Solving the Master Linear Program in Column Generation Algorithms for Airline Crew Scheduling using a Subgradient Method

Per Sjögren

November 28, 2009

Abstract

A subgradient method for solving large linear programs is implemented and analyzed in a column generation framework. A performance comparison of the implementation is made versus the commercial linear programming solver XPress. Computational results from tests using data from airline crew scheduling, in particular crew rostering, show that the method performs very well in a column generation scheme, compared to both simplex and barrier methods.

Acknowledgements

I would like to express thanks to Curt Hjorring and Ann-Brith Strömberg, my supervisors at Jeppesen Systems AB and Göteborgs universitet, respectively. Thanks also go to Tomas Gustafsson, Mattias Grönkvist, Andreas Westerlund, and the other members of the optimization group at Jeppesen who made this thesis possible.

Contents

Chapter 1

Introduction

For an airline company crew salaries and costs constitute the second largest cost next to fuel. Cost-efficient crew scheduling is thus vital for the survival of a company and various optimization models and methods for minimizing personnel costs have been used for a long time.

As an example, one approach to select schedules for crew would be to enumerate all possible schedules during the duty period for each crew member and associate a certain cost to each schedule. Then it would in principle be possible to choose the cheapest combination of schedules such that every task is scheduled to an employee. However, such an approach is not practically possible due to the huge number of possible schedules. Instead a generation technique called column generation may be used [5]. Columns (which in this case represent schedules) are then generated as needed.

One part of the column generation process is solving linear programs (LP). Although effective means to solve linear programs exist, e.g. the simplex and the dual simplex methods, these methods possess an exponential worst case complexity ([9] pp. 434–435). In addition, the solutions produced by the simplex method are in several cases inefficient for the column generation process and approximate (different) solutions might be preferrable [8].

Due to the complexity of scheduling optimization problems many airline companies use optimization products developed by Jeppesen Systems AB to manage their crew scheduling. The optimization process of large-scale scheduling problems at Jeppesen incorporate many different optimization techniques of which column generation is one. The reason for this study is that it was noted at Jeppesen that a large amount of the column generator runtime was spent in the linear programming solver XPress. It was argued that a more specialized solver might be able to solve the linear programs faster. Also, as mentioned above, using a different solution technique rather than the simplex method for the linear program may accelerate the convergence in the column generation process.

In this thesis a subgradient method with variations is described and computational results using this method, both on pure LPs and in a column generation framework, is presented. Most of the development made in this thesis is based on work by Sherali and coauthors [7, 11, 12].

The outline of this thesis is as follows. In the next chapter the mathematical background in optimization theory used in the thesis is presented. In Chapter 3 the crew scheduling process and modelling is described. Chapter 4 contains a brief outline of column generation. Different solution methods for linear programs are presented in

Chapter 5. Chapter 6 presents computational results and conclusions are drawn in Chapter 7.

Chapter 2

Preliminaries

In this chapter a few basic results in optimization theory is presented and most of the notation used in the thesis is introduced. Lagrangian duality is needed for explaining the role of solving linear programs within column generation. Section 5.1 briefly touches LP duality, for which Lagrangian duality is also needed. Subgradients are introduced in Section 2.3 and used in the subgradient methods presented in Section 5.3. Optimality conditions are presented and used in Sections 5.1 and 5.2, respectively.

Lower-case bold letters used as variables or constants, e.g. $\mathbf{x}$, $\mathbf{b}$ and $\boldsymbol{\mu}$ denote real column vectors. Either the context will limit the size of the vectors or their dimensions will be given. Likewise $\mathbf{0}$ denotes the zero-vector and $\mathbf{1}$ denotes a vector of ones. Functions may be vector-valued, e.g. $\mathbf{h} : \mathbb{R}^n \to \mathbb{R}^m$, and are then also written in bold. The letteres A and D denote matrices, L denotes the Lagrangian function as defined in Section 2.1 and other capital letters denote vector spaces.

If $\mathbf{b}$ and $\mathbf{c}$ are two vectors, then $\mathbf{b} \geq \mathbf{c}$ is true if their elements $b_i \geq c_i$ for all i. Other relations, for example $\mathbf{x} = \mathbf{b}$, $\mathbf{x} \leq \mathbf{b}$, are defined analogously. Thus $\mathbf{x} \geq \mathbf{0}$ is true if all components of $\mathbf{x}$ are non-negative.

For a minimization problem

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}), && && \text{(2.1a)} \\ \text{subject to} \quad & \mathbf{h}(\mathbf{x}) &=& \mathbf{b}, && \text{(2.1b)} \\ & \mathbf{g}(\mathbf{x}) &\leq& \mathbf{0}, && \text{(2.1c)} \\ & \mathbf{x} &\in& X, && \text{(2.1d)} \end{aligned}$$

where $\mathbf{h} : \mathbb{R}^n \to \mathbb{R}^m, \mathbf{b} \in \mathbb{R}^m, \mathbf{g} : \mathbb{R}^n \to \mathbb{R}^k, f : \mathbb{R}^n \to \mathbb{R}$ and $X \subseteq \mathbb{R}^n$. If the optimal objective value exists, we denote it by f^*, which is then attained at some feasible point $\mathbf{x}^*$. A point $\mathbf{x} \in \mathbb{R}^n$ is said to be feasible in (2.1) if $\mathbf{h}(\mathbf{x}) = \mathbf{b}$, $\mathbf{g}(\mathbf{x}) \leq \mathbf{0}$ and $\mathbf{x} \in X$.

2.1 Lagrangian Duality

The general optimization problem (2.1) can be very hard to solve depending on the properties of f, $\mathbf{g}$, $\mathbf{h}$ and X. Various relaxation techniques can be used to produce an approximate solution, for example by extending the set X or removing a few of the constraints. One such technique is the Lagrangian relaxation. See [13] Ch. 2 for a definition of relaxation in an optimization context.

Consider the general optimization problem (2.1) where we assume that the inequality constraints (2.1c) are absent, i.e., $k = 0$. Instead of requiring the constraints (2.1b) to

be fulfilled we introduce a cost in the objective function for breaking these constraints. Define the Lagrangian function

$$L(\mathbf{x}, \boldsymbol{\mu}) = f(\mathbf{x}) + \boldsymbol{\mu}^T(\mathbf{b} - \mathbf{h}(\mathbf{x})), \qquad \mathbf{x} \in \mathbb{R}^n,\ \boldsymbol{\mu} \in \mathbb{R}^m,$$

and the Lagrangian dual function

$$\theta(\boldsymbol{\mu}) = \min_{\mathbf{x} \in X} L(\mathbf{x}, \boldsymbol{\mu}), \qquad \boldsymbol{\mu} \in \mathbb{R}^m. \tag{2.2}$$

If $\bar{\mathbf{x}}$ solves (2.2) and $\mathbf{x}^*$ solves (2.1), then the relations

$$\theta(\boldsymbol{\mu}) = f(\bar{\mathbf{x}}) + \boldsymbol{\mu}^T(\mathbf{b} - \mathbf{h}(\bar{\mathbf{x}})) \leq f(\mathbf{x}^*) + \boldsymbol{\mu}^T(\mathbf{b} - \mathbf{h}(\mathbf{x}^*)) = f(\mathbf{x}^*) = f^* \tag{2.3}$$

hold, since $\bar{\mathbf{x}}$ minimizes $L(\mathbf{x}, \boldsymbol{\mu})$ over $\mathbf{x} \in X$. Thus for every $\boldsymbol{\mu} \in \mathbb{R}^m$, $\theta(\boldsymbol{\mu})$ is a *lower bound* on the optimal value f^*. To get the highest possible lower bound we solve the Lagrangian dual problem to

$$\theta^* = \max_{\boldsymbol{\mu} \in \mathbb{R}^m} \theta(\boldsymbol{\mu}). \tag{2.4}$$

From the above follows that $\theta^* \leq f^*$. We define $\Gamma = f^* - \theta^*$ as the *duality gap* for the problem (2.1) with $k = 0$. Also, for any feasible point $\mathbf{x} \in \{\mathbf{x} \in X \,|\, \mathbf{h}(\mathbf{x}) = \mathbf{b}\}$, $f(\mathbf{x})$ is an upper bound on the primal optimal value f^*. If we know that $\Gamma = 0$ (which holds, e.g. for linear programs) and the dual problem is solved to optimality we can easily check how good a feasible point is by comparing its objective value with the lower bound.

The Lagrangian dual function has some nice properties as, e.g. concavity.

Proposition 2.1.1. *The dual function θ is concave.*

Proof. Let $\boldsymbol{\mu}_0, \boldsymbol{\mu}_1 \in \mathbb{R}^m$ and $0 \leq \alpha \leq 1$. It then holds that

$$\begin{aligned}
\theta(\alpha\boldsymbol{\mu}_0 + (1-\alpha)\boldsymbol{\mu}_1) &= \min_{\mathbf{x} \in X} \Big(f(\mathbf{x}) + \left[\alpha\boldsymbol{\mu}_0 + (1-\alpha)\boldsymbol{\mu}_1\right]^T \left[\mathbf{b} - \mathbf{h}(\mathbf{x})\right] \Big) \\
&= \min_{\mathbf{x} \in X} \left[\alpha \left(f(\mathbf{x}) + \boldsymbol{\mu}_0^T[\mathbf{b} - \mathbf{h}(\mathbf{x})] \right) + (1-\alpha) \left(f(\mathbf{x}) + \boldsymbol{\mu}_1^T[\mathbf{b} - \mathbf{h}(\mathbf{x})] \right) \right] \\
&\geq \alpha \min_{\mathbf{x} \in X} \left(f(\mathbf{x}) + \boldsymbol{\mu}_0^T[\mathbf{b} - \mathbf{h}(\mathbf{x})] \right) + (1-\alpha) \min_{\mathbf{x} \in X} \left(f(\mathbf{x}) + \boldsymbol{\mu}_1^T[\mathbf{b} - \mathbf{h}(\mathbf{x})] \right) \\
&= \alpha\theta(\boldsymbol{\mu}_0) + (1-\alpha)\theta(\boldsymbol{\mu}_1).
\end{aligned}$$

The result follows. □

This means that the Lagrangian properties are not dependent on the possible convexity of f. The Lagrangian dual function is, however, non-differentiable and an ordinary Newton method can not be used to solve the Lagrangian dual problem. For example a subgradient method can be used instead, see Chapter 5.

2.2 Optimality conditions

The Karush-Kuhn-Tucker (KKT) optimality conditions constitute a very useful tool for both confirming optimality and designing optimization methods. The KKT optimality conditions are stated without a proof for a special case as needed in this thesis; for a more thorough and general presentation, see for example the textbook [1], Ch. 5.

Consider the minimization program (2.1), where we assume that $X = \mathbb{R}^n$. Assume further that the functions f and $\mathbf{g}$ are convex and differentiable and that the functions $\mathbf{h}$ are linear. Then a feasible point $\mathbf{x} \in \mathbb{R}^n$ is optimal in (2.1) *if and only if* there exist vectors $\boldsymbol{\mu} \in \mathbb{R}^m$ and $\boldsymbol{\lambda} \in \mathbb{R}^k$ such that

$$\begin{aligned} \sum_{i=1}^{k} \lambda_i \nabla g_i(\mathbf{x}) + \sum_{i=1}^{m} \mu_i \nabla h_i(\mathbf{x}) + \nabla f(\mathbf{x}) &= \mathbf{0}, && \text{(2.5a)} \\ \lambda_i g_i(\mathbf{x}) &= 0, \qquad i = 1, \dots k, && \text{(2.5b)} \\ \boldsymbol{\lambda} &\geq \mathbf{0}. && \text{(2.5c)} \end{aligned}$$

Now consider the problem (2.1); the conditions in (2.5) are then simplified. For Lagrangian duality a primal-dual pair $(\mathbf{x}^*, \boldsymbol{\mu}^*)$ is optimal in (2.1) — as in Section 2.1 assuming that the inequality constraints (2.1c) are absent (i.e., $k = 0$) — if $\mathbf{x}^*$ is feasible and

$$\mathbf{x}^* \in \arg\min_{\mathbf{x} \in X} \; L(\mathbf{x}, \boldsymbol{\mu}^*).$$

This follows from the inequality (2.3) and the fact that $L(\mathbf{x}^*, \boldsymbol{\mu}^*) = f(\mathbf{x}^*)$ for feasible $\mathbf{x}^*$. If $k > 0$, i.e. the program contains inequality constraints, the Lagrangian function is defined slightly differently and an additional condition analogous to (2.5b) is needed to ensure that $L(\mathbf{x}^*, \boldsymbol{\mu}^*) = f(\mathbf{x}^*)$ for all feasible $\mathbf{x}^*$. Again, see for example [1] for a thorough presentation of optimality conditions.

