MASTER'S THESIS

Advantages of utilizing new global optimality conditions when constructing column generation algorithms for integer programming problems

Martin Önneflod

Department of Mathematics
CHALMERS UNIVERSITY OF TECHNOLOGY
GÖTEBORG UNIVERSITY
Göteborg Sweden 2004

Thesis for the Degree of Master of Science

Advantages of utilizing new global optimality conditions when constructing column generation algorithms for integer programming problems

Martin Önneflod

CHALMERS | GÖTEBORG UNIVERSITY

Department of Mathematics
Chalmers University of Technology and Göteborg University
SE-412 96 Göteborg, Sweden
Göteborg, October 2004

Matematiskt centrum
Göteborg 2004

Abstract

This thesis investigates the possibilities of considering global optimality conditions, consistent also when a positive duality gap is present, when designing column generation algorithms for integer programming problems. These new optimality conditions state that δ-complementarity and Lagrangian ϵ-optimality must hold at an optimal solution to any problem. Our approach consists of considering these new findings when generating the columns that are to be used in order to solve the original integer problems. We then hope that these columns will be better than columns generated by means of traditional column generation, when searching for integer optimal solutions.

Experiments are conducted on the Generalized Assignment Problem, the Log Cutting Problem and the Crew Pairing Problem. The first two are much smaller and serve as test-problems for our new ideas. The effects of considering the new global optimality conditions when forming the column generation algorithms are presented for each of these problems. Results show that for some test problems, considerable improvements can be reached by considering the new global optimality conditions. The results for the real-world Crew Pairing Problem suggests that, even though more investigation is needed, better integer solutions can in fact be found by making relatively small changes to the original model.

Acknowledgements

I would like to thank my supervisor and examiner, Professor Michael Patriksson, for his ideas and his support during my work with this Master's Thesis. I would also like to thank many of the employees at Carmen Systems, especially my supervisor Curt Hjorring, for sharing their expertise regarding their optimization software.

Contents

Chapter 1

Introduction

This thesis investigates the possibilities of including new theoretical findings together with traditional column generation schemes when solving integer problems, in hope of reaching better solutions. Problems of various types and sizes will be considered: the Generalized Assignment Problem (GAP), the 1-dimensional Log Cutting Problem and finally the real world Crew Pairing Problem. The latter of these problems will be many times larger than the previous two, and experiments with it have been carried out at Carmen Systems, a company that develops software for solving planning and scheduling problems, especially for the airline industry.

1.1 Background

Carmen Systems is a company that develops and implements resource optimization solutions for airlines and railways, solutions which consist of applications based on the latest research combined with services such as analysis, simulation, development and support of clients' business processes.

The company has its headquarters in Göteborg, Sweden. Further, they also have offices located in Austin (Texas), Copenhagen, Leon (Mexico), Madrid, Montreal, Singapore and Stockholm. Carmen Systems works closely with a number of leading universities and this thesis is but one example of this collaboration.

1.2 Purpose of the Thesis

The purpose of this thesis is to investigate the advantages of utilizing a new class of global optimality conditions when designing column generation algorithms for integer programming problems. By utilizing the information given in this new theory, and thereby making changes to the traditional column generation algorithms, we hope to be able to generate columns which will prove to be better for solving the original integer problems.

1.3 Methods

The experiments on the GAP and Log-Cutting problems will be modeled in AMPL[12][15], and CPLEX[13] will be used as a solver for both the LP-relaxed and the integer problems. In order to determine benefits and drawbacks from including our new ideas in the problem formulation of the test problems, comparisons will be made both with respect to solution quality and time consumption.

In order to try our ideas on a real world pairing optimizer, the system at Carmen Systems AB will be used as a foundation. By adjusting parts of their code, written in C/C++, we hope to be able to incorporate our methods in their system and thereby be able to observe the effects that this will have.

As was stated above, results from all of the experiments will be compared with original results from applying traditional column generation based on the traditional global optimality conditions and LP theory. We thereby hope to be able to determine whether or not it is a good idea to include findings concerning new global optimality conditions when designing column generation algorithms. Both solution quality and time consumption will be considered.

1.4 Restrictions

Specific restrictions that are present when experimenting with some of the problems will be explained further in corresponding sections. Even though different types of problems are experimented with, as well as several instances within each problem type, it should be emphasized that the results and conclusions mainly applies to the problems experimented with. When possible, general conclusions will be stated.

Chapter 2

The Airline Planning Process

The planning and scheduling of aircraft and crews in large airlines is a very complex task. Therefore, it is very common to divide the task into several planning stages where the output from one problem defines the data for the following problem (or, step). These stages are known as: Timetable Construction, Fleet Assignment, Tail Assignment, Crew Pairing, and Crew Rostering.

It is important to involve the marketing department when the timetable is produced, in order to meet the customer's travel needs. The output from this part of the process is a number of flight legs which the airline operates.

Then, with the flights as input, the airline company must decide what type of aircraft they want to use on each flight, a process known as Fleet Assignment. The goal is to maximize profit. The result of this planning step is that a specific fleet will be assigned to each flight.

In Tail Assignment, routes are created for individual aircrafts. Here the goal is to minimize costs, which are not as obvious as in the previous steps. One will have to consider penalties for connections which are considered bad for crew or passengers. Also, it is important to see to it that each aircraft gets sufficient maintenance and also to make sure that other operational constraints are considered.

The next step, which is focused upon in this thesis, is the Crew Pairing Problem. The output from the Tail Assignment phase serves as input. The task is to construct a set of pairings with the right complement such that all flight legs are covered at least once. A pairing is defined as a sequence of flight legs for an unspecified crew member starting and ending in the same crew base (often the home city). The objective is to minimize costs such as salaries, hotel costs and deadheading costs. This has to be done while at the same time considering governmental regulations and collective agreements (block time, duty times, night stops etc.), connection times and base constraints (like a limited number of crew per base).

Crew Rostering is the step where the anonymity from the crew pairing is removed and the pairings are assigned to named individuals. The objective is to cover all pairings as well as training requirements, vacations etc. This has to be done while at the same time considering constraints such as work rules and regulations. The objective is to minimize costs. However, it is important to realize that these costs not only come from the crew rostering step, but also from the previous pairing step. Bad pairings will often lead to high costs because it is impossible to make up for them in the rostering phase. So, like in all other steps in the planning process, actions taken in one phase will probably have a great impact in following planning phases, or at least limit the possibilities in them.

There is another step in the planning process, which sometimes is not mentioned, namely operational planning. This is where unforseen events has to be taken care of, like for example delays, bad weather and sicknesses. Here the planners must try to minimize the damage and try and get back to plan as soon as possible, since changes often cause disturbances that propagate through operations. Constructing robust pairings that can withstand some disturbance is an excellent example of so called pro-active planning, where these types of problems are considered throughout the whole of the planning chain in order to prevent the above mentioned problems.

Chapter 3

The Crew Pairing Problem

Due to the fact that the Crew Pairing Problem is the problem in focus in this report, it will now be explained in more detail and a mathematical formulation of the problem will be given.

A pairing is an anonymous work plan, consisting of sequences of flight legs starting and ending at a specified home base, satisfying all rules and regulations. These rules consist of, for example, union rules, maximum duty time and maximum number of days in a pairing. Naturally, these rules vary between airline companies and also between countries. In the optimization phase, a subset of all pairings is chosen in order to minimize costs, while making sure that all flight legs are covered, i.e., gets sufficient crew. Before the mathematical formulation is given, a few definitions need to be made. First of all, a flight is a single nonstop flight. Secondly, a duty period is a sequence of flight legs with short rest stops separating it from other duty periods. Then, as has been explained before, a pairing consists of a sequence of duty periods (therefore also of flight legs) with overnight rests in between.

Let R be the number of pairings and M the number of flight legs that need to be covered. The variables x_r are then defined as:

$$x_r = \begin{cases} 1, \text{ if pairing } r \text{ is in the solution,} \\ 0, \text{ if not.} \end{cases}$$

Further, we let c_r be the calculated cost of using pairing r. We have one set of covering constraints, which corresponds to the fact that each flight leg, l, must be covered by at least b_l crew members. One has to choose between formulating the problem as a set covering or as a set partitioning problem. By choosing the former, one allows for overcovers which has to be converted into deadheads. A deadhead is a crew member who is flying as an ordinary passenger. Of course, deadheads are not a good thing to have too many of, but sometimes it is the only possible option for the planners to take. The coefficients in the covering matrix are:

$$a_{lr} = \begin{cases} 1, \text{ if leg } l \text{ is operated within pairing } r, \\ 0, \text{ if not.} \end{cases}$$

By considering the above defined settings, the following mathematical formulation of the crew pairing problem is stated:

$$\begin{aligned} min \quad & \sum_{r\in R} c_r x_r, \\ s.t. \quad & \sum_{r\in R} a_{lr} x_r \geq b_l, \qquad l \in L, \\ & x_r \in \{0,1\}, \qquad\quad r \in R. \end{aligned} \tag{3.1}$$

This integer problem is NP-complete, meaning that no polynomial time algorithm is known for its solution. Therefore, as the size of the problem grows, the computation time required to solve it becomes very large.

Chapter 4

Theory

When it comes to resource optimization problems, they often give rise to very large and difficult integer problems. Integer programming problems are much more difficult to solve than linear programming problems, and the rules and methods which apply to linear programming problems need to be modified in order to be useful. Often one has to be satisfied by getting near-optimal solutions to integer problems by means of using some clever heuristic approach.

4.1 Integer Programming

In many real world problems decisions are often needed to be made either for or against something. This is particularly true in decision making processes, where management controls a limited amount of resources and wants to maximize some profit. When many different decisions are needed to be made at the same time, the number of final propositions that can be made grows exponentially. If it is possible to evaluate all of them, then management can feel comfortable knowing that they have made the best possible choice of how to utilize their resources. However, when the number of decisions grows large, it is not possible to conduct all these evaluations, and therefore management might have to be satisfied with knowing that their final decision is, if not the best possible, at least very close to being the best possible. Often, if not always, there exists a trade-off between finding good solutions and the time spent on reaching them.

Problems like the ones described above are, when formulated mathematically, included in the integer programming class of problems. For example, decisions that are either yes or no are often modeled by the binary values 1 or 0 in the mathematical problem. Many integer programming problems belong to the so called NP-hard problems, which implies that they are difficult to solve to optimality in a reasonable amount of time. Therefore, it is always very interesting to investigate various heuristic approaches that might result in good solutions while at the same time not being too time-consuming. The Crew Pairing Problem is but one example of an integer programming problem, as are the Generalized Assignment Problem and the Log Cutting Problem.

At this point, an often met concept should be explained, namely the 'integrality gap'. It is the difference between the optimal objective value of the optimal solution to the LP-relaxed problem, z^*_{LP}, and that of the optimal integer solution, z^*_{IP}. The optimal solution to the LP relaxed problem is usually rather straightforward to get. Assuming that a feasible integer solution, z_{IP}, can be obtained, the following can be stated:

$$z^*_{LP} \leq z^*_{IP} \leq z_{IP}$$

If the value of z^*_{IP} can be predicted, this gap can serve as an indicator of how far from optimal a feasible integer solution, corresponding to z_{IP}, is. Depending on the nature of the problem, the size of the integrality gap at an optimal integer solution can vary. If the integrality gap is large for a solution produced by a column generation algorithm, it can often be assumed that no good integer solution can be created by using the generated columns. (Column generation is described in the next section.) This problem is what the ideas in this thesis aim at handling by implementing new ideas in the solution process, making the columns better for solving the original integer problems.

4.2 Column Generation for Integer Programs

Column generation is most often used in integer programming when it is impractical to explicitly consider all variables when optimizing the problem (due to excessive computation time). The main idea is therefore to only consider a subset of these when solving the problem. This can be seen as a restriction of the problem, not allowing it to take on as many different solutions, but it possibly speeds up the solution process considerably. In order for column generation to be successful, it is important to focus on generating good columns, thereby not considering the ones that never would be used when forming the integer solution.

The original integer problem is decomposed into a master problem and a sub problem (also referred to as the pricing problem). The columns (corresponding to variables) in the master problem can be defined as optimal solutions to the Lagrangian relaxed problem. The master problem then becomes a problem of convex combining these variables, while fulfilling some additional constraints, in order to either minimize or maximize the objective function.

