Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

CHUL-YOUNG BYUN

Lower Bounds for Large-Scale Set Partitioning Problems

This research has been supported with funds of the German Ministry for Research and Education, Grant No. 03-GR7ZI1.
Responsibility for the contents of this article is with the author.

ZIB-Report 01-06 (January 2001)

Lower Bounds for Large-Scale Set Partitioning Problems

Diplomarbeit
bei Prof. Dr. M. Grötschel

vorgelegt von Chul-Young Byun
am Fachbereich Mathematik der
Technischen Universität Berlin

Berlin, 08. Januar 2001

Die selbständige und eigenhändige Anfertigung versichere ich an Eidesstatt.

Chul-Young Byun
Berlin, 08. Januar 2001

dedicated to my family

Acknowledgments

The possibility to write this thesis at the Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB) was initiated by *Mr. Thieme*. At the interview it turned out that my skills did not really fit to the job offer, but even though he tried to find a field, which could need a student like me. So I got a further interview with *Ralf Borndörfer* and at the end I started in a field, for which I did not apply. I have never regretted it.

Without the help of the following people this thesis would not have been accepted:

First of all, I wish to thank *Ralf Borndörfer* for reading my thesis again and again in his spare time. The discussions with him afterwards improved my mathematical understanding a lot.
I am thankful for the programming advices of *Andreas Löbel*, who always listened to my problems. Without his motivating help my program would still not properly run.
Special thanks to both of them, because they believed in me at any time.
I thank *Steffen Weider*, who showed me the gaps in my proposed proofs and made constructive suggestions for the content and the layout of this work.
Especially because he is not a mathematician (he is a politician and sociologist) I want to thank *James Sater* for correcting my thesis stylistically.
Furthermore, I wish to thank *Mr. Grötschel* for the opportunity of working at such an inspiring institution. Finally, I would like to thank *Günter Ziegler* for his outstanding lectures, which formed my basic understanding of combinatorial optimization.

Chul-Young Byun
Berlin, January 2001

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Set partitioning is one of the fundamental models in the field of combinatorial optimization. A partial list of applications includes truck deliveries (see Balinski and Quandt [1964], Cullen, Jarvis, and Ratliff [1981]), tanker routing (see Fisher and Rosenwein [1985]), vehicle scheduling (see Borndörfer [1998]), airline crew scheduling, and bus driver scheduling (see Desrosiers, Dumas, Solomon, and Soumis [1993]).

First, we will give a combinatorial description of this problem. Let M be a nonempty and finite set. Let F be a family of acceptable or feasible subsets of M. Associated with each family j of F is a cost c_j. The problem is to find a collection of members of F, which is a partition of M, where the cost sum of these members is minimal.

An integer programming formulation of this problem reads

$$\begin{aligned} \text{(SPP)} \quad \min \; & c^t x \\ & Ax = \mathbf{1} \\ & x \leq \mathbf{1} \\ & x \in \mathbb{N}^n, \end{aligned} \tag{1.1}$$

where x is a solution vector, $\mathbf{0} \leq c \in \mathbb{R}^n$ a cost vector, and $A \in [0,1]^{m \times n}$ a zero-one matrix. M corresponds to the m rows of matrix A and the subsets of M correspond to the columns of this matrix in such a way that $a_{ij} = 1$ if $i \in j$ and $a_{ij} = 0$ if $i \notin j$. The stipulation that each member of M has to be covered once corresponds to the constraint set of (1.1), which defines F.

It is well-known that SPP is $\mathcal{NP}$-hard (see Garey and Johnson [1979]), but we do not discuss further **complexity** issues here and refer the reader to Emden-Weinert, Hougardy, Kreuter, Proemel, and Steger [1996] for more detailed information.

The most widespread application of the set partitioning model seems to be the airline crew scheduling problem. In order to describe this application we need to

explain two technical terms. A **flight leg** defines a flight from a city A to city B at a time t and a **rotation** defines a sequence of flight legs with the same initial and terminal point. Note that rotations must satisfy many regulations in order to be acceptable. Now we can formulate the airline crew scheduling problem, namely, partitioning flight legs by acceptable rotations during a planning period at minimal cost. In this case M corresponds to the set of flight legs, while each subset of M stands for a possible rotation. To set up the problem, we generate the set of all acceptable rotations with their respective costs. This produces a matrix A and a cost vector c, after which we attempt to solve the set partitioning model. If the attempt is successful, the solution yields a collection of acceptable rotations, whose cost sum is minimized and which cover each flight leg exactly once.

Duty scheduling in public transport can be formulated similarly to the airline crew scheduling. Given a bus schedule, the duty scheduling of bus drivers consists of covering tasks by duties, where a duty forms a set of such tasks that corresponds to a legal day of work for an individual driver. M corresponds to the set of tasks and F is the family of all feasible duties. Again we get a matrix A and a cost vector c of a set partitioning problem. If this problem is solvable, then a collection of acceptable duties exists, whose cost sum is minimal and where each task is included in exactly one duty of the collection.

Some of the largest SPP instances, which can be solved by state-of-the-art algorithms, are vehicle scheduling problems reported in Borndörfer [1998], where the corresponding problem matrices A have up to 1771 rows and 146715 columns. Further large instances, which result from an airline crew scheduling application, have been tested by Bixby, Gregory, Lustig, Marsten, and Shanno [1992]. The corresponding problem matrices consist of up to 837 rows and 13 million columns.

The instances of our test bed, which originate from duty scheduling applications of some European public transport companies, are made of up to 3570 rows and 82422 columns. I.e., we double the state-of-the-art number of rows. We focus on determining quickly good lower bounds. This is of interest in a column generation context because this technique needs to calculate among other things a large number of such lower bounds in order to proceed. For more detailed information on column generation the reader is referred to Desrosiers, Dumas, Solomon, and Soumis [1993].

This work is structured as follows. We start in Chapter 2 with a literature survey, which introduces background material; then we discuss different approaches and algorithms for determining lower bounds for SPP. In Chapter 3 we will develop new algorithms for this task, and Chapter 4 describes the implementation details and the computational results for all algorithms. Finally, we will summarize in Chapter 5 the findings and give an outlook.

There is no explanation of **notation** or basic concepts of combinatorial optimization. Instead we have tried to resort to standards and in particular to the book Grötschel, Lovász and Schrijver, Alexander (1988), *Geometric Algorithms and Combinatorial Optimization*, Springer Verlag, Berlin.

Chapter 2

Literature Survey

In the literature there exist two main approaches for determining lower bounds for SPP. They are called **combinatorial bounds** and **linear programming bounds**. In Figure 2.1 we have listed a classification, which we are going to discuss in this chapter.

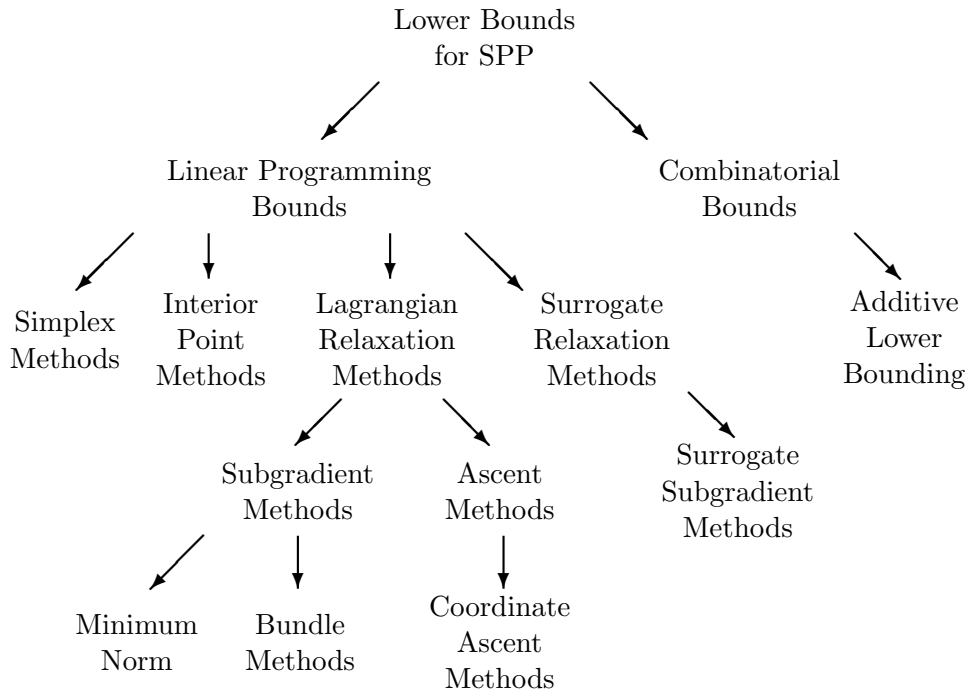

Figure 2.1: Lower bounds for SPP

2.1 Combinatorial Bounds

Many network problems can be modeled as SPP or have SPP relaxations. In this case a lower bound for SPP would be a lower bound for the network problem

at the same time. I.e., among other things set partitioning models can provide lower bounds for certain network problems. The converse implication does, of course, not hold in general. We will mention one general technique that can often be applied in such situations though this technique does not make any essential contribution to lower bounds for SPP in general. We added this technique to the list for the sake of completeness.

2.1.1 Additive Lower Bounding

Let us consider the problem

$$\begin{aligned} (\mathrm{P}) \quad \min \; & c^t x \\ & x \in F \subseteq \mathbb{N}^n, \end{aligned} \tag{2.1}$$

where x and $c \leq \mathbf{0}$ are a solution vector and a cost vector, respectively. For the sake of simplicity, we assume that all the problems considered in this section are feasible and bounded, i.e., $F \neq \emptyset$ and F is finite.

In many situations, several different lower bounding procedures for P are available; each exploits a different substructure of the problem. Clearly a lower bound for P could be obtained by applying each single bounding procedure and taking the maximum of the values computed. In this way, however, only one substructure is fully exploited, while all the others are lost completely. The additive approach tries to partially overcome this disadvantage.

Let $\mathcal{L}^{(1)}, \ldots, \mathcal{L}^{(r)}$ be the lower bound procedures available for problem P. Also suppose for $k = 1, \ldots, r$ that procedure $\mathcal{L}^{(k)}(\hat{c})$—when applied to the instance of problem P with cost vector $\hat{c}$—returns a lower bound value $\delta^{(k)}$ as well as a **residual cost vector** $c^{(k)} \in \mathbb{R}^n$ such that $c^{(k)} \geq \mathbf{0}$ and

$$\delta^{(k)} + c^{(k)\,t} x \leq \hat{c}^{\,t} x \quad \forall x \in F. \tag{2.2}$$

The additive approach generates a sequence of instances of problem P, each obtained from the previous one by considering the corresponding residual costs and applying a different lower bounding procedure.

Initially, procedure $\mathcal{L}^{(1)}(c)$ is applied to obtain a value $\delta^{(1)}$ and the corresponding residual cost vector $c^{(1)}$. We consider the problem

$$(\mathrm{P}^{(1)}) \quad \delta^{(1)} + \min_{x \in F} c^{(1)\,t} x. \tag{2.3}$$

It can be seen easily that $\mathrm{P}^{(1)}$ is a relaxation of P, since the two problems have the same feasible set F and their objective functions are such that

$$\delta^{(1)} + c^{(1)\,t} x \leq c^t x \quad \forall x \in F. \tag{2.4}$$

Note that in problem $\mathrm{P}^{(1)}$ a **residual instance**

$$(\mathrm{R}^{(1)}) \quad \min_{x \in F} c^{(1)\,t} x \tag{2.5}$$

of P has been introduced.

Due to our hypothesis, the objective value of $R^{(1)}$ is non-negative, so any lower bound on its value can be added to $\delta^{(1)}$ to obtain a tighter bound for P. To this end, we apply procedure $\mathcal{L}^{(2)}(c^{(1)})$, which yields a value $\delta^{(2)}$ and the corresponding residual cost $c^{(2)}$. Now, a new relaxation

$$(P^{(2)}) \quad \delta^{(1)} + \delta^{(2)} + \min_{x \in F} c^{(2)\,t} x \tag{2.6}$$

of problem P is available. In fact, it can be verified easily that

$$\delta^{(1)} + \delta^{(2)} + c^{(2)\,t} x \leq \delta^{(1)} + c^{(1)\,t} x \leq c^t x \quad \forall x \in F. \tag{2.7}$$

As before, the current lower bound for P, given by $\delta^{(1)} + \delta^{(2)}$, can be further strengthened by applying procedure $\mathcal{L}^{(3)}(c^{(2)})$. The procedure can be iterated by applying the remaining bounding procedures in sequence. In this way, it is possible to take into account the different substructures of problem P exploited by the available bounding procedures. Consequently, the gap between the objective value of P and $\sum_{k=1}^{r} \delta(k) + \min_{x \in F} c^{(k)\,t} x$ of the current relaxation can decrease after each iteration. Whenever one of the residual problems has the form of an SPP, bounding procedures for set partitioning can contribute in an additive lower bounding procedure.

For more examples and detailed information the reader is referred to Fischetti and Toth [1992], Carpaneto, Fischetti, and Toth [1987], and Carpaneto, Dell'Amico, Fischetti, and Toth [1989], who have applied the additive approach to the symmetric and asymmetric traveling salesman problem and to the multiple depot vehicle scheduling problem.

2.2 Linear Programming Bounds

Another approach for determining lower bounds for SPP is called **linear programming bounds**. The idea of this method is to solve the **linear programming relaxation** of a given IP, which has the form

$$\begin{aligned} (IP) \quad \min \; & c^t x \\ & Ax = b \\ & Bx \leq d \\ & x \in \mathbb{N}^n, \end{aligned} \tag{2.8}$$

where $c \in \mathbb{R}^n$ and $A \in \mathbb{R}^{m \times n}$, and $B \in \mathbb{R}^{m_o \times n}$. The relaxation can be obtained by omitting the integrality constraint, i.e. the linear programming relaxation of IP has the form

$$\begin{aligned} (LP) \quad \min \; & c^t x \\ & Ax = b \\ & Bx \leq d \\ & x \geq \mathbf{0}. \end{aligned} \tag{2.9}$$

IP has the following relationship to LP:

$$\begin{array}{c} \{x|x\in\mathbb{N}^n\} \\ \subseteq \{x|x\geq\mathbf{0}\} \end{array}$$

$$\min_{\substack{Ax=b \\ Bx\leq d \\ x\in\mathbb{N}^n}} c^t x \qquad \geq \qquad \min_{\substack{Ax=b \\ Bx\leq d \\ x\geq\mathbf{0}}} c^t x. \tag{2.10}$$

This technique can be also applied to SPP. By this we get the following formulation for the **linear programming relaxation** of SPP:

$$\begin{aligned} (\mathrm{SPP}^{\mathrm{LP}}) \quad \min\ & c^t x \\ & Ax = \mathbf{1} \\ & x \leq \mathbf{1} \\ & x \geq \mathbf{0}. \end{aligned} \tag{2.11}$$

As for the general case there exists the following relationship between SPP and its linear programming relaxation $\mathrm{SPP}^{\mathrm{LP}}$:

$$\begin{array}{c} \{x|x\in\mathbb{N}^n\} \\ \subseteq \{x|x\geq\mathbf{0}\} \end{array}$$

$$\min_{\substack{Ax=\mathbf{1} \\ x\leq\mathbf{1} \\ x\in\mathbb{N}^n}} c^t x \qquad \geq \qquad \min_{\substack{Ax=\mathbf{1} \\ x\leq\mathbf{1} \\ x\geq\mathbf{0}}} c^t x. \tag{2.12}$$

We assume in the remainder of this work that the matrix A contains no zero columns and SPP is feasible. Under this assumption we can make the following observations.

Remark 2.1. $\mathrm{SPP}^{\mathrm{LP}}$ is feasible because of the feasibility of SPP.

Remark 2.2. $\mathrm{SPP}^{\mathrm{LP}}$ is bounded because of the constraints $\mathbf{0} \leq x \leq \mathbf{1}$.

Remark 2.3. The constraint $x \leq \mathbf{1}$ in $\mathrm{SPP}^{\mathrm{LP}}$ is redundant because the zero-one matrix A does not contain any zero columns, but we will maintain this constraint for simplicity reasons.

There exist at least three approaches, which could determine the optimal objective function value of LP and $\mathrm{SPP}^{\mathrm{LP}}$: **simplex**, **interior point** and **Lagrangian relaxation methods**.

2.2.1 Simplex Methods

The simplex method, due to Danzig [1951], is a fundamental tool in mathematical programming. It is usually described for linear programs in the form

$$\begin{aligned} \min\ & c^t x \\ & Hx = h \\ & x \geq \mathbf{0}. \end{aligned} \tag{2.13}$$

In the case of LP the constraint $Bx \leq d$ can be transformed to an equation by introducing slack variables $x' \geq \mathbf{0}$, i.e., $Bx \leq d$ can be transformed to $Bx + x' = d$, $x' \geq \mathbf{0}$. Therefore, a reformulation of LP could be

$$\begin{array}{lll} (\mathrm{LP}') & \min & c^t x \\ & & Ax \qquad = b \\ & & Bx + Ix' = d \\ & & x,\ x' \geq \mathbf{0}, \end{array} \tag{2.14}$$

where I is an identity matrix. Note that LP$'$ describes the same optimization problem as LP and therefore fits with the formulation needed by the simplex method in order to be solved.

Geometrically, this method proceeds monotonely with respect to the objective function along vertices of the solution polyhedron. Its complexity is exponential (see Schrijver [1989]). A more detailed description of this method is outlined in Schrijver [1989] and Graham, Grötschel, and Lovász [1995].

Here are three computational properties of the simplex method that are important for our purposes:

- Strong degeneracy can slow down the running time of the simplex method.
- For the performance of this method the number of rows are more important than the number of columns.
- SPPs from real-world duty scheduling applications with more than 1000 rows are computationally difficult (see Borndörfer [1998], Löbel [1998], and Marsten and Shepardson [1981])

A well-known implementation of the simplex method is part of **CPLEX** (see Bixby [1992]), which has been used by Bixby, Gregory, Lustig, Marsten, and Shanno [1992] for solving large-scale linear programming relaxations of airline set partitioning instances, which we have already mentioned in the introduction. From this work the exact sizes and the computation time of these instances can be obtained. For our purpose of determining good lower bounds for $\mathrm{SPP}^{\mathrm{LP}}$ instances we will use the dual simplex method, which is also implemented in CPLEX. A description of this method can be obtained from Schrijver [1989].

The dual simplex method implemented in CPLEX is **dualopt**, which is an exact method. It provides optimal lower bounds for $\mathrm{SPP}^{\mathrm{LP}}$, but usually it needs too much computation time (see Table 4.13 and Table 4.15). Our aim in this work is to develop faster alternatives.

2.2.2 Interior Point Methods

Linear programs can also be solved by **interior point methods**. The idea of these methods is as follows. Choose an inner point of a solution polyhedron,

assign to this point a vector, which is increasing with respect to the objective function of the linear program, and move along this vector to a new point. Construct a sequence of interior points iteratively, which converges to an optimal solution.

In contrast to the simplex method interior point methods make no use of the vertices of the solution polyhedron, which allows to devise methods of polynomial complexity (see Karmarkar [1984]). Another crucial difference to the simplex method is that the simplex method determines an optimal vertex of the solution polyhedron, whereas interior point methods determine an inner element of the optimal face.

A well-known class of interior point methods are **barrier function methods**. These methods treat inequality constraints by creating a **barrier function**, which is a combination of the original objective function and a weighted sum of functions with a positive singularity at the constraint boundary. I.e., the feasible but non-optimal points at constraint boundaries are associated with higher costs, which increase the nearer these points get to the boundaries. Many barrier functions have been proposed like for example the **logarithmic barrier function**, first suggested by Frisch [1955]. As the weight assigned to the singularities approaches zero, the minimum of the barrier function approaches the minimum of the original constrained problem. For a more detailed discussion of barrier function methods see Fiacco [1979] and for a brief overview see Gill, Murray, and Wright [1981].

The basic idea of another class of interior point methods called **penalty methods** is to eliminate some or all of the constraints and add to the cost function a penalty term that prescribes a high cost to infeasible points. Associated with these methods is a penalty parameter that determines the severity of the penalty and, as a consequence, the extent to which the resulting unconstrained problem approximates the original constrained problem. As the penalty parameter takes higher values, the approximation becomes increasingly accurate.

Degeneracy is not a practical problem, but as for the simplex method SPPs from real-world applications with more than 1000 rows are typically hard to solve. An implementation of an interior point method by Lustig, Marsten, and Shanno [1990] called **OB1** was also used by Bixby, Gregory, Lustig, Marsten, and Shanno [1992] for testing the same large-scale airline linear programs mentioned above. Beside the dual simplex method we will use the barrier function method **baropt** implemented in **CPLEX** as a further reference.

2.2.3 Lagrangian Relaxation Methods

In contrast to the simplex method, which solves linear programs directly, the **Lagrangian relaxation method** creates a sequence of relaxations from the

original program, whose optima converge to the optimum of the original problem. This sequence can be used to construct a good approximate solution quickly. For a more detailed description of Lagrangian relaxation the reader is referred to Nemhauser and Wolsey [1988] and Schrijver [1989]. In the following we will show how to create such a sequence of programs out of an LP.