2.3 Subgradients of the Lagrangian dual function

Let the function $\theta : \mathbb{R}^m \to \mathbb{R}$ be concave on $\mathbb{R}^m$. A *subgradient* to θ at $\boldsymbol{\mu}_0$ is a vector $\boldsymbol{\lambda} \in \mathbb{R}^m$ such that for all $\boldsymbol{\mu} \in \mathbb{R}^m$ it holds that

$$\theta(\boldsymbol{\mu}) \leq \theta(\boldsymbol{\mu}_0) + \boldsymbol{\lambda}^T(\boldsymbol{\mu} - \boldsymbol{\mu}_0). \tag{2.6}$$

The *subdifferential* of θ at $\boldsymbol{\mu}_0$ is the set of all subgradients at this point and is denoted by $\partial\theta(\boldsymbol{\mu}_0)$. If the subdifferential of θ at $\boldsymbol{\mu}_0$ contains the $\mathbf{0}$-vector then $\boldsymbol{\mu}_0$ is a maximizer of θ [1, p. 159]. The function θ is differentiable at $\boldsymbol{\mu}_0$ if the subdifferential contains only a single element which then equals the *gradient* of θ at $\boldsymbol{\mu}_0$.

Consider the Lagrangian dual function $\theta : \mathbb{R}^m \to \mathbb{R}$. As shown in the proof of Proposition 2.1.1 this function is concave. Let $\mathbf{x}_0 \in \arg\min_{\mathbf{x} \in X} L(\mathbf{x}, \boldsymbol{\mu}_0)$, i.e., $\mathbf{x}_0$ solves the program (2.2). Then $\mathbf{b} - \mathbf{h}(\mathbf{x}_0)$ is a subgradient to θ at $\boldsymbol{\mu}_0$ since, for all $\hat{\mathbf{x}} \in \arg\min_{\mathbf{x} \in X} L(\mathbf{x}, \boldsymbol{\mu})$,

$$\theta(\boldsymbol{\mu}) = f(\hat{\mathbf{x}}) + \boldsymbol{\mu}^T(\mathbf{b} - \mathbf{h}(\hat{\mathbf{x}})) \leq f(\mathbf{x}_0) + \boldsymbol{\mu}^T(\mathbf{b} - \mathbf{h}(\mathbf{x}_0)),$$

and since

$$\theta(\boldsymbol{\mu}_0) = f(\mathbf{x}_0) + \boldsymbol{\mu}_0^T(\mathbf{b} - \mathbf{h}(\mathbf{x}_0)),$$

this implies that

$$\theta(\boldsymbol{\mu}) - \theta(\boldsymbol{\mu}_0) \leq (\mathbf{b} - \mathbf{h}(\mathbf{x}_0))^T(\boldsymbol{\mu} - \boldsymbol{\mu}_0).$$

In this thesis subgradients will mostly be considered with respect to Lagrangian dual functions. We may define subgradients for convex functions analogously; the inequality in (2.6) is then reversed.

Chapter 3

Crew Rostering

In this chapter a brief outline of modelling rostering problems is given and in its last section a solution process is presented. In the rostering process a set of activities consisting of pairings (i.e. sequences of flights for a duty period), reserves, training activities, and various ground duties need to be covered by the available crew. All these activities are referred to as tasks. A collection of tasks for a specific crew member is called a roster.

There are numerous of rules and regulations that the personal rosters must follow. In this thesis we will limit the description of modelling by presenting examples of rules and how to model them and refer the reader to [5] for a more detailed explanation.

In the following subsection the basic rostering model is presented. Section 3.2 extends the model by introducing a few example constraints. Soft and hard constraints are presented in Section 3.3 and in Section 3.4 a brief solution process is outlined.

3.1 The basic model

The rostering problem is built by a set T of tasks and a set C of crew members. For each crew member $k \in C$ we need to assign a legal roster $R_{j_k} \subset T$ such that

$$T = R_{j_1} \cup R_{j_2} \cup \cdots \cup R_{j_{|C|}},$$

where $R_{j_i} \cap R_{j_k} = \emptyset$ if $i \neq k$. This ensures that each task is assigned to exactly one crew member. Additionally, this set of rosters should be the 'best' choice in some sense. What 'best' means is often a combination of, e.g. paid overtime, a fair distribution of attractive rosters, and quality of life for crew members.

In order to formulate the rostering problem as an optimization problem we model a roster R_i as a binary column vector $\mathbf{a}_i$ such that $a_{ji} = 1$ if column i is a roster for crew member $j \in \{1, \ldots, |C|\}$. For each task j that the roster includes $a_{(j+|C|)i}, j \in T$. All other $a_{ji} = 0$. Let then x_i be a binary decision variable denoting whether to use the roster modelled by $\mathbf{a}_i$ or not. Let also c_i be the cost associated to the roster R_i. Finally we define the matrix $A = [\mathbf{a}_1 \ldots \mathbf{a}_k] \in \{0, 1\}^{m \times n}$, where n is the total number of possible rosters and $m = |C| + |T|$ is the sum of the number of tasks and crew members.

The first $|C|$ rows of A make sure that every crew member gets assigned to exactly one roster; these are called *assignment constraints*. The second block of A, called *activity constraints*, makes sure that every task gets assigned exactly once. This is illustrated in Table 3.1.

The matrix A										rel	rhs	Description
Crew 1			Crew 2			...	Crew $\|C\|$					
R_1	R_2	R_3	R_4	R_5	R_6	...	R_{n-2}	R_{n-1}	R_n			
1	1	1	0	0	0	...	0	0	0	=	1	Assignment
0	0	0	1	1	1	...	0	0	0	=	1	constraints
						$\ddots$				$\vdots$	$\vdots$	
0	0	0	0	0	0	...	1	1	1	=	1	
1	1	0	0	1	0	...	1	0	0	=	1	Activity
0	1	1	0	0	1	...	0	1	0	=	1	constraints
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	
1	0	1	1	0	0	...	0	0	1	=	1	

Table 3.1: The matrix A, relations (rel), and right-hand sides (rhs) for the basic rostering model; here three different rosters are generated for each crew member.

From the above we conclude that in its simplest form the rostering problem is a set-partitioning problem (SPP), that is, to

$$\underset{x}{\text{minimize}} \quad \mathbf{c}^T\mathbf{x}, \tag{3.1a}$$

$$\text{subject to} \quad A\mathbf{x} = \mathbf{1}, \tag{3.1b}$$

$$\mathbf{x} \in \{0,1\}^n, \tag{3.1c}$$

where $\mathbf{c} \in \mathbb{R}^n$ and $A \in \{0,1\}^{m\times n}$.

One intuitive solution approach to the rostering problem would be to generate all possible rosters $R_j, j \in J$, that is, to form all possible subsets of T and pick the best ones. This corresponds to generating the full matrix A and solving the problem (3.1). However, this approach fails at first glance because of the huge number of possible rosters for practical instances.

Alternatively one can generate a large number of rosters and solve a set-partitioning problem in order to obtain the best solution consisting of combinations of the rosters generated. Depending on the solution characteristics more rosters (columns) can be generated until the solution is close to optimal. This process, known as column generation, is explained in Chapter 5.

3.2 Extending the model

The basic rostering model can be extended in several ways to include more complicated rules. If more than one crew member is needed for a specific task — for example, a certain training activity may need two captains — the right-hand side of the row representing that task is simply changed from 1 to the required number. For every task requiring an additional qualification a constraint row is added. One example of such a qualification is that a flight between two different countries may need at least one crew member to speak an additional language. Table 3.2 illustrates this.

When a task requires more than one crew member it is preferrable that not only inexperienced crew are assigned to it. How to model this is illustrated in Table 3.3. If, for some reason, two crew members can not work together this is modelled analogously.

The matrix A										rel	rhs	Description
Crew 1			Crew 2			...	Crew $\|C\|$					
R_1	R_2	R_3	R_4	R_5	R_6	...	R_{n-2}	R_{n-1}	R_n			
1	1	1				...				=	1	Assignment
			1	1	1	...				=	1	constraints
						$\ddots$				$\vdots$	$\vdots$	
						...	1	1	1	=	1	
1	1	0	0	1	0	...	1	0	0	=	2	Task 1
0	0	0	0	1	0	...	0	0	0	$\geq$	1	T. 1, language
0	1	1	0	0	1	...	0	1	0	=	2	Task 2
0	0	0	0	0	1	...	0	0	0	$\geq$	1	T. 2, language
1	0	1	1	0	0	...	0	0	1	=	3	Task 3
0	0	0	0	0	0	...	0	0	1	$\geq$	1	T. 3, language

Table 3.2: An extended rostering model including language requirement constraints. Crew member 2 knows the language required for Tasks 1 and 2 but not for task 3. Crew member $|C|$ knows the language required for Task 3.

The matrix A										rel	rhs	Description
Crew 1			Crew 2			...	Crew $\|C\|$					
R_1	R_2	R_3	R_4	R_5	R_6	...	R_{n-2}	R_{n-1}	R_n			
1	1	1				...				=	1	Assignment
			1	1	1	...				=	1	constraints
						$\ddots$				$\vdots$	$\vdots$	
						...	1	1	1	=	1	
1	1	0	0	1	0	...	1	0	0	=	2	Task 1
1	1	0	0	1	0	...	0	0	0	$\leq$	1	T. 1, inexperience
0	1	1	0	0	1	...	0	1	0	=	2	Task 2
0	0	0	0	0	1	...	0	0	0	$\leq$	1	T. 2, Inexperience
1	0	1	1	0	0	...	0	0	1	=	3	Task 3
0	0	0	0	0	0	...	0	0	0	$\leq$	1	T. 3, Inexperience

Table 3.3: An extended rostering model including inexperience constraints. Crew member 1 is inexperienced for Task 1, crew member 2 is inexperienced for Tasks 1 and 2.

The difference is that crew members might be inexperienced for one task but not for another, while incompatibility between two crew members holds for all tasks.

3.3 Soft constraints

When using optimization software for creating a schedule the answer "This problem is infeasible" is seldom an acceptable outcome. What the user wants to know when the problem is infeasible is whether he or she can solve the scheduling problem by, for example, calling in extra staff or paying workers for over-time. For this reason a partition is often made between *hard* and *soft* constraints. Hard constraints may not be broken at all, while soft constraints may be broken at a cost.

Consider the problem to

$$\begin{aligned}
\underset{x}{\text{minimize}} \quad & \mathbf{c}^T\mathbf{x}, \\
\text{subject to} \quad A\mathbf{x} &= \mathbf{b}, && \text{(3.2a)} \\
D\mathbf{x} &\leq \mathbf{d}, && \text{(3.2b)} \\
\mathbf{x} &\in [0,1]^n. && \text{(3.2c)}
\end{aligned}$$

Let us assume that the constraints (3.2a) are soft in the sense that a solution $\mathbf{x}$ such that $A\mathbf{x} \leq \mathbf{b}$ is allowed and that the constraints (3.2b) may be broken. For this setting we introduce the variables $\mathbf{s}$ and $\mathbf{t}$ along with costs for breaking each of the constraints and formulate the problem with soft constraints as that to

$$\begin{aligned}
\text{minimize} \quad & \mathbf{c}_\text{s}^T\mathbf{s} + \mathbf{c}_\text{t}^T\mathbf{t} + \mathbf{c}^T\mathbf{x}, && \text{(3.3a)} \\
\text{subject to} \quad \mathbf{s} + A\mathbf{x} &= \mathbf{b}, && \text{(3.3b)} \\
-\mathbf{t} + D\mathbf{x} &\leq \mathbf{d}, && \text{(3.3c)} \\
\mathbf{x} &\in [0,1]^n, && \text{(3.3d)} \\
\mathbf{s},\ \mathbf{t} &\geq \mathbf{0}. && \text{(3.3e)}
\end{aligned}$$

The values of variables $\mathbf{s}$ and $\mathbf{t}$ denote how much the rules modelled are broken. Decisions must then be made on whether or not the constraints have been violated too much and whether the extra cost is acceptable.

3.4 Solving Rostering Problems

The size of rostering problems solved by Jeppesen software (or of any fairly large scheduling problem) makes a brute force solution approach impossible. Instead some sort of heuristic generation process combined with fixing strategies may be used.

One approach is to put a lot of effort in generating good rosters. This could be done in a column generation scheme connected to some LP solver. When an LP-relaxation of the original problem has been solved, variables with values 'close to 1' can be fixed to 1 and the restricted problem is solved. This will iteratively generate a sequence of solutions which eventually approaches an integer solution.

Obviously a large number of LPs will have to be solved in this process. Thus it is essential that solving LPs is done as fast as possible. Column generation is explained in the following chapter and various LP solvers are discussed in Chapter 5.

Another approach is to use local search to iteratively improve a solution. An initial feasible assignment of all trips is then required. This can in most cases be done by a simple greedy algorithm since the initial assignment can disregard fairness and costs.

Chapter 4

Column Generation

In this chapter we consider the problem to

$$\begin{aligned} \text{minimize} \quad & \mathbf{c}^T\mathbf{x}, && && (4.1a)\\ \text{subject to} \quad & A\mathbf{x} && = \mathbf{b}, && (4.1b)\\ & \mathbf{x} && \in X, && (4.1c) \end{aligned}$$

where the set $X \subset \mathbb{R}^n$ is defined by linear (in)equalities and possibly also integer requirements. The general setting is that the matrix A has too many columns for the program (4.1) to be computationally tractable. Instead we generate columns of A as they are needed. An example of this is the rostering problem, described in Chapter 3, in which A consists of all possible rosters for all crew members. However, before looking at the general column generation scheme we present the classical Dantzig-Wolfe decomposition scheme for linear programs. For Dantzig-Wolfe decomposition, see for example [6], and for column generation in general, see [8].