4.2.1 The Restricted Master Problem

Since the final experiments in this thesis are conducted on the Crew Pairing Problem, the example below is such a problem. The LP master problem of a set covering Crew Pairing Problem can look like the following:

$$\begin{aligned} min \quad & \sum_{r \in R'} x_r c_r, & \\ s.t. \quad & \sum_{r \in R'} a_{lr} x_r \geq b, & l \in L, \\ & x_r \in [0,1], & r \in R'. \end{aligned} \tag{4.1}$$

Here L is the set of flight legs, and the set R' is a subset of all possible pairings, R, generated from the sub problem. In order to be able to generate such pairings (corresponding to columns), the pricing problem must be solved and good pairings must be identified.

4.2.2 The Sub/Pricing Problem

The purpose of the sub problem is to generate a new column, which minimizes the Lagrangian function corresponding to a dual vector obtained from solving the restricted master problem. Also, by 'moving' some of the constraints to the sub problem, it is possible to get columns that are feasible with respect to these constraints. In order for a column to be useful from an LP perspective, the reduced cost associated with it needs to be negative (assuming that the master problem is a minimization problem). If it is, adding the column to the restricted master problem will enable a better LP objective value to be reached.

The procedure of finding new columns is very similar to what is being done in the simplex method. There, a reduced cost criterion is used to select which variable that should enter the basis. If no negative (assuming that a minimization problem is considered) reduced cost variables exist, then the current basis is optimal. See [4]. The same can be proven for column generation. If no more negative reduced cost columns can be found, then the current LP solution is in fact the optimal solution for the entire LP master problem.

So, knowing that the sub problem corresponds to finding columns with a negative reduced cost, the following can be considered as a sub problem corresponding to the above presented Crew Pairing master problem. The formulation is similar to the one presented in [6]:

$$\begin{array}{ll} min & (c_r - \pi^T a_r)x_r, \\ s.t. & x_r \in \{0, 1\}, \end{array}$$

where c_r is the cost of pairing r, π is the vector of dual variables in an optimal dual solution, and a_r is the column associated with the pairing. Finding the pairing that corresponds to the solution of the above defined pricing problem thus equals finding a new variable, with an associated cost and column, that will further improve the objective value of the LP relaxed master problem.

This problem is of different nature for different problems. For Crew Pairing Problems one often solves a k-shortest path problem. When a k-shortest path problem is solved, the cost of the edges in the network gets updated between each iteration using the dual variables, as is described in [11]. This process continues until no more attractive columns are found.

4.2.3 Integer solutions

Since column generation algorithms, as the one described above, only gives a feasible solution to a convexified problem (the master problem), no guarantee exists that this solution will be integer valued. Therefore, when using column generation for solving integer programming problems, some sort of branching is often done in order to reach high-quality integer solutions. It is important to be aware of the fact that is not enough to generate new columns only in the root-node of the search tree. One needs to perform a so called Branch-and-Price, i.e. where new columns are generated in each node after each branching decision. By doing this, better solutions can be reached [3]. It is not necessary to solve the problem to optimality in each node, although that might lead to better bounds. It is a trade-off between solution quality and time consumption.

4.3 Decomposition Methods

Methods closely related to the basic column generation principle are so called decomposition methods. The most famous decomposition method is the one known as Danzig-Wolfe decomposition and is based on the fact that a problem with many independent blocks in the constraint matrix can be decomposed into a number of more or less easily solved subproblems. Therefore, the approach becomes more and more attractive as the numbers of such independent blocks grow, according to [1]. The idea is to use a column generation scheme where the subproblem solutions are added together in a master problem under the restriction of convexity constraints. So, dual variables are sent to the subproblems which are solved independently thereby giving new columns to the master problems.

The LP-relaxed master problem is then solved over new variables, say $\lambda_i \in \{0, 1\}$, which indicate to what extent the subproblem solution i (or column) is used. As stated above, convexity constraints on these new variables must be included. The master problem thus gives a solution consisting of a convex combination of subproblem solutions. This in turn explains the reason for reformulating many problems by applying Danzig-Wolfe decomposition. It is in fact so that the decomposed formulation of the LP relaxed problem often is tighter than the LP relaxation of the standard formulation. This is due to the fact that solutions that are not convex combinations of what is often 0-1 subproblem solutions are not feasible in the decomposed formulation. Therefore, a better primal bound can hopefully be obtained.

As was explained in the section above, there is no guarantee that the optimal solution to the LP-relaxed problem will be integer. Therefore, some sort of heuristic or branching approach might be needed in order to obtain a primal feasible solution to the original problem.

4.4 Obtaining Integer Solutions

Before getting into how the branching step should be performed, remember that the integrality gap can be used as a measure of how close to optimality the currently best solution is. If the gap is large, refinement of the solution is needed - probably consisting of applying some kind of rule.

When it comes to determining what kind of branching decisions that should be used, there are many suggestions to be read about in recent as well as in older publications. To define what a branching scheme is, one can think of it as a method of dividing the solution space in such a way that the current fractional solution is excluded, integer solutions remain intact, and finiteness of the algorithm is ensured, all according to [8]. Further, also in accordance with [8], some general "rules of thumb" are said to be good to have in mind. It is important to 1) balance the search tree, i.e. to produce branches of possibly equal size, 2) to make important branching decisions early in the tree and 3) to try and prevent columns that have been branched on from being regenerated later. Neglecting the last rule could in some cases lead to having to find the k:th best subproblem solution instead of the optimal one. However, when doing so is less complicated than to prevent regeneration it could still be preferable.

Hopefully, applying clever branching rules will enable better integer solution to be found, thereby decreasing the size of the gap $z_{IP} - z_{LP}^*$.

4.4.1 Set Covering Problems

In the context of set covering problems, branching rules are often formulated so that two rows in the cover-constraint matrix are branched upon - one branch where they must be covered by two distinct columns, and one where they must be covered by the same column. These rules can easily be added to the pricing problem, without giving rise to any structural difficulties. In order to make this branching effective, we want to branch on the original variables of the compact formulation, not on the variables in the disaggregated formulation. As is explained in [3], branching on the variables in the disaggregated master problem will probably lead to the same illegal columns getting generated all over again. This will in turn then give rise to the need of finding the next best solution in the pricing problem. At depth l in the branching tree this means that the l^{th} best solution may need to be found for the pricing problem.

4.4.2 Tailing-off

When considering how to construct a Branch-and-Price algorithm is, as mentioned in [8], one should be aware of the so called tailing-off effect. Remember that Branch-and-Price is when column generation is performed in each node of a search tree, i.e., not only in the root node. Tailing-off is a phenomenon which occurs when the column generation process suffers from poor convergence, usually when some sort of simplex-based solver is involved. There are various ways in which to avoid such problems and being aware of them is important throughout the whole of the algorithm design process. For example, one can instruct the algorithm to stop generating columns when tailing-off occurs and to make a branching decision much earlier than otherwise. This can of course be viewed as a trade-off between solution quality and computation time, a trade-off often encountered in integer programming. For further insight I refer to the above mentioned source.

There exist many different branching rules, and it is up to each and every OR practitioner to find which rules apply to his or her problem.

4.5 Lagrangian Duality and Lagrangian Relaxation

Consider the following integer problem:

$$\begin{aligned} min \quad & c^T x, \\ s.t. \quad & Ax \geq b, \\ & x \in X, \end{aligned} \tag{4.2}$$

where $X \subset \Re^n$, $A \sim m \times n$, and $b \sim m \times 1$. Further, $Ax \geq b$ are somewhat complicating constraints, making the problem difficult to solve. These constraints can then be handled by adding them to the objective function with a penalty term $u(b - Ax)$. Here, u is the vector of dual variables or Lagrange multipliers associated with the constraints $Ax \geq b$. The new problem, also referred to as the Lagrangian problem, then looks like:

$$\begin{aligned} min \quad & c^T x + u(b - Ax), \\ & x \in X, \end{aligned} \tag{4.3}$$

where $u = (u_1, ..., u_m) \geq 0$. Hopefully, this problem will be easier to solve. However, depending on the fact that the solution to the Lagrangian relaxed problem might be infeasible in the original problem, the Lagrange multipliers might need to be updated. It is possible to show that there exist multipliers such that an optimal feasible solution can be obtained, if and only if the continuous relaxation has an optimal integer solution. This is seldom the case, and then some heuristic approach is needed!

One common way to get these multipliers is by applying a sub-gradient algorithm to the resulting Lagrangian subproblem. This is an iterative process which is often set to terminate after a fixed number of iterations. However, some sort of heuristic approach often is needed in order to get a feasible solution.

4.5.1 Lagrangian relaxation and Column Generation

Since we are particularly interested in column generation and its relations to Lagrangian relaxation it is also worth pointing out that there exists a very close connection between the two concepts. Let us consider the following problem:

$$\begin{array}{ll} min & c^T x, \\ s.t. & Ax \geq b, \\ & x \in \{0,1\}. \end{array} \tag{4.4}$$

Now we consider Lagrangian relaxing the set covering constraints, with some multiplier u. The Lagrangian dual function $\theta(u)$ then looks like:

$$\theta(u) := min_{x \in [0,1]}\{c^T x + u(b - Ax)\} \tag{4.5}$$

It follows from LP duality theory that the solution of $max_{u>0}(\theta(u))$ gives the same objective value as the LP-relaxation of (4.4). Let A^0 be some initial matrix in the problem including only a subset of all columns in A, and let u^0 be the multipliers related to the optimal solution for this problem. Now, in order to relate this to column generation, consider adding another column a_i with cost c_i to the problem. The minimization then becomes:

$$min_{x^0, x_i}\{u^0 b + (c^0 - u^0 A^0)x^0 + (c^i - u^0 a_i)x_i\} \tag{4.6}$$

It is easily seen that if $(c_i - u^0 a_i) \leq 0$ (i.e., if the the reduced cost is negative), then x_i will be part of the solution. That will improve the objective, given the dual vector u^0.

Since the problem in a column generation sub problem is that of finding the column which has the most negative reduced cost, the connection between column generation and Lagrangian relaxation now becomes quite clear. Let the new column correspond to the variable x_i. If we want to add it to the restricted master problem, it must hold that $(c_i - u^0 a_i) \leq 0$. Then, the new column is said to ‘price out’.

If the new column is added, the matrix A becomes larger and now looks like $A^1 = (A^0, a_i)$.

Resolving the maximization problem (the corresponding Lagrangian dual problem) with this new matrix will give a new set of Lagrange multipliers, u^1, which are in turn used to price out new columns. This procedure goes on until no more negative reduced cost columns are found. So, in light of what has just been said, it is clear that column generation is closely related to duality theory, and Lagrangian relaxation in particular. Minimizing the Lagrangian function over (x^0, x_i) thus equals finding the column a_i with the most negative reduced cost $(c_i - u^0 a_i)$, given that the dual multipliers are known from solving the LP relaxed master problem.

Another way of presenting the similarities between column generation and Lagrangian relaxation, as is done in [10], is that the subproblem which needs to be solved in the column generation procedure is the same as the one we have to solve when using a Lagrangian relaxation approach, except for a constant in the objective function. For column generation, the duals are obtained from solving the LP relaxation of the master problem, while for Lagrangian relaxation, the Lagrangian multipliers are usually updated by sub-gradient optimization.

4.5.2 Duality Gaps

From LP duality theory we know that the optimal value given by maximizing the Lagrangian function, $\theta(u)$, is equal to that obtained by minimizing the primal problem. However, many problems require the primal variables to take on only integer (perhaps even binary) values, especially problems concerning resource optimization and decision making processes. Then, however, strong duality does not always hold. This is a well known phenomenon, and the difference between the optimal value of the original problem and the optimal value of the considered Lagrangian dual problem is called the duality gap. In other words, the duality gap is:

$$\Gamma = f^* - \theta^*, \tag{4.7}$$

where f^* is the optimal value of the primal problem and θ^* the optimal value of the Lagrangian dual problem. Since the global optimality conditions for linear programming problems do not hold when a positive duality gap is present, there exists a need for formulating new conditions which will apply to problems where a positive gap is present. The next section will present such a system, and also explain why and how it can be used in the context of column generation.

