)$. According to proposition 3.2.2, (3.4) is still valid for $P_{BTW}(K_4)$, but no longer facet-defining because it can be written as the

sum of the following valid constraints:

$$\begin{aligned} x_{234} + x_{243} + x_{324} &= 1 \\ x_{132} + x_{134} - x_{234} &\geq 0 \\ x_{142} + x_{143} - x_{243} &\geq 0 \\ x_{123} + x_{124} - x_{324} &\geq 0 \end{aligned}$$

The first line is a triangle equation (see proposition 3.2.1), the last three are odd cycle cut inequalities (see section 3.6 on page 34).

We first suspected that trivial lifting is possible if the subgraph is missing some vertices, but even this claim proved to be wrong: in the section 3.3 one can see that the variable bounds $x_{ijk} \geq 0$ define facets of $P_{BTW}(K_3)$, but not $P_{BTW}(K_4)$. Therefore, we can not even trivially lift inequalities in this case.

The dimension of the betweenness polytope can be computed quite easily.

Proposition 3.2.3. *The dimension of the associated betweenness polytope for clique graphs is given by*

$$\dim P_{BTW}(K_n) = 2 \binom{n}{3}. \quad (3.5)$$

Proof. As noted earlier, there are $r = m(n-2)$ variables which gives

$$r = \binom{n}{2}(n-2) = \frac{(n-2)n!}{(n-2)!2!} = 3 \binom{n}{3}. \quad (3.6)$$

Also, as shown in section 3.3 we have the equation

$$x_{ijk} + x_{ikj} + x_{jik} = 1 \quad (3.7)$$

for every 3-subset $\{i, j, k\} \subset V$.

These $\binom{n}{3}$ equations of type (3.7) are linearly independent since all rows are non-zero and no two rows have any terms in common.

Therefore, we have an upper bound on the dimension:

$$\dim P_{BTW}(K_n) \leq r - \binom{n}{3} \stackrel{(3.6)}{=} 2 \binom{n}{3}. \quad (3.8)$$

On the other hand, for any given 3-subset $\{i, j, k\} \subset V$, we can look at the arrangements

$$\begin{aligned} p_1 &= (i, j, k, u_4, \dots, u_n), \\ p_2 &= (j, i, k, u_4, \dots, u_n), \\ p_3 &= (k, i, j, u_4, \dots, u_n), \\ p_4 &= (k, j, i, u_4, \dots, u_n), \end{aligned}$$

that result in four different, but similar betweenness vectors x^1, x^2, x^3, x^4 . Now let $x^{ijk} = x^1 - x^2 - x^3 + x^4$, then almost all components will cancel out leaving $x_{ijk}^{ijk} = 2$, $x_{jik}^{ijk} = -2$. Reversing the roles of i and k above yields another such vector y^{ijk} with $y_{ijk}^{ijk} = 2$, $y_{ikj}^{ijk} = -2$. We do this for all choices of $\{i, j, k\}$, and get $2\binom{n}{3}$ vectors, two for every triangle in K_n .

Now, let z denote the sum of the betweenness vectors of all arrangements. Then it has the same non-zero value on all components, and (with individual elements scaled appropriately) the set

$$\{z\} \cup \{z + x^{ijk}, z + y^{ijk} \mid \text{for } \{i, j, k\} \subset V\}$$

is affinely independent and contained in the affine hull of $P_{BTW}(K_n)$. Therefore, we have

$$\dim P_{BTW}(K_n) \geq 2\binom{n}{3}.$$

Together with (3.8) this proves the claim of the proposition. \square

Using this proposition, we are able to produce a more general result.

Proposition 3.2.4. *For any graph $G = (V, E)$, the dimension of the associated betweenness polytope is*

$$P_{BTW}(G) = m(n-2) - t \tag{3.9}$$

where t denotes the number of triangles in G , i.e., 3-subsets $\{i, j, k\} \subseteq V$, such that $\{ij, ik, jk\} \subseteq E$.

Proof. We have already seen in the proof of proposition 3.2.3 that every triangle induces a valid equation and that all these equations are linearly independent.

Now assume we find another valid equation $a^T x = b$ that is not induced by all the triangle equations. Then we have two valid inequalities ($a^T x \leq b$ and $a^T x \geq b$) that are also valid for the completion of G , namely $P_{BTW}(K_n)$ (according to proposition 3.2.2). But this would contradict the proposition 3.2.3 so the assumption must be wrong and the triangles provide a minimal system of equations for $P_{BTW}(G)$. \square

3.3 Small Instances

For small numbers of vertices it is possible to compute the complete outer descriptions of the polytope $P_{BTW}(K_n)$ using PORTA [5]. This computation has been done in [22] for $n \in \{3, 4, 5\}$.

For $P_{BTW}(K_3)$ we have the following equations and facets:

$$\begin{aligned} x_{123} + x_{132} + x_{213} &= 1 \\ x &\geq 0 \end{aligned}$$

All these equations are also valid for all 3-subsets of K_4 and K_5 . Furthermore, for $P_{BTW}(K_4)$, we have the following facets for all permutations of $\{1, 2, 3, 4\}$:

$$x_{123} + x_{132} + x_{214} + x_{314} \geq 1$$

Note that the trivial bounds $x \geq 0$ are no longer facet-defining.

Finally, for all permutations of $\{1, 2, 3, 4, 5\}$, these are the facets of $P_{BTW}(K_5)$ in addition to the above facets from $P_{BTW}(K_4)$:

$$x_{132} + x_{154} + x_{245} + x_{315} + x_{324} \geq 1$$

$$\begin{aligned} x_{125} + x_{132} + x_{134} + x_{135} + x_{142} + x_{154} + x_{213} + x_{214} + x_{215} + x_{243} \\ + x_{325} + x_{354} \geq 3 \end{aligned}$$

$$\begin{aligned} x_{125} + x_{132} + x_{143} + x_{153} + x_{213} + x_{214} + x_{215} + x_{234} + x_{243} + x_{345} \\ + x_{354} + x_{425} \geq 3 \end{aligned}$$

$$\begin{aligned} x_{132} + x_{134} + x_{135} + x_{142} + x_{143} + x_{145} + x_{153} + x_{154} + x_{213} + x_{214} \\ + x_{325} + x_{425} \geq 3 \end{aligned}$$

$$\begin{aligned} x_{125} + x_{132} + x_{134} + x_{135} + x_{143} + x_{154} + x_{213} + x_{214} + x_{215} + x_{243} \\ + x_{254} + x_{325} + x_{345} + x_{354} + x_{425} \geq 3 \end{aligned}$$

$$\begin{aligned} x_{124} + x_{125} + x_{132} + x_{134} + x_{143} + x_{152} + x_{213} + x_{214} + x_{243} + x_{245} \\ + x_{315} + x_{324} + x_{354} + x_{435} \geq 4 \end{aligned}$$

$$\begin{aligned} x_{124} + x_{125} + x_{132} + x_{134} + x_{142} + x_{153} + x_{213} + x_{215} + x_{243} + x_{254} \\ + x_{314} + x_{324} + x_{345} + x_{435} \geq 4 \end{aligned}$$

$$\begin{aligned} x_{125} + x_{132} + x_{134} + x_{135} + x_{142} + x_{215} + x_{253} + x_{143} + x_{145} + x_{153} \\ + x_{213} + x_{214} + x_{325} + x_{354} + x_{425} \geq 4 \end{aligned}$$

$$x_{123} + x_{125} + x_{132} + x_{143} + x_{145} + x_{154} + x_{214} + x_{215} + x_{253} \\ + x_{314} + x_{435} \geq 5$$

$$x_{123} + x_{125} + x_{132} + x_{134} + x_{142} + x_{145} + x_{153} + x_{154} + x_{214} + x_{215} \\ + x_{235} + x_{243} + x_{253} + x_{254} + x_{314} + x_{315} + x_{324} + x_{345} + x_{425} + x_{435} \geq 6$$

To separate these facets we can enumerate all k -subsets of V for $k \in \{3, 4, 5\}$ and simply check violation for all permutations of the vertices. For big graphs, this may take too long, so we can abort the search after some number of unsuccessful tries.

Of course, since the polytopes $P_{BTW}(K_n)$ are not full-dimensional (see proposition 3.2.3), every facet admits infinitely many representations via inequalities. We can always add some linear combination of equations and still define the same facet. Therefore, a normal form was used in which all coefficients are non-negative integers. Furthermore, for all triangles $\{i, j, k\}$, at least one of the terms $\{x_{ijk}, x_{ikj}, x_{jik}\}$ must have coefficient 0. Any inequality can easily be transformed to its normal form by adding or subtracting suitable triangle equations (3.3). This normal form does not only give a unique representation for each facet, but also ensures a proper form that will be useful later in the context of proposition 3.5.1 on page 33.