4.1 Dantzig-Wolfe decomposition

Consider the problem to

$$\begin{aligned} \text{minimize} \quad & \mathbf{c}^T\mathbf{x}, && && (4.2a)\\ \text{subject to} \quad & A\mathbf{x} && \geq \mathbf{b}, && (4.2b)\\ & D\mathbf{x} && \geq \mathbf{d}, && (4.2c)\\ & \mathbf{x} && \geq \mathbf{0}, && (4.2d) \end{aligned}$$

with optimal value z^*.

We define the polyhedron $P = \{\mathbf{x} \in \mathbb{R}^n_+ \mid D\mathbf{x} \geq \mathbf{d}\} \neq \emptyset$. If P is bounded then each point in P can be expressed as a convex combination of its extreme points $\{\mathbf{p}_q\}_{q\in Q}$. If P is unbounded a nonnegative linear combination of its extreme rays need to be included as well, but for simplicity we restrict our presentation to the case where P is bounded.[1] Thus we can express

$$P = \left\{ \mathbf{x} \in \mathbb{R}^n \;\middle|\; \mathbf{x} = \sum_{q\in Q} \mathbf{p}_q\lambda_q, \sum_{q\in Q} \lambda_q = 1, \lambda_q \geq 0, q \in Q \right\}$$

[1]The rostering problem introduced in the previous chapter has this property since the variables are continuously relaxed binary variables. That is, all feasible variable values are in the interval $[0, 1]$.

and the problem (4.2) can be expressed as the so called (full) *master problem* to

$$\begin{aligned}
\text{minimize} \quad & \sum_{q\in Q} \lambda_q \mathbf{c}^T \mathbf{p}_q, && && (4.3a)\\
\text{subject to} \quad & \sum_{q\in Q} \lambda_q A\mathbf{p}_q &\geq& \mathbf{b}, && (4.3b)\\
& \sum_{q\in Q} \lambda_q &=& 1, && (4.3c)\\
& \lambda_q &\geq& 0, \quad q \in Q. && (4.3d)
\end{aligned}$$

If only a subset $Q_i \subset Q$ of the extreme points is known the *restricted master problem* (RMP) is defined by substituting Q by Q_i:

$$\begin{aligned}
\text{minimize} \quad & \sum_{q\in Q_i} \lambda_q \mathbf{c}^T \mathbf{p}_q, && && (4.4a)\\
\text{subject to} \quad & \sum_{q\in Q_i} \lambda_q A\mathbf{p}_q &\geq& \mathbf{b}, && (4.4b)\\
& \sum_{q\in Q_i} \lambda_q &=& 1, && (4.4c)\\
& \lambda_q &\geq& 0, \quad q \in Q_i. && (4.4d)
\end{aligned}$$

Columns (i.e., extreme points of P) of the restricted master problem are generated by solving the subproblem

$$\mathbf{p}_{i+1} \in \arg\min \left\{ \left(\mathbf{c}^T - \boldsymbol{\mu}_i^T A\right) \mathbf{x} - v_i \mid D\mathbf{x} \geq \mathbf{d}, \mathbf{x} \geq \mathbf{0} \right\}, \tag{4.5}$$

with optimal value $z_i \geq z^*$, and where $(\boldsymbol{\mu}_i, v_i)$ is an optimal solution to the linear programming dual of the restricted master problem (4.4). Note that p_{i+1} is an extreme point of P. Here, the dual variables $\boldsymbol{\mu}_i$ and v_i correspond to the constraints (4.4b) and the convexity constraint (4.4c), respectively. Defining $Q_{i+1} = Q_i \cup \{\mathbf{p}_{i+1}\}$, we add the column $\left((A\mathbf{p}_{i+1})^T, 1\right)^T$ with cost coefficient $\mathbf{c}^T\mathbf{p}_{i+1}$ to the restricted master problem. The restricted master problem is then solved again producing new dual variable values $(\boldsymbol{\mu}_{i+1}, v_{i+1})$. The optimal value of (4.5) equals the reduced cost of the variable λ_{i+1}, and the process is iterated until the optimal value of (4.5) is non-negative. The latter means that no columns are then added to the restricted master problem, since no column results in an improvement. Thus, the solution to the final restricted master problem is also a solution to the full master problem.

4.2 General column generation

In this section we consider a linear program in a standard formulation with the additional constraint that every variable has an upper bound. This bound is here set to 1 but may vary between variables. Thus, let the following problem be the master problem, MP:

$$\begin{aligned}
z_{\text{MP}} = \text{minimize} \quad & \mathbf{c}^T\mathbf{x} && && (4.6a)\\
\text{subject to} \quad & A\mathbf{x} &=& \mathbf{b}, && (4.6b)\\
& \mathbf{x} &\in& X \equiv [0,1]^n, && (4.6c)
\end{aligned}$$

where $A \in \mathbb{R}^{m \times n}$ and we assume that $n \gg m$. Now, assuming that i columns of A are known, we partition A into one known and one unknown part, i.e., $A = [A_\mathrm{k}, A_\mathrm{u}]$. Analogously we partition $\mathbf{c} = [\mathbf{c}_\mathrm{k}^T, \mathbf{c}_\mathrm{u}^T]^T$ and $\mathbf{x} = [\mathbf{x}_\mathrm{k}^T, \mathbf{x}_\mathrm{u}^T]^T$. This, with $\mathbf{y} = \mathbf{x}_\mathrm{k}$, allows the formulation of the restricted master problem, RMP, as to

$$
\begin{aligned}
z_\mathrm{RMP} = \text{minimize} \quad & \mathbf{c}_\mathrm{k}^T \mathbf{y} && && \text{(4.7a)} \\
\text{subject to} \quad & A_\mathrm{k}\mathbf{y} & = & \mathbf{b}, && \text{(4.7b)} \\
& \mathbf{y} & \in & [0,1]^i. && \text{(4.7c)}
\end{aligned}
$$

The program (4.7) is a restriction of MP since, if $\mathbf{y}$ is feasible in (4.7) and $\mathbf{x} = (\mathbf{y}^T, \mathbf{0}^T)^T$, then $\mathbf{x}$ is feasible in (4.6). Since $[0,1]^i \times \{0\}^{n-i}$ is a subset of X, it follows that $z_\mathrm{MP} \leq z_\mathrm{RMP}$. Relaxing the equality constraints (4.7b) in RMP (as in Section 2.1) we obtain the Lagrangian function

$$
L^\mathrm{k}(\mathbf{y}, \boldsymbol{\mu}) = \mathbf{c}_\mathrm{k}^T \mathbf{y} + \boldsymbol{\mu}^T (\mathbf{b} - A_\mathrm{k}\mathbf{y})
$$

and the Lagrangian dual problem

$$
\underset{\boldsymbol{\mu} \in \mathbb{R}^m}{\text{maximize}} \ \theta^\mathrm{k}(\boldsymbol{\mu})
$$

where

$$
\theta^\mathrm{k}(\boldsymbol{\mu}) = \boldsymbol{\mu}^T \mathbf{b} + \underset{\mathbf{y} \in [0,1]^i}{\text{minimize}} \ (\mathbf{c}_\mathrm{k}^T - \boldsymbol{\mu}^T A_\mathrm{k})\mathbf{y}.
$$

Since MP and RMP have the same number of rows and the columns of A_k is a subset of those of A, a dual vector $\boldsymbol{\mu}$ for RMP is also a dual vector for MP and it can be used to obtain information about MP.

Let $\boldsymbol{\mu}$ be any Lagrangian dual vector (for example obtained by solving the Lagrangian dual to RMP approximately). Then, if the reduced costs $\mathbf{c}_\mathrm{u} - \boldsymbol{\mu}^T A_\mathrm{u}$ are all non-negative the corresponding variables $\mathbf{x}_\mathrm{u}$ in MP will be set to 0 and

$$
\theta^\mathrm{k}(\boldsymbol{\mu}) = \theta(\boldsymbol{\mu}) = \boldsymbol{\mu}^T b + \max_{\mathbf{x} \in [0,1]^n} (\mathbf{c}^T - \boldsymbol{\mu}^T A)\mathbf{x}
$$

is a lower bound on z_MP. If further $(\mathbf{y}, \boldsymbol{\mu})$ is an optimal Lagrangian primal-dual solution for RMP then the Langrangian duality optimality conditions from Section 2.2 yields that $((\mathbf{y}^T, \mathbf{0})^T, \boldsymbol{\mu})$ is an optimal primal-dual solution to MP.

However, if the reduced cost for some variable in $\mathbf{x}_\mathrm{u}$ is negative, then $\theta(\boldsymbol{\mu}) < \theta^\mathrm{k}(\boldsymbol{\mu})$ and $\theta^\mathrm{k}(\boldsymbol{\mu})$ is not necessarily a lower bound on z_MP. We may then let the column corresponding to the smallest reduced cost enter A_k and compute a new dual solution, possibly warm-starting the LP solver with the previous dual variable values. Alternatively we may add the j columns with smallest (but still negative) reduced costs. Assuming that the goal is to produce a lower bound on z_MP, this process will terminate either when all columns in MP are generated or when the reduced costs corresponding to the unknown variables $\mathbf{x}_\mathrm{u}$ are all non-negative.

Whether this process is worthwile depends on the manner in which columns are generated and how the reduced costs are calculated. In a practical rostering problem the total number of columns is often very large and calculating the reduced costs for all possible columns is not possible. Instead it is possible to heuristically generate a set of columns utilizing the dual vector and the (relatively well-known) structure of A. If we do not manage to heuristically generate any columns with negative reduced costs

we may assume that the dual vector is close to optimal, depending on how well the heuristic solution approximates the optimal one.

As an example of a heuristic for generating columns with negative reduced cost, a graph can be generated to represent all the tasks. Arcs with a cost associated to them represent tasks. Solving a k-shortest path problem in the graph will then yield a column with low reduced cost. The correctness of this pricing heuristic is dependent on how well the cost of arcs represent the actual cost of rosters.

Chapter 5

An overview of methods for solving linear programs

The previous chapter presents the column generation principle and defines a sequence of linear programs that need to be solved throughout the column generation procedure. The successive linear programs are similar to each other, thus the solution of a previously solved linear program may be utilized when solving the next one. This warm-start (when possible) often reduces runtime considerably. This chapter describes a few different methods for solving this sequence of linear programs.

We first give a brief description of the simplex method with variations for linear programming. In Section 5.2 barrier methods, such as the interior point method, is described. The following sections are then focused on subgradient methods with variations.

5.1 The Simplex Method and Variations

The simplex method is due to George Dantzig and revolutionized the field of optimization. Since it is a well-known method this section will contain a very brief outline of it and refer the reader to, for example, the textbooks [1] and [9] for details.

This section will consider a linear program in standard form, that is,

$$\begin{aligned} \text{minimize} \quad & \mathbf{c}^T\mathbf{x}, && && \text{(5.1a)} \\ \text{subject to} \quad & A\mathbf{x} &= \mathbf{b}, && \text{(5.1b)} \\ & \mathbf{x} &\geq \mathbf{0}, && \text{(5.1c)} \end{aligned}$$

where $A \in \mathbb{R}^{m \times n}$ and we denote the feasible set by $Q = \{\mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$.

A linear program that has a solution also has a solution at an extreme point of its feasible set ([1, p. 219]). In its simplest form the simplex method initiates at an extreme point of the polytope Q and moves along the edges of Q via extreme points until optimality is verified. Under non-degeneracy assumptions the simplex algorithm terminates after a finite number of steps since there is only finitely many extreme points of a polytope. While in many cases the simplex method is very efficient its worst-case complexity is very bad; see for example [9, pp. 434–435] for an example where every extreme point of Q is visited before termination. This example results in an exponential number of iterations.