4.6 New Global Optimality Conditions for Integer Programming

This section will go through the theoretical foundation upon which the thesis is based. Suppose we have at hand a programming problem such as:

$$\begin{aligned} &min && f(x), \\ &s.t. && g(x) \leq 0^m, \\ &x \in X, && \end{aligned} \tag{4.8}$$

where f and g are real-valued functions, and X is non-empty and closed. The Lagrangian function (i), the dual function (ii) and the Lagrangian dual problem (iii) can then be stated as:

$$\begin{aligned} &(i) \quad L(x,u) = f(x) + u^T g(x) \\ &(ii) \quad \theta(u) := min_{x \in X} \ \{f(x) + u^T g(x)\}, \qquad u \in R^m \\ &(iii) \quad \theta^* := max_{u \in R^m_+} \ \theta(u) \end{aligned} \tag{4.9}$$

The duality gap is defined as: $\Gamma := f^* - \theta^*$. The well-known global optimality conditions are often stated as:

$$\begin{aligned} f(x) + u^T g(x) &\leq \theta(u), \\ g(x) &\leq 0^m, \\ u^T g(x) &= 0, \end{aligned} \tag{4.10}$$

where $(x, u) \in X \times R^m_+$. Such a primal-dual pair, i.e., a pair satisfying these optimality conditions, is also said to be a saddle-point of the Lagrangian function $L(x, u) = f(x) + u^T g(x)$.

As has been stated in previous sections, is is a well known fact that the optimality conditions (4.10) including Lagrangian optimality, primal-dual feasibility and complementarity are inconsistent when the problem at hand has a positive duality gap (i.e., when $\Gamma > 0$). This is often the case when problems such as integer programs are considered, like for example the Crew Pairing Problem.

A suggestion of how to formulate new global optimality conditions, consistent also when a duality gap is present, was presented in [9]. The new conditions look as below for a given pair (x, u), where $(x, u) \in X \times R^m_+$.

$$\begin{aligned} f(x) + u^T g(x) &\leq \theta(u) + \epsilon, \\ g(x) &\leq 0^m, \\ u^T g(x) &\geq -\delta, \\ \epsilon + \delta &\leq \Gamma, \\ \epsilon, \delta &\geq 0, \end{aligned} \tag{4.11}$$

where both ϵ and δ are non-negative numbers. An optimal solution (x, u) satisfying the above system then fulfills the following: Lagrangian ϵ-optimality and (when inequality constraints are present) δ-complementarity (a relaxed complementarity condition). For the proof, see [9].

It is also so that when (u, x) are optimal in the primal and dual problem respectively, then the size of ϵ+δ exactly equals the size of the duality gap, Γ. It is easily seen that these optimality conditions are a generalization of the ordinary ones, and that when ϵ and δ are zero, then the system is in fact the same as the classic one.

A relaxed version of the new optimality conditions, which is consistent also for near-optimal primal-dual pairs (x, u), then looks like:

$$\begin{aligned} f(x) + u^T g(x) &\leq \theta(u) + \epsilon, \\ g(x) &\leq 0^m, \\ u^T g(x) &\geq -\delta, \\ \epsilon + \delta &\leq \Gamma + \kappa, \\ \epsilon, \delta, \kappa &\geq 0. \end{aligned} \tag{4.12}$$

A proposition that is connected with the above relaxed optimality conditions is given in [9]. In order to be able to explain the interpretations it will be presented almost exactly as its original version.

PROPOSITION (near-optimal solutions and the system above)(as presented in [9])

(a) (near-optimality in the primal problem (4.8)).
Let $(x, u) \in X \times R^m_+$. Suppose that, for some $\epsilon, \delta, \kappa \geq 0$ that the system above is consistent. Then x is feasible in the original primal problem (4.8), and

$$f(x) \leq \theta(u) + \Gamma + \kappa.$$

Suppose further that u solves the dual problem (giving $\theta(u) = f^* - \Gamma$). Then,

$$f(x) \leq f^* + \kappa.$$

(b) (near-optimality in the Lagrangian problem).
Suppose that $(x, u) \in X \times R^m_+$ is β-optimal and α-optimal in the primal and dual problem, respectively, for some $\alpha, \beta \geq 0$. Then,

$$\theta(u) \leq f(x) + u^T g(x) \leq \theta(u) + \Gamma + \alpha + \beta.$$

Suppose further that $(x, u) \in X \times R^m_+$ solves the primal and dual problems (4.8) and (4.9) (iii) respectively. Then,

$$\theta(u) \leq f(x) + u^T g(x) \leq f^*.$$

(c) (near-complementarity).
Suppose that $(x, u) \in X \times R^m_+$ is β-optimal and α-optimal in the primal and dual problem respectively, for some $\alpha, \beta \geq 0$. Suppose further that $\epsilon \geq 0$ is such that (4.12) (a) holds with equality. Then, (4.12) holds, with $\delta := \Gamma - \epsilon + \alpha + \beta \geq 0$ and $\kappa := \alpha + \beta$. In fact,

$$\Gamma + \beta + \alpha \leq \epsilon + \delta \leq \Gamma + \kappa$$

always holds when (4.12) is consistent. Suppose further that $(x, u) \in XxR^m_+$ is β-optimal and α-optimal in the primal and dual problem respectively. Then, (4.12) holds, with $\delta := \Gamma - \epsilon \geq 0$

and $\kappa = 0$.

Proposition (a) implies that a vector x that is near-optimal in the Lagrangian problem and near-complementary is a near-optimal solution in the original primal problem. Of course, as soon as the value of κ becomes close to zero, then the solution will be very close to optimal. So, when searching for an optimal solution, we want to minimize κ, making $\epsilon + \delta$ close to the size of the duality gap.

The second part of the proposition (b) shows that a near-optimal solution to the primal problem also must be near-optimal in the Lagrangian problem defined at a near-optimal dual solution.

The third part (c) shows that a near-optimal solution to the primal problem must also be near-complementary. It also shows the relationship between the perturbation parameters ϵ, δ and Γ.

So, an optimal solution to an integer problem might not be optimal in the Lagrangian subproblem and (when inequality constraints are present) it might also violate complementarity. At a feasible solution, the sum of these two perturbations is always at least as large as the size of the duality gap, and at an optimal solution, exactly equal.

Therefore, when applying LP-relaxations and LP-theory to a process where the original problem is integer, it might be beneficial to consider these findings. In a column generation scheme, where columns are generated in a subproblem, this corresponds to including complementary constraints in the master problem (δ-complementarity), and also generating non-optimal columns in the subproblem (ϵ-optimality). Then the Lagrangian optimality perturbation, ϵ, is the sum of perturbations in the independent subproblems. The near-complementarity condition can also be separated into several conditions, one for each relaxed constraint. Then the parameter δ equals the sum of all of these individual perturbations.

4.6.1 Previous Experiments

The above optimality conditions have been experimented on and Lagrangian heuristics which fulfill near-complementarity and Lagrangian near-optimality have been investigated in [9].

The results showed that by considering complementarity fulfillment in the objective associated with a Lagrangian multiplier, it was possible to significantly improve the quality of the resulting primal integer solution. Although this was only tests conducted on greedy strategies for set covering problems, the results have made further investigations concerning complementary fulfillment in other problems interesting. For a full insight in the above discussed experiment, see [9].

4.6.2 Applications to Column Generation

The new global optimality conditions show that at an optimal solution to an integer problem, the size of the duality gap precisely equals the sum of the two perturbations δ and ϵ. The first perturbation, δ equals the complementarity violation of the considered constraints (in the master problem) and the second perturbation, ϵ, is the level of optimality violation with respect to the Lagrangian problem (subproblem). Throughout this section, keep in mind that solving a minimum reduced cost problem in a column generation process is in fact the same as minimizing the Lagrangian function given the multipliers associated with the relaxed constraints.

The applications concerning column generation stem from the idea of adding a new constraint to the restricted master problem which then forces a feasible solution to also satisfy near-complementarity. This is important since a near-optimal solution must in fact, according to the proposition above, be near-complementary. For a disaggregated master problem this extra constraint may look like this:

$$\sum_{j=1}^{n} \sum_{i=1}^{p_j} (\overline{u}^T A_j x_j^i) \lambda_j^i \leq \overline{u}^T b + \delta,$$

where $\overline{u}$ is non-negative and near-optimal in the Lagrangian dual problem, and p_j is the number of points in the set $X_j, j = 1...n$. When designing a column generation algorithm, one could argue that we should add a constraint limiting the level of complementarity violation in the master problem by choosing a parameter δ based on some clever investigation. However, it is not possible (at least not obvious) to know in advance the value of the two perturbations, even if the size of the duality gap is known. As was shown in [9], δ and ϵ can vary, even between optimal solutions to the same problem.

Therefore, when it comes to column generation algorithms, it is probably a good idea to add these new constraints (near-complementarity) to the restricted master problem and to relax them with a fixed positive multiplier, w. Then one could use traditional column generation on the resulting problem, with the only difference being that now the objective also considers near-complementarity fulfillment. The subproblem then has an objective consisting of two terms, the original reduced cost and now also a complementary term. The hope is that better columns will be generated with respect to solving the original integer problem.

However, it is not obvious to what extent near-complementarity should be considered in the objective, especially since the level of complementarity can vary even between optimal solutions to a single problem. In order to generate columns which are both near-complementary and near-optimal in the subproblem, one can also try to let the multiplier, w, vary throughout the generation process.

Another idea could be to consider the complementarity constraint in the sub problem only, thereby not having to consider it at all in the master problem. Then one could relax it into the sub problem objective, which would then become similar to what was described above. However, the master problem will remain the same as it is for a traditional column generation approach.

It is also interesting to note that the authors in [9] prove that it is possible to estimate the quality of the solution to the side-constrained restricted master problem, which we can call f_r^*. They prove that, if we have an optimal integer solution to the complementary constrained restricted master problem, then the difference between that and the value of the original master problem, f^* can be stated as:

$$f_r^* - f^* \leq (\sum_{j=1}^{n} \max_{i=i,\dots,p_j} \overline{c}_j^i) + \delta,$$

where $\overline{c}_j^i$ corresponds to the reduced costs for the variables in the restricted master problem.

Chapter 5

Experiments

This chapter will focus on testing whether or not considering the new global optimality conditions when constructing column generation algorithms will lead to more time-efficient algorithms and/or better solution quality. The experiments are conducted on the Generalized Assignment Problem and the 1-dimensional Log Cutting Problem. By conducting these examples we hope to gain valuable knowledge which also can be used when working with the real-world Crew Pairing Problems at Carmen Systems.

5.1 The Generalized Assignment Problem

Like the Crew Pairing Problem, the Generalized Assignment Problem is an NP-complete optimization problem and therefore suffers from considerable computational effort requirements when the size of the problem increases. This is why methods such as column generation schemes, where the number of possible solutions considered is made much smaller, is of great interest. For these types of problems, there is always a trade-off between solution quality and time consumption.

A Generalized Assignment Problem is characterized by the fact that one shall assign jobs to different agents (or machines) without exceeding the capacity for each agent. Further, each job must be assigned to an exactly one agent. The objective is either to minimize costs or to maximize profits. The basic problem formulation is similar to the one given in [5] and looks as follows:

$$\begin{array}{lll} min & \displaystyle\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij}, & \\ s.t & \displaystyle\sum_{i=1}^{m} x_{ij} = 1, & j \in \{1,...,n\}, \\ & \displaystyle\sum_{j=1}^{n} w_{ij} \leq c_i, & i \in \{1,...,m\}, \\ & x_{ij} \in \{0,1\}, \quad i \in \{1,...,m\}, j \in \{1,...,n\}, & \end{array} \tag{5.1}$$

where c_{ij} is the non-negative cost associated with assigning job j to agent i. The idea which will be used to solve this problem, which is also done by in [5], is to form the so called disaggregated master problem by applying Dantzig-Wolfe decomposition on the standard formulation, placing the knapsack constraints in the subproblem. Let $K_i = \{x_1^i, x_2^i, ..., x_k^i\}$ be the set of all feasible assignments of jobs to agent i, i.e where $x_k^i = \{x_{1k}^i, x_{2k}^i, ..., x_{nk}^i\}$ is feasible with respect to

$$\begin{aligned} &\sum_{j=1}^{n} w_{ij} x_{jk}^i \leq c_i, \\ &x_{jk}^i \in \{0,1\}, \qquad j \in \{1,...,n\}, \end{aligned} \tag{5.2}$$

Then, let y_k^i be a 0-1 variable indicating whether or not assignment x_k^i is selected for agent i. This makes it possible to give the following formulation:

$$\begin{aligned} min \quad & \sum_{i=1}^{m}\sum_{k=1}^{k_i}(\sum_{j=1}^{n} c_{ij} x_{jk}^i) y_k^i, \\ s.t \quad & \sum_{i=1}^{m}\sum_{k=1}^{k_i} x_{jk}^i y_k^i = 1, \qquad j \in \{1,...,n\}, \\ & \sum_{k=1}^{k_i} y_k^i \leq 1, \qquad i \in \{1,...,m\}, \\ & y_k^i \in \{0,1\}, \qquad i \in \{1,...,m\}, \quad k \in K_i. \end{aligned} \tag{5.3}$$

The variables in the above formulation, now also subjected to convexity constraints, are generated by solving the following minimization problem, also referred to as the pricing problem.

$$min_{1 \leq i \leq m}\{z(KP_i) - v_i\}, \tag{5.4}$$

where v_i are dual variables corresponding to the convexity constraints of agent i, and where the $z(KP_i)$ are the optimal values in the following knapsack problem:

$$\begin{aligned} min \quad & \sum_{j=1}^{n}(c_{ij} + u_j) x_j^i, \\ s.t. \quad & \sum_{j=1}^{n} w_{ij} x_j^i \leq c_i, \\ & x_j^i \in \{0,1\}, \qquad j \in \{i,...,n\}. \end{aligned} \tag{5.5}$$

Here u_j is the dual optimal price corresponding to the partitioning constraint for job j from the solution to the RMP. Columns are found interesting if they give a reduced cost which is negative and lowers the objective value of the RMP. This means that a knapsack problem is solved for each agent and then the overlying pricing problem gives the column which will be added to the RMP.