3.4 Rank Inequalities

For all subgraphs $G' = (V', E') \subseteq G$ we have a valid inequality

$$\sum_{ik \in E'} \sum_{j \in V' \setminus \{i, k\}} x_{ijk} \geq \text{lap}(G') - |E'|. \quad (3.10)$$

that we get by substituting (3.1) in the rank inequalities

$$\sum_{ik \in E'} d_{ik} \geq \text{lap}(G'). \quad (3.11)$$

Actually, we do not really need to substitute all terms in (3.1) if the subgraph G' has fewer vertices. For the distance in G' , we have

$$d_{ik}^{G'} = 1 + \sum_{j \in V'} x_{ijk}. \quad (3.12)$$

So we can restrict ourselves to adding only those x_{ijk} where j is already in the subgraph G' , thus getting a stronger inequality. By adding up only ‘‘partial’’

distances we should get an improvement over the model using distance variables d_e that always uses the “full” distances.

Many special classes of graphs have known optimal values and are easy to enumerate as subgraphs. We present some examples that are used in our algorithm. All of these are variations of results in [2].

Stars A p -star graph consists of a center vertex c and p outer vertices u_1, \dots, u_p with edges cu_1, \dots, cu_p . We have

$$\sum_{i=1}^p d_{cu_i} \geq \text{lap}(K_{1,p}) = \left\lfloor \frac{(p+1)^2}{4} \right\rfloor, \quad (3.13)$$

which gives the valid inequality

$$\sum_{i=1}^p \sum_{j \neq i} x_{cu_i u_j} \geq \left\lfloor \frac{(p-1)^2}{4} \right\rfloor. \quad (3.14)$$

Proposition 3.4.1. *Given a p -star $K_{1,p}$, the inequality (3.14) is facet-defining for $P_{BTW}(K_{1,p})$ if and only if p is odd.*

Proof. We first show that the inequality is not facet-defining if p is even. Let $p = 2k$, $k \in \mathbb{N}$ and $K_{1,p} \setminus \{u_i\}$ be the subgraph of $K_{1,p}$ after removing one of the outer vertices. For each of these subgraphs, (3.14) holds and we can sum up all of these inequalities to get another valid inequality of the form

$$\sum_{i=1}^p \sum_{j \neq i} (p-2)x_{cu_i u_j} \geq p \left\lfloor \frac{(p-2)^2}{4} \right\rfloor.$$

The coefficient of $p-2$ at $x_{cu_i u_j}$ is the result of summing up p inequalities, where the term is missing twice: once for u_i and once for u_j . We can now divide by the greatest common divisor of the coefficients, $p-2$, and get the new right-hand side

$$\begin{aligned} \frac{p}{p-2} \left\lfloor \frac{(p-2)^2}{4} \right\rfloor &= \frac{2k}{2k-2} (k^2 - 2k + 1) = k^2 - k \\ &= \left\lfloor \frac{4k^2 - 4k + 1}{4} \right\rfloor = \left\lfloor \frac{(p-1)^2}{4} \right\rfloor, \end{aligned}$$

which is in fact equal to the right-hand side of (3.14) for $K_{1,p}$. Therefore, (3.14) can not be facet-defining.

Let us now consider (3.14) for any odd $p = 2k + 1$, $k \in \mathbb{N}$.

The arrangement shown in figure 3.3 is denoted by $(i_k, \dots, i_1, c, j_0, j_1, \dots, j_k)$ and is optimal for all permutations of the outer vertices.

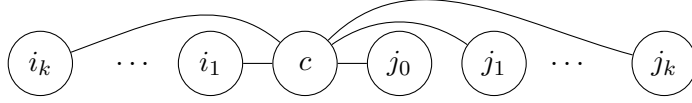


Figure 3.3: Optimal arrangement of odd p -stars $K_{1,p}$.

Let us write $a^T x \geq a_0$ for (3.14) and let us assume that $b^T x \geq b_0$ is a valid inequality for $P_{BTW}(K_{1,p})$ with

$$\{x \in P_{BTW}(K_{1,p}) \mid a^T x = a_0\} \subseteq \{x \in P_{BTW}(K_{1,p}) \mid b^T x = b_0\}. \quad (3.15)$$

Let $\{u, v, w\} \subset \{u_1, \dots, u_p\}$ denote arbitrary outer vertices of $K_{1,p}$. We will now consider different optimal arrangements:

$$\begin{aligned} p_0 &= (i_k, \dots, i_2, w, c, v, u, j_2, \dots, j_k), \\ p_1 &= (i_k, \dots, i_2, w, c, u, v, j_2, \dots, j_k), \\ p_2 &= (j_k, \dots, j_2, v, c, u, w, i_2, \dots, i_k), \\ p_3 &= (i_k, \dots, i_2, v, c, u, w, j_2, \dots, j_k), \\ p_4 &= (j_k, \dots, j_2, w, c, u, v, i_2, \dots, i_k). \end{aligned}$$

To these arrangements correspond vectors of betweenness variables, denoted x^0, x^1, x^2, x^3, x^4 that all satisfy $a^T x = a_0$, and hence $b^T x = b_0$. Therefore, we have

$$\begin{aligned} 0 &= b^T x^0 - b^T x^1 \\ &= (b_{cuv} + \sum_{l=2}^k b_{cwi_l} + b_{cvj_l} + b_{cuj_l}) - (b_{cuv} + \sum_{l=2}^k b_{cwi_l} + b_{cuj_l} + b_{cvj_l}) \\ &= b_{cvu} - b_{cuv} \end{aligned}$$

and also

$$\begin{aligned} 0 + 0 &= b^T x^1 - b^T x^2 - b^T x^3 + b^T x^4 \\ &= (b_{cuv} + \sum_{l=2}^k b_{cwi_l} + b_{cuj_l} + b_{cvj_l}) - (b_{cuv} + \sum_{l=2}^k b_{cvj_l} + b_{cui_l} + b_{cwi_l}) \\ &\quad - (b_{cuv} + \sum_{l=2}^k b_{cvi_l} + b_{cuj_l} + b_{cwj_l}) + (b_{cuv} + \sum_{l=2}^k b_{cwj_l} + b_{cui_l} + b_{cvi_l}) \\ &= 2b_{cuv} - 2b_{cuv}. \end{aligned}$$

Because we have chosen $\{u, v, w\}$ arbitrarily, we know that $b^T = (f, \dots, f)$, for some $f \in \mathbb{R}$ and thus $b = fa$, $b_0 = fa_0$. Note that $P_{BTW}(K_{1,p})$ is full-dimensional according to proposition 3.2.4 on page 27 since $K_{1,p}$ does not contain any triangles. Also, f has to be positive or else $b^T x \geq b_0$ would be violated by all non-optimal arrangements. So, (3.14) is indeed facet-defining for $P_{BTW}(K_{1,p})$. \square

Bicliques The p -stars are special cases $K_{1,p}$ of complete bipartite graphs, also called bicliques. The p, q -biclique $K_{p,q}$ has a partition of vertices into two subsets V_1, V_2 with $|V_1| = p$ and $|V_2| = q$.¹ All vertices $v_1 \in V_1$ are connected to all vertices $v_2 \in V_2$, but no other edges exist in the graph. The optimal value is given by

$$\text{lap}(K_{p,q}) = \begin{cases} p(3q^2 + 6pq - p^2 + 4)/12 & \text{if } p + q \text{ is even} \\ p(3q^2 + 6pq - p^2 + 1)/12 & \text{if } p + q \text{ is odd} \end{cases}$$

from which we get the inequalities

$$\sum_{i \in V_1} \sum_{k \in V_2} \sum_{j \in V \setminus \{i,k\}} x_{ijk} \geq \begin{cases} p(3q^2 + 6pq - p^2 + 4)/12 - pq & \text{if } p + q \text{ is even} \\ p(3q^2 + 6pq - p^2 + 1)/12 - pq & \text{if } p + q \text{ is odd.} \end{cases}$$

Doublestars For another special case, namely $K_{2,q}$, also called q -double stars, with q odd, there exist stronger inequalities. Let $V_1 = \{i, j\}$,

$$\sum_{k \in V_2} 2d_{ik} + d_{jk} \geq 3(q^2 + 4q - 1)/4 \quad (3.16)$$

yielding

$$\sum_{k \in V_2} \sum_{l \in V \setminus \{i,k\}} 2x_{ilk} + \sum_{k \in V_2} \sum_{l \in V \setminus \{j,k\}} x_{jlk} \geq 3(q^2 - 1)/4. \quad (3.17)$$

Note that one gets the usual inequality for bicliques by adding up two inequalities of type (3.17) and dividing by the greatest common divisor of the coefficients, which is 3.

All these classes can be searched for as subgraphs, given fixed bounds on p and q . A more detailed description of our separation routines will be given later, in section 4.1.4 on page 42.

3.5 Completion

In this section we will be able to construct an artificial solution to the linear programming relaxation on the completion K_n of our problem graph G . Separated inequalities on this solution can then be used for our original problem.