A solution of RMP given by the simplex method, an extreme point of the feasible set, is not the dual solution needed for column generation. However, the simplex method produces both primal and corresponding dual solutions. Lagrangian relaxing the constraints (5.1b) (as in Chapter 2) results in the Lagrangian dual problem to

$$\underset{\boldsymbol{\mu}\in\mathbb{R}^m}{\text{maximize}} \left(\mathbf{b}^T\boldsymbol{\mu} + \underset{\mathbf{x}\geq\mathbf{0}}{\text{minimize}} \, (\mathbf{c}^T - \boldsymbol{\mu}^T A)\mathbf{x} \right). \tag{5.2}$$

If a dual solution $\hat{\boldsymbol{\mu}}$ results in a negative reduced cost $(\mathbf{c} - \hat{\boldsymbol{\mu}}^T A)_i$ for any variable x_i the corresponding Lagrangian subproblem solution becomes unbounded. This dual variable value $\hat{\boldsymbol{\mu}}$ can then obviously not be optimal in the Lagrangian dual problem. Thus we can restrict the dual problem by excluding dual variable values producing unbounded subproblems. This restriction does not change the optimal value and gives an equivalent formulation of the Lagrangian dual problem, the *linear programming dual* to

$$\begin{aligned} \text{maximize} \quad & \mathbf{b}^T\boldsymbol{\mu}, && && (5.3a)\\ \text{subject to} \quad & A^T\boldsymbol{\mu} && \leq \quad \mathbf{c}, && (5.3b)\\ & \boldsymbol{\mu} && \in \quad \mathbb{R}^m. && (5.3c) \end{aligned}$$

The linear programming primal and dual problems are closely related. As mentioned in Section 2.1, the corresponding duality gap is zero. The KKT optimality conditions for a primal-dual pair $(\mathbf{x}, \boldsymbol{\mu})$ may thus be expressed as

$$\begin{aligned} A\mathbf{x} &= \mathbf{b}, && (5.4a)\\ A^T\boldsymbol{\mu} &\leq \mathbf{c} && (5.4b)\\ \mathbf{x}^T(\mathbf{c} - \boldsymbol{\mu}^T A) &= \mathbf{0}, && (5.4c)\\ \mathbf{x} &\geq \mathbf{0}. && (5.4d) \end{aligned}$$

Thus $\overline{\mathbf{x}} \in Q$ is optimal in (5.1) if there exists a point $\overline{\boldsymbol{\mu}}$ feasible in (5.3) and such that (5.4c) holds. Therefore, another way to describe the simplex method is: for a primal-dual pair $(\mathbf{x}, \boldsymbol{\mu})$ the simplex method maintains primal feasibility and approaches dual feasibility. From this we can easily derive the dual simplex algorithm, which does principally the same but maintains dual feasibility and approaches primal feasibility.

Simplex is often considered as an ultimate method for LPs, but, as mentioned above, in some cases it performs extremely slowly. Not only that, the 'random nature' of the dual extreme point solutions produced by the simplex method is inefficient for column generation. This is due to an oscillating property of the dual variable values during the column generation [3, p. 19]. Other methods, such as barrier or subgradient methods, may increase the over-all performance of column generation, see [8] and [3, p. 19].

5.2 Barrier Methods

Consider the general minimization program to

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}), && && (5.5a)\\ \text{subject to} \quad & \mathbf{g}(\mathbf{x}) && \leq \quad \mathbf{0}, && (5.5b)\\ & \mathbf{x} && \in \quad \mathbb{R}^n, && (5.5c) \end{aligned}$$

with optimal value $f^* = f(\mathbf{x}^*)$, where $f : \mathbb{R}^n \to \mathbb{R}$ and $\mathbf{g} : \mathbb{R}^n \to \mathbb{R}^m$. Letting $W = \{\mathbf{x} \in \mathbb{R}^n | \mathbf{g}(\mathbf{x}) \leq \mathbf{0}\}$ denote the feasible set, barrier methods introduce a penalty in the objective function that makes solutions close to the boundary of W intractable. If the boundary is approached from the interior of the feasible set, the method is a so-called interior point-method; it is an exterior point-method if it is approached from outside the feasible set.

We note that program (5.5) is equivalent to the problem to

$$\text{minimize } f(\mathbf{x}) + \chi_W(\mathbf{x}),$$

where

$$\chi_W(\mathbf{x}) = \begin{cases} 0, & \text{if } \mathbf{x} \in W, \\ \infty, & \text{otherwise,} \end{cases}$$

is a so-called indicator function.

A function such as χ_W is obviously not easy to handle computationally since it is differentiable only in the interior of W and an approximation of it is therefore required. For the interior point method one such approximation is the logarithmic function which results in the problem to

$$\underset{\mathbf{x} \in \mathbb{R}^n}{\text{minimize}} \; f(\mathbf{x}) - \nu \sum \log\left(-g_i(\mathbf{x})\right), \tag{5.6}$$

where $\nu > 0$. If any of the elements of $\mathbf{g}(\mathbf{x})$ approaches 0 the objective increases very fast. Letting $\mathbf{x}_\nu^*$ be the solution to the program (5.6) for a certain value of $\nu > 0$, then $\mathbf{x}_\nu^* \to \mathbf{x}^*$ as $\nu \to 0$, see for example [1, Ch. 13]. The interior point method is then to iteratively choose smaller ν, that is $\nu_t > \nu_{t+1} > \cdots > 0$ such that $\nu_t \to 0$ as $t \to \infty$. Solving (5.6) for each ν_t results in a sequence $\mathbf{x}_{\nu_t} \to \mathbf{x}^*$ as $t \to \infty$.

Turning our focus of interior point methods to their application to linear programming we consider the linear programming dual (5.3), i.e. $f(\boldsymbol{\mu}) = \mathbf{b}^t\boldsymbol{\mu}$ and $\mathbf{g}(\boldsymbol{\mu}) = \boldsymbol{\mu}^T A - \mathbf{c}$. If the $\mathbf{c} > \mathbf{0}$, the solution $\boldsymbol{\mu} = \mathbf{0}$ is a feasible interior point that may initialize the interior point method. However, since $A \in \mathbb{R}^{m \times n}$ each iteration of the barrier method, i.e. solving (5.6) for a certain value of ν, includes solving a system of m polynomial equations of degree n. Furthermore, the vector $\mathbf{c}$ may include both negative as well as 0 values.

Since the system of non-linear equations (5.4) can not be solved by Newton's method [1, p. 337], we add the slack variables $\mathbf{s}$ to the program (5.3) to obtain a different formulation of the optimality conditions. The slack variables transform the program to

$$\begin{aligned} \underset{\mathbf{s},\boldsymbol{\mu}}{\text{maximize}} \quad & \mathbf{b}^T\boldsymbol{\mu}, && && \text{(5.7a)} \\ \text{subject to} \quad & A^T\boldsymbol{\mu} + \mathbf{s} &=& \mathbf{c}, && \text{(5.7b)} \\ & \mathbf{s} &\geq& \mathbf{0}., && \text{(5.7c)} \\ & \boldsymbol{\mu} &\in& \mathbb{R}^m. && \text{(5.7d)} \end{aligned}$$

The optimality conditions (5.4) are then transformed into

$$\begin{aligned} A\mathbf{x} &= \mathbf{b}, && \text{(5.8a)} \\ A^T\boldsymbol{\mu} + \mathbf{s} &= \mathbf{c} && \text{(5.8b)} \\ \mathbf{x}^T\mathbf{s} &= \mathbf{0}, && \text{(5.8c)} \\ \mathbf{x}, \mathbf{s} &\geq \mathbf{0}. && \text{(5.8d)} \end{aligned}$$

Changing the sign of the objective function (5.7a), relaxing the constraints (5.7c) and adding a penalty $\nu > 0$ to the objective function yields the program to

$$\underset{\boldsymbol{\mu},\mathbf{s}}{\text{minimize}} \quad -\mathbf{b}^T\boldsymbol{\mu} - \nu\sum_{i=1}^{n}\log(s_i), \text{subject to} \quad A^T\boldsymbol{\mu} + \mathbf{s} = \mathbf{c}, \tag{5.9a}$$

The KKT conditions for the program (5.9) are given by

$$\begin{aligned} A\mathbf{x} &= \mathbf{b}, && (5.10a)\\ A^T\boldsymbol{\mu} + \mathbf{s} &= \mathbf{c}, && (5.10b)\\ x_i s_i &= \nu, \qquad i = 1,\ldots,n, && (5.10c) \end{aligned}$$

where $\mathbf{s} > \mathbf{0}$ and $\nu > 0$ imply $\mathbf{x} > 0$. Since it is assumed that there exists an interior point to the program (5.3), there exists a $\mathbf{s} > \mathbf{0}$ such that $A^T\boldsymbol{\mu} + \mathbf{s} = \mathbf{c}$.The system (5.10) of equations is solvable by for example Newton's method, see [1, pp. 336–337].

One of the main drawbacks of using a barrier method for column generation is that there is no possible warm start. Since the linear programs solved in two subsequent column generation iterations are similar a warm start would be preferrable.

5.3 Subgradient Methods

Consider the optimization problem to

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}), && (5.11a)\\ \text{subject to} \quad & \mathbf{h}(\mathbf{x}) = \mathbf{b}, && (5.11b)\\ & \mathbf{x} \in X. && (5.11c) \end{aligned}$$

A subgradient algorithm is a generalization of a gradient based descent algorithm to non-differentiable functions. For subgradient methods a line-search can in some cases be made[1] but usually a predetermined steplength rule is used.

A basic subgradient algorithm applied to the Lagrangian dual of (5.11)

$$\theta^* = \underset{\boldsymbol{\mu}\in\mathbb{R}^m}{\text{maximize}}\ \theta(\boldsymbol{\mu}), \tag{5.12}$$

where

$$\theta(\boldsymbol{\mu}) = \underset{\mathbf{x}\in X}{\text{minimize}}\ \left(f(\mathbf{x}) + \boldsymbol{\mu}^T[\mathbf{b} - \mathbf{h}(\mathbf{x})]\right), \tag{5.13}$$

works in the following manner. The sequence of iterates is defined by

$$\boldsymbol{\mu}^{k+1} = \boldsymbol{\mu}^k + \alpha_k\mathbf{g}^k, \tag{5.14}$$

where $\alpha_k > 0$ denotes the steplength and

$$\mathbf{g}^k = \mathbf{b} - \mathbf{h}(\mathbf{x}^k),$$

where $\mathbf{x}^k$ solves (5.13) for $\boldsymbol{\mu} = \boldsymbol{\mu}^k$, is a subgradient to θ at $\boldsymbol{\mu}^k$.

[1]For example by using a predetermined steplength rule and for a set number of iterations increase the step if this results in an improvement.

Here,

$$\mathbf{x}^k \in \arg\min_{\mathbf{x}\in X} \left(f(\mathbf{x}) + (\boldsymbol{\mu}_k)^T \left[\mathbf{b} - \mathbf{h}(\mathbf{x})\right]\right), \tag{5.15}$$

is easily solved if (5.11) is a linear program. From the discussion on subgradients in Section 2.3 we know that $\mathbf{g}^k$ is a subgradient to θ at $\boldsymbol{\mu}^k$.

The choice of the steplength α_k in a subgradient method can be made to fulfill theoretical convergence criteria and/or for its practical behavior. Two examples of step-length rules are (see [1, Ch. 6]), the divergent series step-length rule

$$\alpha_k \to 0, \text{ as } k \to \infty, \qquad \sum_{k=1}^{\infty} \alpha_k = \infty, \qquad \sum_{k=1}^{\infty} \alpha_k^2 < \infty, \tag{5.16}$$

and the socalled Polyak step lengths,

$$\alpha_k = s_k \frac{\theta^* - \theta(\boldsymbol{\mu}_k)}{(\mathbf{g}^k)^T(\mathbf{g}^k)}, \tag{5.17}$$

where $s_k \in [\epsilon_1, 2-\epsilon_2], \epsilon_1 > 0, \epsilon_2 > 0$ is a step length parameter. The optimal value θ^* of (5.12) is usually not known but an approximation of it may be used. For example the objective value $\bar{\theta} > \theta^*$of the best known feasible solution to the primal problem may be used in place of θ^* in (5.17). The quality of the approximation determines how close to θ^* the objective values $\theta(\boldsymbol{\mu}^k)$ can be guaranteed to reach, see [10, p. 19] Theorem 4.

Finally, we state a convergence result for subgradient methods; for a proof, see for example [1, pp. 169–170].

Theorem 5.3.1. *The algorithm (5.14) in combination with any of the step length rules (5.16) or (5.17) converges to a solution to the Lagrangian dual (5.12).*

The theorem above applied to the step-length rule (5.17) is valid when the optimal value θ^* is used in this formula. If θ^* is approximated by $\hat{\theta}$ then the subgradient method (5.14), (5.17) converges to a solution with objective value $\geq \hat{\theta}$ if $\hat{\theta} \leq \theta^*$ and with objective value $\leq 2\theta^* - \hat{\theta}$ otherwise. See [10] for details.

5.3.1 Inequality constraints and restricting the dual space

The minimization problem (5.11) has no inequality constraints which means that the dual feasible set $\Pi = \mathbb{R}^m$. Assume that (5.11) is a linear program including inequality constraints, i.e. the program to

$$\begin{aligned} \text{minimize } & \mathbf{c}^T\mathbf{x}, && && (5.18a)\\ \text{subject to } & A\mathbf{x} &=& \mathbf{b}, && (5.18b)\\ & D\mathbf{x} &\leq& \mathbf{d}, && (5.18c)\\ & \mathbf{x} &\in& X. && (5.18d) \end{aligned}$$

Lagrangian relaxing the constraints (5.18b), (5.18c) yields the Lagrangian function

$$L(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda}) = \mathbf{c}^T\mathbf{x} + \boldsymbol{\mu}^T(\mathbf{b} - A\mathbf{x}) + \boldsymbol{\lambda}(D\mathbf{x} - \mathbf{d}).$$

To ensure that the function L yields a relaxation of f the additional dual variable values $\boldsymbol{\lambda}$ are required to be non-negative. Thus we define the dual space $\Pi = \mathbb{R}^m \times \mathbb{R}^k_+$ and change the step (5.14) to

$$(\boldsymbol{\mu}^{i+1}, \boldsymbol{\lambda}^{i+1} = P_\Pi \left((\boldsymbol{\mu}^i, \boldsymbol{\lambda}^i) + \alpha_i \mathbf{g}^i \right),$$

where P_Π denotes the Euclidean projection onto the set Π, in this case $P_\Pi(\boldsymbol{\mu}, \boldsymbol{\lambda}) = (\boldsymbol{\mu}, (\max\{0, \lambda_j\})_{j=1}^k)$, and $\mathbf{g}^i = (\mathbf{b} - A\mathbf{x}, D\mathbf{x} - \mathbf{d})$. This change ensures that no infeasible dual variable values, which give no information on the optimal value of (5.18), are evaluated.