The underlying reason to why the standard formulation of the GAP is decomposed using Danzig-Wolfe decomposition is, according to [3], that the disaggregated version produces better and tighter LP bounds. This can be understood by considering the fact that fractional solutions that are not convex combinations of solution to the knapsack problem are allowed in the original problem, whereas they are not in the disaggregated formulation. When comparing the two formulations with respect to solution process simplicity, the LP relaxation of (5.3) may in fact be harder to solve than that of (5.1), but can be motivated by the above mentioned tighter bounds.

5.1.1 Original Settings

In order to start the generation process it is necessary to have some sort of initial solution to the restricted master problem. An obvious approach, which is also suggested in [5], is to use one column for each machine corresponding to an optimal knapsack solution, and also one extra dummy column corresponding to all ones but with a large cost (or zero profit) associated with it. This will ensure that the problem is feasible and that duals can be generated and sent to the pricing problem, thereby enabling the column generation process to start.

Once the column generation process has started, the RMP is solved and the duals are sent to the pricing problem. Here a new possible assignment is generated if and only if it has a negative reduced cost. If this is not the case, then the generation terminates and the original integer problem is solved using some integer heuristic. In our case, the CPLEX MIP (mixed integer programming) solver is used.

The above described column generation procedure was implemented in AMPL, and the problems were obtained from [14]. Results showed that for some problem instances, relatively good integer solutions were found, but rarely the optimal ones. Perhaps more interesting was that this procedure failed to even produce a feasible integer solution for some problems.

Having these failures and non-optimal solutions in mind while also considering the new global optimality conditions, it seems reasonable to somehow also consider them and to update our column generation algorithm accordingly. Theory suggests that, for problems such as this with a duality gap, we must consider the fact that Lagrangian ϵ-optimality and δ-complementarity must hold at optimal integer solutions. Since the GAP problem only has equality constraints, the only thing we can benefit from is by considering Lagrangian ϵ-optimality. In a column generation scheme, on obvious way of doing this is to generate not only optimal solutions (or columns) in the pricing problem, but also non-optimal ones. For the integer optimal solution, it must hold that the size of the duality gap, Γ, exactly equals the sum of all optimality perturbations, ϵ, of the variables in the optimal solution.

5.1.2 Considering ϵ-optimality

One way of generating non-optimal solutions to the pricing problem is by dynamically adding constraints saying that the objective (reduced cost) cannot be as good as it was for the best column, then resolve the problem. This can be done several times and all generated columns can be added to the master problem, not only the best one. The obvious drawback from using this method is the fact that the pricing problem needs to be re-optimized again and again when we are looking for more that one column, thus implying that much more time needs to be spent in the pricing problem. Another problem is that we will increase the computational effort required in order to solve the restricted master problems. However, it is interesting to see whether or not better solution quality can be obtained.

5.1.3 Results

The lower part of Table 5.1 shows the results where one extra column has been generated per machine and per pricing run. The upper part is traditional column generation as described in [5]. No branching is ever considered. The lower part of the table corresponds to the results when one extra column, i.e. the next best column, also is generated and added to the master problem in each iteration. It should also be explained that problem type x-y means the problem of assigning x jobs to y machines (or agents). The examples used in this experiment are randomly generated and taken from [14]. They are maximization problems (which is realized by studying Table 5.1) which have been reformulated and solved as minimization problems.

Problem Type	*LP*	*IP*	*GAP*	*Optimal IP*	*Time*
24-8	564	563	1	563	10
32-8 a	762.1	758	4.1	761	20
32-8 b	760	758	2	759	26
48-8	1133.54	none	-	1133	93
24-8	564	563	1	563	27
32-8 a	762.1	759	3.1	761	46
32-8 b	760	759	1	759	39
48-8	1133.54	1132	1.54	1133	170

Table 5.1: Method Comparison

Depending on how many extra columns we want to generate, the time spent in the pricing routine increases. This means that the time needed in order to reach a final integer solution increases considerably, actually (and naturally) in proportion to how many extra columns we generated in each generation phase. But still, the experiments show that by also generating non-optimal reduced cost columns from the subproblem, we are in fact able to reach better integer solutions. For the largest problem, only generating the best columns were not sufficient for even getting a feasible integer solution. Our experiment showed that by generating one extra column for each machine in each pricing run, a very high quality solution was in fact found (1132, where the optimal value is 1133).

Since the time consumption increases by a factor which is equal to the number of extra columns generated, it is interesting to try other approaches in order to get good integer solutions faster. One idea is to use standard column generation until no more negative reduced cost columns are found. Then, when normally some sort of branching would be applied, we continue generating more columns which are non-optimal from the LP-perspective. We consider the largest of the tested problems, where 48 jobs are to be assigned to 8 machines. When using traditional column generation, the IP solver fails to find a feasible solution (see the upper part of Table 5.1). The table below presents the results that were obtained when a predetermined number of extra column were generated, after the traditional column generator stopped. This means that columns with a positive reduced cost were generated and added to the restricted master problem.

Number of extra columns	*LP*	*IP*	*Relative Time*
0	1133.54	-	0.47
5	1133.54	1132	0.56
10	1133.54	1132	0.64
15	1133.54	1132	0.77
20	1133.54	1132	0.87
2 in each	1133.54	1132	1

Table 5.2: Generating extra columns in the end

It is clear that by generating extra columns only after the original column generation process has terminated, it is possible to reach good solutions relatively quickly. We see that by only generating five extra columns (per machine) we are able to reach the same integer result as when one extra column was generated in each iteration. Apparently, creating a somewhat greater restricted master problem by generating only 5 non-optimal (from an LP-perspective) columns made it possible to quickly reach a near-optimal integer solution. However, the optimal solution was not found, even though up to as many as 20 extra column were generated.

It should be pointed out that the experiments conducted here are but some examples of how one can utilize the information stated in the new global optimality conditions, given in section 4.6, when applying column generation algorithms to an equality constrained integer problem.

5.2 The Log Cutting Problem

A well known problem that often appears in the literature concerning column generation is the 1-dimensional Log Cutting Problem. The idea is to minimize some objective function, which in some cases is the number of used logs. The constraints that have to be fulfilled are that the demand on each width must be satisfied and that a log can not be cut in pieces whose length sum up to more than its length. The mathematical formulation of the problem looks like the following.

$$\begin{aligned}
min \quad & \sum_{j=1}^{n} y_j, & & \\
s.t. \quad & \sum_{j=1}^{n} y_j x_j^i \geq d^i, & \forall i \in \{1, ..., m\}, & \\
& \sum_{i=1}^{m} w^i x_j^i \leq W_{log}, & & \\
& x_j \in Z^+, & i \in \{1, ..., m\}, & \quad j \in \{1, ..., n\}, \\
& y_j \in \{0, 1\}, & j \in \{1, ..., n\}, &
\end{aligned} \tag{5.6}$$

where j denotes the different cutting patterns and where i denotes the different widths that are wanted. Then, clearly, y_j corresponds to how many times pattern j is cut, and x_j^i how many pieces of width i that are included in pattern j. In our case with identical logs, W_{log} corresponds to the capacity of each log, or, more naturally, the length of it.

This problem can be be attacked by decomposition principles, where the capacity constraints are placed in a subproblem and the remaining problem forms a so called master problem. Then one can relax the integrality constraints on the y-variables and use column generation as a solution approach. The complete master problem then looks like:

$$\begin{aligned}
min \quad & \sum_{j=1}^{n} y_j, & \\
s.t. \quad & \sum_{j=1}^{n} y_j x_j^i \geq d^i & \forall i \in \{1, ..., m\}, \\
& y_j \in [0, 1], & j = 1, ...n,
\end{aligned} \tag{5.7}$$

where x_j^i now corresponds to a feasible pattern. The sub problem takes the following form, in order to find negative reduced cost columns, corresponding to feasible patterns, which then are sent back to the master problem.

$$\begin{aligned} min \quad & 1-\sum_{i=1}^{m} u^i\lambda^i, \\ s.t. \quad & \sum_{i=1}^{m} w^i\lambda^i \le W_{log}, \\ & \lambda^i \in \{0,1\}, \end{aligned} \tag{5.8}$$

5.2.1 Method

The experiments are conducted on both small and large instances of the 1-dimensional Log Cutting Problem. The problems are first solved using traditional column generation, and the results are stored. Then, the problems are solved when also considering the information stated in the new set of global optimality conditions, δ-complementarity in particular. The new approach is compared to the traditional one, and the results will be shown in a scatter plot. A detailed analysis will be made regarding if and how the columns generated, while considering δ-complementarity, perform when searching for good integer feasible solutions.

By using the CPLEX mixed integer solver in all of the experiments in order to obtain integer solutions, we hope that it will become clear how the quality of the generated columns differ between the two considered approaches.

5.2.2 Adding complementarity constraints

The idea is to try and reach better integer solutions by including a near-complementarity constraint into the problem. Then we Lagrangian relax this constraint with some fixed non-negative multiplier, so that it becomes a part of the objective function. The complementary constraint looks like this:

$$\sum_{j=1}^{n}\sum_{i=1}^{m} y_j x_j^i u^i \le \sum_{i=1}^{m} d^i u^i + \delta. \tag{5.9}$$

Lagrangian relaxing the above constraint leads to the following master problem for the log cutting problem:

$$\begin{aligned} min \quad & \sum_{j=1}^{n} y_j + w(\sum_{j=1}^{n}\sum_{i=1}^{m} y_j x_j^i u^i - \sum_{i=1}^{m} d^i u^i - \delta), & \\ s.t. \quad & \sum_{j=1}^{n} y_j x_j^i \ge d^i, & \forall i \in \{1,...,m\}, \\ & y_j \in [0,1], & \in \{0,...,m\}, \end{aligned} \tag{5.10}$$

where w is the corresponding Lagrangian multiplier and δ the level of complementary violation allowed for.

This new term in the master objective also affects how the reduced cost is computed. Remembering that the reduced cost stands for how the objective is changed when the associated new column is introduced in the master problem, we look at the following Lagrangian function (where the demand constraints have been relaxed).

$$\sum_{j=1}^{n} y_j + w(\sum_{j=1}^{n}\sum_{i=1}^{m} y_j x_j^i u^i - \sum_{i=1}^{m} d^i u^i - \delta) + \sum_{j=1}^{n}\sum_{i=1}^{m} u^i(d^i - y_j x_j^i),$$

which thus gives the adjusted reduced cost function:

$$1 - \sum_{i=1}^{m} u^i z^i + \sum_{i=1}^{m} w u^i z^i.$$

Here z^i equals the number of times that width i is cut in the new pattern and w is the same fixed multiplier which comes from the relaxation of the new complementarity constraints. Based on previous experiments and recommendations from [9] and the fact that the objective now consists of both an original reduced cost term and a new term corresponding to complementarity, we form a linear combination of the original reduced cost term and the complementarity term. The objective in the pricing problem then becomes

$$(1-w)(1 - \sum_{i+1}^{m} u^i z^i) + w \sum_{i=1}^{m} u^i z^i.$$

We do the same with the objective in the restricted master problem. The master objective then looks as below, with all constant terms having been removed.

$$(1-w)\sum_{j=1}^{n} y_j + w(\sum_{j=1}^{n}\sum_{i=1}^{m} y_j x_j^i u^i)$$

However, we need to decide how to set the multiplier associated with the relaxation of the near-complementarity constraint. Therefore, different multiplier values are tested when solving the problem. Our hope is to be able to identify intervals for the multiplier where the performance of the solution process, with respect to solution quality and solution speed, is better than when not considering the new global optimality conditions. Although we are experimenting on several problem instances, we only perform this configuration once. This is due to the fact that the different instances are generated using the same random parameters.