We consider LP as defined in (2.9)

$$\begin{array}{lll} \min & c^t x & \\ & Ax = b & \text{(complicating constraints)} \\ & Bx \leq d & \text{(nice constraints)} \\ & x \geq \mathbf{0}. & \end{array} \tag{2.15}$$

Suppose that $Bx \leq d$ is a set of 'nice constraints', say those of a network problem. We transfer the complicating constraints to the objective function and get

$$\begin{array}{rcll} L(y) & := & \min & c^t x + y^t(b - Ax) \\ & & & Bx \leq d \\ & & & x \geq \mathbf{0}, \end{array} \tag{2.16}$$

where $y \in \mathbb{R}^m$. Function L is called **Lagrangian function** of LP and the entries of y are called **Lagrangian multipliers**.

Note that L can be rewritten as follows

$$L(y) = \min_{\substack{Bx \leq d \\ x \geq \mathbf{0}}} c^t x + y^t(b - Ax) = \min_{\substack{Bx \leq d \\ x \geq \mathbf{0}}} (c^t - y^t A)x + b^t y = \min_{\substack{Bx \leq d \\ x \geq \mathbf{0}}} \overline{c}^{\,t} x + b^t y, \tag{2.17}$$

where $\overline{c}^{\,t} := (c^t - y^t A)$ defines the **reduced cost vector**. The following relationship holds for LP and L.

Lemma 2.1. *Let z be the optimal objective value of LP*

$$\begin{array}{rcll} z & := & \min & c^t x \\ & & & \mathrm{A}x = b \\ & & & \mathrm{B}x \leq d \\ & & & x \geq \mathbf{0} \end{array} \tag{2.18}$$

and consider L: $\mathbb{R}^m \mapsto \mathbb{R}$

$$\begin{array}{rcll} L(y) & = & \min & c^t x + y^t(b - \mathrm{A}x) \\ & & & \mathrm{B}x \leq d \\ & & & x \geq \mathbf{0}. \end{array} \tag{2.19}$$

Then it holds

$$z \geq L(y) \qquad \forall y \in \mathbb{R}^m. \tag{2.20}$$

Proof. Let $y \in \mathbb{R}^m$ be fixed. It holds

$$z = \min_{\substack{Ax=b \\ Bx \leq d \\ x \geq \mathbf{0}}} c^t x \overset{(a)}{=} \min_{\substack{Ax=b \\ Bx \leq d \\ x \geq \mathbf{0}}} c^t x + y^t(b - Ax) \overset{(b)}{\geq} \min_{\substack{Bx \leq d \\ x \geq \mathbf{0}}} c^t x + y^t(b - Ax) = L(y). \quad (2.21)$$

Equality (a) holds because $(b - Ax)$ equals zero for all feasible x of LP and inequality (b) is valid since

$$\{x | Ax = b \ \wedge \ Bx \leq d \ \wedge \ x \geq \mathbf{0}\} \subseteq \{x | Bx \leq d \ \wedge \ x \geq \mathbf{0}\}.$$

□

By this lemma we know that for each y the Lagrangian function yields a minimization problem with respect to x, whose corresponding objective value implies a lower bound for LP. In order to obtain the best bound we have to maximize L with respect to y. By this we get the problem

$$\max_y \ L(y) = \max_y \ \min_{\substack{Bx \leq d \\ x \geq \mathbf{0}}} \ c^t x + y^t(b - Ax), \quad (2.22)$$

which is called the **Lagrangian relaxation** of LP. Note that under certain assumptions the Lagrangian function is bounded from above, concave, and piecewise linear. We show this in the next theorem.

Theorem 2.1. *Let* $\{x | \mathrm{B}x \leq d \wedge x \geq \mathbf{0}\}$ *be non-empty and finite. The Lagrangian function*

$$L(y) = \min_{\substack{\mathrm{B}x \leq d \\ x \geq \mathbf{0}}} \ c^t x + y^t(b - \mathrm{A}x)$$

is bounded from above, concave, and piecewise linear.

Proof. $\{x | Bx \leq d \wedge x \geq \mathbf{0}\}$ forms the solution polyhedron, which is by assumption non-empty and bounded. From the theory of linear optimization it is well-known that the optimum of $L(y)$ is attained at a vertex of this polyhedron and that the number of vertices is finite. Let Q be the set of these vertices. For each $x \in Q$ we can define an affine map g_x in y by

$$g_x(y) := c^t x + y^t(b - Ax). \quad (2.23)$$

Due to the fact that Q is finite L can be described as a minimum of a finite number of affine maps, i.e.,

$$L(y) = \min_{x \in Q} g_x(y). \quad (2.24)$$

1. L is concave:

We have to show $L(\alpha\, y + (1-\alpha)\, \tilde{y}) \geq \alpha\, L(y) + (1-\alpha) L(\tilde{y})$ for all $\alpha \in [0,1]$ and $y, \tilde{y} \in \mathbb{R}^m$:

$$\begin{aligned} L(\alpha\, y + (1-\alpha)\, \tilde{y}) &= \min_{x \in Q} g_x(\, \alpha\, y + (1-\alpha)\, \tilde{y}\,) \\ &= \min_{x \in Q} g_x(\alpha\, y) + g_x(\, (1-\alpha)\, \tilde{y}\,) \\ &\geq \alpha \min_{x \in Q} g_x(y) + (1-\alpha) \min_{x \in Q} g_x(\tilde{y}) \\ &= \alpha\, L(y) + (1-\alpha) L(\tilde{y}). \end{aligned} \tag{2.25}$$

2. L is piecewise linear:

   Let $Y_x := \{y \in \mathbb{R}^m | L(y) = g_x(y)\}$ for all $x \in Q$. Note that $\bigcup_{x \in Q} Y_x = \mathbb{R}^m$. If we were able to show that Y_x is a convex set then L would be linear over Y_x, i.e., L would be piecewise linear over $\mathbb{R}^m$. It holds: Y_x is convex $\Leftrightarrow$ $(\alpha\, y + (1-\alpha)\, \tilde{y}) \in Y_x$ for all $\alpha \in [0,1]$ and $y, \tilde{y} \in Y_x$. We will show that the right hand side of this statement holds. Let $y, \tilde{y} \in Y_x$:

$$\begin{aligned} L(\, \alpha\, y + (1-\alpha)\, \tilde{y}\,) &\geq \alpha\, L(y) + (1-\alpha)\, L(\tilde{y}) \\ &= \alpha\, g_x(y) + (1-\alpha)\, g_x(\tilde{y}) \\ &= g_x(\, \alpha\, y + (1-\alpha)\, \tilde{y}\,). \end{aligned} \tag{2.26}$$

3. L is bounded from above:

   We have already shown this in Lemma 2.1.

□

Remark 2.4. The theory of Lagrangian relaxation also holds for IP. Assume that for IP as defined in (2.8) has 'nice and complicating constraints', i.e.,

$$\begin{aligned} \min\ & c^t x \\ & Ax = b \qquad \text{(complicating constraints)} \\ & Bx \leq d \qquad \text{(nice constraints)} \\ & x \in \mathbb{N}^n, \end{aligned} \tag{2.27}$$

then the corresponding Lagrangian relaxation reads the form

$$\begin{aligned} \hat{L}(y) := \min\ & c^t x + y^t(b - Ax) \\ & Bx \leq d \\ & x \in \mathbb{N}^n. \end{aligned} \tag{2.28}$$

Remark 2.5. L and $\hat{L}$ are in general not differentiable.

Lagrangian Relaxation of SPP$^{\mathbf{LP}}$ In the following part we will apply the idea of the Lagrangian relaxation method to set partitioning problems. Consider therefore SPP

$$\begin{aligned} \min\ & c^t x \\ & Ax = \mathbf{1} \\ & x \leq \mathbf{1} \\ & x \in \mathbb{N}^n \end{aligned}$$

as defined in (1.1) and its linear programming relaxation SPP$^{\mathrm{LP}}$

$$\begin{aligned} \min\ & c^t x \\ & Ax = \mathbf{1} \\ & x \leq \mathbf{1} \\ & x \geq \mathbf{0} \end{aligned} \tag{2.29}$$

as defined in (2.11). Note that SPP in (1.1) is a specialization of IP in (2.8). The constraint $Ax = \mathbf{1}$ is considered to be complicating, therefore we relax both problems with respect to this constraint. By this we get a set of integer programs

$$\begin{aligned} \hat{L}(y) &= \min_{x \in \{0,1\}^n} c^t x + y^t(\mathbf{1} - Ax) \quad \forall y \in \mathbb{R}^m \\ &= \min_{x \in \{0,1\}^n} (c^t - y^t A)x + y^t \mathbf{1} \quad \forall y \in \mathbb{R}^m \end{aligned} \tag{2.30}$$

and a set of linear programs

$$\begin{aligned} L(y) &= \min_{\mathbf{0} \leq x \leq \mathbf{1}} c^t x + y^t(\mathbf{1} - Ax) \quad \forall y \in \mathbb{R}^m \\ &= \min_{\mathbf{0} \leq x \leq \mathbf{1}} (c^t - y^t A)x + y^t \mathbf{1} \quad \forall y \in \mathbb{R}^m, \end{aligned} \tag{2.31}$$

i.e., the Lagrangian relaxation of SPP is $\max_y \hat{L}(y)$ and for SPP$^{\mathrm{LP}}$ $\max_y L(y)$, where for each y an optimal solution $x^{(y)}$ of both Lagrangian functions can be easily obtained using the reduced cost $\overline{c}^{\,t} = (c^t - y^t A)$ as follows:

$$x_j^{(y)} := \begin{cases} 0 & \text{if } \overline{c}_j > 0 \\ 0 \text{ or } 1 & \text{if } \overline{c}_j = 0 \\ 1 & \text{if } \overline{c}_j < 0. \end{cases} \quad \forall j \tag{2.32}$$

SPP$^{\mathrm{LP}}$, $\max_y L(y)$, and $\max_y \hat{L}(y)$ have the following relationship.

Theorem 2.2.

$$\min_{\substack{Ax=\mathbf{1} \\ x \leq \mathbf{1} \\ x \geq \mathbf{0}}} c^t x \quad = \quad \max_y L(y) \quad = \quad \max_y \hat{L}(y) \tag{2.33}$$

Proof. The first equation uses the duality theorem of linear programming, which is for example stated in Löbel [1998].

$$\begin{aligned}
\max_y L(y) &= \max_y \left[y^t\mathbf{1} + \left\{ \min_{\mathbf{0}\le x\le\mathbf{1}} (c^t - y^t A)x \right\} \right] \\
&\overset{\text{duality th}^{\text{m}}}{=} \max_y \left[y^t\mathbf{1} + \left\{ \max_{\substack{v^t\le c^t - y^t A \\ v\le\mathbf{0}}} v^t\mathbf{1} \right\} \right] \\
&\overset{z:=-v}{=} \max_{\substack{y^tA - z^t \le c^t \\ z\ge\mathbf{0}}} y^t\mathbf{1} - z^t\mathbf{1} \qquad (2.34) \\
&\overset{\text{duality th}^{\text{m}}}{=} \min_{\substack{Ax=\mathbf{1} \\ -x\ge-\mathbf{1} \\ x\ge\mathbf{0}}} c^t x \\
&= \min_{\substack{Ax=\mathbf{1} \\ x\le\mathbf{1} \\ x\ge\mathbf{0}}} c^t x
\end{aligned}$$

The second equation holds since for each y

$$\begin{aligned}
\hat{L}(y) &= \min_{x\in\{0,1\}^n} (c^t - y^t A)x + y^t\mathbf{1} \\
&\overset{\substack{\text{conv}(\{0,1\}^n) \\ =\{x|\mathbf{0}\le x\le\mathbf{1}\}}}{=} \\
&= \min_{\mathbf{0}\le x\le\mathbf{1}} (c^t - y^t A)x + y^t\mathbf{1} \qquad (2.35) \\
&= L(y).
\end{aligned}$$

□

This theorem and Lemma 2.1 imply that the three optimization problems SPP^{LP}, $\max_y L(y)$, and $\max_y \hat{L}(y)$ yield lower bounds for SPP of equal quality. Due to the fact that L and $\hat{L}$ are describing the same optimization problem (see proof of Theorem 2.2) we will consider just one of them, say L. Consequently, we have two optimization problems of different types, namely, SPP^{LP} and $\max_y L(y)$, which yield the same optimal objective value. We solve SPP^{LP} by the dual simplex method and an interior point method of CPLEX as state-of-the-art references.

In the case of $\max_y L(y)$ we wish to maximize a bounded, concave, and piecewise linear function, which is not differentiable. Therefore, we have to fall back on general non-differential optimization methods like **subgradient methods** or **ascent methods**. In the following sections we will discuss both approaches.

2.2.3.1 Subgradient Bundle Method–SBM

The **subgradient method** is a generalization of the well-known **Gauss algorithm** for convex and differentiable optimization (see Bertsekas [1995]) to a

more general non-differentiable setting. Like the Gauss method, the subgradient method approaches the global optimum in a sequence of line searches along appropriate descent directions. The main difference is the choice of these directions. The gradient, which is used by the Gauss method, does in general not exist in a non-differential setting, i.e., this method can not go further, but the subgradient method can. It resorts to the analogous concept for convex optimization and descends in the direction of a so-called subgradient, which exist in a convex (concave) setting. Surveys on subgradient optimization can be found in Bertsekas [1995]. We will describe in this section the idea of this method for the concave case, which means that we aim to find ascent directions instead of descending ones.

Definition 2.1. Let $f : \mathbb{R}^m \mapsto \mathbb{R}$ be some concave function. A vector $g \in \mathbb{R}^m$ defines a **subgradient** of f at $y_o \in \mathbb{R}^m$ if

$$f(y) \leq g^t(y - y_o) + f(y_o) \quad \forall y \in \mathbb{R}^m. \tag{2.36}$$

The set

$$\delta f(y_o) := \{g \in \mathbb{R}^m | g \text{ is a subgradient of } f \text{ at } y_o\} \tag{2.37}$$

is called **subdifferential** of f at y_o.

In Figure 2.2 and 2.3 we illustrate these concepts. In the first figure f is a

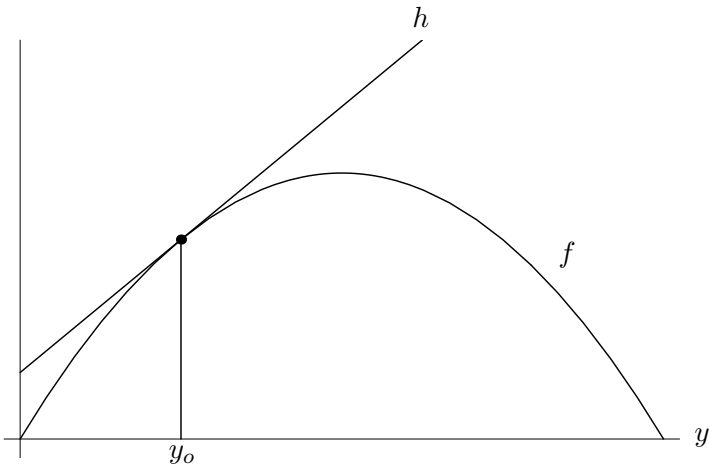

Figure 2.2: A differentiable function

concave and differentiable function in y, whereas in the second figure f is concave but non-differentiable. In the beginning of this section we have mentioned that the subgradient method is a generalization of the gradient method. In the differentiable case in Figure 2.2 we can see this relationship by the fact that the gradient at y_o coincides with the subgradient at y_o. We indicate this subgradient g at y_o by a single function $h(y) := g^t(y - y_o) + f(y_o)$.

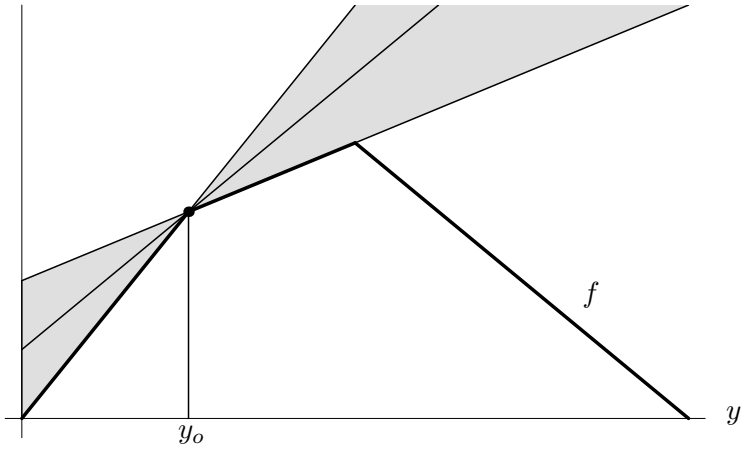

Figure 2.3: A non-differentiable function

In the non-differentiable case (see Figure 2.3) a gradient at y_o does not exists, but instead it is possible to obtain a continuum of subgradients, which is represented by the gray area. It arises the question how the subdifferential of the Lagrangian function L can be determined for each y. We show this in the following theorem.

Theorem 2.3. *Let* $X := \{\mathrm{A}x = b \wedge \mathrm{B}x \leq d \wedge x \geq \mathbf{0}\}$ *and* L *the Lagrangian relaxation of LP. For a given* $y_o \in \mathbb{R}^m$ *let* $X_{(y_o)}$ *be defined as* $X_{(y_o)} := \{x \in X | L(y_o) = c^t x + y_o^t(b - \mathrm{A}x)\}$. *It holds*

$$\delta L(y_o) = conv\{b - \mathrm{A}x | \; \forall \, x \in X_{(y_o)}\}. \tag{2.38}$$

Proof. We have to show two cases.

1. $\delta L(y_o) \supseteq \mathrm{conv}\{b - Ax | \; \forall \, x \in X_{(y_o)}\}$:

   It holds for all $y \in \mathbb{R}^m$ and all $\tilde{x} \in X_{(y_o)}$

$$\begin{aligned} L(y) - L(y_o) &= \min_{\mathbf{0} \leq x \leq \mathbf{1}} c^t x + y^t(b - Ax) \; - \; \min_{\mathbf{0} \leq x \leq \mathbf{1}} c^t x + y_o^t(b - Ax) \\ &= \min_{\mathbf{0} \leq x \leq \mathbf{1}} c^t x + y^t(b - Ax) \; - \; \left(c^t \tilde{x} + y_o^t(b - A\tilde{x})\right) \\ &\leq \quad c^t \tilde{x} + y^t(b - A\tilde{x}) \quad - \; \left(c^t \tilde{x} + y_o^t(b - A\tilde{x})\right) \\ &= \quad (b - A\tilde{x})^{\,t} \, (y - y_o), \end{aligned} \tag{2.39}$$

   which implies that for all $y, y_o \in \mathbb{R}^m$, and $x \in X_{(y_o)}$ the subgradient inequality (2.36) holds.

2. $\delta L(y_o) \subseteq \mathrm{conv}\{b - Ax | \; \forall \, x \in X_{(y_o)}\}$:

   We will present an indirect proof by assuming $u \in \delta L(y_o)$, but $u \notin \mathrm{conv}\{b -$

$Ax|\ \forall x \in X_{(y_o)}\}$. The sets $\{u\}$ and $\text{conv}\{b-Ax|\ \forall x \in X_{(y_o)}\}$ are convex and closed. From the theory of convex sets (see Hiriart-Urruty and Lemaréchal [1993a]) we know that a hyper-plane exists, which separates both sets, i.e., there exists a vector $\tilde{y}$ with $\tilde{y}^t u < \tilde{y}^t v$ for all $v \in \text{conv}\{b-Ax|\ \forall x \in X_{(y_o)}\}$. Set $y := y_o + \alpha\,\tilde{y}$ with $\alpha \in \mathbb{R}$ and consider

$$\begin{aligned} L(y) &= \min_{x\in X} c^t x + y^t(b-Ax) \\ &= \min_{x\in X} c^t x + (y_o + \alpha\,\tilde{y})^t(b-Ax) \\ &= \min_{x\in X} (c^t - y_o^t A - \alpha\,\tilde{y}^t A)x + (y_o + \alpha\,\tilde{y})^t b \end{aligned} \tag{2.40}$$

and

$$\begin{aligned} L(y_o) &= \min_{x\in X} c^t x + y_o^t(b-Ax) \\ &= \min_{x\in X} (c^t - y_o^t A)x + y_o^t b. \end{aligned} \tag{2.41}$$

The reduced costs of both programs differ by the term $-\alpha\,\tilde{y}^t A$. Let $J \subseteq \{1,\ldots,n\}$, such that $j \in J$ if and only if $(c^t - y_o^t A)_j \neq 0$. Furthermore let $p := \min_{j\in J} |(c^t - y_o^t A)_j|$. If we choose α such that $|-\alpha\,\tilde{y}^t A| < p$ then the term $-\alpha\,\tilde{y}^t A$ does not change the sign of the reduced costs, which are non-zero. If all reduced costs were non-zero this would imply that $X_{(y)} = X_{(y_o)}$. But in general there exist reduced costs, which are zero, therefore the conclusion is $X_{(y)} \subseteq X_{(y_o)}$. In this case it holds

$$\begin{aligned} L(y) &= \min_{x\in X}\ c^t x + (y_o + \alpha\,\tilde{y})^t(b-Ax) \\ &\overset{X_{(y)}\subseteq X_{(y_o)}}{=} \min_{x\in X_{(y_o)}}\ c^t x + (y_o + \alpha\,\tilde{y})^t(b-Ax) \\ &\overset{x^* opt.}{=} c^t x^* + y_o^t(b-Ax^*) + \alpha\,\tilde{y}^t(b-Ax^*) \\ &\overset{x^*\in X_{(y_o)}}{>} c^t x^* + y_o^t(b-Ax^*) + \alpha\,\tilde{y}^t u \\ &= \left[\min_{x\in X} c^t x + y_o^t(b-Ax)\right] + (y-y_o)^t u \\ &= L(y_o) + (y-y_o)^t u, \end{aligned} \tag{2.42}$$

which implies $L(y) > L(y_o) + (y-y_o)^t u$ in a small enough neighborhood of y_o, which contradicts the assumption $u \in \delta L(y_o)$. Therefore, it holds $\delta L(y_o) \subseteq \text{conv}(\{b-Ax|\ \forall\, x \in X_{(y_o)}\})$.