There are other cases for which a restriction of the dual space is needed. LP duality from Section 5.1 is one example. The dual space is restricted to force the subproblem in program (5.2) to stay bounded. Consider the minimization problem to

$$\begin{aligned}
\text{minimize} \quad & \mathbf{c}_s^T\mathbf{s} + \mathbf{c}_t^T\mathbf{t} + \mathbf{c}^T\mathbf{x}, && && \text{(5.19a)}\\
\text{subject to} \quad & \mathbf{s} + A\mathbf{x} &=& \mathbf{b}, && \text{(5.19b)}\\
& -\mathbf{t} + D\mathbf{x} &\leq& \mathbf{d}, && \text{(5.19c)}\\
& \mathbf{x} &\in& [0,1]^n, && \text{(5.19d)}\\
& \mathbf{s},\ \mathbf{t} &\geq& \mathbf{0}. && \text{(5.19e)}
\end{aligned}$$

This program contains unbounded slack variables $\mathbf{s}$ and $\mathbf{t}$. Lagrangian relaxing the constraints (5.19b) and (5.19c) yields the Lagrangian dual problem to

$$\underset{\boldsymbol{\mu}\in\mathbb{R}^m, \boldsymbol{\lambda}\in\mathbb{R}_+^k}{\text{maximize}} \ \theta(\boldsymbol{\mu}, \boldsymbol{\lambda}), \tag{5.20}$$

where

$$\begin{aligned}
\theta(\boldsymbol{\mu}, \boldsymbol{\lambda}) &= \boldsymbol{\mu}^T\mathbf{b} - \boldsymbol{\lambda}^T\mathbf{d} && \text{(5.21)}\\
&+ \underset{\substack{\mathbf{x}\in[0,1]^n\\ \mathbf{s}\geq 0\\ \mathbf{t}\geq 0}}{\text{minimize}} \ \left[\left(\mathbf{c}^T - \boldsymbol{\mu}^T A + \boldsymbol{\lambda}^T D\right)\mathbf{x} + (\mathbf{c}_s - \boldsymbol{\mu})^T\mathbf{s} + (\mathbf{c}_t - \boldsymbol{\lambda})^T\mathbf{t}\right],
\end{aligned}$$

Note that the computation of $\theta(\boldsymbol{\mu}, \boldsymbol{\lambda})$ separates into $n+m+k$ single-variable subproblems that are solved by the extreme value of its respective variable.

$$\begin{aligned}
\min_{x_i\in[0,1]} (\mathbf{c}^T - \boldsymbol{\mu}^T A + \boldsymbol{\lambda}^T D)_i x_i, \qquad & i = 1, \ldots, n, && \text{(5.22a)}\\
\min_{s_i\geq 0} (\mathbf{c}_s - \boldsymbol{\mu})_i s_i, \qquad & i = 1, \ldots, m, && \text{(5.22b)}\\
\min_{t_i\geq 0} (\mathbf{c}_t - \boldsymbol{\lambda})_i t_i, \qquad & i = 1, \ldots, k. && \text{(5.22c)}
\end{aligned}$$

Each single-variable subproblem is solved by the upper or lower extreme value of its variable. Thus the unbounded from above slack variables is a problem that needs to be addressed.

For subproblems (5.22b) no restriction of the dual space is needed since the slack variables $\mathbf{s}$ are implicitely bounded by $\mathbf{b}$. For any value of $\mathbf{x}$, $A\mathbf{x}$ is non-negative provided that $A \in \mathbb{R}_+^{m\times n}$, thus $\mathbf{s} \leq \mathbf{b}$ and $\mathbf{b}$ is an upper bound on $\mathbf{s}$ to use when solving the subproblems (5.22b).

For the subproblems (5.22c) the situation is different, there is no easily available upper bound on $\mathbf{t}$. An alternative is to look at the reduced costs $\mathbf{c}_t - \boldsymbol{\lambda}$ for the variables $\mathbf{t}$. If we restrict the dual space so that $\boldsymbol{\lambda} \leq \mathbf{c}_t$, the subproblem (5.22c) will be bounded. Analogously the dual space may be restricted with $\boldsymbol{\mu} \leq \mathbf{c}_s$ which removes the need of an upper bound on $\mathbf{s}$. These two restrictions ensures that subproblems (5.22b) and (5.22c) are always 0. Since an unbounded (from below) solution can not maximize θ,

this restriction of the dual space yields an optimization problem that is equivalent to (5.20), which can then be stated as to

$$\text{maximize } \theta(\boldsymbol{\mu}, \boldsymbol{\lambda}),$$

$$\text{such that } \boldsymbol{\mu} \leq \mathbf{c}_s,$$

$$\mathbf{0} \leq \boldsymbol{\lambda} \leq \mathbf{c}_t,$$

where

$$\theta(\boldsymbol{\mu}, \boldsymbol{\lambda}) = \mathbf{b}^T\boldsymbol{\mu} - \mathbf{d}^T\boldsymbol{\lambda} + \sum_{j=1}^{n} \min_{x_j \in [0,1]} (\mathbf{c}^T - \boldsymbol{\mu}^T A + \boldsymbol{\lambda}^T D)_j x_j.$$

5.3.2 Deflected Subgradients

Although the subgradient method does converge to an optimal solution to the dual problem, it does so very slowly. One reason for this is that the dual iterates often describe a zig-zag pattern. One way to deal with this problem is to remember previous 'good' directions. That is, keeping the previous direction and combine it with the new subgradient. Thus, the search direction is calculated as

$$\mathbf{d}^k = \alpha \mathbf{g}^k + (1 - \alpha)\mathbf{d}^{k-1}, \quad k = 1, 2, \ldots$$

where $\alpha \in (0, 1]$ can be either fixed or altered during the solution process. Note that since $\mathbf{d}^{k-1}$ is a convex combination of a previous direction and a subgradient, by letting $\mathbf{d}^0 = 0$ it follows that

$$\mathbf{d}^k = \alpha \sum_{i=1}^{k} (1 - \alpha)^{k-i} \mathbf{g}^i,$$

where $\alpha_i = \alpha(1-\alpha)^{k-i} \geq 0, \forall i$, $\sum_{i=1}^{k} \alpha_i = 1$. Thus the direction is a convex combination of all previous subgradients. If instead of using predetermined values of α_i the best linear combination of subgradients is computed the so called bundle method is obtained; see for example [4, Ch. 14-15].

One example of a deflected subgradient method is the Volume algorithm by Barahona and Anbil [2]. It uses a fixed deflection parameter for the direction and also the same deflection parameter to produce a sequence of primal values. Although the algorithm often produces primal values of reasonable quality it is not convergent, as demonstrated by Sherali and Lim [12].

5.3.3 Variable Target Value Methods

The basic idea behind the Variable Target Value Method (VTVM) is to use the step-length rule

$$\alpha_k = s_k \frac{\theta^* - \theta(\boldsymbol{\mu}_k)}{(\mathbf{g}_k)^T(\mathbf{g}_k)}$$

in the method (5.14) where an approximate upper bound $\overline{\theta}$, called target value, is used in place of θ^*.

An outer loop controls the target value and keeps track of improvements to the best known solution while an inner loop generates the iterates $\boldsymbol{\mu}^k$. Whenever the objective value of the Lagrangian dual problem (5.12) gets larger than the target value the target

value is increased. If the inner loop has not managed to generate any good iterates the target value is decreased. This ensures that the target value is neither too high nor too low, following the reasoning of step lengths from the beginning of this section. As long as the target value stays below θ^* when it is increased the method will tend to a better value. If the target value is increased above θ^* it might be necessary to decrease it to see an improvement in convergence. See Section (6.1) for a step by step description of the algorithm.

VTVM is a general convergent algorithm for nondifferential convex optimization and can be used in combination with many different step direction strategies. See [11] for VTVM and [12] for VTVM in combination with the volume algorithm for linear programs.

Chapter 6

Computational Tests and Results

This chapter contains computational results from using the VTVM on linear programs from crew rostering problems. Tests are done on stand alone problems comparing solution quality and speed with that of XPress as well as solving the RMP during full column generation runs. The next section gives a detailed explanation of the algorithm VTVM. Section 6.2 describes the different test cases used. The following two sections compares the algorithm with XPress's simplex and barrier algorithms.

6.1 The Algorithm

The idea behind the Variable Target Value Method (VTVM) is described in the previous chapter. In this section the actual algorithm will be stated, including the parameter values used in the test-runs.

For a dual vector $\boldsymbol{\mu}^i$ we denote the objective value to the Lagrangian subproblem $\theta_i = \theta(\boldsymbol{\mu}^i)$. For this iteration $\mathbf{x}^i \in \arg\min_{\mathbf{x}\in X} L(\boldsymbol{\mu}^i, \mathbf{x})$ solves the subproblem and $\mathbf{g}^i = \mathbf{b} - A\mathbf{x}^i$ is a subgradient to θ at $\boldsymbol{\mu}^i$. The algorithm is then as follows [12]:

(0) Initialize the parameters $\epsilon_0 \geq 0$, $\epsilon > 0$, $k_{\max}$, $\beta \in (0, 1]$, $\sigma \in (0, 1/3]$, $r \in (0, 1)$, $\bar{r} = r + 1$, and $\bar{\tau}, \bar{\gamma}$, $\eta \in (0, 1]$. In this implementation we use $\epsilon_0 = 10^{-6}, \epsilon = 0.1, k_{\max} = 3000, \beta = 0.8, \sigma = 0.15, r = 0.1, \bar{\tau} = 75, \bar{\gamma} = 20$ and $\eta = 0.75$. Choose an initial solution $\boldsymbol{\mu}^1$ (possibly from a warm start) and determine θ_i and $\mathbf{g}^i$. Set $\bar{\boldsymbol{\mu}} = \boldsymbol{\mu}^1$ and $\bar{\mathbf{g}} = \mathbf{g}^1$ and the best objective value known $z_1 = \theta_1$. If $||\mathbf{g}^1|| < \epsilon_0$ terminate. Initialize $l = k = 1, \tau = \gamma = \delta = 0$ and $RESET = 1$. Initialize the target value $w_1 = \min\{\text{UB}, \theta_1 + ||\mathbf{g}^1||^2/2\}$ where UB is a known upper bound to $\theta(\boldsymbol{\mu})$, at worst UB$= \infty$. Set $\epsilon_1 = \sigma(w_1 - \theta_1)$.

(1) If $RESET = 1$ set the search direction $\mathbf{d}^k = \mathbf{g}^k$, else $\mathbf{d}^k = \alpha\mathbf{g}^k + (1-\alpha)\mathbf{d}^{k-1}$, where $\beta \leq \alpha \leq 1$. If $||\mathbf{d}^k|| = 0$ set $\mathbf{d}^k = \mathbf{g}^k$. Set the steplength $s = \beta(w_l - \theta_k)/||\mathbf{d}^k||^2$ and set $\boldsymbol{\mu}^{k+1} = P_{\Pi}(\boldsymbol{\mu}^k + s\mathbf{d}^k)$, where P_{Π} denotes Euclidean projection onto the dual space. Increment τ and k by 1, set $RESET = 0$, and compute θ_k and $\mathbf{g}^k$.

(2) If $\theta_k > z_{k-1}$, set $\delta = \delta + (\theta_k - z_{k-1}), z_k = \theta_k, \bar{\boldsymbol{\mu}} = \boldsymbol{\mu}^k, \bar{\mathbf{g}} = \mathbf{g}^k$, and $\gamma = 0$, and go to step 3. Otherwise set $z_k = z_{k-1}$, increment γ by 1, and go to step 4.

(3) If $k > k_{\max}$ or $||\mathbf{g}^k|| < \epsilon_0$, terminate. If $z_k \geq w_l - \epsilon_l$, go to step 5, if $\tau \geq \bar{\tau}$ go to step 6, otherwise return to step 1.

(4) If $k > k_{\max}$ terminate. If $\gamma \geq \bar{\gamma}$ or $\tau \geq \bar{\tau}$ go to step 6. Else, return to step 1.

(5) Compute $w_{l+1} = z_k + \max\{\epsilon_l + \eta\delta, r|z_k|\}$. If $r|z_k| > \epsilon_l + \eta\delta$ set $r = r/\bar{r}$. Let $\epsilon_{l+1} = \max\{(w_{l+1} - z_k)\sigma, \epsilon\}$, put $\tau = \delta = 0$, and increment l by 1. Return to step 1.