5.2.3 Results

Randomly generated test problems were obtained from [14], and 25 experiments were conducted one two problem sets respectively, varying in size. This gave at total of 50 test problems. As was stated above, the first step is to calibrate the multiplier w, in order to be able to reach any good results. A problem is that, even though the problems in each set are created using the same settings, the level of complementarity violation may vary at the optimal solutions. Since the problems in each set are generated using similar parameters, we hope that we can use the same value for the Lagrangian multiplier for all problems coming from the same set. By assuming this we only need to calibrate the multiplier once per set. When calibrating the multiplier for the set of large problems, we got the following results, shown in figure 5.1, w r t solution quality and speed (for problem 77).

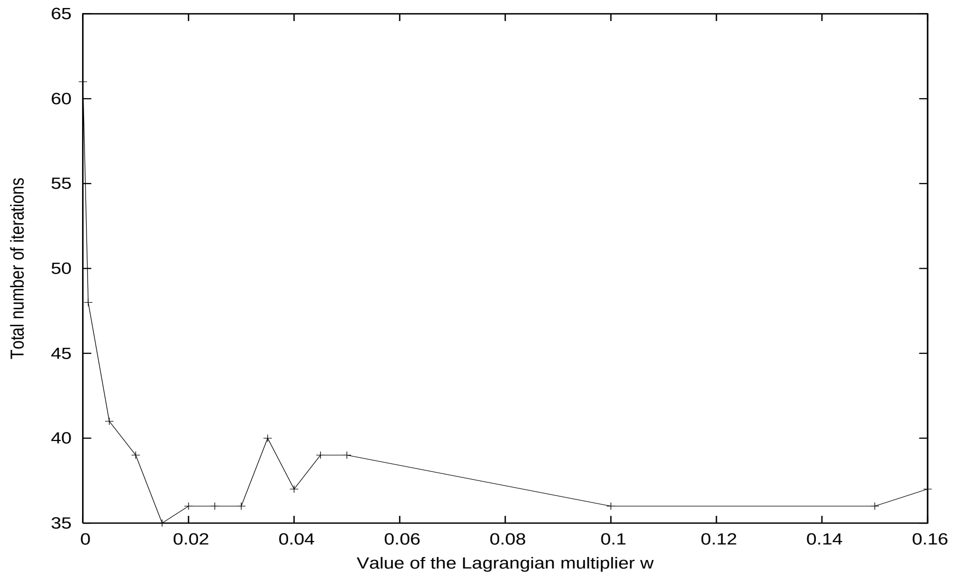

Figure 5.1: Multiplier calibration (large problems)

One can see that the number of iterations needed until the optimal solution is found drops dramatically as soon as one starts considering complementarity. The minimum number of iterations needed to find the optimal integer solution (by the CPLEX MIP solver) is 35, which compared to the original 61 is an incredible increase of efficiency. However, if the multiplier is set to a too large value, then the pricing routine fails to find any attractive rotations even when they do exist, resulting in a very bad final integer solution. This happens already if the multiplier is set to a value greater than 0.165. It should also be stated that in Figure 5.1, the final solution is the same for all instances, namely the optimal solution with an integer objective being 2464. The duality gap is 0.5, and in this case we have that $\Gamma = \delta$ holds. Also note that $w = 0$ corresponds to not considering the complementary term at all. In order to avoid problems which might appear if the multiplier is too large, we let it take on the value of 0.05 for the experiments with the rest of the large log cutting problems.

The multiplier calibration for the smaller test problems is shown in Figure 5.2 below (problem 13).

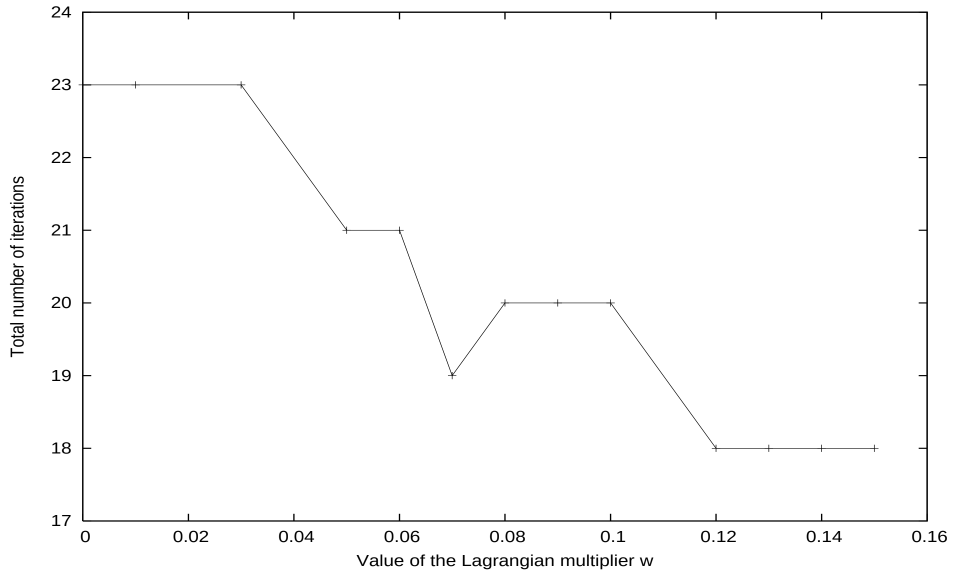

Figure 5.2: Multiplier calibration (small problems)

Here one can see that letting the multiplier increase up to 0.16 again gives a decrease of the number of iterations until the optimal integer solution can be obtained. However, similar to the larger problems, as soon as the multiplier is given a value larger that 0.165 the pricing routine fails. This is probably due to the fact that no more negative reduced cost column are identified. In order to make sure that this problem does not happen when we run our comparison tests, we set the multiplier w to 0.05 for the small problems.

For 3 of the 25 large test problems the traditional column generator failed to finish within reasonable time. Therefore, a valid comparison can only be made using the 22 remaining problems. Comparisons are made with respect to the total number of iteration that the column generator needs in order to reach a state where no more negative reduced cost columns exists. The results were as shown in Figure 5.3 when comparing traditional column generation with our new method which also considers complementarity fulfillment. Each dot represents one problem. The axis represents how many iterations that were needed in order to reach the final solution.

With respect to solution quality, it is important to state that for 17 of these examples the final integer objective was the same for both the traditional and the new column generation method. For four examples a better integer objective was reached (corresponding to a lower δ). For one example the final solution quality, i.e. objective, was slightly worse.

For the small problems, the results shown in Figure 5.4 were obtained, using a multiplier value of 0.05. Here it should be said that the number of different demanded patterns are only half as many as for the large problems. Results show that, again, a remarkable increase in the quality of columns is obtained when complementarity is considered. This is true even though the

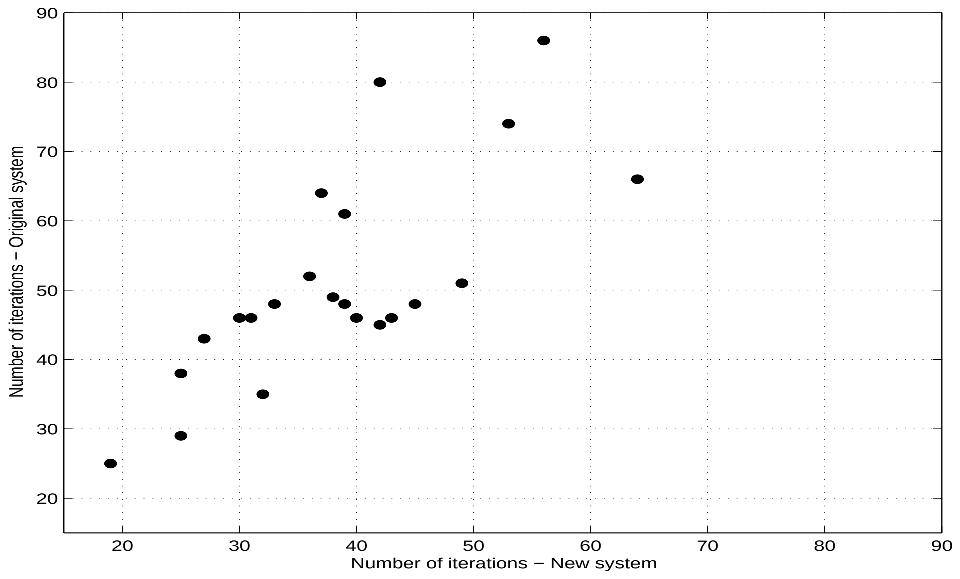

Figure 5.3: Comparison between methods (large problems)

multiplier, w was calibrated while only looking at the solution performance of one of the small problems.

To further illustrate how the new column generation method improves the solution quality throughout the entire process figures 5.5 and 5.6 can be studied. They are two examples from the set of large problems. They show the best integer feasible solution that can be found by the MIP solver throughout the process. The figures clearly show that by using our new method when generating columns it is possible to reach better integer solutions much faster than otherwise. In other words, good columns are generated faster, and thereby sooner added to the restricted master problem.

So, the results clearly show that by considering near-complementarity in the column generation process, by adding a constraint to the RMP and relaxing it, gives a considerable speed-up when searching for the optimal integer solution. Keep in mind that the number of generated columns per iteration is the same as before; it is the way in which they are generated that is different. For both the traditional settings and the new one, the CPLEX mixed integer solver is used, with the original costs as the objective.

It is interesting also to look at which columns that are used when solving the integer problem. When complementarity is considered throughout the generation process, it is clear that many of the generated columns just before termination are in fact used when forming an integer solution. This can not be said to hold for the original system and the columns generated throughout that process.

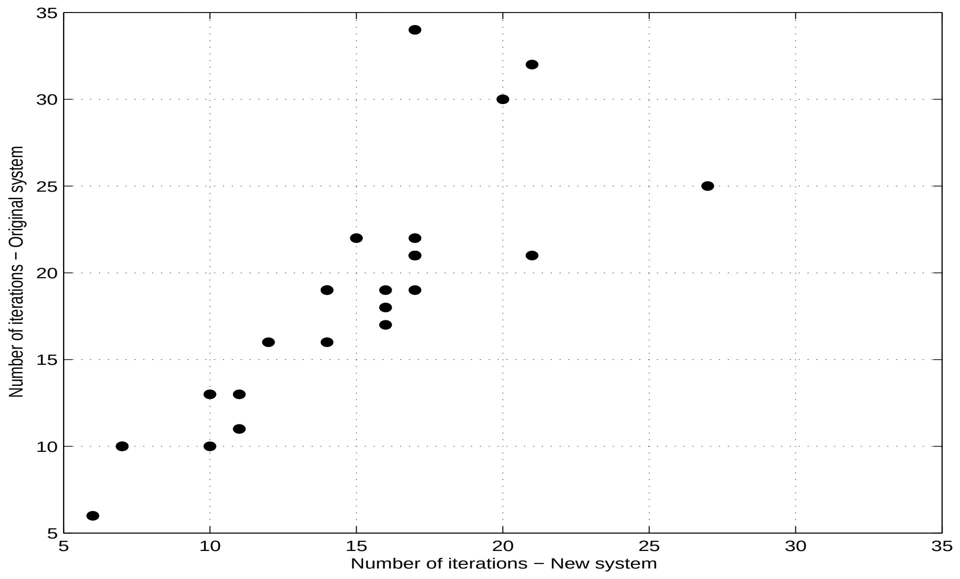

Figure 5.4: Comparison between methods (small problems)

5.2.4 Considering ϵ-optimality in the subproblem

In theory it might be a good idea to let the Lagrangian multiplier vary somewhat in the pricing phase of the column generation process. That would allow us to generate columns that are either near-optimal in the pricing problem, or near-complementary. In these Log Cutting problems it however seems as though we are able to reach optimal integer solutions without doing so, something that suggests that our problems might not need this variation of the multiplier.