□

Remark 2.6. In the case of SPP the subdifferential is defined as $\delta L(y_o) = \text{conv}(\{1 - Ax | \; \forall \; x \in X_{(y_o)}\})$.

It is a well-known fact that the gradient for differentiable functions yields an ascent direction, but a subgradient is not necessarily an ascent direction. In Figure 2.4 we show such a case, where the contours of a concave function are illustrated. Note that this function is non-differentiable. At the left corner we indicate three

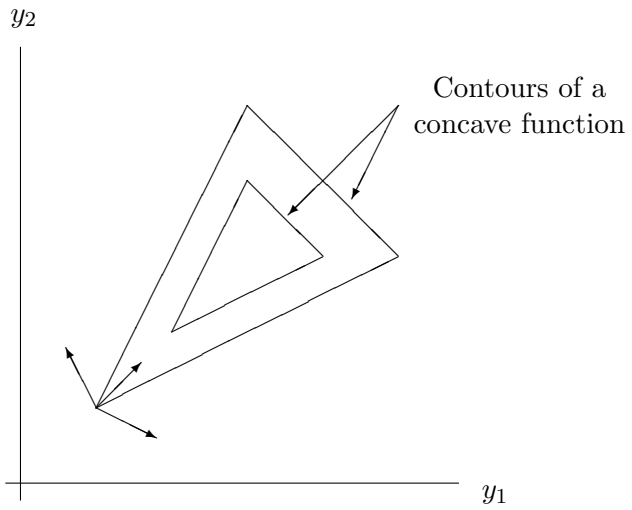

Figure 2.4: Non-ascent subgradient directions

elements of the corresponding subdifferential by three vectors. The middle vector represents an ascent subgradient, whereas both outer vectors represents subgradients, which are non-ascending. Nevertheless, subgradients have a nice property. Subgradients are in an acute angle with the direction toward the optimal set. That means if we take appropriate small step-sizes along these subgradients in Euclidean space then the distance between the current point and the optimal ones can be decreased step by step (see Bazaraa, Sherali, and Shetty [1993]). This implies that methods, which use arbitrary subgradients and appropriate step-sizes, approach optimal points monotonely with respect to the Euclidean distance.

The next theorem shows how the subdifferential indicates optimality in the non-differentiable case. It is a generalization of the well-known fact that a point y^* maximizes a concave differentiable function f, if the first order derivative of f at y^* is zero.

Theorem 2.4. *Let f be a concave function then it holds*

$$\mathbf{0} \in \delta f(y^*) \text{ if and only if } f \text{ reaches its maximum in } y^*.$$

Proof. If $\mathbf{0}$ is a subgradient of f at y^* then it holds

$$f(y) \leq f(y^*) + \mathbf{0}^t(y - y^*) = f(y^*) \quad \forall y \in \mathbb{R}^m, \tag{2.43}$$

which is equivalent to

$$\max_y f(y) = f(y^*) \tag{2.44}$$

□

Now, we have described all ingredients to formulate a generic version of the basic subgradient method.

INPUT: 1. Concave function $f : \mathbb{R}^m \mapsto \mathbb{R}$
2. Start vector $y^{(0)}$
3. $p \in \mathbb{N}$
4. $0 < \epsilon \in \mathbb{R}$

OUTPUT: An optimal or 'nearly' optimal vector y^* and its function value $f(y^*)$

1: Obtain subgradient $g^{(0)} \in \delta f(y^{(0)})$.
2: $k := 0$
3: **while** $\mathbf{0} \notin \delta f(y^k)$ and the last p improvements of the function value are not smaller than ϵ **do**
4: $\quad y^{(k+1)} := y^{(k)} + s^{(k)} \frac{g^{(k)}}{\|g^{(k)}\|}$
5: $\quad k := k + 1$
6: $\quad$ Obtain subgradient $g^{(k)} \in \delta f(y^{(k)})$.
7: **end while**
8: Return $y^{(k)}$ and $f(y^{(k)})$.

Algorithm 1: Subgradient Method

The progression of the basic subgradient method depends on the choice of the step-sizes $s^{(k)}$. Polyak [1967] has shown that for the following step-size criteria this method using only the stop criterion $\mathbf{0} \notin \delta f(y^k)$ converges to an optimal solution. Note that in this case the subgradient method could take infinitely many steps.

Theorem 2.5. *Let $f : \mathbb{R}^m \mapsto \mathbb{R}$ be bounded from above and concave. Furthermore, let $g^{(k)}$ be a subgradient of f at $y^{(k)} \in \mathbb{R}^m$. If the sequence $\{s^{(k)}\}_{k\in\mathbb{N}}$ meets the properties*

1. $s^{(k)} > 0 \qquad s^{(k)} \in \mathbb{R} \quad \forall k \in \mathbb{N}$
2. $\lim_{k\to\infty} s^{(k)} = 0$
3. $\sum_{k=0}^{\infty} s^{(k)} = \infty$,

then the basic subgradient method as described in Algorithm 1 produces a sequence $\{y^{(k)}\}_{k\in\mathbb{N}}$ such that

$$\lim_{k\to\infty} f(y^{(k)}) = \max_y f(y).$$

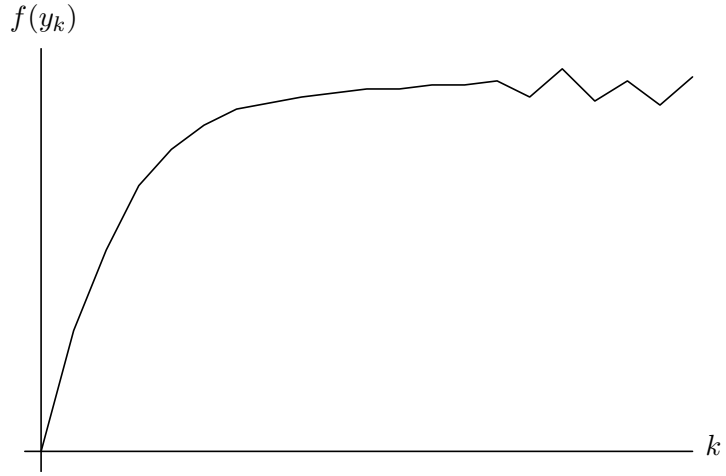

Figure 2.5: Practical behavior of the basic subgradient method

Practically these step-size criteria do not lead to a satisfying convergence rate. In Figure 2.5 a typical progress of the basic subgradient method using such step-sizes is illustrated. In the beginning the improvement rate of the objective value of f is satisfying, but in the long run the improvements begin to fluctuate. This means that theoretical convergence takes seldomly place. Therefore, many methods use heuristical step-sizes in order to get at least faster improvements. These heuristical step-sizes do not necessarily fulfill every criterion proposed by Polyak [1967] (see for example Caprara, Fischetti, and Toth [1996] or Kokott and Löbel [1996]), but they lead to satisfying results.

In addition to the choice of step-sizes, there exist further possibilities to reach fast improvements. We will present two main ideas.

Minimum Norm We could choose a subgradient with the lowest Euclidean norm. Bazaraa, Sherali, and Shetty [1993] have shown that this particular subgradient forms the steepest ascent direction. In combination with a suitable step-size $s^{(k)} \in \mathbb{R}^+$ the update formula $y^{(k+1)} := y^{(k)} + s^{(k)}/\|g^{(k)}\|\, g^{(k)}$ leads to a strict improvement of the objective value, i.e., $f(y^{(k)}) < f(y^{(k+1)})$.

Bundle Methods In some cases it is too time-consuming to calculate the whole subdifferential in order to determine a minimum norm subgradient. Moreover, the basic subgradient method often produces a sequence of $y^{(k)}$, which proceeds in a 'zig-zag' course to the optimal point. A technique called bundling tries to overcome both problems as follows. Instead of using a single subgradient at each iteration, we use a weighted combination of several elements of the subdifferential or several subgradients, which were used in previous iterations. By this we try to approximate a minimum norm subgradient and hope to avoid 'zig-zag' courses. Note that bundling is only a heuristic, but in practice this idea is worth trying (see Crowder [1976] and Kokott and Löbel [1996]). For more information the

reader is referred to Hiriart-Urruty and Lemaréchal [1993b].

We apply the bundle technique to improve the basic subgradient method. In our implementation we use

$$\tilde{g}^{(k)} := 0.6\, g^{(k)} + 0.2\, g^{(k-1)} + 0.1\, g^{(k-2)} + 0.1\, g^{(k-3)}, \tag{2.45}$$

as the new step direction, where the initial subgradients $g^{(-1)}$, $g^{(-2)}$, and $g^{(-3)}$ are set to the first calculated subgradient $g^{(0)}$. In this case, the bundling effect starts with the second iteration. It turns out that this setting of weights seems to be robust. Using these thoughts, we get the following algorithm that we call **subgradient bundle method SBM**.

INPUT: 1. $f : \mathbb{R}^m \mapsto \mathbb{R}$
2. Start vector $y^{(0)}$
3. $p \in \mathbb{N}$
4. $0 < \epsilon \in \mathbb{R}$

OUTPUT: An optimal or 'nearly' optimal vector y^* and its function value $f(y^*)$

1: Obtain subgradient $g^{(0)} \in \delta f(y^{(0)})$.
2: $g^{(-3)} := g^{(-2)} := g^{(-1)} := g^{(0)}$
3: $k := 0$
4: **while** $\mathbf{0} \notin \delta f(y^k)$ and the last p improvements of the function value are not smaller than ϵ **do**
5: $\quad \tilde{g}^{(k)} := 0.6\, g^{(k)} + 0.2\, g^{(k-1)} + 0.1\, g^{(k-2)} + 0.1\, g^{(k-3)}$
6: $\quad y^{(k+1)} := y^{(k)} + s^{(k)} \dfrac{\tilde{g}^{(k)}}{\|\tilde{g}^{(k)}\|}$
7: $\quad k := k + 1$
8: $\quad$ Obtain subgradient $g^{(k)} \in \delta f(y^{(0)})$.
9: **end while**
10: Return $y^{(k)}$ and $f(y^{(k)})$.

Algorithm 2: SBM Algorithm

As for the basic case heuristical step-sizes lead empirically to better results than those proposed by Polyak [1967].

Remark 2.7. Beside set partitioning the **set covering model** is also used in practice. The only difference between both models is that the latter one deals with the constraint $Ax \geq \mathbf{1}$ instead of $Ax = \mathbf{1}$. The Lagrangian relaxation of both are similar, therefore we can also consider applications of set covering problems for describing state-of-the-art applications of set partitioning. In this respect, the FASTER competition is worth mentioning. In 1994 the Italian Railway Company and Italian Operational Research Society organized this competition, where set covering instances consisting of up to 5000 rows and 1 million columns and arising from crew scheduling problems for this railway company should be solved. Note that although set partitioning and set covering are similar the size limits of set covering instances are greater than those of the set partitioning instances.

Two successful competitors are Ceria, Nobili, and Sassano [1995] and Caprara, Fischetti, and Toth [1996], who got the first prize. Caprara, Fischetti, and Toth [1996] have used among other things a subgradient method, which has in each step a linear complexity in the number of non-zeros of the matrix A.

2.2.3.2 Coordinate Ascent Method–CAM

Another approach to solving maximization problems of non-differentiable functions is called **ascent methods** (see Bertsekas [1995]). As other **line search methods** they reduce the often higher-dimensional problem to one-dimensional problems by maximizing in ascent directions. The question is how to get such directions. A possibility is to ascend along a number of fixed directions.

One method, which uses fixed directions, is called **coordinate ascent method** (see Wedelin [1995]), which proceeds along coordinate directions. We will refer to this method by **CAM**. Unfortunately, there is a fundamental difficulty with fixed line ascent methods. It is possible to get stuck at a corner, from which it is impossible to make progress along any fixed direction, i.e., these methods are in general not globally convergent. We illustrate this problem in Figure 2.6. Here, the same contours of the concave function used in Figure 2.4 are plotted. But in this case only the coordinate directions indicated by both vectors can be considered by the coordinate ascent method. It can be seen that both directions are

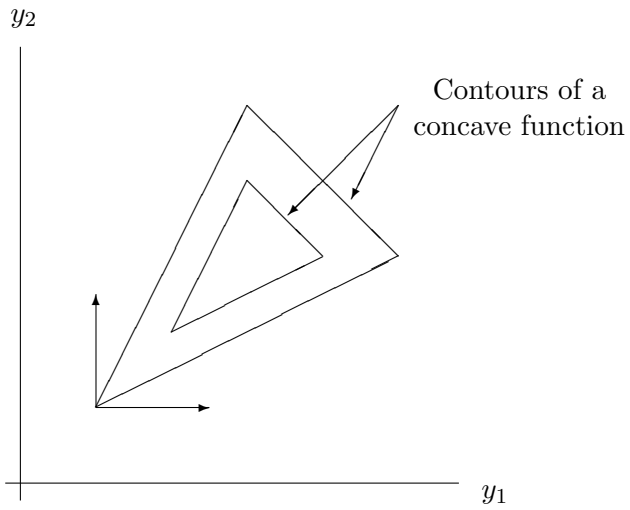

Figure 2.6: Getting stuck at a corner

descending though the maximum has not been reached. Hence, the coordinate ascent method gets stuck at this corner.

Furthermore, convergence may be slow. An example of such a situation is shown in Figure 2.7. In this figure the lines are projections of the edges of a concave and

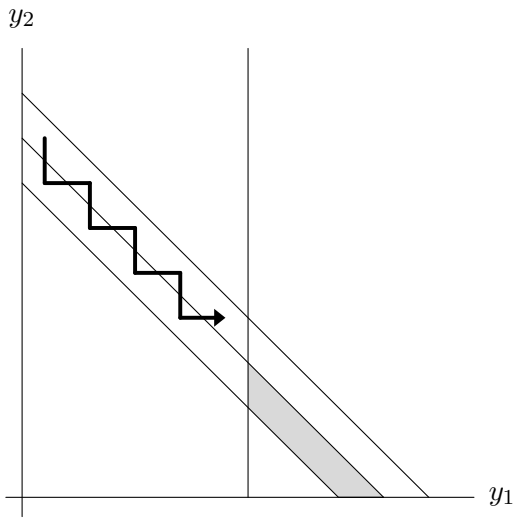

Figure 2.7: Slow convergence

piecewise linear function on its domain. The gray set indicates the set of points, where the concave function reaches its maximum, i.e., over this set the function forms a plateau. We see that the speed of convergence, which is indicated by the thick-lined course, depends on the distance between the three parallel lines. The speed of convergence could also slow down if many of such projections cluster around the optimal set such that the 'zig-zag' course would become more extreme. We call a location, which is clustered in that way, **degenerated**. A problem, which has a large number of such locations, is called a degenerated problem. Real-world problems are often highly degenerated, especially around the set of optimal points. Therefore, the coordinate ascent method can slow down the nearer it gets to an optimal solution.

A possible approach, which overcomes both problems, is called **spacer steps** (see Luenberger [1989]). The idea consists of inserting infinitely many steps of a convergent algorithm (such as the subgradient method) in order to reach a theoretical convergence. The only requirement imposed on the iterations of the original algorithms is that it must not take decreasing steps, which is met by the coordinate ascent method. Therefore, spacer steps can be built in, especially whenever this method gets stuck at a corner or the convergence rate deteriorates.

The main advantages of the coordinate ascent method over other line search methods are the basic operations, which are not very complex (see Bertsekas [1995]), and its good performance. We will show this in the following sections.

The Method We consider equation (2.31):

$$L(y) = \min_{\mathbf{0} \leq x \leq \mathbf{1}} (c^t - y^t A)x + y^t \mathbf{1}$$

In this work we are mainly interested in finding the optimal objective value $L(y^*)$. Therefore, we focus on finding either an optimal Lagrangian multiplier solution or at least a good approximation of it in order to obtain a good lower bound for SPP.

We now explain how y_i, which is the i^{th} component of vector y, is computed. The calculation is repeated for all possible i. Let $i \in \{1, \ldots, m\}$ be fixed and let supp(i) be the support of row i, i.e., supp$(i) := \{j | a_{ij} = 1\}$, where a_{ij} is the element in the i^{th} row and j^{th} column of matrix A. Furthermore, for each fixed row i let r_1 be the lowest and r_2 be the second lowest reduced cost of supp(i). The update formula for y_i is

$$\begin{aligned} y_i &:= y_i + \Big[r_1 + p\,(r_2 - r_1) \Big] \\ \overline{c}_j &:= \overline{c}_j - \Big[r_1 + p\,(r_2 - r_1) \Big] \quad \forall j \in \text{supp}(i), \end{aligned} \tag{2.46}$$

where $p \in (0, 1)$. If supp$(i) = 1$ then we set $r_2 := r_1$. Using (2.46) we obtain the following lemma

Lemma 2.2. *If y_i and $\overline{c}_j$ $\forall j \in$ supp(i) are updated as in* (2.46)*, then there exists an optimal solution $x^{(y)}$ of $L(y)$, which fulfills the i^{th} constraint of $\mathrm{A}x^{(y)} = \mathbf{1}$, i.e., $a_i^t x^{(y)} = 1$, where a_i defines the i^{th} row of matrix A.*

Proof. Considering (2.32) we only have to show that after the update of $\overline{c}$, either there exists exactly one $j \in \text{supp}(i)$ with $\overline{c}_j < 0$ or at least one $j \in \text{supp}(i)$ with $\overline{c}_j = 0$ and $\overline{c} \geq 0$. In the first case all feasible solutions x of $L(y)$ would have a component, which would equal 1 for exactly one $j \in \text{supp}(i)$. In the second case we can construct a feasible solution of $L(y)$, where the solution would have the same property like the solution in the first case. By showing these cases the proof would be concluded.

Let $\overline{c}_j^{(old)}$ denote the reduced cost of variable $x_j^{(y)}$ before the update and $\overline{c}_j^{(new)}$ the new reduced cost for all $j \in \text{supp}(i)$ afterwards. We will distinguish two cases.

1. $r_1 \neq r_2$:
   If $r_1 \neq r_2$, then it holds by definition of r_1 and r_2 that $r_1 < r_2$. The definition of $\overline{c}_j^{(old)}$ yields that $\overline{c}_{j^*}^{(old)} = r_1$ for a index $j^* \in \text{supp}(i)$. Therefore, we obtain

$$\begin{aligned} \overline{c}_{j^*}^{(new)} &= \overline{c}_{j^*}^{(old)} - \Big[r_1 + p\,(r_2 - r_1) \Big] \\ &= r_1 - \Big[r_1 + p\,(r_2 - r_1) \Big] \\ &= p\,(r_1 - r_2) \\ &< 0. \end{aligned} \tag{2.47}$$

For any other index $j \in \text{supp}(i)$ with $j \neq j^*$ it holds by definition $\overline{c}_j^{(old)} \geq r_2$. This yields

$$\begin{aligned} \overline{c}_j^{(new)} &= \overline{c}_j^{(old)} - \Big[r_1 + p\,(r_2 - r_1)\Big] \\ &\geq r_2 - \Big[r_1 + p\,(r_2 - r_1)\Big] \\ &= (1-p)\,(r_2 - r_1) \\ &> 0. \end{aligned} \tag{2.48}$$

2. $r_1 = r_2$:
   In (2.47) and (2.48) we obtain equality in the last equation because of $r_1 = r_2$. That means that $\overline{c}_{j^*}^{(new)} = 0$ and $\overline{c}_j^{(new)} \geq 0 \; \forall j \in \text{supp}(i)$.

□

The proof of Lemma 2.2 implies the following update algorithm for the i^{th} entry of $y^{(k+1)}$.

INPUT:
1. $L : \mathbb{R}^m \mapsto \mathbb{R}$
2. $y_i^{(k)}$
3. $\overline{c}^{\,(k)}$
4. $p \in (0,1)$

OUTPUT:
1. $y_i^{(k+1)}$
2. $\overline{c}^{\,(k+1)}$
3. $L(y^{(k+1)})$

1: Let r_1 and r_2 be the lowest and second lowest reduced cost of $\text{supp}(i)$
2: $\Delta y_i := r_1 + p\,(r_2 - r_1)$
3: $y_i^{(k+1)} := y_i^{(k)} + \Delta y_i$
4: **for all** $j \in \text{supp}(i)$ **do**
5: $\quad \overline{c}_j^{(k+1)} := \overline{c}_j^{(k)} - \Delta y_i$
6: **end for**
7: $L(y^{(k+1)}) := \sum_{j\,:\,\overline{c}_j^{(k+1)} < 0} \overline{c}_j^{(k+1)} + \sum_{i=1,\ldots,n} y_i^{(k+1)}$
8: Return $y^{(k+1)}$ and $f(y^{(k+1)})$.

Algorithm 3: Update algorithm of $y_i^{(k+1)}$

In the next lemma we show that the update of the reduced cost is correct.