(6) Compute $w_{l+1} = (z_k + \epsilon_l + w_l)/2$ and $\epsilon_{l+1} = \max\{(w_{l+1} - z_k)\sigma, \epsilon\}$. If $\gamma \geq \bar{\gamma}$, then set $\bar{\gamma} = \max\{\bar{\gamma} + 10, 50\}$. If $(w_l - w_{l+1}) \leq 0.1$, then set $\beta = \max\{\beta/2, 10^{-6}\}$. Put $\gamma = \tau = \delta = 0$ and increment l by 1. Reset $\boldsymbol{\mu}^k = \bar{\boldsymbol{\mu}}, \theta_k = z_k, \mathbf{g}^k = \bar{\mathbf{g}}$ and put $RESET = 1$. Return to step 1.

Various strategies can be used for the deflection parameter α. Here we use $\alpha = 1$ for most cases. This basically turns the algorithm into a pure subgradient algorithm with an elaborate scheme for calculating the steplength. Of course, the parameters may (and should) be tuned for the problem type solved.

In practice, the following termination criterion is added to the implementation: If an improvement occurs, but the improvement is smaller than some tolerance, the dual variable is assumed to be close to an optimal point and the algorithm is terminated.

6.2 Test cases

The test cases come from crew rostering problems of the types described in Chapter 3. The first set of tests, (see Table 6.1,) is for comparison of solving a single problem, that is, they have a fixed size. Typically they are dumps of the restricted master program of various column generation runs.

Test	Rows	Columns
A	226	4475
B	459	8409
C	437	19273
D	1173	54744

Table 6.1: Test cases for direct comparison between methods solving linear programs.

The second set of test cases, (see Table 6.2,) come from crew rostering problems for full column generation runs. The problem is set up and solved and the solution is analyzed. The number of columns in the problems varies between solution methods. This is because some methods produce the 'right' columns more quickly and the column generation is terminated earlier; see Section 6.4 for details on the different methods. Each of these sets consists of 10 different, although similar, problems. All constraints in these problems are equality constraints and all variables have natural upper bounds.

Describing the size of these problems is intricate since it depends on the order in which (and thereby how many) columns are generated. But, the number of rows is given by what type of, and how many, constraints are used, the number of tasks to be distributed, and the number of crew members in the problem. The approximate problem size given in Table 6.2 is a mean of the size of the final RMPs in the data set. More on how many columns are generated in the problems is described in Section 6.4. All test problems are run on an Opteron 2220, 2.8 GHz Dual-Core with 6 GB memory.

Test set	Description	Approximate size rows, columns
bos_25	Boston, 25 crew	220, 4400
atl_95	Atlanta, 95 crew	460, 8600
bos_50	Boston, 50 crew	440, 19000
atl_254	Atlanta, 254 crew	1170, 50000

Table 6.2: Test sets for column generation with ten test cases in each set. Solving a problem determines monthly rosters for all crew in the problem.

6.3 Comparison with XPress

The behavior of the LP solver determines the success of a column generator. A fast LP solver that produces dual variable values which in turn generate bad columns may be worse than a slow LP solver that generates good columns. In order to analyze how the VTVM performs within column generation it is necessary to know how it performs on single problems.

Table 6.3 shows runtimes for solving the four linear programs given in Table 6.1, solved by both the primal simplex method and the VTVM.

Test problem	Runtime(s)		Error
	The simplex method	VTVM	
A	1.0	1.19	8.0%
B	3.12	1.22	1.7%
C	23.15	3.19	9.5%
D	193.31	10.82	8.2%

Table 6.3: Test results for single problems comparing the simplex method and VTVM. Error given by $(\theta^* - \theta^k)/\theta^*$ where θ^* is the optimal objective value and θ^k is the objective value obtained at termination of the VTVM.

As seen in the table the VTVM becomes increasingly faster than the primal simplex method when the problem sizes increase. This is to be expected, since the simplex method solves several systems of linear equations while the most time consuming operation in VTVM is matrix-vector multiplication.[1] Additionally, VTVM in this implementation terminates after a maximum of 3000 iterations, the simplex method may need more iterations, i.e. more linear equations to solve, when the problem size increases.

The solution produced by VTVM is almost always sub-optimal, that is, has lower objective value compared to the optimal objective value. Further tuning of the parameters given in Section 6.1 results in solutions with objective values closer to optimum at the cost of longer runtimes. However, in the column generation scheme the improved LP solution quality is not worth the increased runtime. Although the solution is sub-optimal the VTVM always produces a feasible dual solution and can be used to generate columns.

[1]Solving a system of linear equations with a sparse matrix (n columns) uses $O(n^2)$ basic, i.e. multiplication and addition, operations. Sparse matrix-vector multiplication only uses $O(n)$ basic operations.

6.4 Results within column generation

As seen in the previous section the speed of the VTVM LP solver is significantly faster compared to the XPress simplex solver. The difference within column generation is not quite as large since the warm-start for the simplex method is very effective[2]. The big gain comes from producing better dual solutions, thereby generating the correct columns earlier. This results in smaller RMPs to solve during column generation, earlier termination of the column generation algorithm, and a smaller integer program to solve when fixing is performed.

One problem with the subgradient method is that it does not produce primal variable values to be used for fixing variable values. Another is that while the subgradient method quickly generates good columns it performs poorly at the end of the column generation process. In the Jeppesen column generator several columns are generated in each iteration. When the column generation process (where the RMP is solved by the subgradient method) terminates, in many cases it does so after quite many iterations in which only a few columns have been generated. That is, it terminates after quite many iterations without much improvement. To address these issues the LP solver switches to the simplex method (using XPress) when the column generation scheme does not improve much.

Figure 6.1 shows the total runtime of solving the LP relaxation using column generation together with the primal and dual simplex methods (XPress), the barrier method (XPress) and the variable target value method. The barrier method and the VTVM produce only dual values[3]. As evident in the figure the subgradient method does not always perform better than the simplex method. The barrier method seems to be a better choice over-all.

Although in some cases the only output required is a lower bound to the problem, for which a dual solution is sufficient, what is ultimately required is an integer solution. For this (continous) primal variable values constitute a good start to be used in various fixing procedures. As mentioned above the VTVM is replaced by the simplex method at the end of the column generation process to terminate the process earlier. Another result of this is that primal values are also produced. For comparison reasons, the barrier method runs a cross-over, also at the end of the column generation process, to generate an optimal solution including both primal and dual variable values. Figure 6.2 shows the result of this. With this change the VTVM is faster in all problem instances, even the smaller ones in the top figure. For the largest problems the column generation with the VTVM is 40%-60% faster but even the smaller problems show significant improvement. For even larger problems the difference between barrier and VTVM may be larger since the barrier method includes more complex operations than VTVM, such as linear equation solving.

The convergence of the column generation process is shown in Figures 6.3 and 6.4

[2]If this was not the case the difference in time between using the simplex method and the subgradient method in the column generation algorithm would be far greater. Consider that the subgradient method is about 20 times faster than the simplex method on the largest stand alone problem of the previous section, a problem of corresponding final size is solved about twice as fast with VTVM compared to the simplex method during the column generation run. See Figure 6.2.

[3]The barrier method is a primal-dual method and produces both primal and dual values. However, since it is an approximate method the objective value given by the dual variable values is lower than the value of any feasible primal solution. Thus there are no primal values corresponding to the lower bound and the dual values.

for two instances in the data sets. Using the subgradient method the column generation process is near convergence before the duals produced by the simplex method produce columns that result in improvement. The spikes in the figures are due to the subgradient method being an approximate method so it may terminate suboptimally. That the subgradient method on rare occations terminates very suboptimal during the column generation process is not a problem. At worst a few unnecessary columns are added and in the next iteration the subgradient method will most probably not terminate too early on the modified RMP. These figures show the strength of the VTVM in this context. Duals produced generating the correct columns early as seen by the figure with objective value versus iteration

The number of columns generated in each instance is shown in Figure 6.5. The total number of columns affect the speed of the RMP solver as well as the fixing procedure after the LP relaxation has been solved. How many columns the final RMP consists of is dependant on the number of column generation iterations, but also how many of those iterations generate 'many' columns.

Figure 6.6 shows the number of column generation iterations for each problem instance. As seen in the figure the VTVM uses in most cases fewer iterations. This of course affects the total runtime. However, all problems in the data sets have fairly easy rules and fast pricing problems (generating columns). Thus, a problem with a more difficult pricing problem should be solved relatively even faster using the VTVM to solve the restricted master problem.

Finally, Figure 6.7 shows the runtime for the largest data set (Atlanta 254) split into the time for pricing and for RMP. As seen in Figure 6.6, for this data set the number of column generation iterations is about the same for both simplex and VTVM. As expected the time spent in the pricer is about the same for these methods as well. Using VTVM to solve the RMP the time spent solving the RMP improves greatly. Using the Simplex method, up to 85% of the runtime is spent solving the RMP. For the VTVM, around 60% of the total runtime is spent solving the RMP. Add to this that the VTVM has the potential to use fewer column generation iterations, thus the total runtime can be much improved.

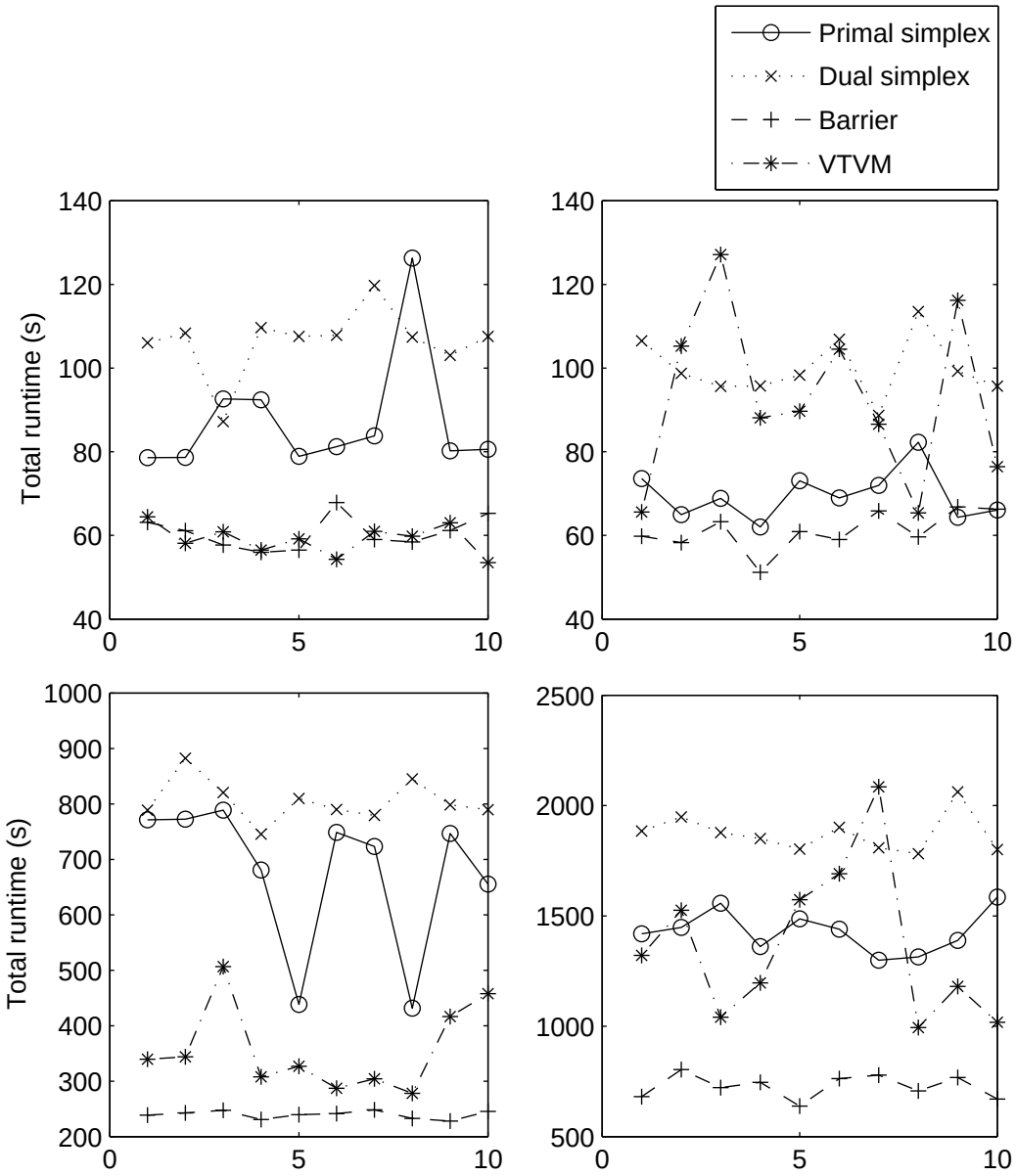

Figure 6.1: Runtimes for the four methods for solving the LP relaxation. The barrier method and VTVM produce only dual variable values. The figures to the left show runtimes for Boston25 (upper) and Boston50 (lower) test sets. The figures to the right show runtimes for Atlanta95 (upper) and Atlanta254 (lower) test sets.