Given an edge $ik \in E$ and any i - k path $P = (i, i_1, \dots, i_l, k)$ in G not using ik , we get a circle C . Now, for any $j \notin C$ and any arrangement p , if j lies between ik it also has to lie between at least one of the edges in P . Thus, we get the inequality

$$\sum_{uv \in P} x_{ujv} \geq x_{ijk}. \quad (3.18)$$

¹without loss of generality, we can assume $p \leq q$.

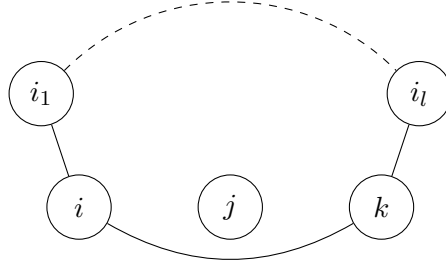


Figure 3.4: Intermediate vertex j , cutting the $i - k$ path.

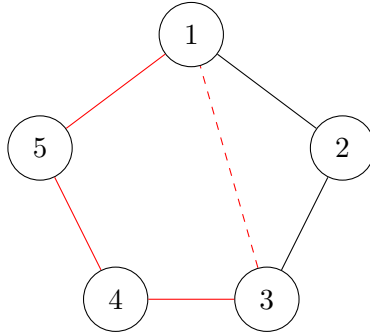


Figure 3.5: Circle graph C_5 with triangle through path.

See figure 3.4 for an illustration.

Using this we can now compute (shortest) paths for all vertices j and all missing edges ik in $G \setminus \{j\}$ using x_{ujv} as weight for the edge uv . That way we can do separation of cuts on the completion of our graph, at least for some kinds of inequalities:

Proposition 3.5.1. *Let $a^T x \geq b$ be any inequality with $a \geq 0$ that is valid for $P_{BTW}(K_n)$. By substituting $\sum_{uv \in P} x_{ujv}$ for x_{ijk} , using any i - k path in G for all edges $ik \notin E$ we get a valid inequality $\bar{a}^T x \geq b$ for $P_{BTW}(G)$.*

Proof. We have already seen that

$$\sum_{uv \in P} x_{ujv} \geq x_{ijk} \quad (3.19)$$

holds for any i - k path P in $G \setminus \{j\}$. Therefore, also $\bar{a}^T x \geq b$ is valid for $P_{BTW}(K_n)$. Now all terms in $\bar{a}^T x \geq b$ with non-zero coefficients correspond to variables that are present in $P_{BTW}(G)$, the projection from $P_{BTW}(K_n)$. \square

Note that all valid inequalities for the dominant $P_{BTW}^D(G)$ fulfill the necessary property of proposition 3.5.1.

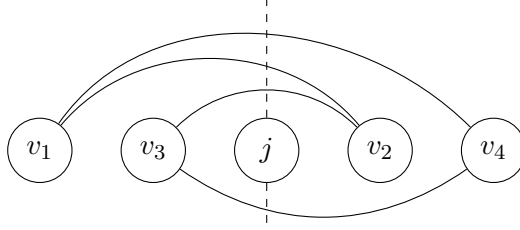


Figure 3.6: Even cut of a circle.

As an example we consider the circle graph C_5 . Suppose we want to separate the triangle equation from $P_{BTW}(K_3)$ for the vertices $\{1, 2, 3\} \subset C_5$:

$$x_{123} + x_{132} + x_{213} = 1 \quad (3.20)$$

Of course, there is no edge connecting 1 and 3 in C_5 , but we can look at the 3-1 path $(3, 4, 5, 1)$ (see figure 3.5). According to (3.18) using the remaining triangle vertex 2 as middle index we get

$$x_{324} + x_{425} + x_{125} \geq x_{123}. \quad (3.21)$$

Substituting the left-hand side of (3.21) in (3.20) we no longer have an equation, but the following valid inequality

$$\underbrace{x_{324} + x_{425} + x_{125}}_{\geq x_{123}} + x_{132} + x_{213} \geq 1. \quad (3.22)$$

Using PORTA we can see that (3.22) is actually a facet-defining inequality of $P_{BTW}(C_5)$.

3.6 Relation to the Cut Polytope

As we have seen in section 3.5, if we have a circle C and some vertex j that lies between some edge $e \in C$ it also has to lie between some other edge $f \in C$. More specifically, the number of edges in a cut of C at some vertex j always has to be even, as illustrated in figure 3.6

Thus, we have a constraint for all vertices $j \in V$ and circles $C \subseteq E(G \setminus \{j\})$

$$\sum_{ik \in C} x_{ijk} \equiv 0 \pmod{2}. \quad (3.23)$$

The constraint (3.23) can be modeled as a system of linear inequalities, explicitly forbidding an odd number of cuts:

$$\sum_{ik \in C \setminus J} x_{ijk} \geq \sum_{ik \in J} x_{ijk} - (|J| - 1) \quad \text{for all odd } J \subseteq C, \quad j \in V. \quad (3.24)$$

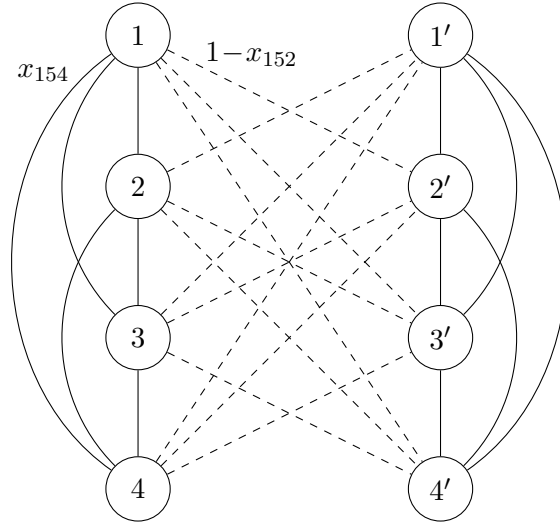


Figure 3.7: Graph \tilde{G} for $G = K_5$, $j = 5$.

Now, if j lies between all edges of one odd subset J it also has to lie between at least one other edge in C .

For the special case $|J| = 1$ we get

$$\sum_{ik \in C \setminus \{uv\}} x_{ijk} \geq x_{ujv} \quad uv \in C, \quad j \in V \quad (3.25)$$

which proved to be most effective and is quite easy to separate. Assuming we have already computed all the shortest paths, as described in section 3.5 we can just compare the value x_{ijk} and the length of the shortest $i - k$ path in $G \setminus \{j\}$ for all edges $ik \in E$. If the path is shorter than this value we have found a violated inequality of the type (3.25).

Separation of the general (3.24) is more elaborate (but still possible in polynomial time). For any $j \in V$ we construct a new weighted graph $\tilde{G} = (\tilde{V}, \tilde{E})$ out of two copies of $G \setminus \{j\}$ in the following way.

$$\begin{aligned} \tilde{V} &= \{u, u' \mid \text{for } u \in V \setminus \{j\}\} \\ \tilde{E} &= \{uv, u'v', u'v, uv' \mid \text{for } uv \in E(G \setminus \{j\})\} \\ w_{uv} &= w_{u'v'} = x_{ujv} \\ w_{u'v} &= w_{uv'} = 1 - x_{ujv} \end{aligned}$$

See figure 3.7 for an example of such a construction. In this graph \tilde{G} we now look for a shortest $i - i'$ path P for all vertices $i \in V \setminus \{j\}$. Note that this path has to choose an odd number of the edges between $\{1, 2, 3, 4\}$ and $\{1', 2', 3', 4'\}$ (the dashed edges in figure 3.7). These correspond to the set J . Interpreted in the

original graph the path will form a circle of length

$$\sum_{uv \in P \setminus J} x_{ujv} + \sum_{uv \in J} (1 - x_{ujv}). \quad (3.26)$$

Now, if (3.26) < 1 we have indeed found a violated inequality of type (3.24).

In fact, there is a more general relation to the **cut polytope**

$$P_{CUT}(G) = \text{conv}\{x \in \{0, 1\}^E \mid x_{ij} = 1 \Leftrightarrow ij \in d(U, W)\}.$$

Take some vertex $j \in V$ and look only at the variables x_{ijk} , $ik \in E$. For different arrangements these variables correspond to different cuts in $G \setminus \{j\}$. One can see this in figure 3.6, where the vertex j 's position yields a partition of $V \setminus \{j\}$: Some vertices are placed on one side of j , the rest of the vertices on the other side of j . In our example we would have $U = \{v_1, v_3\}$, $W = \{v_2, v_4\}$. We can now see that all edges in $d(U, W)$ are really cut by the vertical dashed line at j .

In theory, all constraints for $P_{CUT}(G \setminus \{j\})$ can also be used for our problem, but we restrict ourselves to the odd circle cut inequalities, mentioned above.

One problem with this idea is the fact that we would like to have inequalities of the form $a^T x \geq a_0$ with positive coefficients, to improve the lower bound of the linear programming relaxation. However, such inequalities can not be valid for $P_{CUT}(G)$, since the empty set is a valid cut set, i.e., $0 \in P_{CUT}(G)$.

Also, all constraints from the cut polytope use only variables with the same fixed middle index j . These constraints might be too local to be effective on their own.