But, since we are relaxing the near-complementary constraints along with a 'non-optimal' multiplier, we will in fact generate 'non-optimal' columns which to an extent then also considers near-complementarity.

5.3 Conclusions

The experiments conducted on the GAP problem showed that by generating not only optimal columns in the sub/pricing problem, we were able to obtain an integer solution quality better than what we originally could. The problem was however that the way that this was done meant that the time consumption went up considerably, due to the fact that the time spent in the pricing routine increased proportionally to the number of extra columns that were generated in each iteration. This problem lead us to also investigate the possibilities of only generating extra columns, non-optimal from an LP-perspective, after the original termination criterion (no negative reduced cost columns left) was reached. By implementing such a strategy, the same results as before were reached, but the time needed to reach them decreased considerably.

Since the constraints in the GAP problem are of the equality type, it is impossible to con-

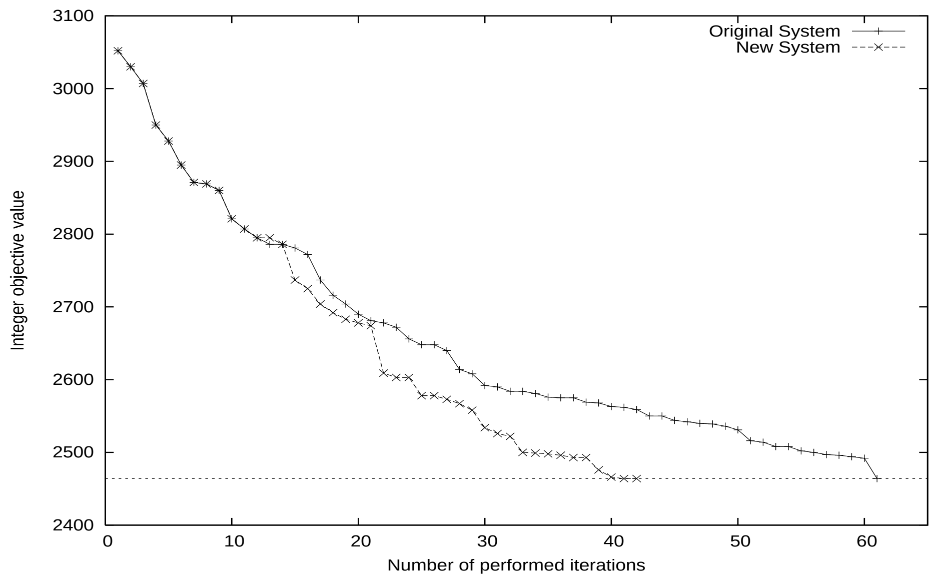

Figure 5.5: Column quality for solving the integer problem (a large problem)

sider adding the complementarity constraint in the master problem. As was explained, this is due to the fact that complementarity must hold when equality constraints are present, otherwise primary feasibility is violated. Therefore, experiments were also conducted on the 1-dimensional Log Cutting Problem. Here, near-complementarity constraints were added to RMP and moved into the objective function along with a Lagrangian multiplier. In the test, comparisons were made between the original formulation and to one which now also considered complementarity. Although it was often impossible to reach a better final integer problem objective value, it showed that we were able to arrive at it earlier when complementarity was considered. So, this showed that by considering the complementarity constraints (which are present in the new global optimality conditions) we were able to speed up the process of finding the columns that were good from an integer problem point of view.

It was also interesting to investigate which columns that were used when the integer problem was to be solved. For the new system, it was possible to see that recently generated columns were used to a large extent when the integer problem was solved. When looking at the original solution process, this was not as obvious. Here columns from all over the iteration process were used. These findings further strengthens the theory that the columns generated by using our new, which considers complementary fulfillment in its objective, are in fact superior when it comes to solving the integer programming problem.

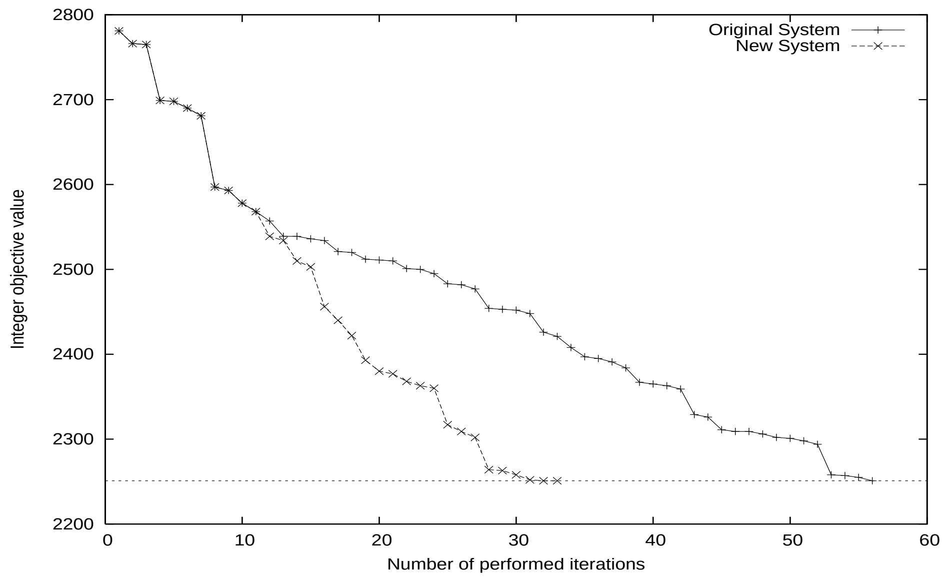

Figure 5.6: Column quality for solving the integer problem (a large problem)

Chapter 6

Carmen Systems

The system at Carmen is a C++ implemented column generating algorithm with several heuristics included in order to make the solutions better. I will go through how the Crew Pairing problem is modeled at Carmen, and also some of the most important heuristics that are being used in order to produce high quality solutions.

6.1 Crew Pairing at Carmen

At Carmen, a Crew Pairing product that uses column generation along with several heuristics in order to solve the existing problems has been developed. These heuristics consist of for example branching through connection fixing, limiting the number of possible pairings, decreasing the problem time horizon, etc. This section will go through the main components of the system developed at Carmen, mainly focusing on the column generator and the connection fixing heuristic. The details presented below are mainly gathered from [7], [2], and internal documents at Carmen, along with my own studies of the actual system.

6.1.1 The Master Problem

The crew pairing master problem at Carmen is modeled as a set covering type problem, similar to the one presented earlier in this thesis:

$$\begin{aligned} min \quad & \sum_{r\in R} c_r x_r, & \\ s.t. \quad & \sum_{r\in R} x_r a_{lr} \geq 1, & \forall l \in L, \\ & \sum_{r\in R} x_r p_{br} \leq q_b, & \forall b \in B, \\ & x_r \in \{0,1\}, & \forall r \in R', \end{aligned} \tag{6.1}$$

where L represents the set of legs and B the set of bases. R') is the set of pairings included in the (restricted) master problem. Besides the by now well known leg-covering constraint there are also productivity constraints for each base. The base constraints represent the fact that there is an upper limit on how much one base can handle during some period. It is also possible to model the problem as a set partitioning problem, but then one has to include all possible deadheads explicitly. For many problem instances, this implies that the model gets very large, thereby slowing down the solution process considerably. When formulating the problem as a set covering one, one needs to convert the overcovers to deadheads after the above problem has been optimized.

6.1.2 The Pricing Problem

Dual information is sent from the master problem to the pricing routine, and new good pairings get identified and added to the set of pairings which are considered in the master problem, i.e., added to R'. A good pairings is a one that has a negative reduced cost associated with it. Adding a good pairing to the set R', and solving the master problem again, will lead to a better master LP-objective value.

The pricing problem is formulated and solved as a k-shortest path problem, where the underlying network consists of duties and where some generated paths might be illegal. The costs of these duties change when the duals change, which is why different rotations can be found good in different pricing steps. The duties need to be generated before the pricing problem can be solved. Depending on the number of legs in the problem, this step can be quite time consuming. Therefore, in an initial phase, only a small number of duties are used in the network. Then, when some pairings are generated, i.e., after some iterations, more duties are generated and added to the network. Naturally, the number of different pairings that can be generated then also increases.

It would of course be possible to include constraints in the pricing problem in order to make sure that each generated rotation is in fact legal, but there is a trade-off between doing so and the fact that the pricing problem then becomes more difficult to solve. Therefore, the choice has been to define a relaxed network and check for legality after a pairing has been generated.

6.1.3 The Network

The network from which good pairings are to be identified is built up of duties and is a central component of the column generator. Here legality and cost information is stored using the duty periods as the blocks of which the network is constructed. This information is a must in order for the pricing problem to be solvable. Therefore one has to generate duties from the leg set. This can be quite time consuming, which is why in an initial phase, only a limited duty network is considered.

There are two types of arcs and two types of nodes in the duty network that are used when solving the pricing problem. The arcs are either duty periods or the connections between them. The nodes are either connection nodes or bases.

It is important to realize that even if it might be possible to generate all duty periods, it is not possible to consider all connections between them. This problem has given rise to investigations which have led to the use of a so called relaxed duty network. Instead of having separate connections between all duties the network now only considers one connection arc between duties having a particular start-leg to duties having some other particular end-leg. This reduces the number of connection arcs in the network, but also allows for illegal pairings to be generated. However, this relaxed version of the network is faster than the original one with respect to solving the pricing problem. As has been stated before, a k-shortest path algorithm is applied in order to find the best new pairings. When the best pairing turns out to be illegal, the next best is generated, and so on. As soon as a legal pairing is found it is added to the master problem, as long as the reduced cost function of it is negative.

If it is possible to identify certain illegal sub-chains in pairings which tend to be generated rather often, it might be a good idea to try to forbid these sub-chains in the pricing problem. The network, upon which the pricing problem is solved, is then refined by splitting some connection nodes and introducing new connection arcs.

Another important thing to point out is that the cost of a pairing must be additive over duty periods. If this is not true, then the cost can be either underestimated or overestimated. It is easily realized that if the cost is underestimated, then bad pairings might be generated and sent to the master problem, thereby slowing down the overall solution process. If the costs are overestimated, then some good pairings might never get generated, something that might lower the quality of the final solution. According internal studies at Carmen, costs in the pairing problems are rarely overestimated, but sometimes underestimated.

6.1.4 Connection Fixing Heuristics

Due to the fact that in reality, the solution of the LP relaxed problem rarely is integer, there exists a need for good integer heuristics when searching for integer solutions. The integrality gap then serves as a measure of how close to optimal the current best IP solution is. If it is large, some branching scheme can be used in order to try and reduce it.

At Carmen, data from the LP solution is used to determine which flight-connections to lock. In order to do so, connections between flights that have a large coverage (near one) in the LP solution are locked. Then the LP is solved again using column generation until no more columns are found, followed by some integer heuristic. Then, hopefully, a better integer solution (i.e. having a smaller integrality gap) is found. This fixing process can be done repeatedly until the solution quality is satisfactory. Important to understand is that if some connections are fixed, and the LP value of the resulting problem is worse than the so far best integer solution, then some connections must be unfixed in order to bring down the LP solution.

6.1.5 PAQS

Instead of solving the restricted master problem by using some simplex based solver, such as in either CPLEX or XPRESS, a heuristic solver called PAQS [2] is often used in the column generation process. The reason for this is twofold. First of all, the time needed in order to solve the LP master using XPRESS greatly increases when solving relatively large problems, which is very impractical. Secondly, PAQS generates dual feedback in a rather special way, focusing more on integer solutions. This is done by performing a rowwise update of the reduced costs in order to get a well defined primal solution to the Lagrangian problem, where the covering constraints have been relaxed. This is basically being done by making sure that for each constraint, only one variable has a negative reduced cost. This variable corresponds to the pairing that will be set to cover the corresponding constraint (leg). Due to the fact that

PAQS does not solve the problem to dual optimality, the quality of the duals sent to the pricing problem might turn out to be quite poor. A sub-gradient method is also included in order to avoid this problem and thereby to obtain high-quality duals. This is important since we want to generate good pairings when solving the pricing problem. A complete description of the PAQS algorithm is given in [2].