Lemma 2.3. *$\overline{c}^{\,(k+1)}$ evaluated in Algorithm 3 corresponds to the reduced cost with respect to $y^{(k+1)}$, i.e., $\overline{c}^{\,(k+1)^t}$ equals $c^t - y^{(k+1)^t} A$.*

Proof. It holds

$$c^t - y^{(k+1)^t} A \overset{\text{step (3)}}{=} c^t - y^{(k)^t} A - \Delta y_i \, a_i. \tag{2.49}$$

Therefore, it follows for all $j \in \{1, \ldots, n\}$

$$\left(c^t - y^{(k+1)^t} A\right)_j = \begin{cases} \overline{c}_j^{(k)} - \Delta y_i & \text{if } j \in \operatorname{supp}(i) \\ \overline{c}_j^{(k)} & \text{if } j \notin \operatorname{supp}(i). \end{cases} \tag{2.50}$$

Step 4 until step 6 finally imply the assertion. □

Geometric Interpretation Theorem 2.1 shows that L is a concave and piecewise linear function. We will illustrate, how CAM—using the update algorithm for each row as described in Algorithm 3—proceeds.

For each y the i^{th} partial derivative can be written as

$$\frac{\delta L(y)}{\delta y_i} = 1 - a_i^t x^{(y)}, \tag{2.51}$$

where $x^{(y)}$ is an optimal solution to $L(y)$. The existence of the partial derivative is guaranteed by the fact that function L is a piecewise linear and continuous function. This means that at each y there exists for all y_i at least a one-sided partial derivative. As in the previous section shown y_i is chosen such that the optimal solution $x^{(y)}$ satisfies $a_i^t x^{(y)} = 1$, which yields $\frac{\delta L(y)}{\delta y_i} = 0$. Since L is concave we may therefore conclude that y is also a maximum point along the y_i-axis. Note that p determines the location on that plateau along y_i. If for example $p = 0.5$ then the new point would be at the middle of the corresponding plateau.

Example 2.1. We consider the problem

$$\begin{gathered} \max 6x_1 + 9x_2 + 2x_3 + 3x_4 + 5x_5 + 2x_6 + 5x_7 \\ \begin{array}{rcrcrcrcrcrcl} x_1 & + & x_2 & + & x_3 & + & x_4 & + & x_5 & & & = & 1 \\ x_1 & + & x_2 & & & & & + & x_6 & + & x_7 & = & 1 \end{array} \\ \mathbf{0} \leq x \leq \mathbf{1}. \end{gathered} \tag{2.52}$$

With respect to (2.11) the corresponding cost vector c^t is

$$c^t = (6, 9, 2, 3, 5, 2, 5)$$

and the corresponding matrix A is

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}.$$

Since we have two constraints L becomes a function of two variables, say $\left(\begin{smallmatrix} y_1 \\ y_2 \end{smallmatrix}\right) =: y$. Figure 2.8 illustrates function L in Lagrangian multiplier space. The lines correspond to the constraints $(c^t - y^t A)_j = 0$, which divide the Lagrangian multiplier

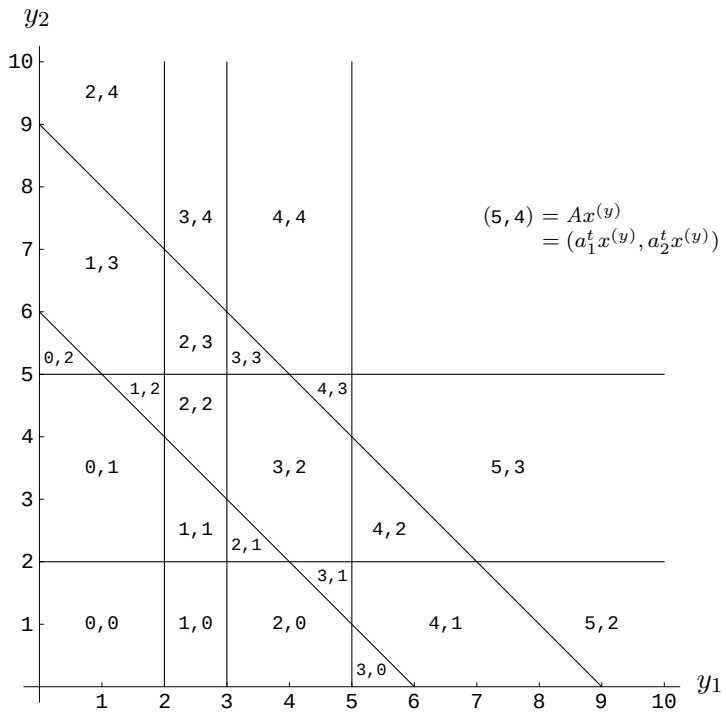

Figure 2.8: The constraints $(c^t - y^t A)_j = 0$ in Lagrangian multiplier space

space in several sets. Each set represents a set of points y, which have the common property of yielding the same optimal solutions x to $L(y)$. For example $\{y|y_i \geq 5\ \forall i = \{1,2\}\}$ is such a set. The calculation of the corresponding solution $x^{(y)}$ can be done in the following manner. $(7,7)$ is an element of the set $\{y|y_i \geq 5\ \forall i = \{1,2\}\}$. It holds

$$\begin{aligned} L(\binom{7}{7}) &= \min_{\mathbf{0}\leq x\leq \mathbf{1}} (c^t - \binom{7}{7}^t A)x + \binom{7}{7}^t \mathbf{1} \\ &= \min_{\mathbf{0}\leq x\leq \mathbf{1}} (-8,-5,-5,-4,-2,-5,-2)\, x + 14, \end{aligned} \tag{2.53}$$

which implies that the solution with respect to set $\{y|y_i \geq 5\ \forall i = \{1,2\}\}$ is $\mathbf{1}$. This holds for all $y \in \{y|y_i \geq 5\ \forall i = \{1,2\}\}$ because the corresponding reduced cost has only strictly negative entries. It holds $Ax^{(y)} = A\mathbf{1} = (5,4)^t$, i.e., the set $\{y|y_i \geq 5\ \forall i = \{1,2\}\}$ is associated with the pair $(5,4)$. We can conclude that each such set corresponds to a unique pair $(a_1^t x^{(y)}, a_2^t x^{(y)})$. At points, which lie on one or more lines, the corresponding solution $x^{(y)}$ is not unique because at least one component of the corresponding reduced cost is zero. In this case there exist at least two possible optimal solutions $x^{(y)}$. This implies that such points do not yield unique pairs $(a_1^t x^{(y)}, a_2^t x^{(y)})$.

In Figure 2.9 we illustrate the plot of L, the constraints $(c^t - y^t A)_j = 0$ by lines at the bottom of the plot and the pairs associated with each set, which result from these constraints. Note that these lines at the bottom of the plot are in fact

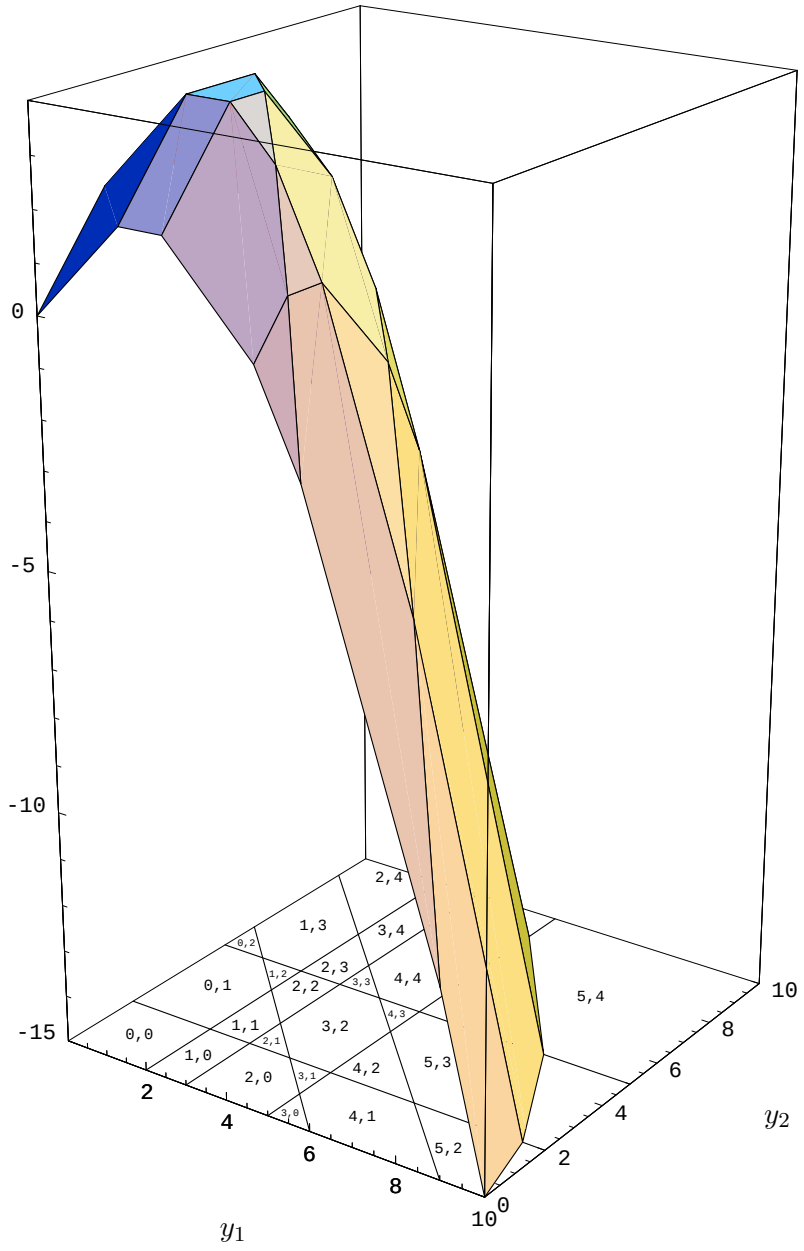

Figure 2.9: Plot of L and the constraints $(c^t - y^t A)_j = 0$

projections of the edges of L on the domain of L or Lagrangian multiplier space, respectively. However, from any point CAM proceeds horizontally (or vertically) until a set is reached, which has a 1 in the first (or second) entry. That means that from any point we move either in y_1-coordinate direction until we reach a y, where the corresponding optimal solution $x^{(y)}$ fulfills the first constraint of (2.52) or in y_2-coordinate direction until we reach a y, where the corresponding optimal solution $x^{(y)}$ fulfills the second constraint of (2.52). We interchangeably repeat this iteration until the set $(1,1)$ is reached, which corresponds to the set of optimal variables y^*.

The explanation for the optimality of set $(1,1)$ is as follows. Let Y be the set of y variables associated with set $(1,1)$, where $y^* \in Y$. It is not hard to see that $x^{(y^*)\,t} = (0,0,1,0,0,1,0)$ is the corresponding solution to set $(1,1)$. However, this solution is feasible to (2.52). That means particularly that $Ax^{(y^*)} = \mathbf{1}$. Therefore, it holds for all $y \in Y$

$$\begin{aligned} L(y) &= c^t x^{(y^*)} + y^t(\mathbf{1} - Ax^{(y^*)}) \\ &\overset{Ax^{(y^*)}=\mathbf{1}}{=} c^t x^{(y^*)} + y^t \mathbf{0} \\ &= c^t x^{(y^*)} \\ &= 4. \end{aligned} \tag{2.54}$$

Note that L has the same value for all $y \in Y$, therefore L forms a plateau over the set $(1,1)$ (see Figure 2.9). That means that L yields its maximum over this set. In (2.54) we have calculated that $L(y^*) = 4$ is a lower bound for problem (2.52). Furthermore, this lower bound equals the objective value of a primal solution of the original problem, namely, $x^{(y^*)}$. Therefore, the solution $x^{(y^*)} = (0,0,1,0,0,1,0)^{\,t}$ associated with the set $(1,1)$ is an optimal solution of problem (2.52).

In Figure 2.10 and Figure 2.11 we illustrate two possible courses for $p = 0.5$. Let us start for example at $y^{(0)} := \binom{8.5}{6}$, and assume that we first move along the y_1-axis. We have already mentioned that for $p = 0.5$ the algorithm always chooses the middle of the plateau along the considered axis. In this case we go from $\binom{8.5}{6}$ to $\binom{1}{6} =: y^{(1)}$, because over the line from (0,6) to (2,6) L defines a plateau. Note that $y^{(1)}$ is an element of set (1,3), i.e., the first entry equals 1. After three of those iterations we reach the optimal set $(1,1)$. Another possibility would be to move first in y_2-coordinate direction. Again, after 3 iterations we reach the optimal set $(1,1)$ (see Figure 2.11).

In higher dimensional problems the set of global maximum points Y forms a polyhedron in Lagrangian multiplier space. However, L attains a constant value for all $y \in Y$. Let us assume first that the polyhedron is full-dimensional. Using the fact that L is a concave and piecewise linear function we can conclude that all $y \in Y$ are optimal Lagrangian multipliers (see Luenberger [1989]). Optimality can also be proved by the subgradient theory. For all $y \in Y$, which are inner

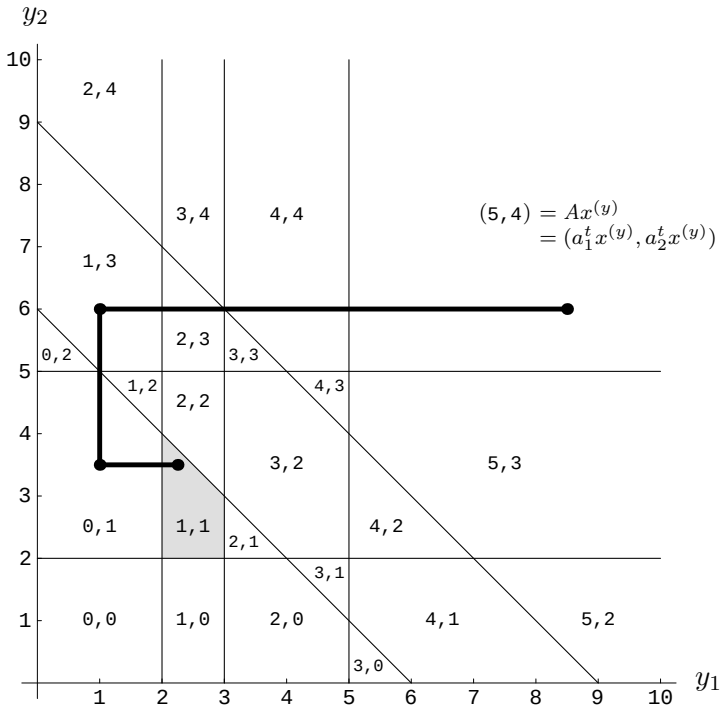

Figure 2.10: CAM run—starting with y_1

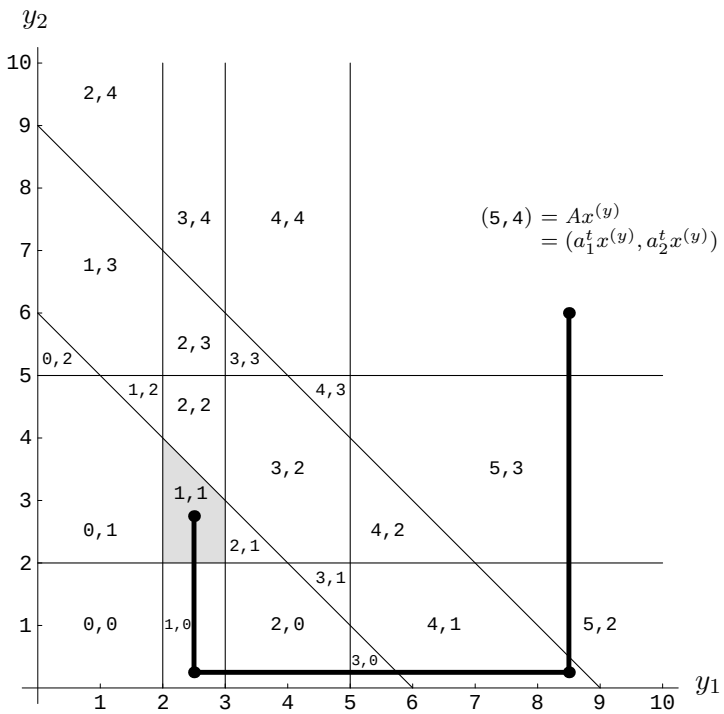

Figure 2.11: CAM run—starting with y_2

points of Y, it holds $\nabla L(y) = \mathbf{0}$. We have already shown that $\nabla L(y)$ can be identified as a subdifferential of L at y. For the inner points $y \in Y$ the corresponding subdifferential consist only of the zero vector and by Theorem 2.4 optimality is proven for this case.

It could also happen that the dimension of the polyhedron corresponding to the set of the optimal variables y^* is not full-dimensional. In that case, at least one component of the reduced cost with respect to y^* is zero. I.e., the corresponding solution would not be unique, but this would not have any impacts on the optimality of $L(y^*)$. In the full-dimensional case we can recognize the optimality of y^* since $\nabla L(y^*) = \mathbf{0}$. But if the optimal polyhedron is not full-dimensional, it could be very time-consuming to check whether we have reached optimality or not since $\nabla L(y)$ could be a polyhedron with exponentially many vertices. In order to avoid this we use a heuristic stop criterion. We check regularly the improvement of the lower bound and if the improvements get too small the algorithm has to be terminated.

Convergence Aspects We have explained in the beginning of this section why CAM is in general not convergent. It arises the question, whether the set partitioning problem forms an exception to this rule. In order to answer this question we have to check whether it is possible that a non-optimal Lagrangian multiplier vector exists, where the corresponding Δy_i is zero for all $i \in \{1, \ldots, m\}$, i.e., we have to check whether at a non-optimal y the update algorithm described in Algorithm 3 could lead to

$$\Delta y_i \;=\; r_1 + p\,(r_2 - r_1) \;=\; 0 \quad \forall i \in \{1, \ldots, m\}. \tag{2.55}$$

There exist two cases.

1. $r_1 \neq r_2\ \forall i \in 1, \ldots, m$:
   If $r_1 \neq r_2$, then we can state p as follows

$$\begin{aligned} & r_1 + p\,(r_2 - r_1) &&= 0 \\ \Leftrightarrow \quad & p &&= \frac{-r_1}{r_2 - r_1} \\ \Leftrightarrow \quad & p &&= \frac{r_1}{r_1 - r_2}. \end{aligned} \tag{2.56}$$

   We assumed $r_1 \neq r_2$ and by definition of r_1 and r_2 it holds $r_1 \leq r_2$. This implies that $r_1 < r_2$ or $r_1 - r_2 < 0$, which leads to $p < 0$. This contradicts to $p \in (0, 1)$, therefore $\Delta y_i \neq 0$ if $r_1 \neq r_2$.

2. $r_1 = r_2\ \forall i \in 1, \ldots, m$:
   This case yields $p = -r_1$. If $(-r_1) \in (0, 1)$ we can perturb p in order to avoid getting stuck. If $(-r_1) \notin (0, 1)$ and r_1 is non-zero then Δy_i is non-zero, i.e., we are able to proceed, but if r_1 is zero, which yields that r_2 is zero, too, then Δy_i is also zero.

Our considerations imply that the update algorithm produces $\Delta y_i = \mathbf{0}$ if and only if $r_1 = r_2 = 0$. This means geometrically that the corresponding Lagrangian multiplier vector y is located at the maximal vertex along the considered axis, i.e., we can not even move to a point, which yields the same lower bound along this axis. If this happens for all axes, we have either reached optimality or we have fallen into a local trap, from which we can not make any improvements along the axes.

This geometric intuition is algebraically substantiated as follows. If for a Lagrangian multiplier vector y the equation $r_1 = r_2 = 0$ holds for all $i \in \{1, \ldots, m\}$ then it follows that for a set of columns the reduced costs are zero and for the other columns they are greater than zero or equal zero. We assume that they are strictly greater than zero. In this case the corresponding entry of $x^{(y)}$ is zero. For the columns with zero reduced costs we can choose whether the corresponding entry of $x^{(y)}$ should be zero or one. In other words we get a feasibility problem of our set partitioning problem. If this feasibility problem has a solution, then we have reached optimality. The reason for that is that in this case $\mathbf{1} - Ax^{(y)} = 0$ holds, i.e., the zero vector would be an element of the subdifferential, which implies by Theorem 2.4 optimality. If the considered feasibility problem has no solution, then $\mathbf{1} - Ax^{(y)} \neq 0$. In this case Theorem 2.4 proves that we have not reached optimality, which means that it holds $\Delta y = \mathbf{0}$ and $\mathbf{1} - Ax^{(y)} \neq 0$, i.e., we have got stuck at a corner.

By this consideration we can construct a counterexample. We need a matrix A, which implies at least one feasible solution. Furthermore, A must contain a sub-matrix consisting of columns of A with at least two non-zero entries in each row. A possible matrix is

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

We will use the first three columns as the sub-matrix mentioned above. Next, we need a cost vector and Lagrangian multipliers. They must yield reduced costs of zero for the first three columns and for example a reduced cost of one for the last column. These demands can be satisfied by the following system of equations:

$$\begin{array}{lllllllll} c_1 & - & y_1 & & & - & y_3 & = & 0 \quad (= \overline{c}_1) \\ c_2 & - & y_1 & - & y_2 & & & = & 0 \quad (= \overline{c}_2) \\ c_3 & & & - & y_2 & - & y_3 & = & 0 \quad (= \overline{c}_3) \\ c_4 & & & & & - & y_3 & = & 1 \quad (= \overline{c}_4). \end{array} \tag{2.57}$$

In this system we have four equations and seven variables. If we set $y_1 := y_2 := y_3 := 1$ we get a cost vector of $(2, 2, 2, 2)$. The resulting optimization problem has only one feasible primal solution, namely, $x^* = (0, 1, 0, 1)^t$. This implies that this solution is also an optimal solution. The corresponding optimal objective value is 4.