Note that even if we add all constraints from $P_{CUT}(G)$ we still do not have a complete integer programming formulation. This can easily be seen in any graph containing a triangle: If we look at all the different cuts corresponding to middle indices $j \in V$, individually, the empty cut would be feasible. But we already know that in a triangle at least one of the three vertices has to lie between the other two.

3.7 Feasibility

Since a complete integer programming formulation (using betweenness variables) is not known to us, not all binary solution vectors x satisfying all the above inequalities are actually feasible. The algorithm described in this section is able to check the feasibility of any binary vector. In theory we could even solve the Minimum Linear Arrangement Problem by explicit enumeration of all binary vectors using this test.

First we transform the problem to a Consecutive Ones Problem [22]. Here a 0/1 matrix M is given, with the task to permute the columns such that in each row, the entries with a 1 appear consecutively.

In our case, M will be a matrix with rows of dimension n . For every edge $ik \in E$ we look at all the vertices in $V_{ik} = \{j \mid x_{ijk} = 1\}$. If $V_{ik} = \emptyset$ we know that i and k should be adjacent in any arrangement, so we insert a row r , with $r_i = r_k = 1$ and 0 otherwise. Else all the vertices in V_{ik} are adjacent to each other as well as i and k , with the latter ones lying at the margins. So we add two rows, one with ones at index i and $j \in V_{ik}$ and another with ones at index k and $j \in V_{ik}$. Now there is a permutation to put M in consecutive ones form if and only if there is an arrangement that is described by x .

To solve the Consecutive Ones Problem we use the PQ -tree algorithm that has linear runtime and outputs all feasible permutations. An implementation is provided by [19].

If no such permutation is found x can not be feasible and should be cut off by adding a suitable linear inequality.

Let $I = \{1, \dots, r\}$ be the index set for x , $J = \{i \in I \mid x_i = 1\}$, then

$$\sum_{i \in J} x_i - \sum_{i \in I \setminus J} x_i \leq |J| - 1 \quad (3.27)$$

is violated by x but no other binary vector of dimension r .

Chapter 4

Implementation

We implemented our algorithm using SCIP¹ as a branch-and-cut framework. In the following we highlight some implementation details and their respective impact on the performance of the algorithm.

4.1 SCIP with SoPlex

SCIP was written with the goal of solving Constraint Integer Programs, a superclass of Mixed Integer Programs and a subclass of Constraint Programs. It is based on a mixed integer solver, but allows more general constraints in the model. In addition to the established methods in Mixed Integer Programming, SCIP applies ideas from Constraint Programming that proved to be useful, e.g., in satisfiability problems [1].

For the relaxation, SCIP relies on an external linear programming solver. There are interfaces to many commercially available solvers, but we use SoPlex², which is shipped together with SCIP (and the modeling language ZIMPL) in the ZIB optimization suite [26]. All these tools are freely available for academic use.

4.1.1 Contribution to Program Code

All of the infrastructure needed for the implementation of a branch-and-bound algorithm is already provided by SCIP. It manages the tree of subproblems and automatically updates the bounds. The rest of the functionality is available via plugins.

In the file `cppmain.cpp` we can choose which plugins to include, load a settings

¹Solving Constraint Integer Programs.

²Sequential Object-oriented Simplex.

file with program parameters and proceed to begin optimization. We have written four new plugins that we include in addition to a default set of plugins, useful for general mixed integer programs. These are:

- **ReaderLAP**: reads the input file and creates problem instance.
- **ConshdlrPQtrees**: ensures feasibility of solutions.
- **ConshdlrComplete**: improves lower bound by separating inequalities.
- **HeurMLS**: yields upper bound.

We now describe the function of every plugin in more detail.

4.1.2 ReaderLAP

SCIP already has file readers for many different formats, like the MPS or CPLEX LP formats for mixed integer programs.

We use a simple sparse format to save the graph of an instance in a text file whose name should have the ending “.lap” so that SCIP knows which reader to use. The first two numbers specify the number of vertices and edges, respectively. We then have a list of edges by specifying the tail and head vertices of each edge, one in every line. Note that we start naming the vertices at 1, using consecutive integers. No weights are given for the edges, we only consider the unweighted case $c_{ij} = 1$, $ij \in E$.

This data is first converted to both adjacency lists and an adjacency matrix and then saved in an object **ProbDataLAP** that will be available globally throughout the algorithm.

Next we create the appropriate binary variables x_{ijk} for every edge ik and vertex j . There are also some (artificial) variables x_{iik} , fixed to 1 from the beginning. These take care of the constant term in (3.1) to get the correct edge length in the objective function.

Then we search the graph for triangles and add the matching equations. This is not really necessary since the relaxed triangle inequalities will be separated anyways, but experiments have shown that it still helps to add the inequalities in the other direction as well.

Consider the problem instance **fidap005** in table 5.1 on page 48. It has the highest density of all graphs and 288 triangles, with only 27 vertices. Therefore, it comes as no surprise that the triangle equations are very important here. Indeed, these equations alone are sufficient to enforce the optimal lower bound: No separation step is needed in this case.

name	with equations		without equations	
	time	subproblems	time	subproblems
fidap005	3.78	1	211.48	1
fidapm05	700.62	1	3812.75	1
pores_1	24.85	1	33.21	1

Table 4.1: Impact of the triangle equations.

Table 4.1 shows the runtime for some of the densest graphs (in seconds), with and without adding all triangle equations at the beginning. For all of these examples we can see an improvement by adding the equations. In one of them we gain a speed up by a factor of over 50.

Finally, we add the two constraint handlers already mentioned above to the problem: `ConshdlrPQtree` and `ConshdlrComplete`.

In SCIP, constraint handlers have several tasks and are responsible for a whole class of constraints. For example, they have to provide callback functions that allow to deal with potential solutions of subproblems. The function `scip_check` gets a solution x and returns `true` if and only if this solution satisfies all constraints of the type that is represented by this handler. This would already be sufficient to run an exact branch-and-bound algorithm, enumerating all possible solutions. To speed up the process, more callback functions can be provided. For example, `scip_sepalp` tries to separate a certain class of valid inequalities to improve the linear programming relaxation. Later, `scip_enfolp` has to deal with infeasible solutions of the relaxation. The options include adding more constraints, branching or declaring the whole subproblem as infeasible. The different constraint handlers are called in order of given priorities. This way some constraint handlers with fast separation routines can be called early and often; while others, with more elaborate constraints can be called later, e.g., after integrality of the solutions is enforced.

4.1.3 ConshdlrPQtree

This constraint handler is necessary to prove feasibility of some potential solution because we do not have a complete integer programming formulation of the linear arrangement problem. It is possible that our relaxation produces a binary vector x , i.e., without fractional values that still does not correspond to any arrangement p .

We use the PQ-tree algorithm described in section 3.7 on page 36 to enumerate all arrangements that match the given betweenness values. If there is one such arrangement the upper bound can be improved. Otherwise the infeasible x is cut

off by a (weak) linear inequality.

The priority of `ConshdlrPQtree`'s `scip_enfolp` is given a negative value so that the integrality constraint (with priority 0) is always satisfied before the PQ-tree algorithm is used. In practice, this means that we will (have to) branch on variables when no more cuts can be separated or the tailing off is activated.

Still, this constraint handler's function `scip_check` is always called for solutions produced by the different primal heuristics employed in SCIP. As we can see in table 4.3 on page 45, almost no time is spent performing this test although we call it quite often.

4.1.4 ConshdlrComplete

Here we try to add more valid inequalities derived for $P_{BTW}(G)$ to cut off the current solution x of the relaxation and improve the lower bound.

First we compute all shortest paths in the graphs $G \setminus \{j\}$ for all vertices $j \in V$, using x_{ijk} as weight for edge $ik \in E$. Now, in addition to the LP solution values x_{ijk} that are already in the problem, we get artificial values x_{ijk} for all pairs $i, k \in V$ even if the edge ik is not in G . After separating some inequality on this extended set of variables, i.e., in the completion of the graph, we substitute the sum of all variables in the shortest path for all artificial values. This was already explained in section 3.5 on page 32.

At the beginning, the odd cut inequalities are searched for (see section 3.6 on page 34) by enumerating all edges $ik \in E$ and all possible middle indices $j \in V$. We can simply check (3.25) for violation using the shortest i - k path with j in between, computed above. Other odd sets are not separated since they did not prove useful.

After that, we separate rank inequalities for subgraphs actually occurring in G . At the moment these include stars, doublestars and bicliques with a smaller subset of 3 or 4 vertices. This separation is done heuristically, following an idea proposed in [3]. For stars we enumerate all possible center vertices $c \in V$ and compute for all neighboring vertices $i \in V$ the distance of the edge ci according to (3.1) on page 24. After sorting the neighbors we try adding more and more of the nearest neighbors, always checking for a violated (3.14). While this method is exact for the d_{ij} variables in [3], it is only heuristical in our case. This is because we compute the edge lengths according to their total length, while only using the partial distance later on (see (3.12) on page 29). Similar approaches are used for the other types of graphs as well.