Chapter 7

Implementation of Ideas at Carmen Systems

A big part of this work has been to decide the best way in which to incorporate the new ideas, concerning the new global optimality conditions, into the already up and running Carmen pairing system. Similarly to what was done with the tests conducted on the Log Cutting problem, we want to try and generate columns which are good for solving the original integer problems.

7.1 Methods

The initial experiments will not use the heuristic solver PAQS in order to solve the LP master problem. Instead, the simplex based XPRESS solver is used. At this initial stage the connection fixing heuristics will also be turned off. These simplifications are being made in order to make things as simple as possible, when for the first time trying our ideas on a real-world Crew Pairing Problem.

However, due to the fact that XPRESS becomes too time expensive when large problems are to be solved, we also implemented our ideas when using PAQS to solve the LP master. Here connection fixing is used in order to reach good final integer solutions. When connection fixing is used, our new ideas concerning mainly near-complementarity fulfillment is only considered in the early stages of the solution process, before any fixing is done. This is because the duals tend to change quite a lot when connection fixing is applied.

7.2 Restrictions

The restrictions here follows from the methods described above. We do not reach better results when using XPRESS, compared to existing best known solutions found by using PAQS. Therefore, we do not compare XPRESS-based results with PAQS-based results.

7.3 Test Problems

We have only used the 'p05 Delta problem' and the '320 Areomexico problem' for our experiments with the Crew Pairing Problem. This is due to the fact that in order to be able to reach any conclusions from the experiments, a lot of runs must be made on each test problem. These runs are rather time consuming.

The p05 Delta problem is rather small in comparison to other pairing problems. It has 385 legs and can be solved to cover all these legs without any deadheads. The second problem is the 320 Areomexico problem and it consists of 4459 legs, thus being much larger than the previous problem. Since these two problems are very different we hope to be able to observe when and how our new ideas can improve solutions to the Crew Pairing Problem.

7.4 Experiments - XPRESS

This section experiments with the p05 Delta problem, and uses XPRESS in order to solve the LP master problem. The ideas are much similar to those used for improving the solution process for the Log Cutting problem previously described.

7.4.1 Updating the Master Problem

In order to be able to take the new global optimality conditions into consideration we first state the original master problem for the crew pairing problem, as it has been presented before. The only difference is that, here, the base constraints are turned off, since they are not present in our test problems.

$$\begin{aligned} min \quad & \sum_{r\in R} c_r x_r, & & \\ s.t. \quad & \sum_{r\in R} a_{lr} x_r \geq 1, & & \forall l \in L, \\ & x_r \in \{0,1\}, & & \forall r \in R'. \end{aligned} \tag{7.1}$$

By considering the new global optimality conditions which are consistent for also discrete problems, we decide to introduce a new constraint, corresponding to near-complementarity fulfillment into the formulation of the restricted master problem. The changed restricted master problem then looks like:

$$\begin{aligned} min \quad & \sum_{r\in R'} c_r x_r, & & \\ s.t. \quad & \sum_{r\in R'} a_{lr} x_r \geq 1, & & \forall l \in L, \\ & \sum_{r\in R'} \sum_{l\in L} \overline{u}^T (a_{lr} x_r - 1) \leq \delta, & & \\ & x_r \in \{0,1\}, & & \forall r \in R', \end{aligned} \tag{7.2}$$

where $\overline{u}_l$ are the duals corresponding to the covering constraints obtained from solving the previous master problem. Similar to the experiments carried out on the 1-dimensional Log Cutting Problem, we now proceed by Lagrangian relaxing this new constraint with a fixed positive multiplier, w. The problem then becomes:

$$\begin{aligned} min \quad & \sum_{r\in R'} c_r x_r + w(\sum_{r\in R'}\sum_{l\in L} \overline{u}_l(x_r a_{lr} - 1) - \delta), & & \\ s.t. \quad & \sum_{r\in R'} a_{lr} x_r \geq 1, & \forall l \in L, & \qquad (7.3) \\ & x_r \in \{0,1\}, & \forall r \in R'. & \end{aligned}$$

So, the above objective now also consists of a term corresponding to near-complementary fulfillment. Since we are optimizing over x we can reformulate the above by removing constant terms and we get:

$$\begin{aligned} min \quad & \sum_{r\in R'} c_r x_r + w(\sum_{r\in R'}\sum_{l\in L} \overline{u}_l x_r a_{lr}), & & \\ s.t. \quad & \sum_{r\in R'} a_{lr} x_r \geq 1, & \forall l \in L, & \qquad (7.4) \\ & x_r \in \{0,1\}, & \forall r \in R'. & \end{aligned}$$

This formulation of the master objective corresponds to adding an extra cost for each variable that equals the sum over all legs of the duals times the coefficients to the original cost of the variable. Note that we do not construct a linear combination of the two terms as was done when experimenting with the 1-dimensional Log Cutting Problem. This is simply because it makes no difference, apart from the fact that the best multiplier value will not be the same for both settings. Some small test indicated that the performance for the Crew Pairing Problems were in fact better, with respect to solution quality, when we did not construct this linear combination.

7.4.2 Updating the Pricing Problem

As was described in section 6.1.2, a k-shortest path problem is solved each time new columns are being priced out. The reduced cost of a pairing is nothing but the real cost of the pairing minus a term consisting of the sum of the duals in the pairing. This implies that when one is working with a duty network one could choose to calculate the reduced cost of each duty and then just add up the reduced cost terms of all duties in a pairing - then giving the total reduced cost of that pairing. This is what is being done in the system at Carmen.

The idea is to add a new term in the pricing objective, corresponding to the changes that have been made in the master problem. Since there already exists a term corresponding to the sum of all duals in an arc, it is possible to use that in order to make the pricing problem correspond to the changes that have been made in the master problem. The new objective in the pricing problem then looks like this:

$$min \quad \sum_{l\in L}((c_r - a_{lr}\overline{u}_l) + w a_{lr}\overline{u}_l), \qquad (7.5)$$

i.e. the new objective now consists of a combination of the original reduced cost and a term corresponding to complementarity. The vector $\overline{u}$ corresponds to the dual vector obtained by solving the restricted master problem.

We now have a restricted master problem where near-complementarity fulfillment is included in the objective with a positive multiplier w. This in turn gives a pricing problem which therefore also, to an extent set by the multiplier w, considers near-complementary when generating new columns. We hope that by considering near-complementarity fulfillment throughout parts of the generation process, we will be able to generate columns that can be used to form a high-quality integer solution.

7.4.3 Numerical Results

Since we do not know in advance whether or not complementarity is violated to a large extent at an optimal integer solution, we must try many different multiplier values in order to find any good ones. By observing the size of the duals and also the level of complementarity at different stages, a clue regarding the relative size of our parameter w is given. We observe that letting the value of w be set relatively large (>0.1) gives us very bad results, indicating that the complementarity term is given a too high weight.

When scanning some intervals for good multiplier values, we found that for the p05 Delta problem, a level of complementarity consideration corresponding of letting w take on values around 0.0001 seemed to perform rather well. Figure 7.1 further illustrates how the average integer objective value for our problem changes in the multiplier interval [0.00001,0.00020]. The original total cost, obtained by using the original XPRESS method without connection fixing, is shown as a horizontal line.

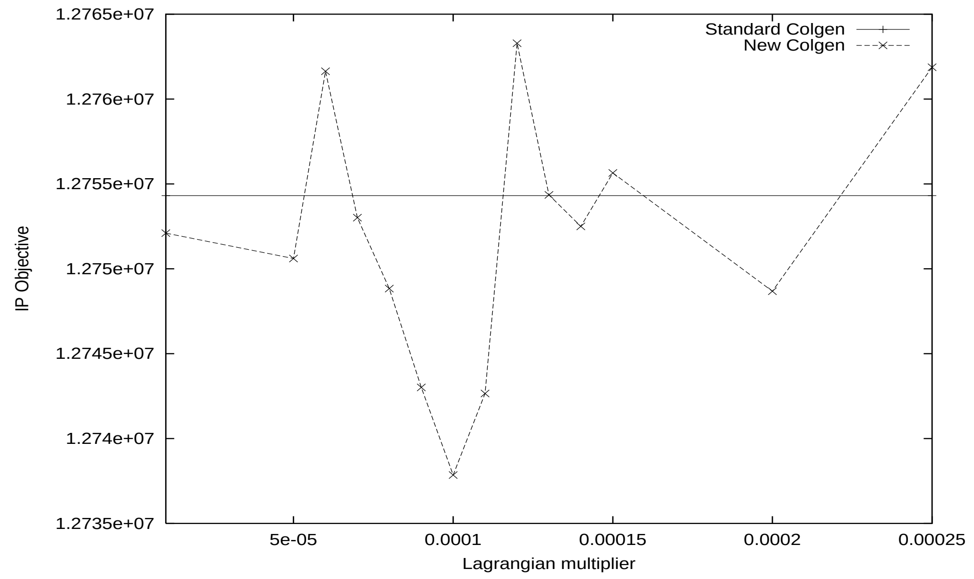

Figure 7.1: Multiplier Calibration

It is easily seen that the multiplier value which gives the best integer objective is 0.00010. Further, it is also clear that some multiplier values give worse results than the original system. However, for the small interval [0.00007,0.00011] all runs give better IP objectives than the original settings did, something that indicates that we are in fact able to generate columns which are somewhat better for solving the integer problem. The results are not as obvious as for the log cutting test problems, but, that is only natural since the pairing system at Carmen is far more complex than the log cutting test programs.

Some tests were made when connections fixing was turned on, but the results were unclear. Both somewhat better and somewhat worse solutions were found. We tried first generating columns while considering complementarity along with the best found multiplier 0.00010, and then turning it off before any connection fixing took place. This procedure was compared with the original process with connection fixing. Here we found that by considering near-complementarity fulfillment in early stages of the process (before connection fixing began) we were able to reach somewhat better solutions that otherwise, 12706706 compared to 12710210.

Also, some experiments were made where the multiplier was varying throughout parts of the generation process. These experiments were not very extensive and the results were unclear. For example, the multiplier was set to increase after each iteration, up to some predefined maximum value. Also, the experiments were conducted where the multiplier was large in the beginning and decreasing after each iteration. No better results were found.

7.5 Experiments - PAQS

Here experiments are conducted when using PAQS instead of XPRESS, along with the connection fixing heuristics, in order to solve the LP master problems. The considered problems are the p05 Delta problem and the 320 Areomexico problem. It should be said that, at least for the p05 Delta problems, PAQS together with connection fixing produces high-quality solutions.

7.5.1 The Master Problem

We choose not to make any changes to the restricted master problem when PAQS is used to solve the LP master. Instead, we treat the complementarity constraints as having been completely moved to the pricing problem. There they are relaxed into the objective, resulting in an objective function which is similar to the one used for the XPRESS experiments. The master problem therefore remains as the following:

$$\begin{aligned} min \quad & \sum_{r \in R} c_r x_r, & \\ s.t. \quad & \sum_{r \in R} a_{lr} x_r \geq 1, & \forall l \in L, \\ & x_r \in \{0,1\}, & \forall r \in R', \end{aligned} \tag{7.6}$$

where R' is the set of generated pairings and L the set of legs. For our problems, no base constraints are present.

7.5.2 Updating the Pricing Problem

The near-complementarity constraints, which are being considered in the pricing phase, are relaxed into the objective along with a non-negative multiplier, w. The objective in the pricing problem can then be stated as:

$$min \quad \sum_{l \in L}((c_r - a_{lr}\overline{u}_l) + wa_{lr}\overline{u}_l), \tag{7.7}$$

where $\overline{u}$ is a non-negative dual variable corresponding to leg l, given by PAQS or by a sub-gradient method. So, we now consider near-complementary fulfillment in the pricing objective.

7.5.3 Numerical Results

The results in Figure 7.2 illustrates how the average integer solution quality, for the p05 Delta problem, varies for different choices of multiplier values.