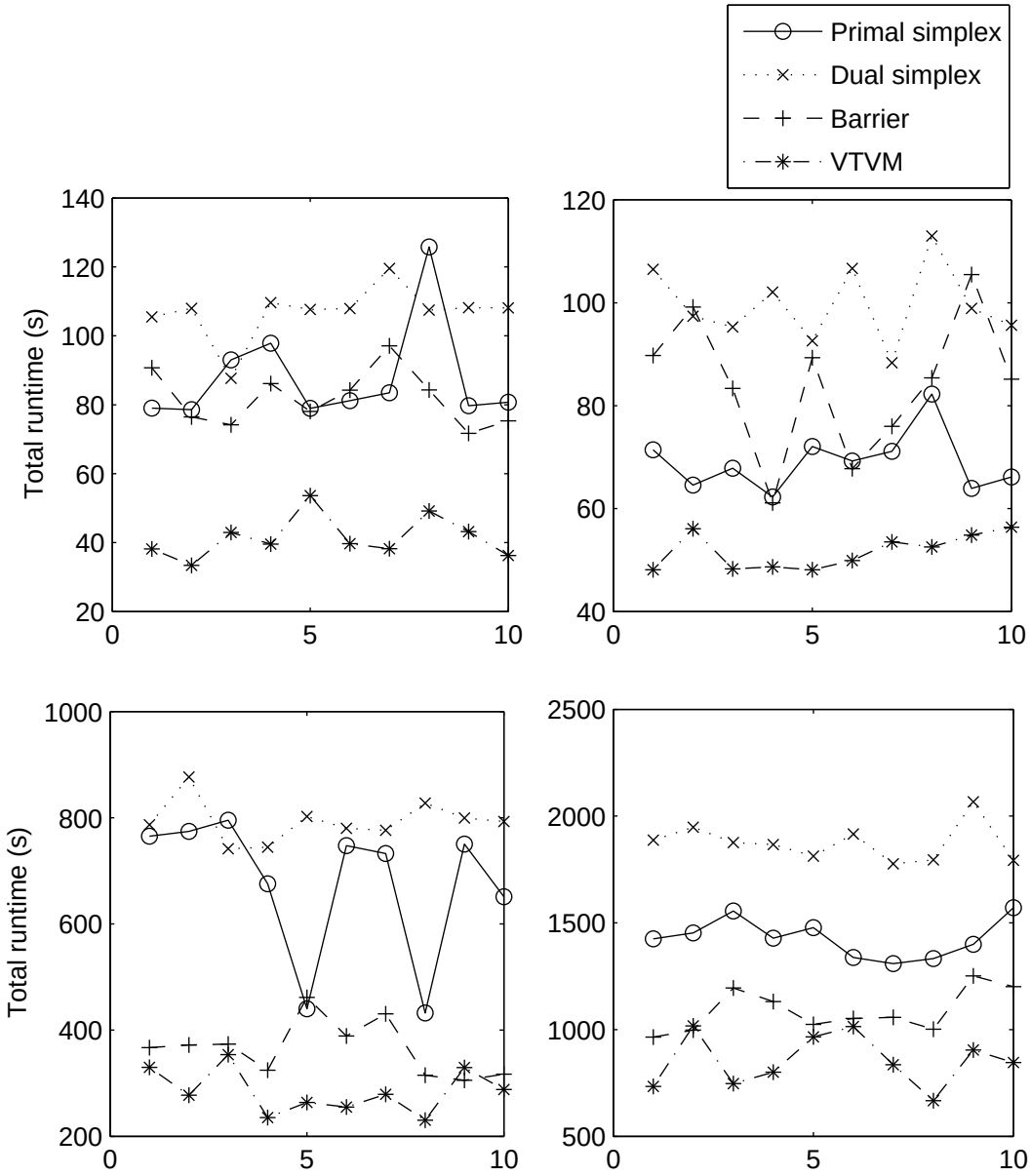

Figure 6.2: Runtimes for the four methods solving the LP relaxation. These are the same test cases as in Figure 6.2. The only difference is that instead of VTVM and the barrier method, the column generation scheme always uses the simplex method at the end when not much improvement is made. This yields primal values at termination as well as faster convergence for VTVM.

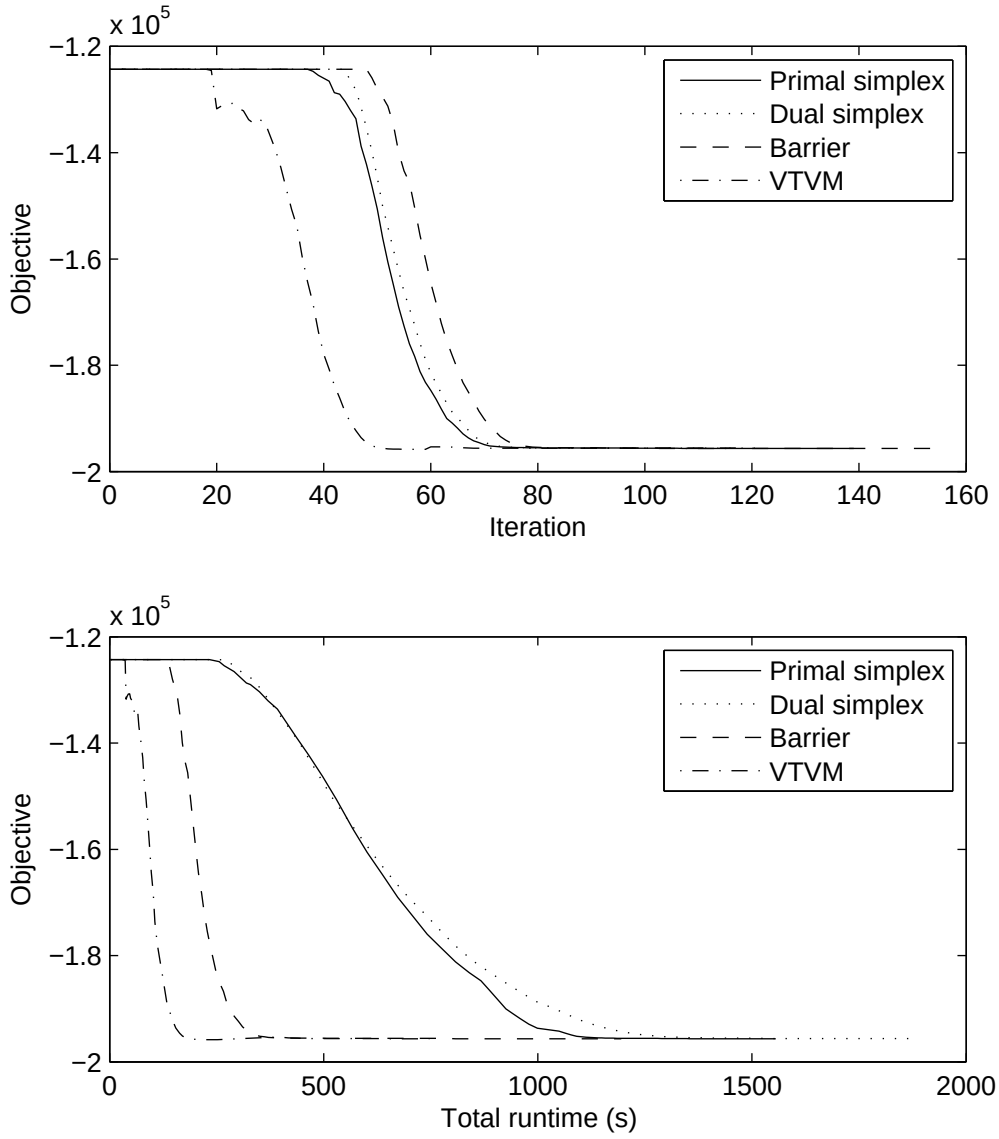

Figure 6.3: Convergence in column generation for one instance of the Atlanta254 data set. The subgradient method generates the correct columns earlier and is almost converged when the dual from the simplex method starts producing the correct columns. The upper figure displays objective value versus iteration number and the lower figure displays objective value versus total runtime.

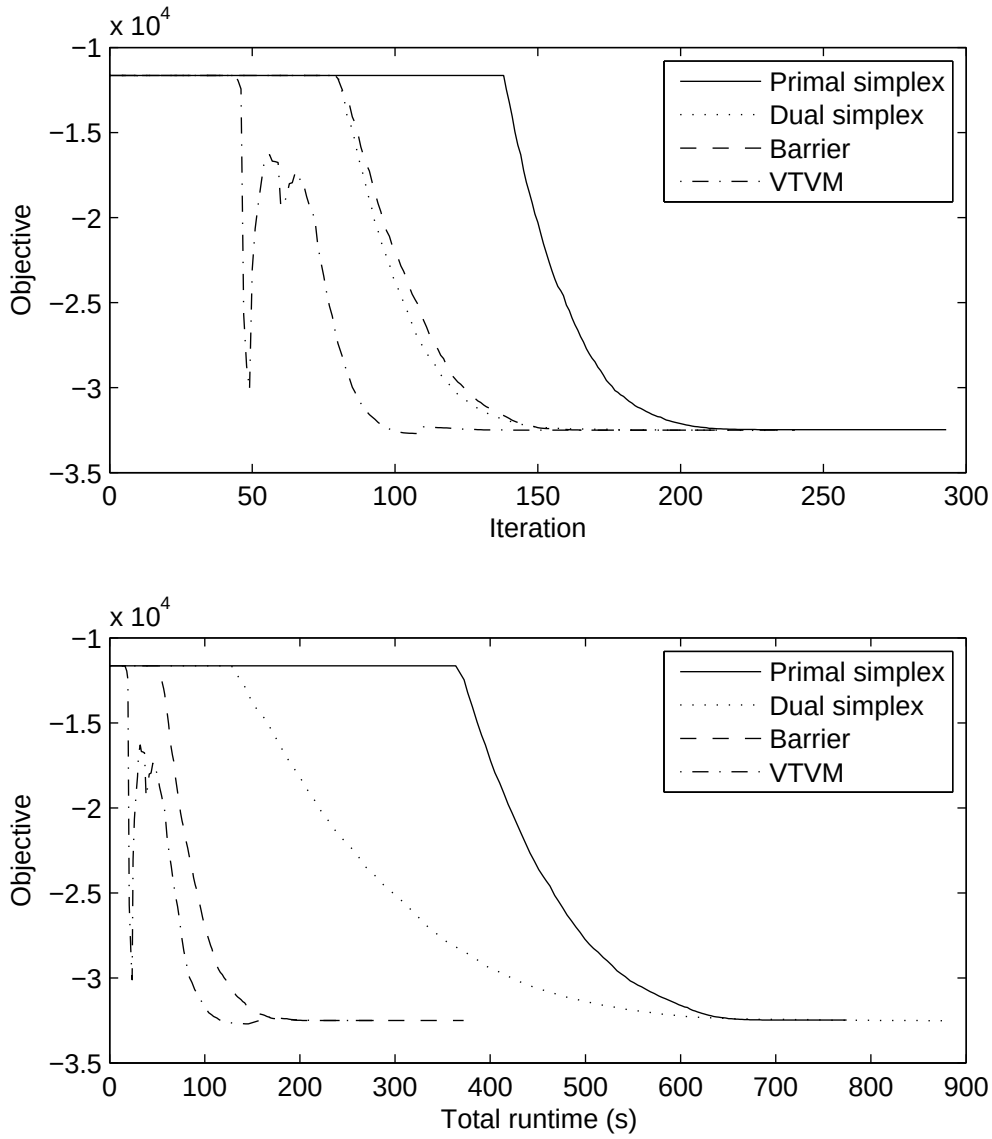

Figure 6.4: Convergence in column generation for one instance of the Boston50 data set.