Then we enumerate all 3-, 4- and 5-subsets of V and check the facets of $P_{BTW}(K_5)$ for violation. Since we only save one element of each class of facets, we have to check all possible permutations of the vertices.

Finally, we also separate some rank inequalities in the completion of the graph. Here we restrict ourselves to odd stars and odd doublestars that give relatively stronger inequalities, as we showed in proposition 3.4.1 on page 30 and (3.17) on page 32, respectively. The enumeration scheme is analogous to the separation of rank inequalities in G .

Altogether a lot of inequalities can be separated in a very short amount of time, but not all of them should be used in the relaxation. In the different separation routines the enumeration is stopped if a certain number of violated inequalities is found or a limit of unsuccessful tries is reached. Furthermore, not all found cuts are actually added to the linear program. Instead, a subset is selected according to three different criteria:

The inequality $a^T x \geq a_0$ should cut off as much as possible, so we are looking for a high distance of x to the hyperplane $\{x \mid a^T x = a_0\}$.

We also want to avoid adding too many similar inequalities that slow down the solving of the linear program. Therefore, we prefer cuts that are highly orthogonal to the already active inequalities.

Finally, for our new cut to improve the lower bound as much as possible, it is advantageous to have a to be aligned with the objective vector c .

A weighted sum of these criteria is used to determine a useful subset of appropriate size. We use the default setting in SCIP, that is, while the distance and orthogonality have the same weight of 1.0, the parallelity to the objective is deemed less important and weighted with 0.0001.

4.1.5 HeurMLS

At the beginning of the branch-and-cut algorithm we call an external heuristic once that provides an upper bound for the problem. It is a multi-start local search heuristic developed by G. Reinelt. Starting from random permutations several different local search heuristics are applied [24]. Very often this solution turns out to be optimal later. As we can see in the following examples, the upper bound is quite efficient in reducing the number of nodes in the tree of subproblems and speeding up the solving process.

Let us consider two doublestars, $K_{2,10}$ and $K_{2,11}$. The heuristic finds an optimal solution for both problems quite fast. Also, because we separate the rank inequalities for the doublestars, our lower bound jumps to the optimal value after just one separation step. This is, of course, also true when not using the heuristic. But in this case we can not simply stop the algorithm (having bounded the value from both sides). Rather, the linear programming relaxation has to find a feasible solution which still takes some time. For $K_{2,10}$, a feasible solution is found after

name	with heuristic		without heuristic	
	time	subproblems	time	subproblems
$K_{2,10}$	0.15	1	60.81	9
$K_{2,11}$	0.19	1	3.22	1

Table 4.2: Impact of the upper bound heuristic.

branching 8 times. But with $K_{2,11}$, we are lucky, because the plugin `feasumpump`³ finds a feasible solution while still in the root node of the branch-and-bound tree. See table 4.2 for the respective runtimes (in seconds). The effect is particularly strong in these graphs, probably because they are highly symmetric and there exist many different optimal arrangements.

4.1.6 General Cutting Planes and Primal Heuristics

Out of the box a lot of separators for general mixed integer programs are provided as plugins in SCIP. However, they do not seem to be of much help for our specific problem. The same is true for primal heuristics. Different kinds of rounding heuristics are called whenever the linear programming solution is solved, but they rarely yield a feasible solution that improves our upper bound. This does not really do any harm, for the time consumption of all these plugins is negligible. In table 4.3 we show how much time is spent in the active, i.e., actually called components of the algorithm. We use the same settings “all” as in tables 5.3 and 5.4 on pages 51 and 51 with a time limit of 21600 seconds.

The first four rows show the time spent in the different constraint handlers. `integral` is used for the integrality constraint and manages the branching, for example. `pqtree` performs our feasibility test on potential solutions. In `complete`, we do the separation of all our problem-specific cuts. The `linear` constraint handler manages the linear constraints, e.g., the triangle equations that are added at the beginning and only exist for problems `gd95c` and `gd96c`.

The next group of rows is for the general cutting plane separators. Note that in instance `gd95c` the upper bound from the MLS heuristic is higher than the actual optimal value. In this case the first feasible (integer) solution is found by the linear programming relaxation, which suggests that more time is spent in separation here.

Next we list all active primal heuristics. Other than our problem specific heuristic `MLS`, no feasible solutions are produced for these four instances. However, in some cases an optimal solution was found here, e.g., for the doublestars we considered earlier in section 4.1.5.

³`Feasibility pump`, built into SCIP.

	gd95c	gd96b	gd96c	gd96d
integral	0.00	0.00	0.00	0.00
pqtree	0.00	0.00	0.00	0.00
complete	8.14	379.37	16.24	74.81
linear	0.00	-	0.00	-
redcost	0.00	0.14	0.01	0.20
gomory	0.00	-	-	-
strongcg	0.00	-	-	-
cmir	0.19	-	-	-
flowcover	0.70	-	-	-
clique	0.00	-	-	-
mcf	0.00	-	-	-
oneopt	0.00	0.02	0.01	-
MLS	31.24	54.69	32.50	180.44
trivial	0.00	0.02	0.01	0.04
simplerounding	0.00	0.08	0.00	0.04
rounding	0.01	1.01	0.13	0.97
shifting	0.05	2.96	0.18	5.15
dual LP	43.86	21112.68	1557.98	21305.77
total	87.79	21600.00	1617.20	21600.00

Table 4.3: Time consumption (in seconds).

As we can see in the last row, most of the time is spent in solving the linear programming relaxation, in our case using a dual simplex algorithm.

SCIP provides a long list of parameters for all its features together with some parameter sets for different purposes. We normally use the default settings, but also considered two more parameter sets, namely `emph/hardlp` and `emph/optimal`. Hopefully, this would improve the time spent solving the linear programming relaxation, or prove optimality of our heuristical solution faster.

The results of this comparison can be seen in table 4.4. Using `emph/hardlp` clearly gives very poor performance compared to the default settings. Only the first instance can be solved to optimality, but with a longer runtime. For all other instances the lower bound is significantly weaker. With these settings, the algorithm will start branching very early. For instance, to solve `gd95c` 68 subproblems were considered. Of course, the problem can be solved in the root node of the branch-and-bound tree when sufficiently many cuts are separated. On the other hand, it is not obvious whether the setting `emph/optimal` will generally improve the results. For these tests it yields poorer results for `gd95c` and `gd96b`, but better results for `gd96c` and `gd96d`. It seems like this set of

name	default		emph/hardlp		emph/optimality	
	lb	time	lb	time	lb	time
gd95c	506	87.8	506	558.1	506	93.0
gd96b	1401.43	limit	1273.97	limit	1398.01	limit
gd96c	519	1617.2	484.98	limit	519	1454.0
gd96d	1499.42	limit	1106.77	limit	1525.11	limit

Table 4.4: Comparison of parameter settings with different emphasis.

parameters does not have a big influence on the behavior of the algorithm here. In general, practically all instances are either solved in the root node, or not at all. Branching was only helpful in some specially constructed and very regular graphs as we have seen in section 4.1.5 with the doublestars.

4.1.7 Presolving

A variety of presolving routines can help a lot when solving general mixed integer programs. Unfortunately, we can not use it in our case because we have very few constraints in the relaxation at the beginning. Not even all variables occur in these so they would be fixed to 0 and removed from the problem prematurely.

4.2 ABACUS with Clp

A previous implementation used ABACUS as the framework for the branch-and-cut algorithm. Its design is presented in [17]. Most of the information about the separation of inequalities is also valid for this implementation. When translating this older code to the new version using SCIP, however, some details (mostly regarding the choice and amount of separation) have changed. Therefore, a direct comparison of the two programs would not yield meaningful insights about the amount of overhead each framework introduces.

Again, many external linear programming solvers are supported, via COIN-OR's⁴ OSI⁵. We decided to use Clp, the default solver that is shipped with OSI [9].

Some features already mentioned above are not present in the ABACUS code by default and had to be provided separately. Most notably, we implemented a ranking of violated cuts, to choose the best subset to be added the relaxation. See [5] for details on the methods used here.

⁴Computational Infrastructure for Operations Research

⁵Open Solver Interface

Chapter 5

Computational Results

Using the implementation of the algorithm described above we get some new results. Mostly, the lower bound of many instances is improved, but we can even solve the Minimum Linear Arrangement Problem for some of the graphs to proven optimality.

5.1 Problem Instances

We use instances from different benchmarks so that we can compare our results. Because of the large number of variables, graphs with many vertices and edges can not be solved efficiently. We therefore filter out those instances having more than 1000 edges. From [10] we use the smallest four of the **gd*** instances. In [3] the authors also provided lower bounds for some graphs that were previously used in the Minimum Bandwidth Problem. The last block of instances was used in [24] in addition to other instances already mentioned. Due to the abundance of appropriately sized problems, no random graphs were generated.