Now we consider the Lagrangian relaxation of this problem at $(y_1, y_2, y_3) = (1, 1, 1)$. Due to our construction CAM can not move further, but it is easy to show that $L(\ (1,1,1)^{\,t}\ )$ equals 3. This means that we have not reached the maximum of L. Therefore, CAM has got stuck at this corner. By this we have proven that CAM is not globally convergent for set partitioning problems.

In the next section we describe the last type of methods for determining linear programming bounds, which is called surrogate relaxation methods.

2.2.4 Surrogate Relaxation Methods

We consider linear programs of the form

$$\begin{aligned} \min\ & c(x) \\ A\ & x\ = b \\ & x\ \leq d \\ & x\ \geq \mathbf{0}, \end{aligned} \tag{2.58}$$

where c is a cost function in x. Note that in the case of LP the cost function c is linear. The corresponding Lagrangian relaxation has the form

$$\max_{y} \hat{L}(y) = \max_{y} \min_{\substack{x \leq d \\ x \geq \mathbf{0}}}\ c(x) + \ y^t(b - Ax). \tag{2.59}$$

As the Lagrangian relaxation the **surrogate relaxation method** maximizes over a set of functions, but in this case they are defined as

$$\hat{\hat{L}}(y, x) := c(x) + \ y^t(b - Ax), \tag{2.60}$$

where $x \leq d$, $x \geq \mathbf{0}$, and y denotes the **surrogate multiplier vector**. These functions are called **surrogate functions**. Note the important differences between surrogate and Lagrangian functions. The surrogate function depends on both variables x and y, whereas the Lagrangian function depends only on y because vector x depends on y. Therefore, the surrogate relaxation is a generalization of the Lagrangian relaxation. The reader should not be confused because of the two names of y. We call y a Lagrangian multiplier vector with respect to the Lagrangian relaxed problem and a surrogate multiplier vector with respect to the surrogate relaxed problem. By this, we emphasize that both methods deal with the same y, but in different manners. A method, which resolves the surrogate relaxation method, is called **surrogate subgradient method**. This method is related to the **subgradient method**, which solves the Lagrangian relaxed problem. In the next section we will explain how surrogate subgradient methods work.

2.2.4.1 Surrogate Subgradient Methods

Consider

$$\begin{aligned} \min\ & c(x) \\ A\ & x\ = b \\ & x\ \leq d \\ & x\ \geq \mathbf{0}\ . \end{aligned} \tag{2.61}$$

Note that c must not be linear in x. We decompose $x := (x_{(1)}, \ldots, x_{(l)})$, $c(x) := (c_{(1)}(x), \ldots, c_{(l)}(x))$, $A := (A_{(1)}, \ldots, A_{(l)})$, and $d := (d_{(1)}, \ldots, d_{(l)})$. Thus, the Lagrangian function, which has the structure

$$\hat{L}(y) := \min_{\substack{x \leq d \\ x \geq \mathbf{0}}}\ c(x) + \ y^t(b - Ax) \tag{2.62}$$

can be rewritten in terms of l individual subproblems:

$$\hat{L}_j(y) = \min_{\substack{x_{(j)} \leq d_{(j)} \\ x_{(j)} \geq \mathbf{0}}}\ c_{(j)}(x_{(j)}) - y^t A_{(j)} x_{(j)} \quad j \in 1, \ldots, l \tag{2.63}$$

$$\hat{L}(y) = \sum_{j=1}^{l} \hat{L}_j(y) + y^t b. \tag{2.64}$$

Thus the Lagrangian relaxation can be formulated as

$$\max_y \hat{L}(y) = \max_y \sum_{j=1}^{l} \hat{L}_j(y) + y^t b. \tag{2.65}$$

As mentioned before problem (2.65) can be solved by subgradient methods, where subgradients are in acute angle with the direction toward y^*, and the distance between the current multipliers and the optimal y^* can be decreased step by step. However, in order to get a subgradient, we have to solve all subproblems, which could be very time-consuming, especially for problems of large size. Therefore, it is desirable to obtain a proper direction with less effort. The main idea of the surrogate subgradient method is to obtain such directions, which we will call **surrogate subgradient directions**, without solving all the subproblems.

For the surrogate function of (2.58) the corresponding surrogate subgradient can be introduced as

$$\tilde{g}_x := b - Ax, \tag{2.66}$$

where $x \leq d$ and $x \geq \mathbf{0}$ is fixed but arbitrary. Luh, Wang, and Zhao [1999] have shown that, if the surrogate function is less than $\hat{L}(y^*)$, then the surrogate subgradient is in acute angle with the direction towards y^*; therefore it is a proper direction.

Lemma 2.4. *Given the current point $(y^{(k)}, x^{(k)})$, if the surrogate function is less than the optimal value of the Lagrangian relaxation, i.e.,*

$$\hat{\hat{L}}(y^{(k)}, x^{(k)}) < \hat{L}(y^*) \tag{2.67}$$

then the surrogate subgradient satisfies

$$0 \leq \hat{L}(y^*) - \hat{\hat{L}}(y^{(k)}, x^{(k)}) \leq (y^* - y^{(k)})^t (b - \mathrm{A}x^{(k)}) \tag{2.68}$$

This lemma implies that it is possible to find a proper direction and that the distance between the current multipliers and the optimal y^* can be decreased step by step, if an appropriate step-size is used. Due to the similarity of the surrogate subgradient method to the subgradient method the step-criteria described in Theorem 2.5 are also suited to the former subgradient method. This fact leads to the following surrogate subgradient method, where $\| \ldots \|$ denotes the Euclidean norm.

INPUT: 1. Concave function $\hat{L} : \mathbb{R}^m \mapsto \mathbb{R}$
2. Start vector $y^{(0)}$
3. Termination parameter ϵ_1 and ϵ_2

OUTPUT: An optimal vector y^* and its function value

1: Let $g^{(0)}$ be a subgradient of $\hat{L}$ at $y^{(0)}$ with $x^{(0)} \in X_{(y^{(0)})}$
2: Set $\hat{\hat{g}}^{(0)} := g^{(0)}$ and $\hat{\hat{L}}(y^{(0)}, x^{(0)}) := \hat{L}(y^{(0)})$
3: $k := 0$
4: **while** $\|y^{(k+1)} - y^{(k)}\| \geq \epsilon_1$ and $\|x^{(k+1)} - x^{(k)}\| \geq \epsilon_2$ **do**
5: $\quad y^{(k+1)} := y^{(k)} + s^{(k)}\, \hat{\hat{g}}^{(k)}$
6: $\quad$ Perform an 'approximate optimization' by obtaining $x^{(k+1)}$ such that $\hat{\hat{L}}(y^{(k+1)}, x^{(k+1)}) < \hat{\hat{L}}(y^{(k+1)}, x^{(k)}) = c^t x^{(k)} + y^{(k+1)\,t}(b - Ax^{(k)})$
7: $\quad$ **if** such a $x^{(k+1)}$ can not be obtained **then**
8: $\quad\quad x^{(k+1)} := x^{(k)}$
9: $\quad$ **end if**
10: $\quad k := k + 1$
11: $\quad$ Obtain a surrogate subgradient $\hat{\hat{g}}^{(k)}$
12: **end while**

Algorithm 4: Surrogate Subgradient Method

Luh, Wang, and Zhao [1999] have shown that this algorithm leads to a sequence of $\left\{(y^{(k)}, x^{(k)})\right\}_{k \in \mathbb{N}}$, such that the surrogate multipliers move closer to y^* step by step. Furthermore it can be shown that if $y^{(k+1)} = y^{(k)}$ and $x^{(k+1)} = x^{(k)}$, then

$$\hat{L}(y^{(k)}) = \hat{L}(y^*) \tag{2.69}$$

holds and $x^{(k)}$ is the minimum solution of the subproblem $\hat{L}y^{(k)}$. In this case $(y^{(k)}, x^{(k)})$ would be an optimal solution of the Lagrangian relaxed problem.

The major difference between the surrogate subgradient method and the subgradient method is that the subgradient method must solve all subproblems in order to proceed, whereas the surrogate subgradient method needs only an approximate optimization, which must fulfill

$$\hat{\hat{L}}(y^{(k+1)}, x^{(k+1)}) < \hat{\hat{L}}(y^{(k+1)}, x^{(k)}) = c^t x^{(k)} + y^{(k+1)\,t}(b - Ax^{(k)}). \tag{2.70}$$

Since there are many ways to implement an approximate optimization, this method provides a framework allowing creative variations. Compared with solving all the subproblems, the computational requirements for the approximate optimization to satisfy (2.70) are much smaller. This can be a computational saving for problems with complicated subproblems.

The effort to obtain a direction is one thing; the quality of the direction obtained is another thing. Although it is difficult to quantify the quality of the surrogate subgradient directions, it can be shown that 'zig-zag' phenomena that occur often in the basic subgradient method happen less.

It arises the question whether the surrogate relaxation is worth trying for solving SPP. We have noticed that computational savings originate from treating many complicating subproblems differently. SPP consists of many subproblems, but they are not complicated at all. We do not expect substantial savings from approximate optimization and have therefore not considered this method in this work.

Chapter 3

New Algorithms

3.1 Overview

The result of our literature survey was that for determining good lower bounds for SPP two methods are worth mentioning, namely, the basic subgradient method and the coordinate ascent method. The basic subgradient method is an exact method and many efforts have been made to improve its performance. In this respect we have extracted the bundling technique from the literature and applied it. The resulting algorithm is called the subgradient bundle method. The other class of algorithms are ascent methods. One well-known representative of this class is the coordinate ascent method. Although this method is not exact it turns out that it has a good performance. Motivated by the successful application of the bundle technique to the basic subgradient method we have also used this technique for developing two variations of the coordinate ascent method. These new methods are called **coordinate convex bundle method** and **coordinate bundle method**, respectively. Both methods are variations of the coordinate ascent method because they make use of the information, which are available from the coordinate ascent method. All in all we have four algorithms for determining the lower bounds for SPP:

1. SBM—Subgradient bundle method
2. CAM—Coordinate ascent method
3. CCBM—Convex coordinate bundle method
4. CBM—Coordinate bundle method

In the following we describe the new algorithms CCBM and CBM.

3.2 Coordinate Convex Bundle Method–CCBM

Assuming a point in Lagrangian multiplier space CAM chooses one axis and determines the direction and the step-size in order to get to the next iteration point.

A disadvantage is that this method does not use the available information to go along other axes at the same time. The new approach tries to take this information at least partly into account by constructing a new direction from coordinate directions, which are strictly ascending.

Let y be a non-optimal Lagrangian multiplier vector. For each $i \in 1, \ldots, m$ we denote Δy_i as described in step 2 of Algorithm 3, i.e., Δy_i denotes the step-size along the i^{th} axis to proceed from point y. Let $\Delta y := (\Delta y_1, \ldots, \Delta y_m)^t$ and e_i the i^{th} unit vector. We will show in the next lemma that $\sum_{i=1}^{m} \frac{\Delta y_i}{m}\, e_i$ is an ascent direction at y.

Lemma 3.1. *Let $f : \mathbb{R}^m \mapsto \mathbb{R}$ be some concave function, $y \in \mathbb{R}^m$ a fixed vector and $\Delta y \in \mathbb{R}^m$, such that $f(y + \Delta y_i\, e_i) \geq f(y)$ for all $i \in \{1, \ldots, m\}$, where e_i is the i^{th} unit vector. Then $f(y + \sum_{i=1}^{m} \frac{\Delta y_i}{m}\, e_i) \geq f(y)$ holds, i.e., $\sum_{i=1}^{m} \frac{\Delta y_i}{m}\, e_i$ is an ascent direction of f at y.*

Proof. We proof this by induction over the number of descent directions.

$$
\begin{aligned}
k = 1 \quad &: \quad \text{trival} \\
k - 1 \to k \quad &: \quad f(y + \sum_{i=1}^{k} \tfrac{\Delta y_i}{k}\, e_i) \\
&= f(y + \tfrac{1}{k}\, \Delta y_1\, e_1 + \tfrac{k-1}{k} \sum_{i=2}^{k} \tfrac{\Delta y_i}{k-1}\, e_i) \\
&= f(\tfrac{1}{k}\, (y + \Delta y_1\, e_1) + \tfrac{k-1}{k}\, (y + \sum_{i=2}^{k} \tfrac{\Delta y_i}{k-1}\, e_i)) \\
&= f(\tfrac{1}{k}\, (y + \Delta y_1\, e_1) + (1 - \tfrac{1}{k})\, (y + \sum_{i=2}^{k} \tfrac{\Delta y_i}{k-1}\, e_i)) \\
&\overset{\text{concavity}}{\geq} \tfrac{1}{k}\, f(y + \Delta y_1\, e_1) + (1 - \tfrac{1}{k})\, f(y + \sum_{i=2}^{k} \tfrac{\Delta y_i}{k-1}\, e_i) \\
&\overset{\text{induction}}{\geq} \tfrac{1}{k}\, f(y) + (1 - \tfrac{1}{k})\, f(y) \\
&= f(y)
\end{aligned}
$$

$k = m$ proves the statement. □

By this lemma we have found a method to construct new ascent directions, which are not restricted to the axes any more. If vector Δy has $0 < k < m$ zero entries then we can neglect them and work on $m - k$ instead on m dimensions. This result forms the basis of CCBM. In Figure 3.1 we present the effects on Example 2.1. We start at $y^{(0)} := (8.5, 6)$. The gray lines at this point indicates the two possibilities CAM could use in order to proceed. We take both possibilities into account by proceeding along the black line, which results from the convex combination of both mentioned gray lines. After two further such iterations we reach

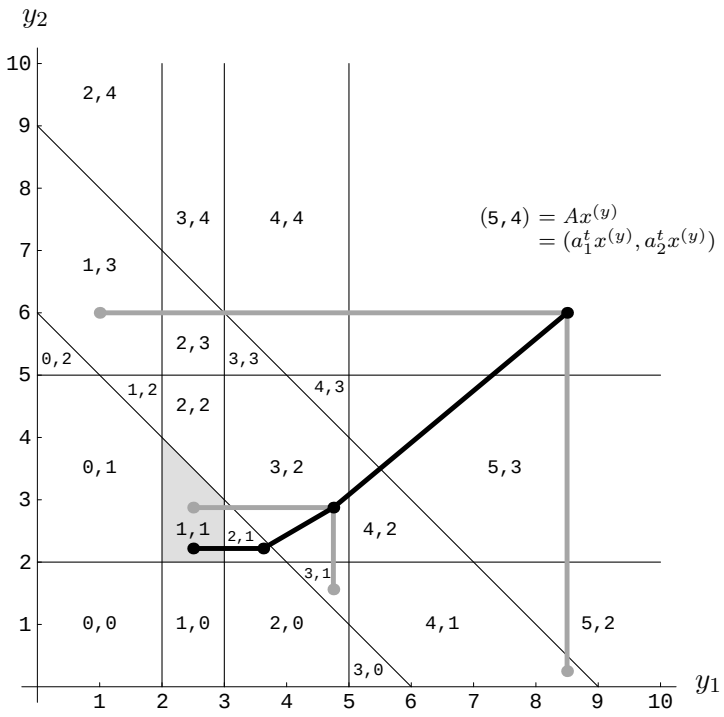

Figure 3.1: CCBM run

the optimal set. Note that due to the bundling idea the ascent directions seem to aim more to the optimal polyhedron than the ascent directions of the CAM.

The CAM counterexample, which we have constructed in the last chapter, also works for CCBM. This means that this method is not convergent, but it is guaranteed that it does not take deteriorating steps. Therefore, **spacer steps** can be inserted, in order to let this method be globally convergent.

3.3 Coordinate Bundle Method–CBM

CCBM has a disadvantage. The rate of improvement of the lower bound is not satisfying enough. This happens because the convex sum of $\Delta y_i e_i$ aims more in the direction of the optimum but shortens the step-size at the same time. Due to this shortening the rate of improvement is low. In order to avoid this we enlarge the step-size while keeping the computed direction by taking vector $\Delta y_i = \sum_{i=1}^{m} y_i \, e_i$ as the new direction. In this case we still get ascent directions, but now it is possible to pass the plateau along these directions. In Figure 3.2 it can be seen that this can happen. We start again at the point $(8.5, 6)$. Note that the black line has now double length, which lets CBM pass the optimal set. However, only one further iteration leads to optimality. In general, it turns out that this change leads empirically to a higher rate of improvement. A disadvantage is that it is possible that in the neighborhood of the optimal set CBM takes too big steps

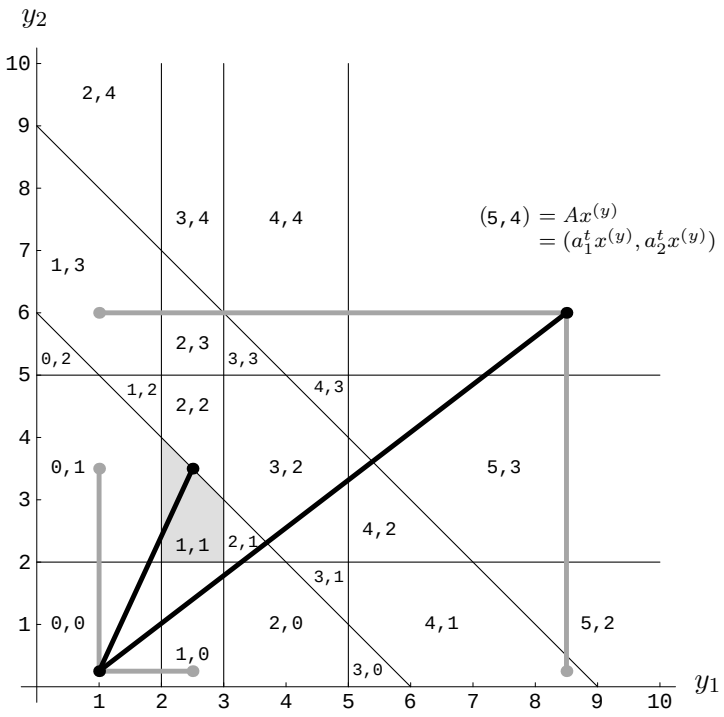

Figure 3.2: CBM run

such that it always passes the optimal set. A solution to this problem would be to stop this method and continue with a convergent one (like the subgradient method) when CBM does not lead to significant improvements any more. However, in most cases this effort is not worthwhile because CBM reaches a good gap of approximately 0.5 percent.

In practice, by these modifications we get a heuristic, which is not monotone ascending, but it leads to satisfying results.

Chapter 4

Computational Results

In this chapter we will present the details of our four optimization methods and their impacts on the performance of our implementations. These details can be understood as improvement techniques, which have been built in step by step, i.e., for each algorithm (SBM, CAM, CCBM, and CBM) we present the basic algorithm and add one improvement after the other. At the end of this chapter these algorithms will be compared with CPLEX dualopt and CPLEX baropt.

4.1 The Test Bed

Our test bed consists of 8 preprocessed large-scale SPP instances arising from real world duty scheduling problems. In the following table the size and the optimal objective values provided by CPLEX dualopt is given. Note that the densities of the problem matrices are on average under 1%.

	Size				
Name	Rows	Cols	NZEs	Density	z^*
ivu41	3570	56867	505195	0.24%	455.93
ivu08	2523	61464	528152	0.34%	347.30
ivu36	1978	40988	493266	0.61%	139.27
ivu37	1978	48138	605873	0.64%	143.56
ivu38	1968	42201	528019	0.64%	116.47
ivu07	1751	59055	590055	0.57%	192.50
ivu28	1295	82422	2375490	2.22%	55.24
ivu06	1177	48058	461360	0.82%	140.18

Table 4.1: Test bed

All instances originate from duty scheduling applications of bus or subway drivers of some European public transport companies. ivu36, ivu37, and ivu38 are in-

stances of a duty scheduling application of subway drivers of a big Italian city. The other instances originate from bus driver duty scheduling of companies located in Germany.

4.2 General Remarks

4.2.1 Data Structure

The matrix of SPP is a zero-one matrix. Moreover, the density of the matrix of most instances is very low. Thus, for the representation of this matrix we have chosen the **row** and **column major format** as ordered and contiguous lists of the non-zero entries (see e.g. CPLEX [1997]). In the column major format we store the row indices of the non-zero entries of each column. This data can be saved in a array, say **ind**[ ]. I.e., **ind**[ ] contains the indices of the non-zero entries of the first column, then the indices of the non-zero entries of the second column, and so on. In order to locate the data of every single column we need two additional arrays, say **beg**[ ] and **cnt**[ ]. Hence, the location of the data of a particular column in the **ind**[ ] array works as follows. **cnt**[**j**] gives the number of non-zero entries in column **j**, and **beg**[**j**] denotes the starting index for data of this column in the array **ind**[ ].

4.2.2 Initialization

As a starting vector for all algorithms we can choose $y^{(0)} = \mathbf{0}$, which yields a lower bound $L(\mathbf{0}) = 0$, but instead of the zero-vector we can also determine $y^{(0)}$ as follows. We compute the average cost of every column by dividing the cost of a column by the number of its non-zero entries. For every i we choose as $y_i^{(0)}$ the lowest average cost of the support of i. This idea was also applied by Caprara, Fischetti, and Toth [1996].

In Figure 4.1 we illustrate the proportion between the initial lower bound and the optimum, where the bright-gray bars represents the optimal objective value as 100% and the dark-gray bars indicates the proportion of the initial lower bound to the optimal objective value. Note that the objective value of the initial start vector y_0 is on average about 75% of the optimum.

4.3 Implementation Details for SBM

4.3.1 Determining a Subgradient

In this section we describe how we determine a subgradient for the Lagrangian relaxation of SPP. As denoted in (2.31) L is defined as

$$L(y) = \min_{\mathbf{0} \leq x \leq \mathbf{1}} (c^t - y^t A)x + y^t \mathbf{1}.$$

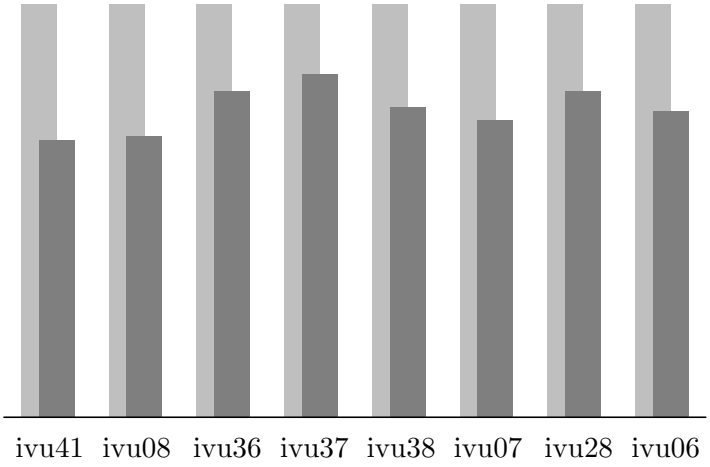

Figure 4.1: Quality of start vector $y^{(0)}$

We have shown in (2.32) that a solution to $L(y)$ can be easily obtained for each y by $x^{(y)}$ whose entries are defined by

$$x_j^{(y)} := \begin{cases} 0 & \text{if } \overline{c}_j > 0 \\ 0 \text{ or } 1 & \text{if } \overline{c}_j = 0 \quad \forall j \\ 1 & \text{if } \overline{c}_j < 0. \end{cases}$$

We do not calculate the whole subdifferential because in the worst case this calculation has a complexity of 2^n operations, where n could get very large. Therefore, we have decided to determine a subgradient out of the subdifferential as follows

$$x_j^{(y)} := \begin{cases} 0 & \text{if } \overline{c}_j > 0 \\ 0 \text{ or } 1 \text{ (randomly chosen)} & \text{if } \overline{c}_j = 0 \quad \forall j \\ 1 & \text{if } \overline{c}_j < 0. \end{cases} \tag{4.1}$$

4.3.2 Updating the Step-size

A step-size update formula suggested by Held and Karp [1971] is defined by

$$s^{(k+1)} := \alpha \, \frac{\overline{L} - L(y^{(k)})}{\|g^{(k)}\|}, \tag{4.2}$$

where $\overline{L}$ is an upper bound for the maximum of L and α is a step-size parameter.

Actually, we do not know an upper bound for L. Therefore, we use $L(y^{(0)})$ (see Section 4.2.2) and guess an upper bound by $\overline{L} := 2\, L(y^{(0)})$. If our guess was too small then we increase $\overline{L}$ by $L(y^{(0)})$. We repeat this whenever we get a lower bound, which is greater than the current value of $\overline{L}$. If the primal problem is feasible and bounded then the relaxed problem is bounded, too. I.e., in this case $\overline{L}$

can not increase to infinity. Moreover, Theorem 2.4 implies $\|g^{(k)}\| \geq 1$ as long the corresponding multipliers are not optimal. This means that the step-size $s^{(k+1)}$ has an upper bound unless optimality is reached. In addition it holds that the step-size gets smaller as the method approaches the set of optimal Lagrangian multiplier vectors. However, if for whatever reason the estimation of $\overline{L}$ is too large, the step-sizes are too large as well. Therefore, we need a factor, which compensates this.

For this task α is used. The classical approach of Held and Karp [1971] is $\alpha := 0.5\alpha$ for every p iterations. Our computational experiments revealed that $\alpha := 0.8\alpha$ for every two consecutive negative improvements of the lower bound is more promising. Note that this step-size update formula does not necessarily fulfill every step-size criterion of Theorem 2.5, but this heuristic leads to more satisfying results. In Figure 4.2 the effects of the settings 0.5 and 0.8 compared.

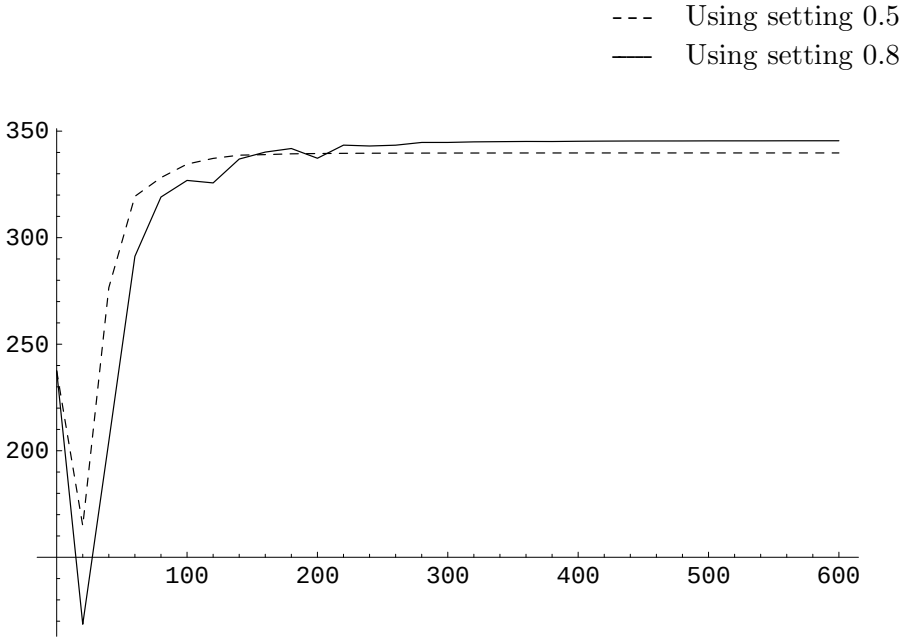

Figure 4.2: SBM—Using step-size parameter setting 0.5 and 0.8 for ivu08

Both settings lead to a similar practical behavior of SBM. Nevertheless, there is an important difference. The settings 0.5 and 0.8 comply with shortening-rates of α. 0.5 is a higher shortening-rate than 0.8, which means that the step-size can be shortened faster. This leads to a faster approximation of the step-sizes. A disadvantage is that in the long run the step-sizes get too small, which means that the rate of improvement slows down quickly. 0.8 has a converse character. Since 0.8 is a lower shortening-rate with this setting the performance of SBM is worse than that using 0.5. But in the long run the rate of improvement is significantly higher than that of 0.5. Due to our focus on high lower bounds for SPP we will

use the latter setting.

From Table 4.2 it can be infered, that for the tested instances setting 0.8 leads on average to an improvement of 2.31%.

	Improvement		
Name	Setting 0.5	Setting 0.8	in %
ivu41	442.44	450.11	1.73%
ivu08	339.77	345.49	1.68%
ivu36	135.54	137.83	1.69%
ivu37	138.22	141.67	2.49%
ivu38	110.63	113.34	2.45%
ivu07	181.76	188.20	3.55%
ivu28	50.01	51.29	2.55%
ivu06	135.49	138.64	2.33%

Table 4.2: SBM—Step-size

4.3.3 Stabilization

Subgradient methods need many iterations to produce good step-sizes because they have to guess them. They are too large after the first iterations, therefore the first steps of SBM often lead to deteriorations of the objective value. In order to compensate this, we bound the step-sizes as follows:

$$s^{(k+1)} := \max\left[\min\left[s^{(k+1)}, cap\right], -cap\right], \tag{4.3}$$

where $cap > 0$ is a parameter, which has to be determined for different classes of problems. For our instances the best value for *cap* is 0.01. We will explain in Section 4.6.2 how a good *cap* could be determined. This general technique is called stabilization and was proposed by Desrosiers, Du Merle, Villeneuve, and Hansen [1997].

In Figure 4.3 we show the positive effect of this technique for ivu38. If we do not use stabilization the first iterations leads to a strong deterioration of the objective value. By using this technique the proceeding fluctuates less and the deterioration is less extreme. Furthermore, stabilization yields higher improvements in the long run.

This effect could be observed for nearly all instances of our test bed (see Table 4.3). On average an improvement of 0.40% can be obtained by using stabilization. Therefore, this technique is worth being built in.

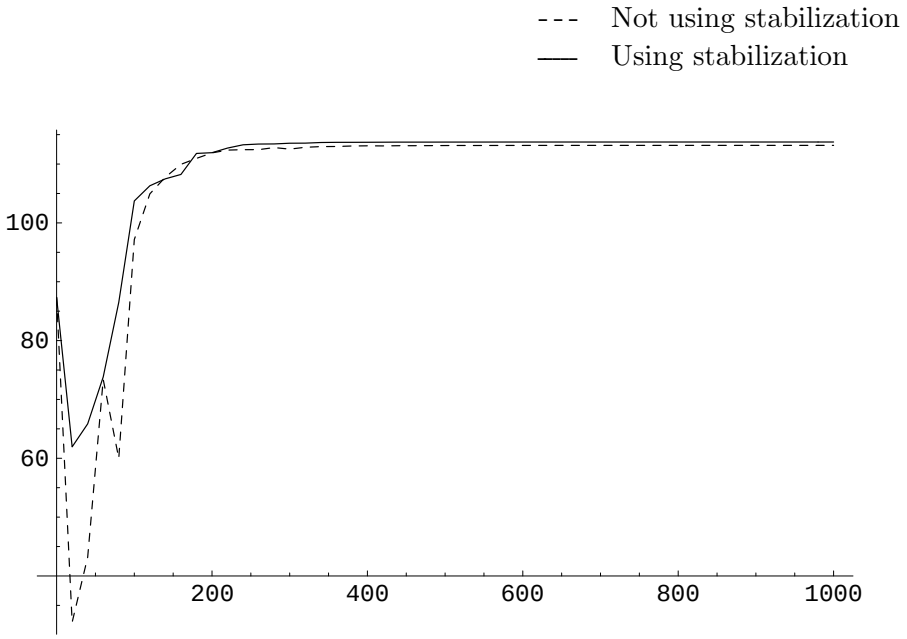

Figure 4.3: SBM—Stabilization for ivu38

	Improvement		
Name	Before	After	in %
ivu41	449.06	450.33	0.28%
ivu08	345.78	345.78	0.00%
ivu36	137.91	138.41	0.36%
ivu37	141.65	142.04	0.28%
ivu38	113.18	113.75	0.51%
ivu07	188.46	191.19	1.45%
ivu28	51.38	51.17	-0.40%
ivu06	138.38	139.33	0.68%

Table 4.3: SBM—Stabilization

4.3.4 Diversification

It could happen that the step-sizes get too small. In addition to the reaching of optimality this occurs when we get in in highly degenerate locations, where most of the subgradients yield directions that do not lead to improvements. For such cases a heuristical approach called **diversification** (see Aarts and Lenstra [1997]) suggests to make big steps in order to escape from such a neighborhood. These

steps do not have to be ascending. Through this technique we try to approach the global maximum from another, less explored location. I.e., we increase the step-size appropriately and enforce the escape, when the convergence gets too slow or the step-sizes too small.

In Figure 4.4 it can be seen that for ivu07 the two negative peaks are initiated by two diversification steps, which have led to a strong decline of the objective value. However, it is important to note that in the end these steps yield a further improvement.

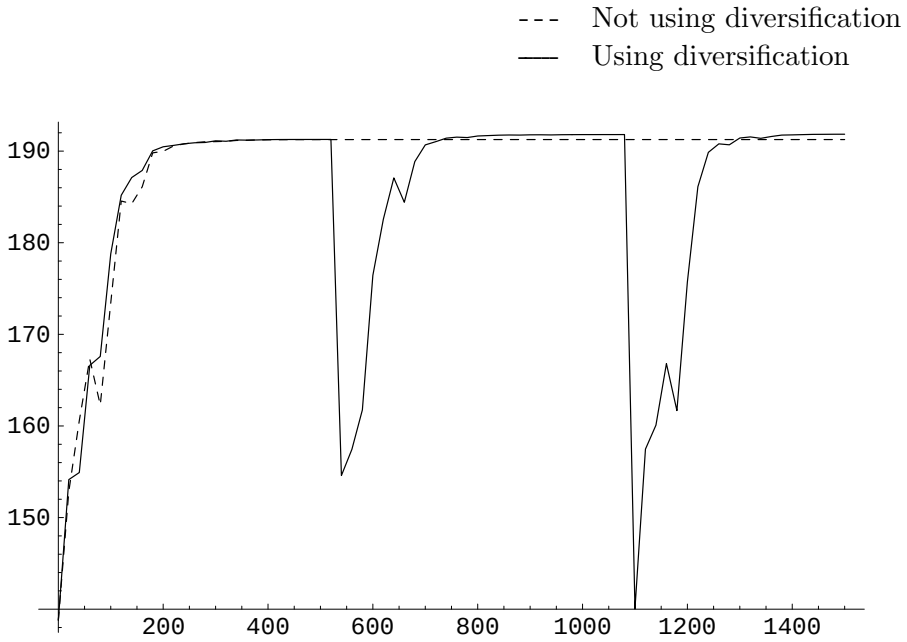

Figure 4.4: SBM—Diversification for ivu07

From Table 4.4 it can be infered that for our test bed diversification leads to an average improvement of 0.53%.

	Improvement		
Name	Before	After	in %
ivu41	449.27	449.27	0.00%
ivu08	346.02	346.81	0.23%
ivu36	138.33	138.77	0.32%
ivu37	142.26	142.64	0.27%
ivu38	114.14	115.28	0.99%
ivu07	191.26	191.85	0.31%
ivu28	51.47	52.54	2.09%
ivu06	139.74	139.83	0.06%

Table 4.4: SBM—Diversification

The Algorithm Our consideration leads us to the following SBM algorithm.

INPUT: L

OUTPUT: An optimal or a 'nearly' optimal vector y^* and its function value Ł(y^*)

for all $i \in \{1, \ldots, m\}$ **do**

$y_i^{(0)} := \min_{j \in supp(i)} \frac{c_j}{\|A_j\|^2}$, where A_j denotes j^{th} column of matrix A

end for

$\overline{c}^{(0)} := c - y^{(0)\,t} A$

for all $j \in \{1, \ldots, n\}$ **do**

$$x_j^{(0)} := \begin{cases} 0 & \text{if } \overline{c}_j^{(0)} > 0 \\ 0 \text{ or } 1 \text{ (randomly chosen)} & \text{if } \overline{c}_j^{(0)} = 0 \\ 1 & \text{if } \overline{c}_j^{(0)} < 0 \end{cases}$$

end for

$g^{(0)} := \mathbf{1} - Ax^{(0)}$

$\tilde{g}^{(0)} := g^{(-1)} := g^{(-2)} := g^{(-3)} := g^{(0)}$

Set $\alpha^{(0)} := 1$ and $\overline{L} := 2\, L(y^{(0)})$

$k := 0$

while $g^{(k)} \neq \mathbf{0}$ **do**

for all $j \in \{1, \ldots, n\}$ **do**

$$x_j^{(k)} := \begin{cases} 0 & \text{if } \overline{c}_j^{(k)} > 0 \\ 0 \text{ or } 1 \text{ (randomly chosen)} & \text{if } \overline{c}_j^{(k)} = 0 \\ 1 & \text{if } \overline{c}_j^{(k)} < 0 \end{cases}$$

end for

$g^{(k)} := \mathbf{1} - Ax^{(k)}$

$\tilde{g}^{(k)} := 0.6\, g^{(k)} + 0.2\, g^{(k-1)} + 0.1\, g^{(k-2)} + 0.1\, g^{(k-3)}$

$y^{(k+1)} := y^{(k)} + \alpha\, \frac{\overline{L} - L(y^{(k)})}{\|\tilde{g}^{(k)}\|^2}\, \tilde{g}^{(k)}$

$\overline{c}^{(k+1)} := c - y^{(k+1)\,t} A$

$L(y^{(k+1)}) := \sum_{j\,:\,\overline{c}_j^{(k+1)} < 0} \overline{c}_j^{(k+1)} + \sum_{i=1,\ldots,n} y_i^{(k+1)}$

if $L(y^{(k+1)}) < 0$ **then**

$y^{(k+1)} := y^{(k)} + \alpha\, \frac{\overline{L} - L(y^{(k)})}{\|g^{(k)}\|^2}\, g^{(k)}$

$\overline{c}^{(k+1)} := c - y^{(k+1)\,t} A$

$L(y^{(k+1)}) := \sum_{j\,:\,\overline{c}_j^{(k+1)} < 0} \overline{c}_j^{(k+1)} + \sum_{i=1,\ldots,n} y_i^{(k+1)}$

end if

Update α and $\overline{L}$ as described in Section 4.3.2 and 4.3.4

$k := k + 1$

end while

Return $y^{(k)}$ and $L(y^{(k)})$

Algorithm 5: The SBM Algorithm

Let q denote the number of non-zero entries of matrix A. The most expensive operations have a running time of $O(q)$. Steps 1-3, 4, 8, 17, 20 and 24 have this total complexity. The overall running time is therefore $O(kq)$, where k is the number of iterations.

4.4 Implementation Details for CAM

4.4.1 Stabilization

The update formula of Δy is

$$\Delta y := r_1 + p\,(r_2 - r_1), \tag{4.4}$$

where Wedelin [1995] suggests $p = 0.5$. However, in most instances CAM does not proceed that well especially during the first iterations, therefore we use (like for SBM) the stabilization technique

$$\Delta y := \max\left[\min\left[r_1 + 0.5\,(r_2 - r_1), cap\right], -cap\right] \tag{4.5}$$

where $cap > 0$. As in the case of SBM it turns out that 0.01 is a good setting for *cap*. These boundaries work against too big steps along the axes, which mainly happen in the first iterations. We will explain in Section 4.6.2 how a good *cap* can be determined.

In Figure 4.5 it can be seen that for ivu37 both methods start from the same initial point, but without the stabilization technique CAM deteriorates and does not come up again. It cannot even compensate the decline for the next 600 iterations. Thus the best lower bound is only 118.71, whereas CAM with stabilization yields in the same time approximately 138, which is an improvement of about 15%.

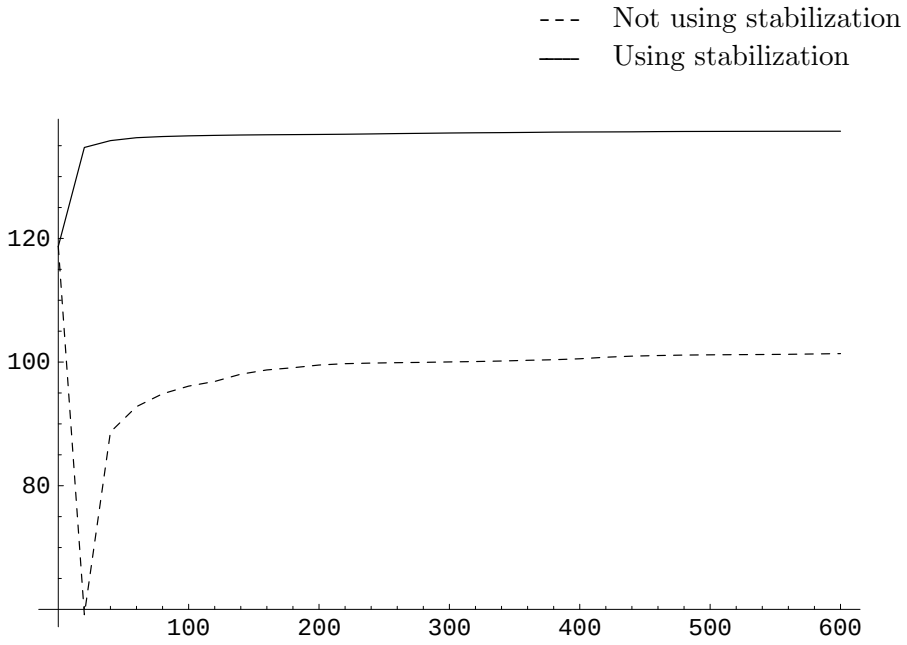

Figure 4.5: CAM—Stabilization for ivu37

The effect of this technique on all instances of our test bed is summarized in Table 4.5. On average the improvement is 25.20%.

	Improvement		
Name	Before	After	in %
ivu41	305.11	432.40	41.72%
ivu08	237.52	341.49	43.77%
ivu36	125.11	134.16	7.23%
ivu37	118.71	137.34	15.70%
ivu38	87.34	109.15	24.97%
ivu07	138.79	188.39	35.73%
ivu28	49.43	49.81	0.75%
ivu06	104.40	137.53	31.74%

Table 4.5: CAM—Stabilization

4.4.2 Random Order

Our computational experiments reveal that it is more effective to go through all coordinate directions in a random order rather than in a fixed or numerical order.

In Figure 4.6 the findings for ivu36 are illustrated. The influence of the random

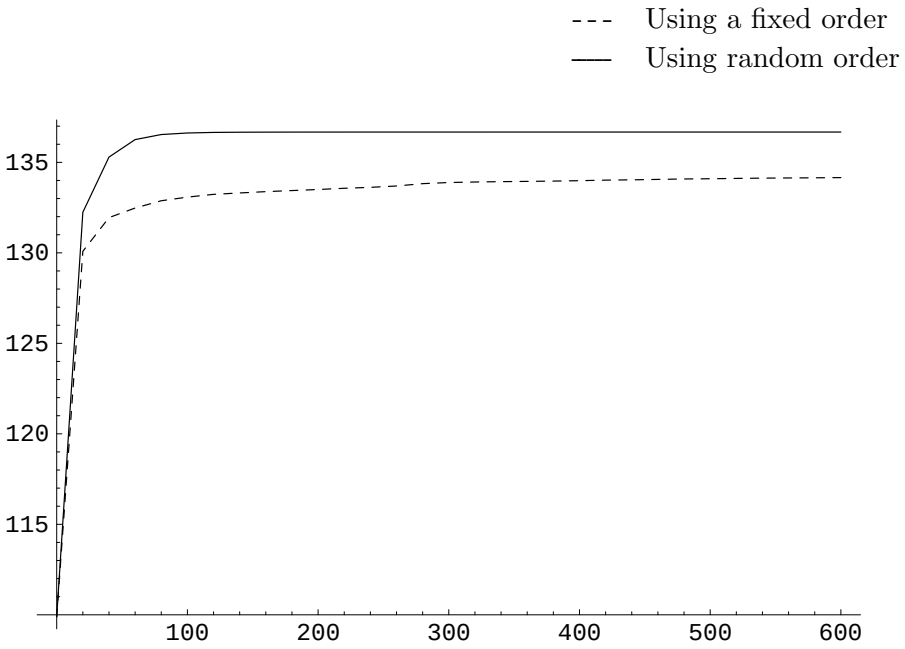

Figure 4.6: CAM—Random order for ivu36

order technique starts right at the beginning and leads to higher improvements.

Table 4.6 presents the improvement rates for the other instances. On average we reach an improvement rate of 2.19%.

	Improvement		
Name	Before	After	in %
ivu41	432.40	448.67	3.76%
ivu08	341.49	343.28	0.52%
ivu36	134.16	136.68	1.88%
ivu37	137.34	140.83	2.54%
ivu38	109.15	111.80	2.43%
ivu07	188.39	191.97	1.90%
ivu28	49.81	51.38	3.16%
ivu06	137.53	139.40	1.36%

Table 4.6: CAM—Random order

Remark 4.1. In general it has not been shown yet, how the efficiency of CAM

correlates with the choice of the coordinate directions. This choice is a degree of freedom, which should be tuned to the treated class of problems.

4.4.3 Updating the Step-size

The update formula of Δy with stabilization is

$$\Delta y := \max\left[\min\left[r_1 + p\,(r_2 - r_1), cap\right], -cap\right]. \tag{4.6}$$

With $p := 0.5$ we get

$$\Delta y := \max\left[\min\left[0.5\,(r_1 + r_2), cap\right], -cap\right], \tag{4.7}$$

but it turns out that

$$\Delta y := \max\left[\min\left[0.95\,(r_1 + r_2), cap\right], -cap\right] \tag{4.8}$$

works better. Note that $\Delta y := 0.95\,(r_1 + r_2)$ would geometrically mean to pass the optimal polyhedron along the considered axis, if $(r_1 + r_2) \neq 0$. In the case of equality $\Delta y := 0.95\,(r_1 + r_2) = 0$ means that we stay at the current location, because we are already on the optimal plateau along the considered axis.

Figure 4.7 shows that for ivu06 this technique leads to a better result. Note that

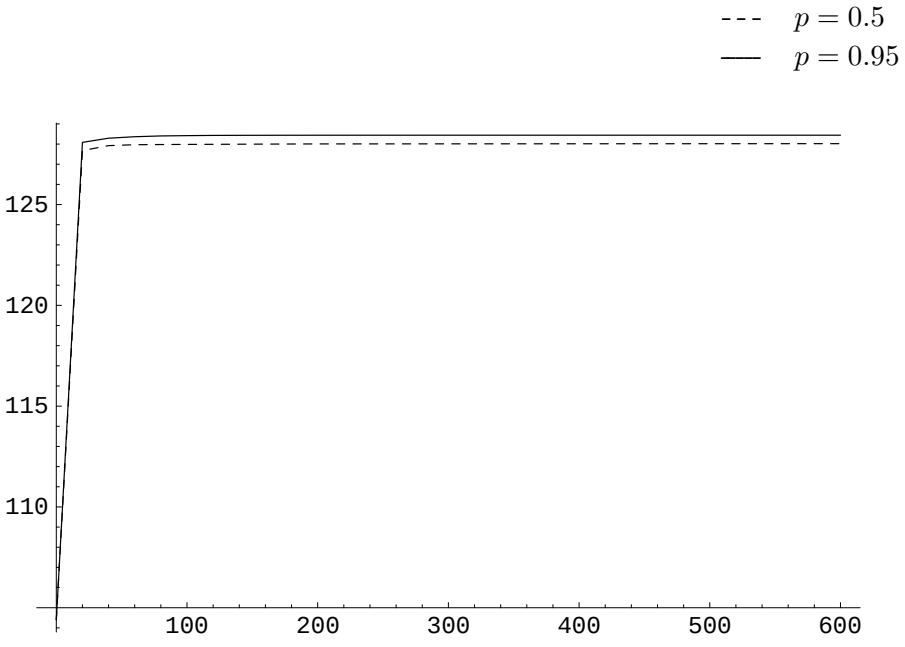

Figure 4.7: CAM—Using step-size parameter setting 0.5 and 0.95 for ivu06

in the first iterations the performances of both variants are similar because of the stabilization technique. The influence of p begins when $|\Delta y_i|$ gets smaller than

cap, which happens in the neighborhood of the optimal set.

The effect of this technique can be infered from Table 4.7. The average improvement is 0.10%.

	Improvement		
Name	$p = 0.5$	$p = 0.95$	in %
ivu41	427.80	427.40	-0.09%
ivu08	327.03	327.75	0.22%
ivu36	132.31	132.12	-0.14%
ivu37	135.69	135.76	0.05%
ivu38	107.17	107.47	0.28%
ivu07	175.39	176.20	0.46%
ivu28	49.43	49.29	-0.28%
ivu06	128.02	128.44	0.33%

Table 4.7: CAM—Step-size parameter

4.4.4 Spacer Steps

We use spacer steps whenever $\Delta y = \mathbf{0}$. This modification guarantees that CAM does not get stuck. Furthermore, we use this technique if too many small improvements occur successively, in order to avoid slow convergences as described in Section 2.2.3.2. Spacer steps could be also seen as a diversification technique: they lead to unexplored locations, from which the optimum could be reached better.

In the following figure it can be observed, how diversification and spacer steps improves the convergence. It can be seen that the diversification technique causes two negative peaks, but it leads to higher lower bounds in the long run.

From Table 4.8 the improvement of this technique for all tested instances can be infered. On average the improvement rate is 0.88%.

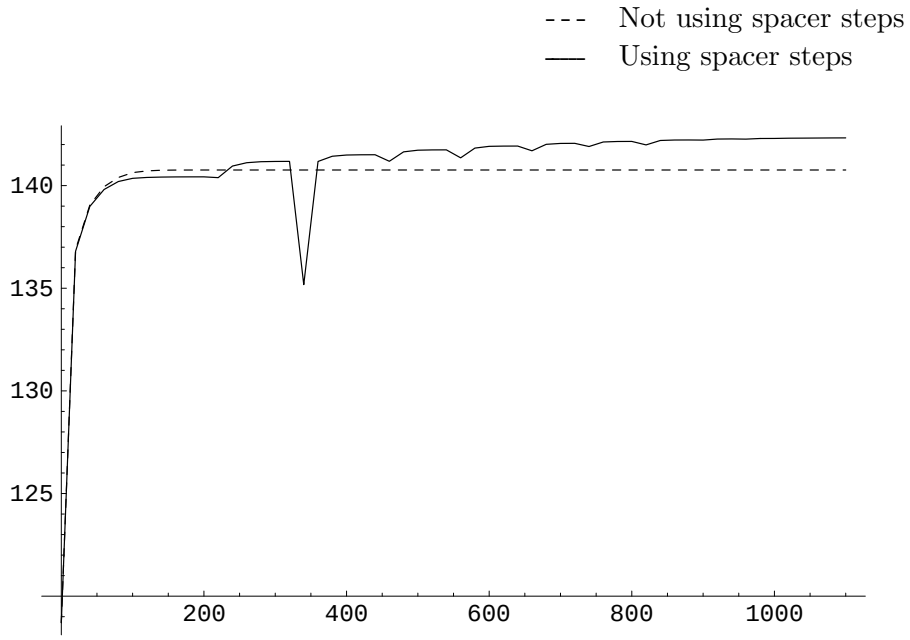

Figure 4.8: CAM—Spacer steps for ivu37

	Improvement		
Name	Before	After	in %
ivu41	450.79	451.71	0.20%
ivu08	342.86	345.96	0.90%
ivu36	136.92	138.17	0.92%
ivu37	140.76	142.33	1.11%
ivu38	111.97	113.99	1.80%
ivu07	192.17	192.15	-0.01%
ivu28	51.60	52.70	2.12%
ivu06	139.96	140.00	0.03%

Table 4.8: CAM—Spacer steps

4.4.5 Lazy Check

In contrast to SBM it is not necessary for CAM to evaluate the lower bound in order to proceed, but this method needs these evaluations to check the development of the lower bound. Therefore, it is sufficient to compute it, say, every 20 iterations. This saves a lot of computation time, because the calculation of the lower bound needs $O(n)$ operations where n could get very large for most

practical instances.

The Algorithm We will present the algorithm of our implementation of CAM.

INPUT: L
OUTPUT: An optimal or a 'nearly' optimal vector y^* and its function value $L(y^*)$

1: Steps 1 until 4 from SBM (see Algorithm 5)
2: $k := -1$
3: **while** the last iterations are satisfying **do**
4: $k := k + 1$
5: **for all** $i \in \{1, \ldots, m\}$ in a random order **do**
6: Let r_1 and r_2 be the lowest and second lowest reduced cost on the support of row i
7: Define $\Delta y := \max\left[\min\left[0.95\ (r_1 + r_2), cap\right], -cap\right]$
8: $y_i^{(k+1)} := y_i^{(k)} + \Delta y$
9: **for all** $j \in \text{supp}(i)$ **do**
10: $\overline{c}_j^{(k+1)} := \overline{c}_j^{(k)} - \Delta y$
11: **end for**
12: **end for**
13: **if** Δy was zero for all i **or** the last improvements were too small **then**
14: Determine a subgradient $g^{(k)}$ and let $\alpha := 0.8\alpha$
15: $y^{(k+1)} := y^{(k)} + \alpha \dfrac{g^{(k)}}{\|g^{(k)}\|}$
16: $\overline{c}_j^{(k+1)} := \overline{c}_j^{(k)} - \alpha \dfrac{g^{(k)}}{\|g^{(k)}\|}$
17: **end if**
18: **if** $k \bmod 20 = 0$ **then**
19: $L(y^{(k+1)}) := \sum_{j\,:\,\overline{c}_j^{(k+1)} < 0} \overline{c}_j^{(k+1)} + \sum_{i=1,\ldots,n} y_i^{(k+1)}$
20: **if** $L(y^{(k+1)})$ is the best found lower bound **then**
21: $y^* := y^{(k+1)}$
22: **end if**
23: **end if**
24: **end while**
25: Return y^* and $L(y^*)$

Algorithm 6: The CAM Algorithm

As for SBM, the most expensive operations of this method have a running time of $O(q)$. Steps 1, 5-6, 9-11 and 14 have this total complexity. The total running time is $O(kq)$.

4.5 Implementation Details for CCBM

4.5.1 Choosing the Size of the Packages

A degree of freedom is the number of rows, which can be considered in order to construct a bundle-direction. Let *packagesize* be the considered number of rows represented in percent, e.g., 0.20% corresponds to $packagesize := 0.002\,m$. If *packagesize* is less than 1, then we simply round *packagesize* up. Thus, in order to evaluate CCBM only for different settings of *packagesize*, ivu28 and ivu06 have no entries in the column 0.04%.

	packagesize					
Name	0.04%	0.12%	0.20%	10%	50%	100%
ivu41	449.81	427.93	427.56	374.01	327.47	317.34
ivu08	345.83	322.76	321.99	293.63	257.69	249.08
ivu36	138.27	131.51	131.30	126.51	120.30	117.31
ivu37	142.28	135.03	134.60	131.54	126.05	123.98
ivu38	113.59	107.69	109.11	103.72	97.53	94.35
ivu07	192.19	173.17	172.69	168.49	155.63	149.31
ivu28	——	52.59	48.93	48.56	48.04	47.53
ivu06	——	139.54	126.61	123.89	115.72	112.12

Table 4.9: CCBM—Lower bounds for different settings of *packagesize*

Note that for $packagesize = 1$ CCBM corresponds to CAM, therefore from Table 4.9 it can be infered that CAM is in fact better than CCBM. Nevertheless, we have mentioned this method because a heuristical approach of CCBM seems to be more promising than CAM; but first we will give a description of the CCBM algorithm.

The Algorithm The only algorithmic differences between CCBM and CAM are the frequency of updates of the reduced cost and the way of updating the multipliers. If we update the reduced cost after every update of y_i, then we get CAM. However, if we do these updates every *packagesize* (> 1) updates of y_i then we get CCBM. This algorithm can be described as follows:

INPUT: L
OUTPUT: An optimal or a 'nearly' optimal vector y^* and its function value Ł(y^*)

Steps 1 until 4 from SBM (see Algorithm 5)
$k := -1$
while the last iterations are satisfying **do**
  $k := k + 1$
  Randomize the order of the m rows to $\{i_1, \ldots, i_m\}$
  for all $l \in \{1, \ldots, \frac{m}{packagesize}\}$ **do**
    $nonzero := 0$
    for all $i \in \{i_l, \ldots, i_{l+packagesize}\}$ **do**
      Let r_1 and r_2 be the lowest and second lowest reduced cost on the support of row i
      Define $\Delta y_i := \max\left[\min\left[0.95\ (r_1 + r_2), cap\right], -cap\right]$
      if $|\Delta y_i| > 0$ **then**
        $nonzero := nonzero + 1$
      end if
    end for
    for all $i \in \{i_l, \ldots, i_{l+packagesize}\}$ **do**
      $y_i^{(k+1)} := y_i^{(k)} + \Delta y / \max[nonzero, 1]$
      for all $j \in \mathrm{supp}(i)$ **do**
        $\overline{c}_j^{(k+1)} := \overline{c}_j^{(k)} - \Delta y / \max[nonzero, 1]$
      end for
    end for
  end for
  if Δy was zero for all i **or** the last improvements are too small **then**
    Determine a subgradient $g^{(k)}$ and let $\alpha := 0.8\alpha$
    $y^{(k+1)} := y^{(k)} + \alpha \dfrac{g^{(k)}}{\|g^{(k)}\|}$
    $\overline{c}_j^{(k+1)} := \overline{c}_j^{(k)} - \alpha \dfrac{g^{(k)}}{\|g^{(k)}\|}$
  end if
  if $k \bmod 20 = 0$ **then**
    $L(y^{(k+1)}) := \sum_{j\,:\,\overline{c}_j^{(k+1)} < 0} \overline{c}_j^{(k+1)} + \sum_{i=1,\ldots,n} y_i^{(k+1)}$
    if $L(y^{(k+1)})$ is the best found lower bound **then**
      $y^* := y^{(k+1)}$
    end if
  end if
end while
Give out y^* and $L(y^*)$

Algorithm 7: The CCBM Algorithm

Because CCBM is a variation of CAM the most expensive operations of CCBM

have a running time of $O(q)$. Steps 1, 8-9, 17-19 and 25 have this total complexity. The overall running time is $O(kq)$.

4.6 Implementation Details for CBM

4.6.1 Relaxing the Convex Combination

We mentioned already in the previous chapter the reason for the unsatisfying performance of CCBM: the convex sum of the considered vectors leads to a bundle vector, which turns out to be too short in order to proceed well, i.e., we get a slow convergence. As described in Section 3.3 we try to overcome this obstacle by taking just the sum of the considered vectors instead of a convex combination. In the following table we can see that except for ivu07 and ivu06 the peak is in the column 0.20%, i.e., for 0.20% of m rows CBM performs better than CAM.

	packagesize					
Name	0.04%	0.12%	0.20%	10%	50%	100%
ivu41	451.49	451.70	452.32	451.82	440.35	379.45
ivu08	342.96	343.88	346.20	341.62	328.20	281.55
ivu36	136.91	136.93	138.53	134.60	122.51	112.45
ivu37	140.68	141.12	142.76	138.00	119.29	119.37
ivu38	112.10	112.51	114.58	111.30	93.22	89.44
ivu07	192.20	191.57	191.25	187.42	174.21	147.95
ivu28	——	52.69	53.98	49.97	43.77	43.77
ivu06	——	140.02	139.33	136.96	127.63	106.93

Table 4.10: CBM—Lower bounds for different settings of *packagesize*

The Algorithm CBM has the following algorithmic structure.

INPUT: L
OUTPUT: An optimal or a 'nearly' optimal vector y^* and its function value $L(y^*)$

1: Steps 1 until 4 from SBM (see Algorithm 5)
2: $k := -1$
3: **while** the last iterations were satisfying **do**
4: $k := k + 1$
5: Randomize the order of the m rows to $\{i_1, \ldots, i_m\}$
6: **for all** $l \in \{1, \ldots, \frac{m}{packagesize}\}$ **do**
7: **for all** $i \in \{i_l, \ldots, i_{l+packagesize}\}$ **do**
8: Let r_1 and r_2 be the lowest and second lowest reduced cost on the support of row i
9: Define $\Delta y_i := \max\left[\min\left[0.95\ (r_1 + r_2), cap\right], -cap\right]$
10: **end for**
11: **for all** $i \in \{i_l, \ldots, i_{l+packagesize}\}$ **do**
12: $y_i^{(k+1)} := y_i^{(k)} + \Delta y$
13: **for all** $j \in \mathrm{supp}(i)$ **do**
14: $\overline{c}_j^{(k+1)} := \overline{c}_j^{(k)} - \Delta y$
15: **end for**
16: **end for**
17: **end for**
18: **if** Δy was zero for all i **or** the last improvements were too small **then**
19: Determine a subgradient $g^{(k)}$ and $\alpha := 0.8\alpha$
20: $y^{(k+1)} := y^{(k)} + \alpha \dfrac{g^{(k)}}{\|g^{(k)}\|}$
21: $\overline{c}_j^{(k+1)} := \overline{c}_j^{(k)} - \alpha \dfrac{g^{(k)}}{\|g^{(k)}\|}$
22: **end if**
23: **if** $k \bmod 20 = 0$ **then**
24: $L(y^{(k+1)}) := \sum_{j\,:\,\overline{c}_j^{(k+1)} < 0} \overline{c}_j^{(k+1)} + \sum_{i=1,\ldots,n} y_i^{(k+1)}$
25: **if** $L(y^{(k+1)})$ is the best found lower bound **then**
26: $y^* := y^{(k+1)}$
27: **end if**
28: **end if**
29: **end while**
30: Give out y^* and $L(y^*)$

Algorithm 8: The CBM Algorithm

Due to the similarity of CCBM and CBM the total running time of CBM is $O(kq)$.

4.6.2 Determining a Good Stabilization Parameter

CBM has been used to find a good *cap* for the Δy update of all methods proposed in this work. We set *cap* to a certain value and start the CBM algorithm for some instances of a problem class. After 40 iterations we stop the algorithm. We repeat this procedure for different values of *cap*. The value of *cap*, which leads to the highest improvement will be taken for *cap*. For nearly all of our instances $cap := 0.01$ turned out to be the best choice (see the following table).

	cap					
Name	100	10	1	0.1	0.01	0.001
ivu41	305.11	305.11	305.11	440.46	429.50	372.77
ivu08	237.52	237.52	237.52	324.16	334.86	284.56
ivu36	109.89	109.89	109.89	124.73	135.04	125.52
ivu37	118.71	118.71	118.71	127.06	138.93	130.89
ivu38	87.34	87.34	87.34	101.08	110.60	102.90
ivu07	138.79	138.79	138.79	168.98	183.57	165.31
ivu28	43.77	43.77	43.77	44.82	51.10	49.28
ivu06	104.40	104.40	104.40	123.72	134.73	119.89

Table 4.11: Determining a good *cap*

Note that the first three entries of each row have the same value. This happened because for *cap* greater than 1 CBM makes negative improvements such that the initial lower bound stays always as the best one. Therefore, the entries are all the same, but for *cap* less than 1 we can observe that CBM works better, which is indicated by higher lower bounds. Note that the best choice of *cap* depends on the class of problems.

4.7 Comparison

In this section we will compare the six methods:

1. CPLEX dualopt—Dual simplex method implemented in CPLEX
2. CPLEX baropt—Barrier function method implemented in CPLEX
3. SBM—Subgradient bundle method
4. CAM—Coordinate ascent method
5. CCBM—Coordinate convex bundle method
6. CBM—Coordinate bundle method

The tested instances are ivu41, ivu08, ivu36, ivu37, ivu38, ivu07, ivu28, and ivu06, which originate from duty scheduling applications of different European public transport companies.

The source codes of SBM, CAM, CCBM, and CBM are written in *ANSI C* and have been compiled with the *GNU project C Compiler gcc* with *optimization -O2*. For the dual simplex method and the barrier function method we have fallen back on *CPLEX 6.5.3*. All computations were made on a *SUN UltraSPARC 60 Model 2360*.

In Table 4.12 the performances of the dual simplex method and the barrier function method are described. We have listed the lower bounds and the computation times of both methods. Note that their lower bounds have an accuracy of 10^{-6}. For this reason the entries of the columns 'Gap' are empty. That means that we can consider these bounds as references for the optimal objective value of the instances of our test bed. In contrast to the accuracy the computation times of both methods differ a lot. The barrier function method is much faster than the dual simplex method. This observation supports the fact that interior points methods allow to devise methods of polynomial complexity (see Karmarkar [1984]) while simplex methods are in general exponential.

The performances of the other four algorithms are listed in Table 4.13. Note that SBM, CAM, and CBM have similar performance, whereas CCBM has performed not that well. The explanation for this is that the convex combination of several ascent coordinate directions leads to improvements per iteration, which are too small. By using a simple combination instead (see CBM) we get a better performance. An interesting observation is that on average CBM reaches higher lower bounds than SBM, which is based on an exact method. This means that our heuristical approach is at least as effective as subgradient bundle methods. If we were to rank all six methods, CPLEX would be one followed by CBM, SBM, CAM, and CCBM.

Another aspect is the running time for reaching a fixed gap. We have drawn such comparisons for the gaps of 1%, 2%, and 5%. The results are listed in Table 4.14 and Table 4.15. If a method has not reached a gap the corresponding entry is empty. As we have already noticed CCBM has a bad performance. This method has even problems to reach the 5% margin. If we compare all methods the ranking would be CBM, CAM, SBM, CPLEX baropt, CPLEX dualopt, and CCBM. This holds for nearly all gaps and instances. In few cases (compare the 1% columns) CPLEX baropt has a better performance than CBM. In this case CBM needs more computation time than CPLEX baropt in order to reach tighter gaps. But, it can be also noticed that the number of rows seems to have a stronger influence on the running of CPLEX baropt than on that of CBM. I.e., the higher the number of rows of an instance the better CBM performs in comparison with CPLEX baropt.

As a result we can state that the performance of CBM and partly that of CAM seem to be better than that of SBM, which is derived from an exact method, and that of CPLEX, which can be considered as the state-of-the-art. Applications, which profit from this discovery, are, for example, duty scheduling applications, which use column generation techniques. Such applications need to solve Lagrangian relaxations of large-scale SPP iteratively, where the Lagrangian multipliers of these problems are needed in order to proceed. These multipliers do not necessarily have to be optimal for column generation. This fact supports the use of CBM instead of CPLEX baropt especially for large-scale problems.

Name	Size				CPLEX dualopt			CPLEX baropt		
	Rows	Cols	NZEs	Density	z^*	Gap	sec.	z^*	Gap	sec.
ivu41	3570	56867	505195	0.25%	455.93	——	641	455.93	——	256
ivu08	2523	61464	528152	0.34%	347.30	——	2912	347.30	——	226
ivu36	1978	40988	493266	0.61%	139.27	——	388	139.27	——	92
ivu37	1978	48138	605873	0.64%	143.56	——	892	143.56	——	81
ivu38	1968	42201	528019	0.64%	116.47	——	396	116.47	——	91
ivu07	1751	59055	590055	0.57%	192.50	——	4050	192.50	——	129
ivu28	1295	82422	2375490	2.23%	55.24	——	2949	55.24	——	205
ivu06	1177	48058	461360	0.82%	140.18	——	1506	140.18	——	63

Table 4.12: Lower bounds computed by CPLEX

Name	Size				SBM			CAM			CCBM			CBM		
	Rows	Cols	NZEs	Density	$\underline{z}$	Gap	sec.	$\underline{z}$	Gap	sec.	$\underline{z}$	Gap	sec.	$\underline{z}$	Gap	sec.
ivu41	3570	56867	505195	0.25%	449.69	1.37%	218	448.73	1.58%	138	426.78	6.39%	170	452.89	0.67%	205
ivu08	2523	61464	528152	0.34%	346.60	0.20%	233	345.31	0.57%	176	331.19	4.64%	416	345.66	0.47%	177
ivu36	1978	40988	493266	0.61%	138.64	0.45%	199	138.01	0.90%	174	134.00	3.78%	224	138.26	0.73%	178
ivu37	1978	48138	605873	0.64%	142.77	0.55%	352	142.34	0.85%	226	137.37	4.31%	296	142.49	0.74%	232
ivu38	1968	42201	528019	0.64%	115.45	0.87%	305	113.93	2.18%	185	109.68	5.83%	181	115.32	0.99%	392
ivu07	1751	59055	590055	0.57%	191.89	0.32%	212	192.13	0.20%	189	177.83	7.62%	327	191.16	0.70%	164
ivu28	1295	82422	2375490	2.23%	52.92	4.20%	1325	52.51	4.94%	798	49.99	9.50%	1050	53.70	2.79%	1205
ivu06	1177	48058	461360	0.82%	139.86	0.23%	168	139.40	0.56%	114	126.04	10.09%	99	139.31	0.62%	114

Table 4.13: Lower bounds computed by SBM, CAM, CCBM, and CBM

Name	Size				CPLEX dualopt			CPLEX baropt		
	Rows	Cols	NZEs	Density	1%	2%	5%	1%	2%	5%
ivu41	3570	56867	505195	0.25 %	233	152	60	125	118	114
ivu08	2523	61464	528152	0.34 %	1647	1263	674	126	117	106
ivu36	1978	40988	493266	0.61 %	172	119	55	62	58	53
ivu37	1978	48138	605873	0.64 %	487	355	172	47	43	42
ivu38	1968	42201	528019	0.64 %	211	147	62	53	51	46
ivu07	1751	59055	590055	0.57 %	2023	1501	848	73	67	57
ivu28	1295	82422	2375490	2.23 %	1769	1491	708	139	135	125
ivu06	1177	48058	461360	0.82 %	773	593	390	40	36	32

Table 4.14: Time comparison of CPLEX for passing the 1%, 2%, and 5% gap in sec.

Name	Size				SBM			CAM			CCBM			CBM		
	Rows	Cols	NZEs	Density	1%	2%	5%	1%	2%	5%	1%	2%	5%	1%	2%	5%
ivu41	3570	56867	505195	0.25 %	——	52	31	——	55	11	——	——	——	117	51	12
ivu08	2523	61464	528152	0.34 %	39	33	23	64	16	8	——	——	487	38	17	8
ivu36	1978	40988	493266	0.61 %	46	25	17	131	13	7	——	——	98	55	14	7
ivu37	1978	48138	605873	0.64 %	66	39	29	161	21	4	——	——	229	106	21	4
ivu38	1968	42201	528019	0.64 %	263	147	23	——	——	11	——	——	——	330	87	11
ivu07	1751	59055	590055	0.57 %	37	30	24	43	22	9	——	——	——	44	22	9
ivu28	1295	82422	2375490	2.23 %	——	——	707	——	——	664	——	——	——	——	——	153
ivu06	1177	48058	461360	0.82 %	27	19	16	26	13	7	——	——	——	27	13	7

Table 4.15: Time comparison of SBM, CAM, CCBM, and CBM for passing the 1%, 2%, and 5% gap in sec.

Chapter 5

Summary

In this work we concentrated on developing methods, which determine good lower bounds for SPP instances in an appropriate amount of time. We found out that it makes sense to use the Lagrangian relaxation method for this task. The Lagrangian relaxed problem of SPP has a simple structure, which leads to algorithms and heuristics, whose total complexity per iteration depends linearly on the number of non-zeros of the problem matrix of SPP. In contrast, other methods like simplex methods or interior points methods have a complexity of higher order. Because the problem matrices of our tested instances are sparse (see our test bed presented in Table 4.1 or for another example Borndörfer [1998]) the linear dependence becomes an advantage for the algorithms and heuristics mentioned above.

As a reference for the state-of-the-art we have applied the dual simplex method and the barrier function method, implemented in CPLEX. The methods, which we have developed and compared with those of CPLEX, are SBM, CAM, CCBM, and CBM. SBM is a subgradient bundle method derived from the basic subgradient method, which is a global convergent method for determining the maximum of concave functions. CAM is a coordinate ascent method, where the convex coordinate bundle method CCBM and the coordinate bundle method CBM are derivatives from CAM.

We observed that the basic subgradient and the coordinate ascent method are improved if bundling techniques can be used. But the motivation for bundling differs for both approaches. In the former case bundling helps to approximate a minimum norm subgradient, which provides a steepest ascent direction, in order to speed up the performance. In the latter case bundling enables proceeding along directions, which are not restricted on the coordinate directions. By this the performance is accelerated.

Among all used techniques stabilization is worth mentioning. Stabilization improves the performance especially at the beginning by avoiding too big steps during the proceeding. This leads to a more stabilized progression. Stabilization

was successfully applied to SBM, CAM, CCBM, and CBM.

As an overall result we conclude the following:

1. CPLEX computes the optimal objective values, whereas SBM and CBM has on average a gap of under 1.5%.

2. In comparison to CPLEX baropt, SBM, CAM, and CBM the algorithm CCBM has a slow convergence because of the convex combination of ascent coordinate directions. An alternative is to relax the convex combination to a simple sum of the corresponding directions. This idea is realized in CBM.

3. If we focus on the running time rather than on optimality then CBM is on average the fastest algorithm.

Note that methods like SBM or CBM are applied on static SPP instances in order to determine a good lower bound. For solving SPP we need dynamical methods. Due to the complex topic of dynamical methods we will not discuss them, but a certain technique is worth mentioning. It is called column generation. We have indicated that this technique needs good Lagrangian multipliers of the corresponding SPP instances in order to generate further columns (in our case duties), which are added to the current SPP instance. Those multipliers are by-products of methods like our six considered methods. Due to the large number of such generation steps the running time depends on the computation time of these methods. Therefore, CBM fits more to this technique than CPLEX baropt or SBM.

To sum it up it can be said that applications such as a duty scheduling can be described as set partitioning problems, whose lower bound can be solved by simplex, interior points, subgradient, or coordinate ascent methods. It turns out that the interior points method CPLEX baropt and the heuristic CBM have good performances. Furthermore, good Lagrangian multipliers, which are by-products of these methods, can be used by techniques like column generation. For this particular technique it also turns out that among our tested algorithms CBM is the most efficient one. In general we can state that real-world applications, which have to solve a large number of Lagrangian relaxed SPP instances can improve their performance by using CBM.

Zusammenfassung

In dieser Arbeit entwickeln wir schnelle Algorithmen zur Bestimmung guter unterer Schranken von Set Partitioning Problemen. Es hat sich herausgestellt, daß für dieses Problem sich die Methoden der Lagrangerelaxierung anbieten. Die Lagrangerelaxierung eines Set Partitioning Problems (SPP) hat eine relativ einfache Struktur, die zu Verfahren führen, deren Komplexität pro Iteration linear in der Anzahl der Einsen der Problemmatrix von SPP ist. Im Gegensatz zu ihnen haben andere Methoden, wie zum Beispiel der Simplexalgorithmus oder das Innere-Punkte-Verfahren, eine höhere Komplexität aufzuweisen. Aufgrund der Tatsache, daß die Problemmatrizen relativ dünn sind (siehe Tabelle 4.1 oder Borndörfer [1998]), erweist sich daher die lineare Komplexität der Lagrangerelaxierungsmethoden als vorteilhaft.

Die von uns entwickelten Algorithmen sind SBM (Subgradient Bundle Method), CAM (Coordinate Ascent Method), CCBM (Coordinate Convex Bundle Method) und CBM (Coordinate Bundle Method). Sie wurden mit dem dualen Simplexalgorithmus und der Barrierefunktionsmethode, die in CPLEX implementiert sind, verglichen. Somit ist ein Vergleich mit aktuellen Algorithmen gewährleistet. SBM ist eine Subgradienten-Bündelmethode, die von der allgemeinen Subgradientenmethode abgeleitet wurde. CAM ist ein Koordinatenaufstiegsverfahren, wobei CCBM und CBM Weiterentwicklungen von CAM bilden.

Es läßt sich beobachten, daß man durch die Anwendung von Bündeltechniken die Effektivität der Subgradientenmethode und des Koordinatenaufstiegsverfahrens erhöhen kann. Jedoch ist die Anwendung dieser Technik bei beiden Methoden verschieden motiviert. Für die Subgradientenmethode wird durch diese Technik ein Subgradient mit minimaler Norm approximiert. Die Attraktivität solcher Subgradienten liegt darin, daß sie strikte Aufstiegsrichtungen darstellen. Koordinatenaufstiegsverfahren nutzen die Bündeltechnik dazu, andere Richtungen als die der Koordinatenrichtungen zu benutzen. Dadurch erhöhen sie ihre eigene Flexibilität.

Von allen angewendeten Techniken zur Verbesserung unserer Algorithmen ist vor allem die Stabilisierungstechnik (stabilization) erwähnenswert. Die Stabilisierung verbessert die Algorithmen vor allem in ihren ersten Iterationen, indem sie zu große Schrittweiten verhindert. Dadurch ergeben sich stabilere Verläufe. Auf SBM, CAM, CCBM und CBM brachte die Stabilisierungstechnik immense Ver-

besserungen ein.

Als Gesamtergebnis unserer Arbeit können wir folgendes festhalten:

1. CPLEX berechnet die optimalen Funktionswerte mit einer Genauigkeit von 10^{-6}. Der Gap von SBM und CBM beträgt im Durchschnitt 1.5%.

2. Im Vergleich zu SBM, CAM, CBM und der Barrierefunktionsmethode von CPLEX besitzt CCBM eine relativ langsame Konvergenz. Dieses Verhalten erklärt sich durch die Konvexkombination der Koordinatenrichtungen. Eine Alternative bildet hierbei CBM, die die Richtungen einfach aufsummiert.

3. Wenn die höhere Priorität der Laufzeit gilt, dann bildet CBM den schnellsten Algorithmus von allen sechs betrachteten Algorithmen.

Für die Lösung von SPP benötigt man dynamische Methoden. Aufgrund der Themenkomplexität von dynamischen Methoden werden wir hier nicht weiter darauf eingehen. Die Spaltengenerierungmethode ist jedoch erwähnenswert. In vorigen Kapiteln haben wir bereits angedeutet, daß sie gute Lagrangemultiplikatoren benötigt, um weitere Spalten (in Fall der Dienstplanoptimierung Dienste) zu generieren, die der aktuellen SPP-Instanz hinzugefügt werden. Die dazu genutzten Lagrangemultiplikatoren sind Nebenprodukte der sechs miteinander verglichenen Algorithmen. Aufgrund der großen Anzahl von Spaltengenerierungsschritten hängt die Laufzeit dieses Verfahrens von der Berechnungszeit von guten aber nicht unbedingt optimalen Lagrangemultiplikatoren ab. CBM eignet sich aufgrund seiner schnellen Laufzeit daher am besten für dieses Verfahren.

Zusammenfassend kann man festhalten, daß gewisse Anwendungen, wie zum Beispiel die Dienstplanoptimierung, als Set Partitioning Probleme mathematisch modelliert werden können. Das Problem der Bestimmung von unteren Schranken solcher Set Partitioning Probleme kann durch Verfahren, wie zum Beispiel Simplexmethoden, Innere-Punkte-Verfahren, Subgradientenmethoden oder Koordinatenaufstiegsverfahren, gelöst werden. Es hat sich herausgestellt, daß unter den betrachteten Algorithmen die Barrierefunktionsmethode von CPLEX und die CBM-Heuristik die effektivsten sind. Aufgrund des besseren Laufzeitverhaltens von CBM ist dieser Algorithmus jedoch besser für das Spaltengenerierungsverfahren geeignet. Man kann also den Schluß ziehen, daß Praxisanwendungen, die eine große Anzahl von SPP-Lagrangerelaxierungen lösen müssen, ihre Effektivität mit CBM merklich erhöhen können.

Bibliography

Aarts, E. and Lenstra, J. (1997). *Local Search in Combinatorial Optimization*. Wiley-Interscience.

Balinski, M. L. and Quandt, R. E. (1964). On an integer program for a delivery problem. *Operations Research*, 12:300–304.

Bazaraa, M. S., Sherali, H. D., and Shetty, M. (1993). *Nonlinear Programming - Theory and Algorithms*. John Wiley & Sons, Inc.

Bertsekas, D. P. (1995). *Nonlinear Programming*. Athena Scientific, Belmont, Massachusetts.

Bixby, R. E. (1992). Implementing the simplex method: The initial basis. *ORSA J. on Comp.*, 4(3):267–284.

Bixby, R. E., Gregory, J. W., Lustig, I. J., Marsten, R. E., and Shanno, D. F. (1992). Very large-scale linear programming: A case study in combining interior point and simplex methods. *Op. Res.*, 40(5):885–897.

Borndörfer, R. (1998). *Aspects of Set Packing, Partitioning, and Covering*. Berichte aus der Mathematik. Shaker, Aachen. PhD thesis, Tech. Univ. Berlin.

Caprara, A., Fischetti, M., and Toth, P. (1996). A heuristic algorithm for the set covering problem. Proc. of the 5th Int. IPCO Conf., pages 72–84, Vancouver, British Columbia, Canada.

Carpaneto, G., Dell'Amico, M., Fischetti, M., and Toth, P. (1989). A branch and bound algorithm for the multiple depot vehicle scheduling problem. *Networks*, 19:531–548.

Carpaneto, G., Fischetti, M., and Toth, P. (1987). New lower bounds for the symmetric travelling salesman problem. Technical Report. DEIS, University of Bologna (to appear in *Math. Program B.*

Ceria, S., Nobili, P., and Sassano, A. (1995). A lagrangian-based heuristic for large-scale set covering problems. Working Paper, University of Roma La Sapienza.

Christofides, N., Mingozzi, A., Toth, P., and Sandi, C. (1979). *Combinatorial Optimization*. Wiley Interscience.

CPLEX (1997). *Using the CPLEX Callable Library*. ILOG CPLEX Division, 889 Alder Avenue, Suite 200, Incline Village, NV 89451, USA. Information available at URL http://www.cplex.com.

Crowder, H. (1976). Computational improvements for subgradient optimization. In *Symposia Mathematica*, volume 19. Istituto Nazionale di Alta Matematica, Academic Press London and New York.

Cullen, F., Jarvis, J., and Ratliff, H. (1981). Set partitioning based heuristics for interactive routing. *Networks*, 11:125–143.

Danzig, G. (1951). Maximization of a linear function of variables subject to linear inequalities. Chapter 21 in: T.C. Koopmans (ed.), *Activity Analysis of Production an Allocation* (Cowles Commission Monograph No.13), Wiley, New York, pages 339–347.

Desrosiers, J., Du Merle, O., Villeneuve, D., and Hansen, P. (1997). Stabilisation dans le cadre de la génération de colonnes.

Desrosiers, J., Dumas, Y., Solomon, M. M., and Soumis, F. (1993). Time constrained routing and scheduling. Working paper.

Emden-Weinert, T., Hougardy, S., Kreuter, B., Proemel, H., and Steger, A. (1996). *Einführung in Graphen und Algorithmen*[1].

Fiacco, A. (1979). Barrier methods for nonlinear programming. in: A. Holzman (ed.), *Operations Research Support Methodology,* Marcel Dekker, New York, pages 377–440.

Fischetti, M. and Toth, P. (1992). An additive bounding procedure for the asymmetric TSP. *Mathematical Programming*, 53:173–197.

Fisher, M. L. and Rosenwein, M. B. (1985). An interactive optimization system for bulk cargo ship scheduling. Working paper.

Frisch, K. (1955). The logarithmic potiential method of convex programming. University Institute of Economics, Oslo, Norway.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York.

Gill, P., Murray, W., and Wright, M., editors (1981). *Practical Optimization.* Academic Press, London.

Graham, R., Grötschel, M., and Lovász, L., editors (1995). *Handbook of Combinatorics*, volume 2. Elsevier Science B.V., Amsterdam.

Grötschel, M., Lovász, L., and Schrijver, A. (1988). *Geometric Algorithms and Combinatorial Optimization.* Springer Verlag, Berlin.

Held, M. and Karp, R. M. (1971). The traveling-salesman problem and minimum spanning trees: part ii. *Mathematical Programming*, 1:6–25.

Hiriart-Urruty, J.-B. and Lemaréchal, C. (1993a). *Convex Analysis and Minimization Algorithms I: Fundamentals.* Springer Verlag.

[1] http://www.informatik.hu-berlin.de/Institut/struktur/algorithmen/ga/

Hiriart-Urruty, J.-B. and Lemaréchal, C. (1993b). *Convex Analysis and Minimization Algorithms II: Advanced Theory and Bundle Methods*. Springer Verlag.

Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395.

Kokott, A. and Löbel, A. (1996). Lagrangean relaxations and subgradient methods for multiple-depot vehicle scheduling problems. Preprint SC 96-22, Konrad-Zuse-Zentrum für Informationstechnik Berlin. Available at URL http://www.zib.de/ZIBbib/Publications.

Löbel, A. (1998). *Optimal Vehicle Scheduling in Public Transit*. Berichte aus der Mathematik. Shaker, Aachen. PhD thesis, Tech. Univ. Berlin.

Luenberger, D. G. (1989). *Linear and nonlinear programming*. Addison-Wesley, second edition.

Luh, B., Wang, J., and Zhao, X. (1999). Surrogate gradient algorithm for lagrangian relaxation. *Journal of Optimization Theory and Applications*, 100(3):699–712.

Lustig, I. J., Marsten, R., and Shanno, D. (1990). On implementing Mehrotra's predictor-corrector interior point method for linear programming. *SIAM Journal on Optimization*, 2(3):435–449.

Marsten, R. E. and Shepardson, F. (1981). Exact solution of crew scheduling problems using the set partitioning model: Recent successful applications. *Networks*, 11:165–177.

Nemhauser, G. L. and Wolsey, L. A. (1988). *Integer and Combinatorial Optimization*. John Wiley & Sons, Inc.

Polyak, B. T. (1967). A general method of solving extremum problems. *Soviet Mathematics Doklady*, 8:593–597. English translation.

Schrijver, A. (1989). *Theory of Linear and Integer Programming*. John Wiley & Sons Ltd., Chichester.

Wedelin, D. (1995). An algorithm for large scale 0-1 integer programming with application to airline crew scheduling. *Ann. Oper. Res.*, 57:283–301.