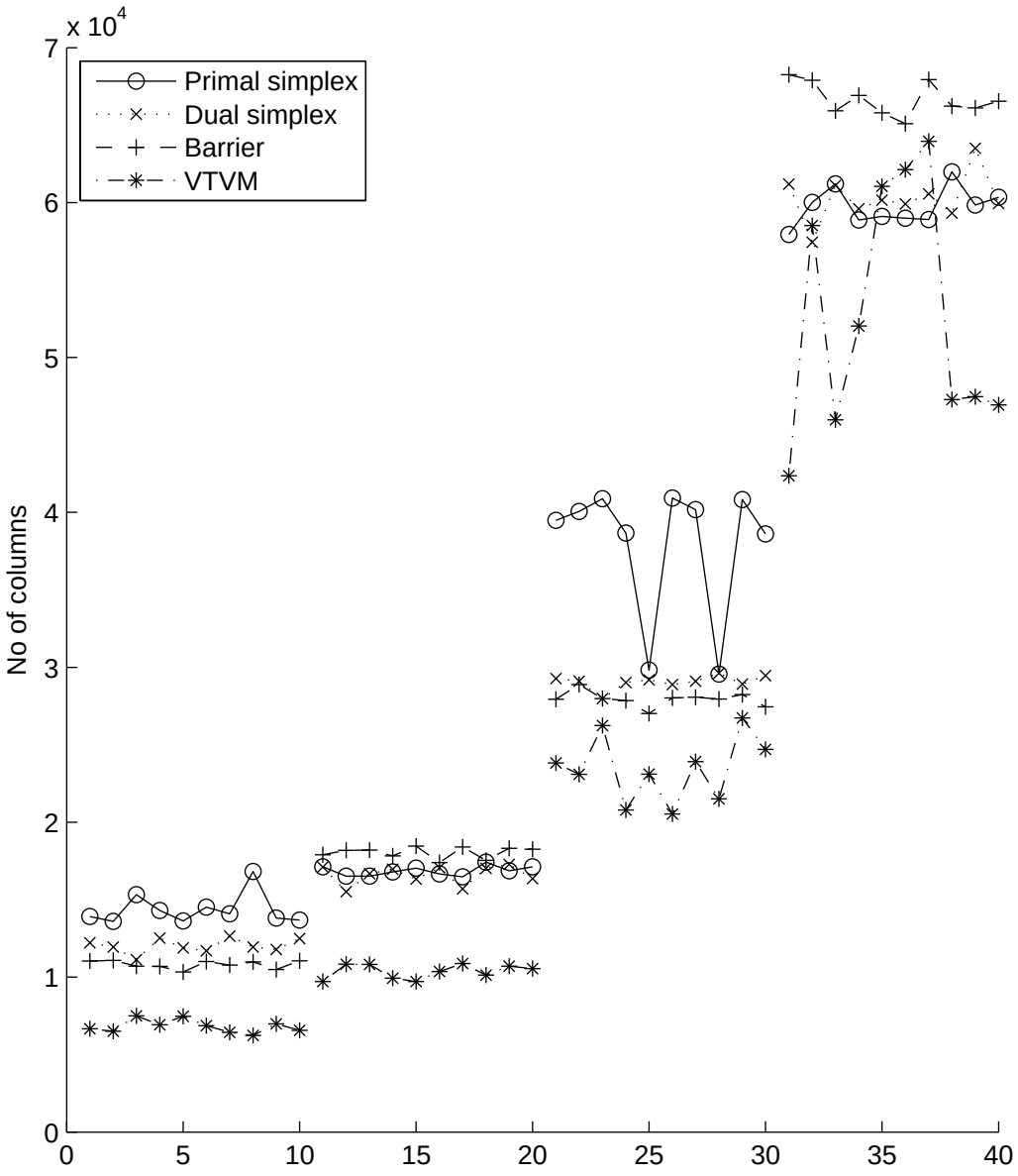

Figure 6.5: Total number of columns generated for all data sets. The subgradient method produces in most cases a final RMP that is considerably smaller.

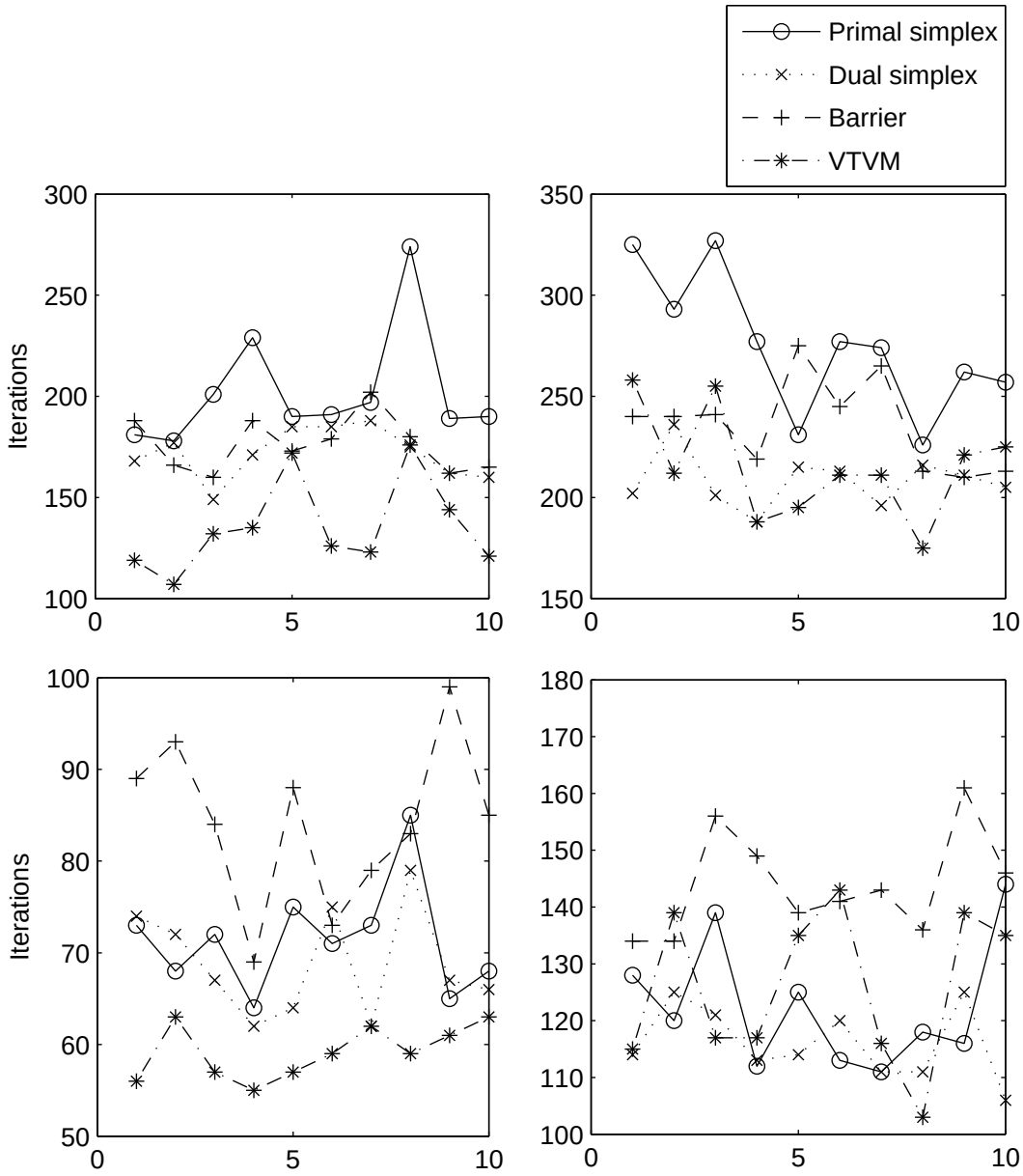

Figure 6.6: Total number of column generation iterations for all problem instances. The left figures show the Boston25 (upper) and Boston50 (lower) data sets and the right figures show the Atlanta95 (upper) and Atlanta 254 (lower) data sets.

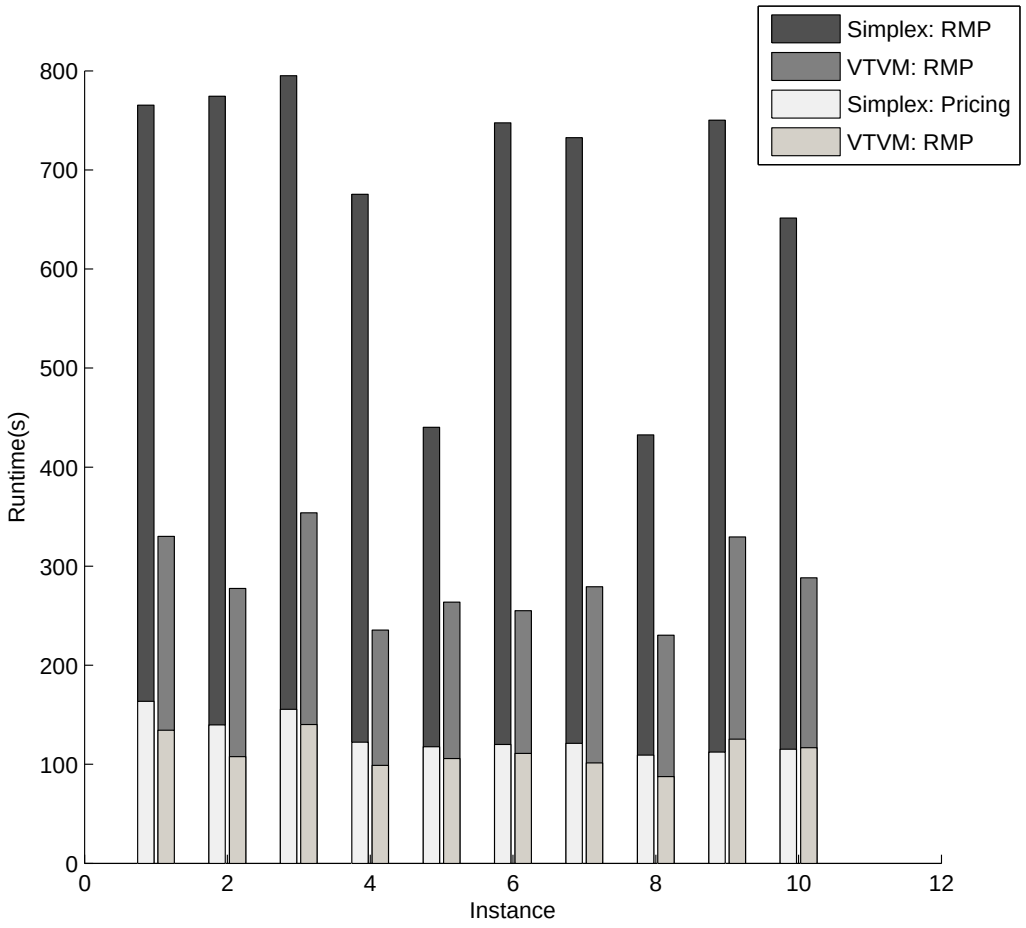

Figure 6.7: Runtimes of the largest data set, split into pricing time and RMP time.

Chapter 7

Conclusions

A variable target value method has been implemented and tested in a column generation setting for crew rostering problems. For solving stand alone problems the method in this implementation performs many times faster than the simplex method. However, this is at the cost of the solution not being optimal and without generating any primal solutions. This suggests that the VTVM is effective for quickly producing a rough lower bound for linear programs. However, the lack of primal values and suboptimal termination make the VTVM a poor choice for solving linear programs.

On the other hand, when used in a column generation setting the method performs very well, resulting in earlier convergence of the column generation and in smaller problems to solve in fixing procedures. Combining the VTVM with the simplex method terminates the column generation process faster as well as producing primal values to use for fixing. The method is faster with respect to both runtime and number of iterations, which is promising for its use in solving more difficult pricing problems.

Comparing the subgradient method to the barrier method, the subgradient method requires less memory and results in totally fewer columns generated. Thus the memory requirements in total is much smaller when using the subgradient method.

In conclusion the VTVM performs very well within a column generation context, with regard to both computational speed and memory requirements. When combined with the simplex method a nearly seamless change of LP optimizer may be made, that is, the solution quality is equivalent and primal solutions are generated. Runtimes have been cut in half for full column generation runs, where most of the time earned comes from solving the RMP more efficiently. However, only a few less iterations of the column generation scheme — which often happens when using the VTVM — may result in a drastic reduction of total runtime due to the size of the final RMP.

Further development may include a development of the algorithm to produce near-feasible primal solutions analogously with the volume method. These primal solutions could then be used in a fixing procedure to obtain a near-optimal integer solution.

Bibliography

[1] N. Andréasson, A. Evgrafov, and M. Patriksson. *An Introduction to Continuous Optimization*. Studentlitteratur, Lund, 2005.

[2] F. Barahona and R. Anbil. The volume algorithm: Producing primal solutions with a subgradient method. *Mathematical Programming*, 87(3):385–399, 2000.

[3] G. Desaulniers, J. Desrosiers, and M. Solomon. *Column Generation*. Springer, New York, 2005.

[4] J.-B. Hiriart-Urruty and C. Lemaréchal. *Convex Analysis and Minimization Algorithms II*. Springer-Verlag, Berlin, 1993.

[5] N. Kohl and S. E. Karisch. Airline crew rostering: Problem types, modeling and optimization. *Annals of Operations Research*, 127:223–257, 2004.

[6] L. S. Lasdon. *Optimization theory for large systems*. MacMillan, New York, 1970.

[7] C. Lim and H. D. Sherali. Convergence and computational analyses for some variable target value and subgradient deflection methods. *Computational Optimization and Applications*, 34(3):409–428, 2006.

[8] M. E. Lübbecke and J. Desrosiers. Selected topics in column generation. *Operations Research*, 53(6):1007–1023, 2005.

[9] K. G. Murty. *Linear Programming*. John Wiley & Sons, New York, 1983.

[10] B. T. Polyak. Minimization of unsmooth functionals. *USSR Computational Mathematics and Mathematical Physics*, 9:14–29, 1969.

[11] H. D. Sherali, G. Choi, and C. H. Tuncbilek. A variable target value method for nondifferentiable optimization. *Operations Research Letters*, 26:1–8, 2000.

[12] H. D. Sherali and C. Lim. On embeding the volume algorithm in a variable target value method. *Operations Research Letters*, 32:455–462, 2004.

[13] L. A. Wolsey. *Integer Programming*. John Wiley & Sons, New York, 1998.