In the table 5.1, we list some of the graphs' properties. Using the number of vertices and edges one can compute the **density** of the graph

$$\text{dens}(G) = \frac{2m}{n(n-1)}.$$

And given the number t of triangles, proposition 3.2.4 on page 27 tells us the dimension of the associated betweenness polytope $P_{BTW}(G)$.

The last column shows the best upper bound to our knowledge. Most of the feasible solutions were produced by the heuristic. In some cases our linear programming relaxation found a better feasible solution. Provably optimal values are marked using a **bold font**.

name	n	m	dens	t	dim	ub
gd95c	62	144	0.076	88	8552	506
gd96b	111	193	0.032	0	21037	1416
gd96c	65	125	0.060	20	7855	519
gd96d	180	228	0.014	0	40584	2391
bccspwr01	39	46	0.062	2	1700	106
bccspwr02	49	59	0.050	3	2770	161
bccspwr03	118	179	0.026	23	20741	662
bccspwr04	274	669	0.018	582	181386	4727
bccsttk01	48	176	0.156	160	7936	1132
can__24	24	68	0.246	60	1436	210
can__61	61	248	0.135	396	14236	1137
can__62	62	78	0.041	2	4678	210
can__73	73	152	0.057	32	10760	1100
can__96	96	336	0.073	320	31264	2703
can__144	144	576	0.055	912	80880	3224
can__161	161	608	0.047	592	96080	6696
can__187	187	652	0.037	620	120000	5191
can__229	229	774	0.029	690	175008	9836
dwt__59	59	104	0.060	30	5898	496
dwt__66	66	127	0.059	62	8066	192
dwt__72	72	75	0.029	0	5250	167
dwt__87	87	227	0.060	147	19148	932
dwt__162	162	510	0.039	464	81136	2431
dwt__209	209	767	0.035	707	158062	9263
dwt__221	221	704	0.028	608	153568	4470
dwt__245	245	608	0.020	374	147370	3883
bwm200	200	298	0.014	0	59004	496
fidap005	27	126	0.358	288	2862	414
fidapm05	42	239	0.277	671	8889	1003
lshp-265	265	744	0.021	480	195192	6088
nos4	100	247	0.049	36	24170	1031
pde225	225	420	0.016	0	93660	3040
rdb200	200	460	0.023	0	91080	3808
saylr1	238	445	0.015	0	105020	3107
steam3	80	424	0.134	992	32080	1416
tub100	100	148	0.029	0	14504	246
curtis54	54	124	0.086	78	6370	454
ibm32	32	90	0.181	28	2672	485
impcol_b	59	281	0.164	285	15732	2076
pores_1	30	103	0.236	88	2796	383
will57	57	127	0.079	94	6891	335

Table 5.1: Problem instances from the literature.

We summarize the best lower bounds for all these instances in table 5.2 comparing our results with those in [3] and [24]. Note that we pick the best values among different settings from [3]. A “-” denotes missing data when an instance was not considered in one approach.

For our code two different settings are used. In the first column we force the adding of separated rank inequalities from subgraphs in the graph. The next setting does not necessarily add all these inequalities. Rather, we let SCIP choose a subset among all found cuts. For most instances the latter option is more successful, though not by a wide margin.

Please note that a branch-and-cut-and-price algorithm was used in [24]. This means that the lower bound is not reliable if the time limit was reached, since some important variables might not yet be priced in. The value 559 that resulted for the instance `gd95c` is greater than the optimal value of 506, for example.

5.2 New Results

As we can see in table 5.2, most of the smaller sized graphs can be solved to optimality. Apart from `dwt__66`, none of these benchmark instances could be solved before. In some cases the optimal solution we found is an improvement on the best known solution given by heuristics, e.g., `gd95c`, where the feasible solution of the heuristic has a value of 508.

In some more cases we can still compute a better lower bound, namely for `gd96b`, `can__144`, `dwt__162` and `impcol_b`.

But then with larger graphs, the number of variables grows very quickly and the linear programs take very long to solve. For that reason the d_{ij} formulation still provides a better lower bound for about half of the instances even when their code hits the time limit.

5.3 Comparisons

We also study the impact of individual classes of inequalities. Several subsets of the separators are used. See table 5.3 for the definition. The first setting, 1, only uses the triangle equation, both those occurring in the graph and those in the completion. This seems to be the most basic and by far most important class of inequalities. The next setting, 2, additionally uses the odd circle cut inequality separation. Based on 2 we create two more settings, 3 and 4, that add the rank inequalities for subgraphs in G and facets from the polytopes $P_{BTW}(K_4)$ and $P_{BTW}(K_5)$, respectively. Then there is setting 5 that builds on setting 3 but also separates rank inequalities on the completion. Finally, the last setting “all”

name	x_{ijk} with force		x_{ijk} without force		d_{ij} [3]		d_{ijk} [24]	
	lb	time	lb	time	lb	time	lb	time
gd95c	506	109.7	506	101.4	443	68.3	559.00	limit
gd96b	1389.85	limit	1404.63	limit	1281	493.5	1619.73	limit
gd96c	519	2368.3	519	4162.0	402	218.1	525.31	limit
gd96d	1554.44	limit	1602.89	limit	2021	1642.2	2494.67	limit
bcspr01	106	6.5	106	5.7	91	0.7	103.91	limit
bcspr02	161	16.0	161	14.9	144	1.8	172.93	limit
bcspr03	662	44641.5	662	52216.7	588	189.5	-	-
bcspr04	3004.98	limit	2948.30	limit	3700	limit	-	-
bcsstk01	1132	40852.8	1132	44838.3	972	38481.1	1151.62	limit
can__24	210	4.7	210	3.3	203	2.8	203.86	1080
can__61	1137	1371.9	1137	1125.1	1119	538.0	1137.00	limit
can__62	210	49.6	210	69.2	187	4.2	221.99	limit
can__73	965.00	limit	961.49	limit	971	2016.8	-	-
can__96	2039.33	limit	2020.31	limit	2105	8280.6	-	-
can__144	2776.44	limit	2841.97	limit	2304	1710.6	-	-
can__161	3042.42	limit	3361.71	limit	5657	limit	-	-
can__187	3064.09	limit	3143.20	limit	3827	limit	-	-
can__229	2196	limit	3857.33	limit	7461	limit	-	-
dwt__59	289	40.9	289	39.4	258	55.4	-	-
dwt__66	192	34.4	192	34.2	192	1.7	-	-
dwt__72	167	55.3	167	38.4	150	6.7	-	-
dwt__87	932	2542.6	932	2507.0	897	18430.7	-	-
dwt__162	2356.38	84362.2	2361.90	81662.8	2032	limit	-	-
dwt__209	3492.57	limit	3748.87	limit	5905	limit	-	-
dwt__221	3086.59	81124.8	3061.20	56702.7	3603	limit	-	-
dwt__245	2696.90	limit	2727.34	limit	3422	limit	-	-
bwm200	496	538.2	496	538.6	495	2409.1	-	-
fidap005	414	4.1	414	4.1	412	4.2	-	-
fidapm05	1003	1516.9	1003	365.6	998	805.2	-	-
lshp-265	3338.00	limit	3305.44	limit	5602	limit	-	-
nos4	1031	3990.0	1031	4671.37	976	59118.0	-	-
pde225	1883.80	limit	1884.75	limit	2539	70519.0	-	-
rdb200	2306.54	limit	2304.38	limit	3066	73463.6	-	-
saylr1	2010.90	limit	2069.90	limit	2676	limit	-	-
steam3	1416	164.3	1416	164.4	1406	limit	-	-
tub100	246	67.8	246	131.0	245	71.5	-	-
curtis54	454	69.6	454	101.5	-	-	511.91	limit
ibm32	485	1241.3	485	1434.9	-	-	462.36	30677
impcol_b	1825.30	limit	1999.67	limit	-	-	2357.81	limit
pores_1	383	29.9	383	20.6	-	-	351.02	15340
will57	335	56.0	335	50.3	-	-	352.00	limit

Table 5.2: Best lower bounds with a time limit of 86400 seconds. Provably optimal values are highlighted using a **bold font**. For `dwt__162` and `dwt__221`, the time limit could not be reached, because of insufficient memory.

	setting	triangle	odd cut	rank in G	small facets	rank in K_n
1	✓					
2	✓	✓				
3	✓	✓	✓			
4	✓	✓		✓		
5	✓	✓	✓		✓	
all	✓	✓	✓	✓	✓	

Table 5.3: Different subsets of separated inequalities.

	1		2		3	
name	lb	time	lb	time	lb	time
gd95c	506	281.3	506	98.4	506	97.0
gd96b	1163.63	limit	1144.62	limit	1403.95	limit
gd96c	519	4020.7	519	1639.1	519	1143.2
gd96d	1524.89	limit	1449.20	limit	1711.63	limit
	4		5		all	
name	lb	time	lb	time	lb	time
gd95c	506	111.2	506	100.2	506	87.8
gd96b	1163.30	limit	1379.45	limit	1401.43	limit
gd96c	519	2356.4	519	2979.1	519	1617.2
gd96d	1419.88	limit	1433.71	limit	1499.42	limit

Table 5.4: Comparison of different separators.

combines all these classes of inequalities together.