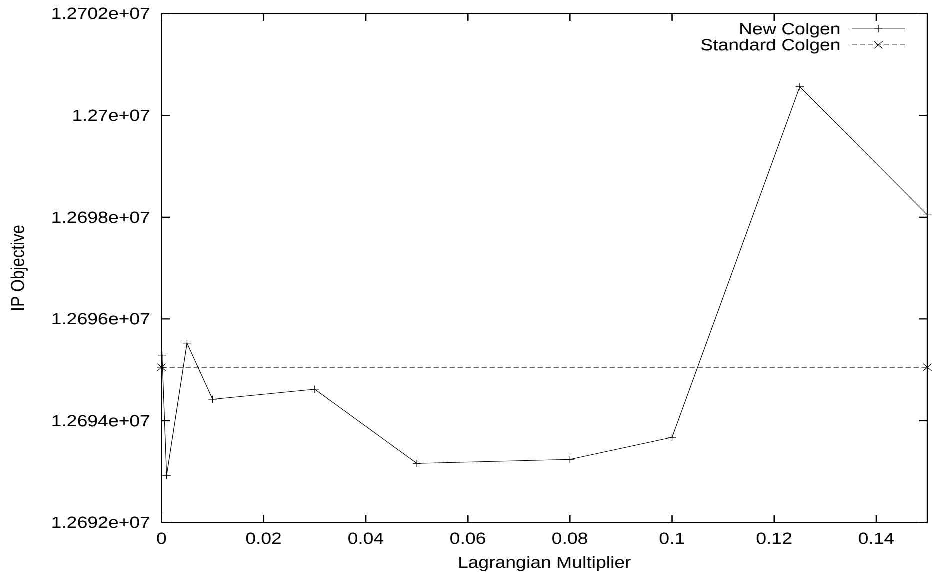

Figure 7.2: Multiplier Calibration

When studying Figure 7.2 it is clear that for all of the tested multiplier values in the interval [0.01,0.1], the system which also considers complementarity performs better than the original, with respect to the final integer objective value. For very low multiplier values it is impossible to draw any conclusions. It is however surprising that for a multiplier value as low as 0.001, we actually get the best improvement of the objective. On the other hand, this is similar to the results from the previous experiments, where XPRESS was used instead of PAQS. There, 0.0001 was actually found to perform best. For values of w larger that 0.1, the quality of the solutions decrease, suggesting that the complementarity term is given a too high weight. The problem then probably is that good pairings will not be generated due to the fact that the system finds them unattractive.

It should also be mentioned that the new system performs similarly to the old one with respect to time. Also, one can look at the number of generated rotations. In the standard run, an average of 5216 rotations were generated throughout the entire column generation process. For the new system, with a multiplier value of 0.05 (i.e. in the interval where the new system performs well) an average of 5240.5 rotations were generated. So, it seems as if the performance of the system is unaffected by our new ideas and the changes they imply.

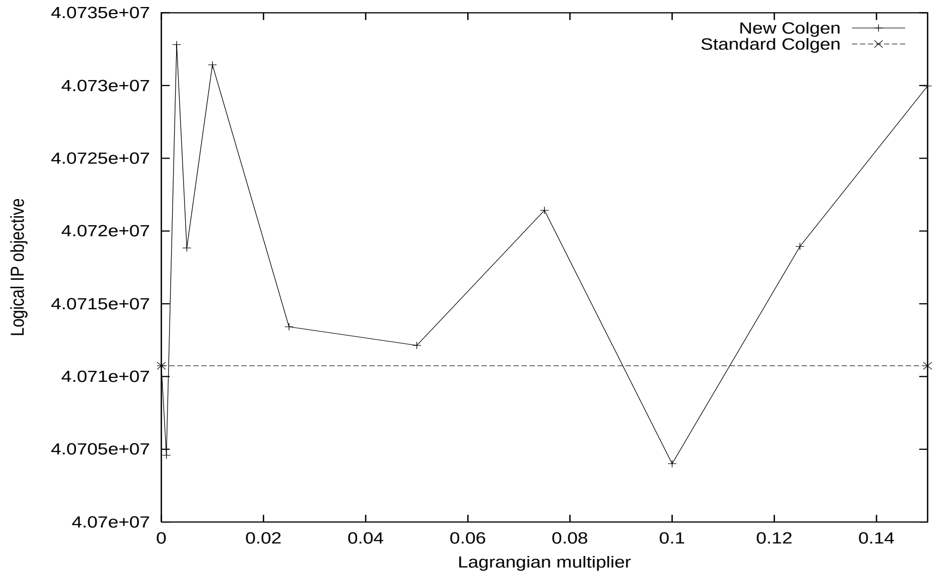

Figure 7.3: Multiplier Calibration

For the 320 Areomexico problem the results are as presented in Figure 7.3. Here one can see that even though better solutions are reached in some runs, no good intervals seem to exist. The two multiplier values that give better results than a standard run is 0.001 and 0.01. It is also interesting to see whether or not our new ideas have any effect on the resulting number of deadheads. Deadheads correspond to rows (legs) being overcovered. Keep in mind that we do not know in advance how many deadheads there are in the optimal solution. There might exist several near-optimal solutions between which the number of deadheads vary. The results, with respect to the number of deadheads, are presented in Table 7.1. The average total number of deadheads decreases (330.875 for w=0), although not proportionally, when also considering near-complementarity (for $w > 0$). This holds for all multiplier values tested. However, as Figure 7.3 and Table 7.1 together illustrate, it is possible to reach good solutions that have both many and few deadheads (where some have been converted from overcovers) in their final solutions.

7.6 Conclusions

It is clear that different multiplier values give rise to different columns in the column generation process, and hence also different results when solving the original integer problem. It is rather time expensive to check different levels of the Lagrangian multiplier since each test requires a separate run.

Multiplier Value	*Average Number of DHs*
0	330.875
0.001	326.250
0.003	330.125
0.005	328.375
0.010	328.125
0.025	328.750
0.050	324.625
0.075	327.375
0.100	329.375
0.125	329.375
0.150	323.875

Table 7.1: Deadhead Variation

The experiments using XPRESS, on the p05 Delta problem, show that when a too large multiplier is chosen, the column generation process fails to produce good solutions, probably due to the fact that no columns will be attractive. This in turn depends on the fact that the pricing problem now also considers near-complementarity in its objective, and if this term becomes too large, no negative reduced cost columns will be found. However, we found that by letting the multiplier take on the value of 0.0001, we were in fact able to reach somewhat better integer solutions. The original settings gave an average total cost was 12,754,310.625, which can be compared with 12,737,851.875 when our new ideas were included. Given the fact that the best LP bound was approximately 12,680,000 this is a rather big decrease of the average percentage size of the integrality gap (0.58 percent to 0.46 percent, being a decrease of approximately 21 percent). It should also be said that the comparisons are made when the connection fixing heuristics are turned off.

When experimenting with PAQS, on both the p05 Delta and the 320 Areomexico problems, we decided to only consider near-complementarity in the pricing problem, i.e. where the columns are generated. By doing this we could consider near-complementarity without affecting how the duals were generated by PAQS when solving the restricted master.

The p05 Delta problem, which does not have any overcovers (converted to deadheads) in its optimal solution, performed well when near-complementarity was considered along with multiplier values in the interval [0.01, 0.1]. It is interesting to observe that although near-complementarity was not considered throughout the entire generation process, it still considerably affected the final integer solutions quality. (See Figure 7.2). It should also be pointed out that when the multiplier is set too large, the results becomes very poor. The reason is the same as for the XPRESS experiments, namely that no negative reduced cost columns will then be found.

For the 320 Areomexico problem, the results were not as clear. This is probably due to the fact that many good final integer solutions exist, between which the level of complementarity varies. Even though we were able to reach better average solutions in some cases, no conclusions with respect to good multiplier intervals could be made. By including near-complementarity considerations in the generation phase, we reduced the average total number of deadheads for all tested multiplier values. However, there was no clear correlation between the final number of deadheads and the total cost. Good solutions could have both many and few deadheads in them. However, in order to check the real effect with respect to the multiplier value and its impact on deadheading costs, a deeper investigation in the total deadhead flying time should be done. This has not been done here.

Chapter 8

General Conclusions and Discussion

Based on the experiments which have been conducted some conclusions can be drawn regarding the use of considering the new global optimality conditions presented in [9]. Applying these ideas to experiments conducted on the GAP problems lead to the idea of generating also non-optimal columns. The results showed that for some problems the new columns, not found by traditional column generation, made it possible for the IP solver to find better integer solutions. Perhaps most interesting was the fact that in some cases where traditional column generation failed to produce even a feasible solution, the new method worked very well.

Considerable improvements were made by also considering complementarity fulfillment in the column generation process for the Log Cutting problem, at least with respect to solution speed. The results showed that the columns generated by our new approach often were much better in terms of solving the original integer problem. The good columns were also found earlier on in the generation process. For some examples we were also able to reach better integer objective values by decreasing the level of complementarity violation, δ, at the final integer solution. This was what we hoped for since we wanted to control the level of complementarity violation, all in accordance with the new optimality conditions. It is also important to keep in mind that nothing regarding the IP solver or the original integer problem was altered.

When implementing our new ideas at Carmen Systems, and running some tests on real world pairing problems, several things were observed. For problems such as the p05 Delta problem, where it is practically known in advance that no overcovered rows will exist at an optimal integer solution, it is possible to find good multiplier intervals. However, for problems such as the 320 Areomexico problem, this is much more difficult. Even though some multiplier values, w, gave better integer solutions, no intervals of good multiplier values were found. It is probably necessary to implement some strategy regarding how to update the multiplier during the generation process, in order to reach some pre-determined level of near-complementarity. However, this was not tested in our experiments.

The experiments conducted in this thesis show that it is possible to reach better integer solutions by considering the new global optimality conditions when designing column generation algorithms. However, the difficulty of doing this varies between different problems, and also between problem instances.

8.1 Future Research

The experiments showed that it is possible to benefit from considering the new global optimality conditions when designing column generation algorithms for integer programming problems. However, they also showed that it sometimes is not easy to simply include these new findings into an already up and running optimization system. In order to be able to capitalize on the possibilities which most certain exist, it is very important that the new optimality conditions are recognized throughout the whole of the algorithm design process. An interesting venture for future research could be to not only consider the new global optimality conditions when generating columns, but also when searching for an integer solution. This combination, and the latter in particular, would probably give a better control of the complementarity violation at the final integer solution.

In order to generate even better columns in the pricing step, some clever strategy regarding how to update the multiplier, w, during the column generation process, should be developed. By checking how columns corresponding to certain multiplier values affects the objective of the original integer problem, one might get an idea regarding how the multiplier value should be updated.

Bibliography

[1] L. S. Lasdon *Optimization Theory for Large Systems*, Macmillan, New Yory, NY, 1970.

[2] D. Wedelin, *An algorithm for large scale 0-1 integer programming with applications to airline crew scheduling*, 1995.

[3] C. Barnhart et al, *Branch-and-Price: Column Generation for Solving Huge Integer Programs*, Georgia Institute of Technology, Atlanta, 1996.

[4] S.G. Nash, A. Sofer, *Linear and Nonlinear Programming*, McGraw-Hill Companies, 1996

[5] M. Savelsbergh, *A Branch-and-Price Algorithm for the Generalized Assignment Problem*, Georgia Institute of Technology, Atlanta, 1997.

[6] P. Vance et al, *A Heuristic Branch-and-Price Approach for the Airline Crew Planning Problem*, 1997.

[7] T. Gustavsson, *A heuristic approach to column generation for airline crew scheduling*, Chalmers Univeristy of Technology and Göteborg Univerisy, Göteborg, 1999.

[8] M. Lubbecke, J. Desrosiers *Selected Topics in Column Generation*, Barunschwieg University of Technology, HEC Mantreal and GERAD, 2002.

[9] M. Patriksson, T. Larsson, *Global optimality conditions for discrete and nonconvex optimization. With applications to Lagrangian heuristics and column generation*, Chalmers University of Technology and Linköpings Universitet, 2003.

[10] Huisman et al, *Combining Column Generation and Lagrangian Relaxation*, Rotterdam, 2003

[11] E. Jacquet-Lagreze, M. Lebar, *A column generation model for a scheduling problem with maintenance constraints*, Euro-Decision and Ecole Centrale Prais, 2000.

[12] Fourer et al *AMPL : A Modeling Language for Mathematical Programming*, Brooks/Cole, 2002

[13] *ILOG CPLEX 7.1, User's Manual*, ILOG, France, 2001.

[14] J. E. Beasley, *OR-LIBRARY*, www.ms.ic.ac.uk/info.html, 25 Jan 2004

[15] 4er, *AMPL Modeling Language for Mathematical Programming*, www.ampl.com, 25 Jan 2004