The lower bounds and runtimes for all these settings used on the four smallest **gd*** instances are shown in table 5.4. Again, we only use the four smaller **gd*** instances with a time limit of 21600 seconds. The most obvious fact is that the two smaller instances, **gd95c** and **gd96c** can be solved to optimality with all settings, although not always in the same time. For these two, setting 1 gives the worst results. However, for the two larger instances, **gd96b** and **gd96d**, the first setting is actually better than the second, with the added odd circle cut inequalities. Surprisingly, setting 3 gives the best results on the last two instances, even by a wide margin. This shows the high importance of rank inequalities for the lower bound. Adding some inequalities in settings 4 and 5 yields worse results than setting 3, for all instances! It seems like some bad choices are made, when adding separated cuts to the relaxation, slowing down the solving of the linear program. Combining the two in the setting “all” remedies this loss.

Chapter 6

x_{ijkl} Approach

In chapter 3, we presented a model for the linear arrangement using betweenness variables x_{ijk} . With these we could express the distance variables d_{ik} and inherit all valid inequalities on these. But we could still add more inequalities, this way improving the bound for the problem.

Let us now consider another iteration of this idea, with a similar type of variables x_{ijkl} , indexed by four vertices of the graph G . In theory, this would give us an even finer control on the arrangement. However, this model was computationally inefficient, most likely because of the huge number of variables.

6.1 Variables

We introduce a variable x_{ijkl} for every edge $il \in E$ and every pair of vertices $\{j, k\} \subset V$, not necessarily distinct from i and j . For a given arrangement p , $x_{ijkl} = 1$ if the following holds:

- j and k lie between il ,
- j and k are consecutive, i.e., $|p(j) - p(k)| = 1$.

Obviously, if j lies between il and j is consecutive to k then k also has to lie between il .

In addition, for every pair $\{j, k\} \subset V$ with $jk \notin E$ we have a variable x_{jjkk} that only considers whether j and k are consecutive in one arrangement. These do not have a contribution to the objective function.

The following relation connects these new variables to the betweenness variables:

$$2x_{ijl} = \sum_{k \in V \setminus \{j\}} x_{ijkl} \quad il \in E, j \in V \setminus \{i, l\}. \quad (6.1)$$

Note that the two terms x_{ijl}, x_{ijll} also occur in the sum in (6.1). If the vertex j lies between il , then $x_{ijl} = 1$. Also, j is consecutive to exactly two other vertices (possibly i and/or l) so exactly two of the summands on the right-hand side are 1. On the other hand, if $x_{ijl} = 0$, then none of the x_{ijkl} can be 1.

Let us just consider the variables of the type x_{iijj} (regardless of $ij \in E$). These variables have been used in formulations of the Traveling Salesman Problem (TSP) already and the arrangement is fully determined by their values alone [21].

But not all variables occur in the objective function

$$\text{lap}(G, p) = \sum_{il \in E} \sum_{j < k \in V} x_{ijkl}. \quad (6.2)$$

In this formulations there is no constant ter, like with the betweenness variables.

We can now look at some valid inequalities and formulate an integer linear program.

6.2 Constraints

The following two classes of constraints consider only the TSP style variables and ensure that an arrangement is formed (by a Hamiltonian path).

Neighbor Degree In the (implicit) Hamiltonian path, every vertex has one or two neighbors:

$$1 \leq \sum_{l \in V \setminus \{i\}} x_{iill} \leq 2 \quad i \in V \quad (6.3)$$

Subtour Elimination The Hamiltonian path has to be connected, so we need exactly $n - 1$ edges; but we also forbid the existence of shorter tours:

$$\sum_{\{i,l\} \subset V} x_{iill} = n - 1 \quad (6.4)$$

$$\sum_{\{i,l\} \subset S} x_{iill} \leq |S| - 1 \quad S \subsetneq V, 3 \leq |S| \leq n - 2 \quad (6.5)$$

But we still need some constraints, to make sure that the arrangement encoded by the TSP variables also yields the correct objective value.

Path Constraints Let us consider an edge $il \in E$. Then the section of the arrangement from $p(i)$ to $p(l)$ forms an $i - l$ path. That is, i and l each have exactly one consecutive vertex lying between il . All other vertices between il have exactly two consecutive vertices between il . Formally, we get

$$\sum_{j \in V} x_{ijl} = 1 \quad il \in E \quad (6.6)$$

$$\sum_{j \in V} x_{ijl} = 1 \quad il \in E \quad (6.7)$$

$$\sum_{k \in V \setminus \{i, l, j\}} x_{ijkl} \in \{0, 2\} \quad il \in E, j \in V \setminus \{i, l\} \quad (6.8)$$

The parity constraint can again be linearized, like (3.23) on page 34. Thus, instead of (6.8) we can add

$$\sum_{k \in V \setminus \{i, l, j\}} x_{ijkl} \leq 2 \quad il \in E, j \in V \setminus \{i, l\} \quad (6.9)$$

$$\sum_{k \in V \setminus \{i, l, j, h\}} x_{ijkl} \geq x_{ijhl} \quad il \in E, \{j, h\} \subset V \setminus \{i, l\} \quad (6.10)$$

The inequalities listed so far are already sufficient:

Proposition 6.2.1. *The constraints (6.3)-(6.10), together with the binary condition provide an integer programming formulation for the linear arrangement problem.*

Proof. Clearly, these constraints are valid for all possible arrangements p .

But we also have to check that every solution x really corresponds to an arrangement. Let us first consider only the TSP variables $x_{iil}, \{i, l\} \subset V$. Let P be a graph with $V(P) = V$ and $E(P) = \{il \mid x_{iil} = 1\}$. We know that P has to be connected because according to (6.4), P has exactly $n - 1$ edges. These would not fit in (6.5) if P had more than one connected component. We know that P is a tree because we have only $n - 1$ edges. Furthermore every vertex in P has degree at most 2 so P is a (Hamiltonian) path. Vertices that are connected in P are consecutive in our arrangement.

Now we have to make sure that the other variables x_{ijkl} have the correct value, contributing to the objective. Let $il \in E$. If i and l are consecutive then $x_{iil} = 1$ already gives the correct length of $|p(i) - p(l)|$. Else some other vertices have to lie between them. Suppose we have a partial arrangement (i, j_1, \dots, j_k, l) . Then according to (6.6) and (6.7) we know that for one vertex j_i we have $x_{ij_i l} = 1$, thus i and j_i are consecutive; analogously for l and some j_l . But with (6.8) we know that the vertices j_i and j_l must have another consecutive vertex between i

and l and so on, forming a path from i to l . Because of the minimized objective, this path will be the shortest possible, namely, of length $k + 1$. Of course, j_i and j_l could coincide, in which case the length $|p(i) - p(l)| = 2 = x_{iijl} + x_{ijll}$. So, every edge il gets the correct length. \square

We could find some more valid inequalities that are useful for the problem.

Narrow Edge If i and l are consecutive for some edge $il \in E$ then no other vertex can lie between them. Therefore,

$$x_{ijkl} \leq 1 - x_{iill} \quad il \in E, \{i, l\} \neq \{j, k\} \subset V. \quad (6.11)$$

Betweenness Similarly a pair of vertices $\{j, k\}$ has to be consecutive for it to be consecutive in between some edge, giving

$$x_{ijkl} \leq x_{jjkk} \quad il \in E, \{j, k\} \subset V. \quad (6.12)$$

Odd Cuts In analogy to section 3.6 on page 34, we can add inequalities like (3.24) that ensure an even number of edges in cuts by pairs of consecutive vertices $\{j, k\} \subset V$ of circles $C \subseteq E(G \setminus \{j, k\})$. We have

$$\sum_{il \in C \setminus J} x_{ijkl} \geq \sum_{il \in J} x_{ijkl} - (|J| - 1) \quad \text{for all odd } J \subseteq C, \{j, k\} \subset V. \quad (6.13)$$

6.3 Implementation

Since we already have a complete integer programming formulation, we can just enumerate all these inequalities and solve the problem with a general IP solver. We could only solve very small instances (with up to 6 vertices) this way.

Then, we implemented a branch-and-cut algorithm using ABACUS that was already mentioned in section 4.2 on page 46. The branching is done on variables, preferably of the TSP type x_{iill} , hoping to settle for a particular arrangement quickly. Also, fixing some variables x_{iill} to either 0 or 1 will at the same time fix many more variables, according to (6.11) and (6.12). Still, the branch-and-bound tree becomes very deep and we have a lot of variables (roughly n times as many as with betweenness variables).

Chapter 7

Outlook

In this thesis we discussed a novel approach to solving the Minimum Linear Arrangement Problem using betweenness variables. A branch-and-cut algorithm was implemented to conduct computational experiments with benchmarks from the literature. Especially for smaller graphs, the lower bound could be improved and the optimality of our solutions could be proven. We will now suggest some ideas for potential further research on the subject.

We already explained the separation on the completion of the graph, using (shortest) paths. In our case the artificial variables are only introduced implicitly because we always substitute the sum of variables in a path for the missing edge. But we could also use variables x_{ijk} for all 3-subsets $\{i, j, k\} \subset V$. This has two main advantages. First, we only need to separate every inequality once, instead of several times, each with a different path. Secondly, when using the new variables explicitly, we can combine them with many different paths, so that the currently shortest one is always binding. But, of course, we have to pay by adding still more variables and the high number of variables seems to be our formulation's biggest limitation. For the distance variables d_{ij} , [3] experimented with this dense formulation, but could only observe a generally higher runtime.

The rank inequalities have been shown to be of great importance for the improvement of the lower bound. However, so far only a small number of classes of graphs are used for their separation, namely bicliques (including stars and doublestars). The optimal values $\text{lap}(G)$ are known for many other classes of graphs G , and can be computed efficiently for even more classes, e.g., tree graphs (maybe using a minimal spanning tree for separation). All these could also be incorporated in the separation.

We also saw that all inequalities for the cut polytope can be used for our problem. Only odd cuts of circles were separated so far, so there is still a potential improvement, given the abundance of literature on the Max Cut Problem.

In our branch-and-cut algorithm, branching is always done on variables. We did some experiments with an alternative branching scheme that fixed two of the vertices at the extreme positions in the arrangement. Unfortunately, the number of subproblems $\binom{n}{2}$ is very large and usually, this fixing did not help improving the lower bound significantly. Still, some other idea might be appropriate.

Besides the betweenness variables, we also described a finer formulation of variables with 4 indices. Yet another formulation could use variables that represent the product of betweenness variables, i.e., $x_{ijkl} = x_{ijl}x_{ikl}$. But the experiments with medium sized graphs have shown that we probably can not increase the number of variables any more without being restricted to very small instances.

Chapter 8

Appendix

The source code of our branch-and-cut algorithm is attached on a CD-ROM. To build it, SCIP has to be installed first. The easiest way is to download and install the ZIB Optimization Suite [26]. Extract our source into the `examples` folder of the SCIP directory. Build the program with `make depend` and `make`. The executable can then be started with `bin/sciplap -f <problem.lap>`.

All problem instances used in chapter 5 are available in the `problems` folder, together with some constructed regular graphs.

We also provide the different parameter settings in the folder `settings`.

List of Figures

1.1	Example of an integer program.	10
1.2	Branching on one variable.	11
1.3	Applying Gomory Cutting Planes.	13
2.1	Example of a graph G with one possible arrangement p	15
3.1	One arrangement of C_4 with corresponding betweenness variables.	23
3.2	The 3-star H (solid edges) is a subgraph of K_4	25
3.3	Optimal arrangement of odd p -stars $K_{1,p}$	31
3.4	Intermediate vertex j , cutting the $i - k$ path.	33
3.5	Circle graph C_5 with triangle through path.	33
3.6	Even cut of a circle.	34
3.7	Graph \tilde{G} for $G = K_5$, $j = 5$	35

List of Tables

4.1	Impact of the triangle equations.	41
4.2	Impact of the upper bound heuristic.	44
4.3	Time consumption (in seconds).	45
4.4	Comparison of parameter settings with different emphasis.	46
5.1	Problem instances from the literature.	48
5.2	Best lower bounds with a time limit of 86400 seconds. Provably optimal values are highlighted using a bold font . For <code>dwt_162</code> and <code>dwt_221</code> , the time limit could not be reached, because of insufficient memory.	50
5.3	Different subsets of separated inequalities.	51
5.4	Comparison of different separators.	51

Bibliography

- [1] T. Achterberg, T. Berthold, T. Koch, and K. Wolter. Constraint integer programming: A new approach to integrate CP and MIP. *Lecture Notes in Computer Science*, 5015:6, 2008.
- [2] A.R.S. Amaral, A. Caprara, A.N. Letchford, and J.J. Salazar-Gonzalez. A New Lower Bound for the Minimum Linear Arrangement of a Graph. *Electronic Notes in Discrete Mathematics*, 30:87–92, 2008.
- [3] A. Caprara, A.N. Letchford, and J.J. Salazar-Gonzalez. Decorous lower bounds for minimum linear arrangement. To appear in *INFORMS Journal on Computing*, 2010.
- [4] P.P.S. Chen. The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems*, 1:9–36, 1976.
- [5] T. Christof. *Low-dimensional 0/1-polytopes and branch-and-cut in combinatorial optimization*. PhD thesis, Ruprecht-Karls-Universität Heidelberg, Shaker Verlag, Aachen, 1997.
- [6] F.R.K. Chung. Optimal linear arrangements of trees. *Computations & Mathematics with Applications*, 10(1):43–60, 1983.
- [7] F.R.K. Chung. Labelings of graphs. In *Selected topics in graph theory*, volume 3, chapter 7, pages 151–168. Academic Press, 1988.
- [8] J. Cohen, F. Fomin, P. Heggenes, D. Kratsch, and G. Kucherov. Optimal linear arrangement of interval graphs. *Lecture Notes in Computer Science*, 4162:267, 2006.
- [9] COIN-OR. Open Solver Interface. <https://projects.coin-or.org/0si>, 2009.
- [10] J. Díaz, J. Petit i Silvestre, and M. Serna. A survey of graph layout problems. *ACM Computing Surveys (CSUR)*, 34(3):313–356, 2002.
- [11] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

- [12] S. Even and Y. Shiloach. NP-completeness of several arrangement problems. *Department of Computer Science, Israel Institute of Technology, Haifa, Israel, Technical Report 43*, 1975.
- [13] M.R. Garey, D.S. Johnson, and L. Stockmeyer. Some simplified NP-complete problems. In *Proceedings of the sixth annual ACM symposium on Theory of computing*, page 63. ACM, 1974.
- [14] R.E. Gomory. An Algorithm for Integer Solutions to Linear Programs. In R.L. Graves and P. Wolfe, editors, *Recent Advances in Mathematical Programming*, pages 269–302. McGraw-Hill, New York, 1963.
- [15] L.H. Harper. Optimal assignments of numbers to vertices. *Journal of the Society for Industrial and Applied Mathematics*, pages 131–135, 1964.
- [16] J. Petit i Silvestre. *Layout Problems*. PhD thesis, Universitat Politècnica de Catalunya, 2001.
- [17] M. Jünger and S. Thienel. The ABACUS system for branch-and-cut-and-price algorithms in integer programming and combinatorial optimization. *Software Practice and Experience*, 30:1325–1349, 2000.
- [18] Y. Koren and D. Harel. A multi-scale algorithm for the linear arrangement problem. *Lecture Notes in Computer Science*, 2573:296–309, 2002.
- [19] S. Leipert. PQ-Trees, An Implementation as Template Class in C++. Technical Report 259, Zentrum für Angewandte Informatik Köln, Lehrstuhl Jünger, 1997.
- [20] W. Liu and A. Vannelli. Generating lower bounds for the linear arrangement problem. *Discrete applied mathematics*, 59(2):137–151, 1995.
- [21] G.L. Nemhauser and L.A. Wolsey. *Integer and combinatorial optimization*. Wiley New York, 1999.
- [22] M. Oswald. *Weighted Consecutive Ones Problems*. PhD thesis, Ruprecht-Karls-Universität Heidelberg, 2003.
- [23] I. Safro. The minimum linear arrangement problem on proper interval graphs. *Arxiv preprint cs.DM/0608008*, 2002.
- [24] H. Seitz. *Contributions to the Minimum Linear Arrangement Problem*. PhD thesis, Ruprecht-Karls-Universität Heidelberg, 2010.
- [25] A. Vannelli and G.S. Rowan. An eigenvector-based approach for multistack VLSI layout. In *Proceedings of the Midwest Symposium on Circuits and Systems*, volume 29, pages 435–439, 1986.
- [26] Zuse Institut Berlin. ZIB Optimization Suite (Version 1.2). <http://zibopt.zib.de>, 2009.

Acknowledgments

First, I would like to thank Prof. Dr. Gerhard Reinelt. It is through his lectures that I was introduced to the field of Combinatorial Optimization.

I am deeply grateful to Marcus Oswald, Hanna Seitz and Emiliano Traversi, who have been very helpful for the writing of this thesis. I thank them for all advice and explanations.

Special thanks go to Alexander Buchner and Michael Herting, who helped with proof-reading drafts of this thesis.

I thank all members of the Research Group Combinatorial Optimization, for providing useful hints and a friendly environment.

Above all, I would like to thank my family. Their support made possible my studies.

Erklärung

Hiermit versichere ich, dass ich meine Arbeit selbstständig unter Anleitung verfasst habe, dass ich keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, und dass ich alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entlehnt sind, durch die Angabe der Quellen als Entlehnung kenntlich gemacht habe.

Ort, Datum:

Heidelberg, 10. März 2010

Unterschrift: