# A Lagrangean Relaxtion Based Algorithm for Solving Set Partitioning Problems

van Krieken, M.G.C.; Fleuren, H.A.; Peeters, M.J.P.

*Publication date:*
2004

No. 2004–44

# A LAGRANGEAN RELAXATION BASED ALGORITHM FOR SOLVING SET PARTITIONING PROBLEMS

By M.G.C. van Krieken, H.A. Fleuren, M.J.P. Peeters

April 2004

ISSN 0924-7815

# A Lagrangean Relaxation Based Algorithm for Solving Set Partitioning Problems

M.G.C. van Krieken*, H.A. Fleuren, M.J.P. Peeters

Tilburg University, P.O. Box 90153, 5000 LE Tilburg, The Netherlands

* Corresponding author, email: M.G.C.vanKrieken@uvt.nl

Manuscript version 27-04-2004

---

## Abstract

In this paper we discuss a solver that is developed to solve set partitioning problems. The methods used include problem reduction techniques, lagrangean relaxation and primal and dual heuristics. The optimal solution is found using a branch and bound approach. In this paper we discuss these techniques. Furthermore, we present the results of several computational experiments and compare the performance of our solver with the well-known mathematical optimization solver Cplex.

*Keywords:* Integer programming, Set partitioning, Branch and bound, Lagrangean relaxation

---

## 1. Introduction

### *1.1 The set partitioning problem*

Given a collection of subsets of a certain ground set and costs associated to these subsets, the set partitioning problem is the problem of finding a minimum costs partition of the ground set (Atamtürk et al, 1996). Formally, we can write the set partitioning problem as follows:

$$z_{SP} = \min \sum_{j \in J} c_j \cdot x_j \qquad [1]$$

Subject to:

$$\sum_{j \in J} a_{rj} \cdot x_j = 1 \qquad r \in R \qquad [2]$$

$$x_j \in \{0,1\} \qquad j \in J \qquad [3]$$

Here, R is the set of *constraints* or *rows* (ground set) and J is the collection of *subsets* or *columns*. The matrix $A = \{ a_{rj} \}$ is defined such that $a_{rj}$ is equal to 1 if subset j contains row r and 0 otherwise. The

costs of a subset j are given by $c_j$. Furthermore, we define R(j) as the set of rows that are contained in subset j and J(r) as the collection of subsets that contain row r. Without loss of generality we assume that the costs vector c is integer.

### *1.2 Practical applications*

Due to the special structure of the set partitioning problem, it is possible to solve to optimality large problems in a reasonable amount of time. Therefore, much research has been focused on the application of set partitioning problems in real-life situations. Nawijn (1987) discusses an application of the set partitioning problem to optimize the performance of a blood analyzer. Baldacci et al. (2002) describe an approach to solve capacitated location problems using set partitioning. Another field in which set partitioning has been applied successfully is that of vehicle routing, see Le Blanc et al. (2004) and Fleuren (1988). The most famous application of set partitioning problems described in literature is crew scheduling, see for example Mingozzi et al. (1999) and Hoffman and Padberg (1993).

### *1.3 Motivation*

Solving set partitioning problems has been a subject of research for decades. However, to our knowledge, Hoffman and Padberg (1993) were the first to discuss an algorithm that was able to solve large set partitioning problems to proven optimality. Since then, other researchers have reported on successful algorithms for solving large set partitioning problems to optimality, see for example Borndörfer (1998). However, the algorithms that are fast and able to solve very large problems are all linear programming (LP) based methods. Since the LP relaxations of large set partitioning problems are highly degenerate and hard to solve, a high quality LP solver is needed to solve these relaxations. Hoffman and Padberg (1993) as well as Borndörfer (1998) use Cplex to solve the relaxations. Since this type of high quality LP solver is expensive, the goal of our research is to examine whether we can achieve the same performance on solving set partitioning problems without using any other mathematical programming solver.

### *1.4 Outline of the paper*

Section 2 is devoted to the use of preprocessing in solving set partitioning problems. First, an overview of the literature on preprocessing will be given. Next we discuss the techniques that are used in the solver.

Section 3 deals with finding lower bounds for the set partitioning problem. We will give a brief overview of the literature and discuss the Lagrangean relaxation approach and the dual heuristics that are used in the solver.

Section 4 covers the search for solutions for set partitioning problems. Again we start with an overview of the literature, followed by the discussion of the primal heuristic and branch and bound algorithm used in the solver.
Section 5 describes the actual composition of the solver, i.e. how the methods are used in the solver.
Section 6 discusses the performance of the solver. We use a common test set to compare our solver to results in the literature and to the well-known mathematical programming solver Cplex.
Finally, in Section 7, we will conclude and give some recommendations for further research.

## 2. Preprocessing

Preprocessing is a generic term for all techniques that are designed to improve the formulation of linear or integer programs, such that they can be solved faster by some solution method. Mostly, these techniques use logical implication to simplify a problem in an automated way. In general, this results in a reduction in the number of rows and/or columns of the problem.

### *2.1 Literature*

Preprocessing set partitioning problems has received much attention in literature. Already in 1976, Balas and Padberg report on the "equal columns" and "contained rows" preprocessing rules (Balas and Padberg, 1976). More recently, Atamtürk, Nemhauser and Savelsbergh (Atamtürk et al, 1996) pay attention to preprocessing and probing techniques for set partitioning problem. Borndörfer (Borndörfer, 1998) gives an overview of all preprocessing techniques for set partitioning problems that appear in the literature up to 1998. Finally, Van Krieken, Fleuren and Peeters (Van Krieken et al., 2003) present two new preprocessing techniques for set partitioning, the cut rule and the row combination heuristic, and give extensive computational results for these and other preprocessing techniques.

### *2.2 Preprocessing in the solver*

In this section, we will briefly discuss the preprocessing techniques that are used in the solver. For more information and computational results concerning these techniques see Van Krieken et al. (2003).

**PP1: Equal Columns.** If column j is equal to column k, i.e. R(j) = R(k), with $c_j \geq c_k$, then column j can be removed from the problem.

**PP2: Contained Rows.** If row r is contained in row s, i.e. $C(r) \subseteq C(s)$, then all columns that are in C(s), but not in C(r) and row s can be removed from the problem.

**PP3**: **Clique.** If all columns that cover row r have one or more elements in common with a column j that does not cover row r, then we can remove column j.

**PP4: Equal Rows.** If row r is equal to row s, i.e. J(r) = J(s), than row s can be removed from the problem.

**PP5: Cut.** If there is a set of three rows {r,s,t} and a row w, for which holds that row w is only covered by columns that cover at least two of the rows r, s and t, then we can remove all columns that cover at least two of the rows r, s and t, but not row w.

**PP6: Row combinations.** If for two rows r and s, we add combinations of all columns that cover only one of these rows to the problem, we can subsequently remove all columns that cover only one of these rows. Furthermore, we can remove one of the rows. This is particularly interesting for pairs of rows that differ only on a few elements. The technique that we implemented to make row combinations works as follows:

1. Max_growth = $\frac{p}{100} \cdot$ number of columns.
2. For each $r_1, r_2 \in R$ we define:
   $C(r_1, r_2) = \{j \in J(r_1) \mid j \notin J(r_2)\}$ and
   $f(r_1, r_2) = |C(r_1, r_2)| \cdot |C(r_2, r_1)| - |C(r_1, r_2)| - |C(r_2, r_1)|$
   This function gives an upperbound on the increase in number of columns when rows $r_1$ and $r_2$ are combined. Now let {s,t} be the set of rows for which $f(r_1, r_2)$ is minimal.
   If ($f(s, t) >$ Max_growth) then stop.
3. Combine rows *s* and *t*. Now delete all columns $k \in \{j \in J(s) \mid j \notin J(t)\}$ and all columns $m \in \{j \in J(t) \mid j \notin J(s)\}$. Go to step 2.

This implementation uses the parameter p, a percentage that denotes the maximal allowed growth in the number of columns. In the solver, we apply the row combination technique with p = 0.5, a value that is determined by extensive testing. For more information on row combinations, see Van Krieken et al. (2003).

## 3. Lower bounds

In methods that use branch and bound or branch and cut to solve set partitioning problems to optimality, a good lower bound is of great value. This section discusses methods for determining lower bounds for the set partitioning problem. A brief overview of literature is given, followed by a discussion of the methods used in the solver

### *3.1 Literature*

Most attention in literature on solving set partitioning problems has been on branch and cut solvers that use the linear programming relaxation to determine a lower bound. Examples can be found in Hofmann and Padberg (1993) and Borndörfer (1998), where in both cases a commercial software

package is used to solve the linear programming problem. An alternative to the use of linear programming relaxations is Lagrangean relaxation, which will be discussed in the next paragraph. This method can also be found in literature, for example Fleuren (1988) applies Lagrangean relaxation to determine lower bounds for set partitioning problems. Beasley and Cao (1996) apply Lagrangean relaxation to the more general crew scheduling problem. The Lagrangean relaxation method that is used in the solver is based partly on Held et al. (1974) and Hunting (1998).

### *3.2 Lagrangean relaxation*

To obtain the Lagrangean relaxation model (LR) of the set partitioning problem, we relax the equality constraints of the problem. The constraints are taken into the objective with a so-called Lagrangean multiplier $\lambda_r$:

$$z_{LR}(\lambda) = \min \sum_j \left( c_j - \sum_r a_{rj} \cdot \lambda_r \right) \cdot x_j + \sum_r \lambda_r \qquad [4]$$

subject to [3]

Define the Lagrangean costs of a column j to be:

$$cl_j = c_j - \sum_{r \in R} a_{rj} \cdot \lambda_r \qquad [5]$$

Now the solution to the relaxed problem, given vector $\lambda$, is given by:

$$x_j = \begin{cases} 1 & \text{if } cl_j \leq 0 \\ 0 & \text{otherwise} \end{cases} \qquad [6]$$

The best lower bound we can thus find with this relaxation is given by:

$$LB = \max_{\lambda} z_{LR}(\lambda) \qquad [7]$$

It is shown by Geoffrion (1974) that the value of the solution to this maximization problem equals the value of the solution to the linear programming relaxation of the original problem. Since the maximization problem given by [7] is too time-consuming to solve to optimality, it is common practice to use heuristic methods to find a good value of the vector $\lambda$. In our solver, we use a subgradient search method, which is discussed in the next paragraph.

### *3.3 Subgradient search*

The goal of the subgradient search method is to determine a sequence of vectors $\lambda^0$, $\lambda^1$,..., $\lambda^k$ that converges to the optimal vector that gives the lower bound [7]. To this end, the following iteration scheme is used:

$$\lambda_r^0 = \min_{j \in J(r)} \frac{c_j}{\sum_{t \in R} a_{tj}} \qquad [8]$$

$$\lambda_r^{k+1} = \lambda_r^k + \text{stepsize}^k \cdot gr_r^k \qquad [9]$$

Here, the vector $gr^k$ represents the vector of subgradients and $stepsize^k$ the stepsize used in the $k^{th}$ iteration of the algorithm:

$$gr_r^k = 1 - \sum_j a_{rj} \cdot x_j^k \qquad r \in R \qquad [11]$$

and

$$stepsize^k = \frac{C_k}{\sqrt{\sum_r \left(gr_r^k\right)^2}} \qquad [12]$$

$C_k$ is determined by:

$$C_k = (\alpha)^k \cdot C_0 \qquad [13]$$

The subgradient search algorithm uses two parameters $C_0$ and $\alpha$. $C_0$ is a large number and $\alpha$ is a number between 0 and 1, close to 1. This means that we start with a $C_0$ that we expect to be far from the optimal value, but by multiplying $C_k$ with $\alpha$ in every iteration, we come closer to the optimal value. In our implementation, we start with values $C_0 = 100000$ and $\alpha = 0.95$. The closer the value of $\alpha$ is to 1, the smaller the steps that we take in our convergence sequence, and the closer we converge to the optimum. Therefore, we adjust the value of $\alpha$ twice during the subgradient search. This is done in the following way:

$$\text{If } \left|\frac{z_{LR}\left(\lambda^{k+1}\right) - z_{LR}\left(\lambda^k\right)}{z_{LR}\left(\lambda^{k+1}\right)}\right| \leq 0.5 \text{ we set } \alpha = 0.975. \qquad [14]$$

$$\text{If } \left|\frac{z_{LR}\left(\lambda^{k+1}\right) - z_{LR}\left(\lambda^k\right)}{z_{LR}\left(\lambda^{k+1}\right)}\right| \leq 0.25 \text{ we set } \alpha = 0.99875 \qquad [15]$$

The algorithm is stopped when $z_{LR}(\lambda^{k+1}) = z_{LR}(\lambda^k)$.

Since the speed of the subgradient search depends on the number of columns, we do not take all the columns into account at the start of the search. Instead, we only take the $N_r$ columns with the lowest costs for every row. For this set of columns we perform the subgradient search. If the resulting $\lambda$ gives the same lower bound for the whole set of columns as for the subset of columns, we keep this $\lambda$ as the final solution. If this is not the case, we take a larger set of columns and start again.

The maximum number of columns taken into account per row is given by:

$$N_r = \min\left(Q \cdot \frac{\sum_{s \in R}\sum_{j \in J} a_{sj}}{|J|},\ |J(r)|\right) \qquad [16]$$

At the start Q = 25, and every time that we restart the subgradient search, this value is multiplied by 2.

### *3.4 Dual feasibility*

The dual of the linear programming relaxation of the set partitioning problem is given by:

$$z_{DLP} = \max \sum_r u_r \qquad [17]$$

subject to:

$$\sum_r a_{rj} \cdot u_r \le c_j \qquad \forall j \in J \qquad [18]$$

$$u_r \text{ unrestricted} \qquad \forall r \in R \qquad [19]$$

If a dual feasible vector $u^*$ is available, a lower bound for the set partitioning problem is given by:

$$LB(R, J, u^*) = \sum_{r \in R} u_r^* \qquad [20]$$

For every partial problem with R' $\subseteq$ R and C' $\subseteq$ C we can also define a lower bound by:

$$LB(R', J', u^*) = \sum_{r \in R'} u_r^* \qquad [21]$$

This property of dual feasible solutions has the advantage that we have lower bounds for every partial problem that we encounter during the branch and bound procedure. However, the Lagrangean vector $\lambda$ that results from our subgradient search does not necessarily have to be dual feasible. Therefore we add a step to the algorithm to make the resulting vector $\lambda$ dual feasible. In this step, we sort the columns on increasing costs and for every column j with negative Lagrangean costs $cl_j$, we find the first row r that this column covers and add $-cl_j$ to the Lagrangean multiplier $\lambda_r$ of this row. When applied to the instances in the test set discussed in Paragraph 6.1, it appeared that the value of the lower bound is hardly affected by this adjustment.

### *3.5 Dual heuristics*

Starting with a dual feasible vector $\lambda$, we apply two dual heuristics to raise the value of the lower bound, while attaining a dual feasible solution.

**DH1: Simple updating heuristic.** If all columns that cover row r have a positive Lagrangean costs, then we can raise $\lambda_r$ with:

$$\Delta = \min_{j \in J(r)} cl_j \qquad [22]$$

After this step, the vector $\lambda$ still satisfies the constraints in [16] and the lower bound, given by the sum of $\lambda_r$, is higher.

**DH2: 3OPT.** The idea for this heuristic stems from Fisher and Kedia (1986). The 3-opt heuristic is a local improvement heuristic that begins with a dual feasible solution and tries to improve this solution by simultaneously changing three $u_r$ – values. Let $u_{r1}$, $u_{r2}$ and $u_{r3}$ be these three values. We now want to increase the value of the lower bound by simultaneously decreasing $u_{r1}$ and increasing $u_{r2}$ and $u_{r3}$, all by the same amount $\Delta$. This will increase the value of the lower bound by $\Delta$. This concept is implemented as follows.

Let $J^b$ be the collection of the columns for which the restrictions of the dual problem are binding:

$$J^b = \{j \in J \mid cl_j = 0\} \quad [23]$$

We now have to make sure that two conditions are met to ensure feasibility of the resulting vector u:

1. $\forall j \in J^b : \text{if } a_{r_2 j} = 1 \text{ or } a_{r_3 j} = 1 \text{ then } a_{r_1 j} = 1$ [24]
2. $\forall j \in J^b : a_{r_2 j} \cdot a_{r_3 j} = 0$ [25]

Note that the proposed improvement is allowed if and only if these two conditions are met. If we have found three rows $r_1$, $r_2$ and $r_3$ for which the conditions hold, we determine the maximum allowed value of $\Delta$ such that the constraints in [16] hold for all columns.

The heuristic consists of a complete search of all combinations of three rows in the problem. Although further improvements are possible when this method is applied iteratively, we apply it only once for every possible combination of three rows.

## 4. Upper bounds and solutions

In this paper we describe an algorithm that is aimed at finding an optimal solution to a set partitioning problem. This algorithm uses a greedy primal heuristic that is designed to find a solution quickly and use this as the starting point of the branch and bound procedure. In this section, we first pay some attention to the literature on algorithms that are directed at finding solutions to set partitioning problems. Then we describe the primal heuristic and the branch and bound method used in our solver.

### *4.1 Literature*

While technological and theoretical progress has been immense in the last decades, it is now possible to solve fairly large set partitioning problems to optimality in a reasonable amount of time. Hoffman and Padberg (1993) describe a successful branch and cut algorithm, that is enhanced by Borndörfer (1998). Ryan (1992) and Fleuren (1988) describe implementations of branch and bound algorithms. Many of the exact methods discussed in literature incorporate a (greedy) primal heuristic to find an upper bound to the problem, see for example Hoffman and Padberg (1993). A successful implementation of a stand-alone heuristic that finds nearly optimal solutions for set partitioning problems can be found in Atamtürk et al. (1996).

### *4.2 Primal heuristic*

Before the start of the branch and bound algorithm, we apply a greedy primal heuristic to find an upper bound to the problem. This upper bound can be used to restrict the number of columns that have to be considered during the branching process. If we have a dual feasible vector $\lambda$ and a corresponding lower bound, given by [18], we can disregard all columns j for which $cl_j >$ upper bound – lower bound.

The greedy primal heuristic extends a partial solution with the column with the lowest Lagrangean costs that covers a particular uncovered row. This implies that we consider the rows in a certain order. We consider three different row orderings:

- The rows are sorted on decreasing dual values $\lambda_r$. This row ordering is based on the perception that rows with a high dual value have great influence on the objective value of the problem and thus are considered first.
- The rows are ordered on increasing number of non-zeros. This row ordering is based upon the idea that rows with a small number of non-zeros are more difficult to cover and thus can be best considered in the beginning of the heuristic.
- The rows are ordered on cover frequency. The cover frequency of row r with row s, cf(r,s) is the number of times that row s is covered by the columns that cover row r and can be seen as a measure for the overlap between rows r and s. The ordering is created as follows:

    1. row_order[0] = first row of problem
       $R' = \{row\_order[0]\}$
       i = 1
    2. $row\_order[i] = \arg\max_{r \in R \setminus R'} cf(row\_order[i-1], r)$
       $R' = R' \cup \{row\_order[i]\}$
       i = i + 1
    3. If R' = R then stop, else go to 2

For every row ordering, we perform 200 iterations. In every iteration, we consider the rows in the given sequence and add the column with the lowest Lagrangean costs that covers the next row to the partial solution. The iteration ends if either the problem becomes infeasible or we find a feasible solution. In the first case, the first row in the ordering that cannot be covered is put in front of the sequence and the next iteration is started. In the second case, the row that is in the middle of the ordering is put in front and the next iteration is started. If the primal heuristic does not find a solution to the problem, the upper bound is set to infinity.

### *4.3 Branch and bound*

Given the dual feasible vector $\lambda$, the lower bound given by [18], the Lagrangean costs vector given by [5] and the upper bound resulting from the primal heuristic, a branch and bound procedure is used to find the optimal solution. In every node of the branch and bound tree, a column is added to a partial solution. In contrast to most linear programming based algorithms, we do not branch on a variable basis, but on a row basis. In every node of the tree, we choose the row with the least number of active elements and branch on the active columns that cover this row. A column j is inactive if either it has nonzero's in common with one or more columns in the partial solution, or if for the Lagrangean costs $cl_j$ and the partial solution vector x it holds that:

$$cl_j > \text{upper bound} - \sum_r \lambda_r - \sum_k cl_k \cdot x_k \qquad [27]$$

When a partial solution is fathomed, we remove the last added column from the partial solution. There are two reasons why we can fathom a certain node:

1. The problem is infeasible because there is a row r that is only covered by inactive columns.
2. All rows are covered and the partial solution is a feasible solution to the problem.

Obviously, the speed of the branch and bound procedure depends heavily on the quality of the lower and upper bounds.

## 5. Solver composition

The sequence in which methods are applied can have a large influence on the performance of the solver. For example, when the row combination technique is applied, the knowledge of a lower- and upper bound can speed up the process, since columns j for which [24] holds do not have to be added to the problem. Examples of the interdependencies between preprocessing rules can be found in Van Krieken et al. (2003).

The composition of our solver is given below. A schematic overview is given in Figure 1.

1. Preprocessing techniques PP1, PP2, PP3 and PP4
2. Lagrangean relaxation and subgradient search
3. Dual heuristics DH1 and DH2
4. Primal heuristic
5. Preprocessing techniques PP6 and PP1
6. Lagrangean relaxation and subgradient search
7. Dual heuristics DH1 and DH2
8. Primal heuristic
9. Preprocessing techniques PP5, PP3 and PP4
10. Branch and bound

In step 2, parameter $\alpha$ is not adjusted in the way that is given by [14] and [15], but kept constant at 0.95, such that a lower bound is found very quickly. In step 6, we do the more sophisticated subgradient search to find a higher lower bound. In the next section, we discuss the performance of the algorithm.

## 6. Computational experiments

In this section, we discuss the performance of the solver. For a test set of 60 problems we show the performance in terms of the lower- and upperbound compared to the optimal values and the computational times compared to the time of the well-known Cplex solver. The computational

experiments are performed with a code that is written entirely in C++ and is tested on a normal desktop computer, running on MS Windows XP with a 2.4 Ghz Pentium processor and 1536 MB RAM.

### *6.1 The test set*

The test set that we use consists of 60 problems. From this set, 55 instances are real-life set partitioning problems that stem from the OR-library of Beasley (Beasley, 1990). This is the same set as is used in Hoffman & Padberg (1993) and Borndörfer (1998). The other 5 problems are set partitioning formulations of puzzles. Three of them, Heart, Meteor and Delta, are parts of the well-known Eternity puzzle (http://www.eternity-puzzle.co.uk). A description of the Bill's snowflake puzzle can be found at http://www.johnrausch.com/PuzzleWorld/puz/bills_snowflake.htm. Finally, the Exotic Fives puzzle is described at http://www.puzzles.force9.co.uk/gall2/exotic5.htm. Interested readers are invited to contact the authors to obtain the instances used in the experiments.

These puzzles are modeled as set partitioning problems as follows. The compartments of the puzzle are represented by the rows of the set partitioning problem. Every piece of the puzzle has several columns in the set partitioning tableau, representing the different ways that piece can be placed in the puzzle. The constraints make that no more than one piece covers each compartment. Moreover, we add one constraint for every piece to make sure that this piece is not used more than once.

To solve a puzzle, we just need a feasible solution to this problem. This is modeled by giving all the columns equal costs, such that we minimize the number of pieces used. This number is equal for all feasible solutions, since we have to use all the pieces.

The problem characteristics of the 60 instances are given in Table 1, where the density of a problem denotes the percentage of nonzero's in the constraint matrix

### *6.2 Results*

We first note that we leave two problems out of consideration at this point, aa01 and aa04. We deal with these problems in Paragraph 6.4. The results of the solver on the remaining 58 problem instances are given in Table 2. The columns 'lower' and 'upper' denote the lower- and upper bounds before branch and bound. Furthermore, 'time lower' denotes the time used to determine the lower bound, i.e. the time needed for the subgradient search and the dual heuristics. The last three columns of Table 2 show the total time needed by the solver to solve the problem, the total time needed by Cplex and the number of nodes used in the branch and bound tree before the optimal solution was found. For 27 out of the 58 problems, the optimal solution is found before the start of the branch and bound procedure. On average, the time needed to determine the lower bound of a problem is about one-third of the total time needed to solve the problem. The number of nodes in the branch and bound tree grows fast when the gap between lower- and upper bound increases. We cannot generally say that the time needed to solve the problems grows with the number of rows or columns, although the results do show a trend in

that direction. The most remarkable exceptions are the puzzles, whose results we will discuss in more detail in the next paragraph.

### *6.3 Comparison with Cplex*

Table 3 gives a summary of the comparison between our solver and Cplex. The total time of Cplex over the 58 problems is 627 seconds, while our solver takes 188 seconds, a difference of 439 seconds or 70%. The maximum time benefit of our solver on one instance is 140 seconds, while the maximum time difference in the advantage of Cplex is only 3 seconds. As can be seen by the results in Table 2, the difference in time between our solver and Cplex on the puzzle-instances is remarkable. As discussed in Paragraph 6.1, these problems are essentially feasibility problems and not optimization problems, meaning that a fast branching procedure is much more effective than a good lower bound. In the Cplex MIP solver, the accent is much more on lower bound determination than in our solver, while our solver takes advantage of the set partitioning structure in the branch and bound procedure. Note that, when we disregard the puzzle-instances, the time benefit is still over 50%.

The relative performance of the two solvers is illustrated by the performance profile in Figure 2. The concept of performance profiles to compare optimization methods is discussed in Dolan and Moré (2002). The profile shows that the set partitioning solver is faster than Cplex on 80% of the problems in the test set. Moreover, it shows that the calculation time of the set partitioning solver is within a factor two of the time of the best solver for all problems. On the other hand, the solution time of Cplex is within a factor two of the time of the best solver for about 60% of the problems. The profile indicates that the performance of the set partitioning solver is better than the performance of Cplex on this test set.

### *6.4 Problematic instances*

The two instances that are left out of consideration in the above comparison, aa01 and aa04, are much more difficult to solve. This observation can also be found in literature, for example Hoffman and Padberg (1993) say that they "require significantly more computational effort than the rest" and refer to them as "problem children". Borndörfer (1998) refers to them as "hard problems" and also gives an argument why these problems are more difficult: "closing the gap from the dual side seems to be what makes the instances (…) aa04 and aa01 hard". Hoffman and Padberg (1993) report that they solve aa01 in 4.01 hours and aa04 in 38.7 hours, while Borndörfer (1998) solves both problems in less than 10 minutes. Unfortunately, we were not able to solve both problems within a period of 24 hours with our solver. For aa04, we do find a solution, however the value of the solution is more than 10% away from the optimal solution. For aa01, we do not find a solution within 24 hours.

## 7. Conclusions and further research

This paper discusses a solver that is developed by the authors to solve set partitioning problems. The solver uses Lagrangean relaxation and dual heuristics to determine a lower bound, a primal heuristic to determine an upper bound, preprocessing to reduce the size of the problem and branch and bound to find the optimum. Apart from two hard cases, the solver performs very well on the test set of 60 problems. While the total time of the mathematical programming solver Cplex is 627 seconds, the time of the solver of the authors is only 188 seconds, a difference of 439 seconds, or 70%. The large gap between the calculation times of these solvers indicate that the development of specific solvers for set partitioning problems is worthwhile. Moreover, these results show that, with current technology, it is possible to solve large set partitioning problems to proven optimality in a reasonable amount of time. Comparable conclusions were taken in the past considering algorithms that use very fast commercial solvers to solve linear programming problems in a branch and cut setting. The solver discussed in this paper does not make use of any other mathematical programming solvers.

Further research is recommended on the two problem instances discussed in this paper that cannot be solved in a reasonable amount of time by our solver. Since several branch and cut solvers are able to solve these problems, more insight in the difficulty of these problems is desirable.

Most methods discussed in this paper can be applied to more general problems. We therefore recommend further research in the application of the methods discussed here, and extended versions of these methods, to more general problems. One can think of mixed set packing/set partitioning problems, but also problems that have constraints with low-integer coefficients and right-hand sides.

## References


Atamtürk, A., G.L. Nemhauser and M.W.P. Savelsbergh (1995) A combined Lagrangian, linear programming, and implication heuristic for large-scale set partitioning problems. Journal of heuristics, 1: 247-259.

Balas, E. and M.W. Padberg (1976) Set partitioning: A survey. SIAM Review, 18: 710-760.

Baldacci, R., E. Hadjiconstantinou, V. Maniezzo and A. Mingozzi. (2002) A new method for solving capacitated location problems based on a set partitioning approach. Computers & Operations Research, 29: 365-386.

Beasley, J.E. and B. Cao (1996) A tree search algorithm for the crew scheduling problem. European Journal of Operations Research, 94: 517-526.

Borndörfer, R. (1998) Aspects of set packing, partitioning and covering. Dissertation, Technical University of Berlin, Germany.

Dolan, E.D. and J.J. Moré (2002) Benchmarking optimization software with performance profiles. Mathematical Programming, 91: 201-213.

Fisher, M.L. and P. Kedia. (1990) Optimal solutions of set covering/partitioning problems using dual heuristics. Management Science, 36: 674-688.

Fleuren, H.A (1988) A computational study of the set partitioning approach for vehicle routing and scheduling problems. Dissertation, University of Twente, The Netherlands.

Geoffrion, A.M. (1974) Lagrangean relaxation for integer programming. Mathematical programming study, 2: 82-114.

Held, M., P.Wolfe and H.P. Crowder (1974) Validation of the subgradient approach. Mathematical Programming, 6: 62-88.

Hoffman, K.L and M. Padberg (1993) Solving airline crew scheduling problems by branch-and-cut. Management Science, 39: 657-682.

Hunting, M. (1998) Relaxation techniques for discrete optimization problems. Dissertation, University of Twente, The Netherlands.

Le Blanc, H.M., M.G.C. van Krieken, H.A. Fleuren and H.R. Krikke (2004) Collector Managed Inventory, a proactive planning approach to the collection of liquids coming from end-of-life vehicles. CentER Discussion Paper 2004-22.

Mingozzi, A., M.A. Boschetti, S. Ricciardelli and L. Bianco (1999) A set partitioning approach to the crew scheduling problem. Operations Research, 47: 873-888.

Nawijn, W.M. (1987) Optimizing the performance of a blood analyser: Application of the set partitioning problem. University of Twente Memorandum 626.

Ryan, D.M. (1992) The solution of massive generalized set partitioning problems in aircrew rostering. Operational Research Society, 43: 459-467.

Van Krieken, M.G.C., H.A. Fleuren and M.J.P. Peeters (2003) Problem reduction in set partitioning problems. CentER Discussion Paper 2003-80.

*Figure 1: Solver composition*

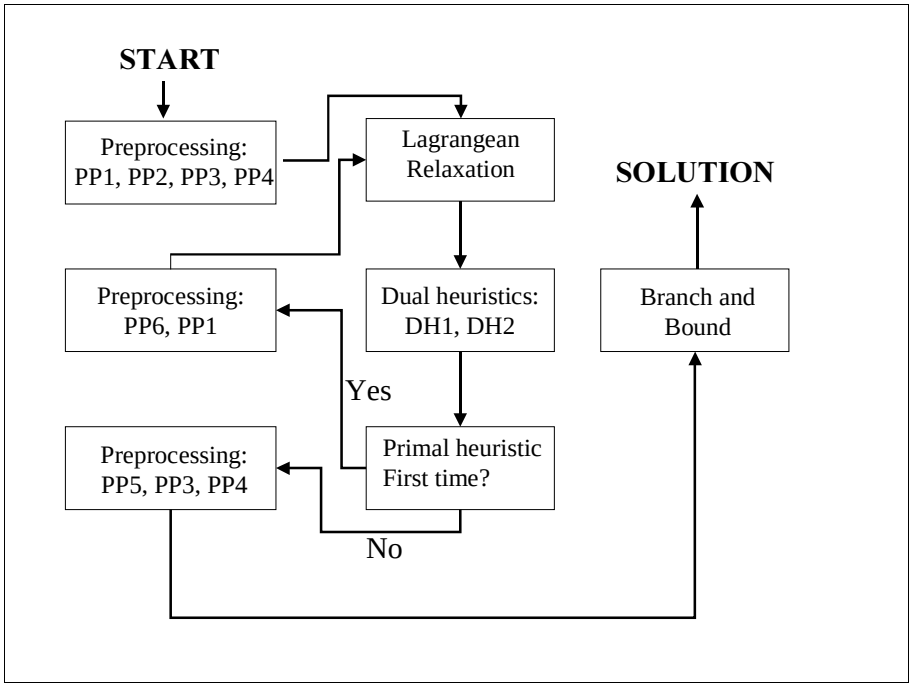

*Figure 2: Performance profile the SPP solver and Cplex*

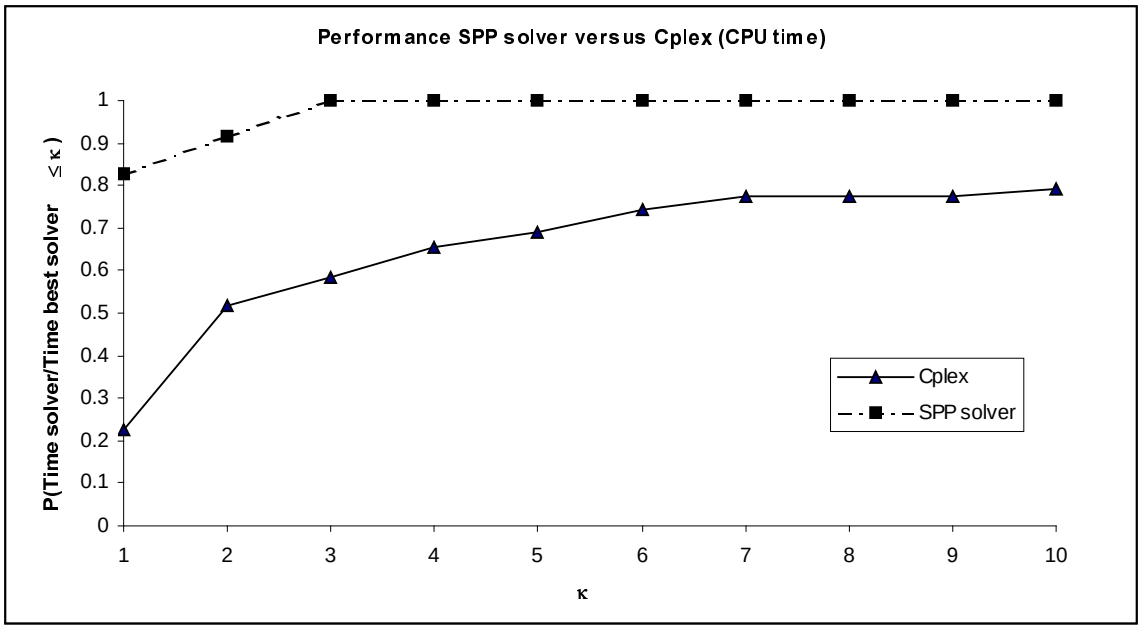

*Table 1: Problem characteristics*

| Problem | Columns | Rows | Elements | Density |
|---|---|---|---|---|
| nw41 | 197 | 17 | 740 | 22% |
| nw32 | 294 | 19 | 1357 | 24% |
| nw40 | 404 | 19 | 2069 | 27% |
| nw08 | 434 | 24 | 2332 | 22% |
| nw15 | 467 | 31 | 2830 | 20% |
| nw21 | 577 | 25 | 3591 | 25% |
| nw22 | 619 | 23 | 3399 | 24% |
| nw12 | 626 | 27 | 3380 | 20% |
| nw39 | 677 | 25 | 4494 | 27% |
| nw20 | 685 | 22 | 3722 | 25% |
| nw23 | 711 | 19 | 3350 | 25% |
| nw37 | 770 | 19 | 3778 | 26% |
| nw26 | 771 | 23 | 4215 | 24% |
| nw10 | 853 | 24 | 4336 | 21% |
| nw34 | 899 | 20 | 5045 | 28% |
| Heart | 926 | 180 | 8334 | 5% |
| nw43 | 1072 | 18 | 4859 | 25% |
| nw42 | 1079 | 23 | 6533 | 26% |
| Delta | 1194 | 126 | 10746 | 7% |
| nw28 | 1210 | 18 | 8553 | 39% |
| nw25 | 1217 | 20 | 7341 | 30% |
| nw38 | 1220 | 23 | 9071 | 32% |
| nw27 | 1355 | 22 | 9395 | 32% |
| nw24 | 1366 | 19 | 8617 | 33% |
| nw35 | 1709 | 23 | 10494 | 27% |
| nw36 | 1783 | 20 | 13160 | 37% |
| Bill's snowflake | 2300 | 585 | 103938 | 8% |
| Exotic fives | 2440 | 72 | 14640 | 8% |
| Meteor | 2464 | 60 | 14784 | 10% |
| nw29 | 2540 | 18 | 14193 | 31% |
| nw30 | 2653 | 26 | 20436 | 30% |
| nw31 | 2662 | 26 | 19977 | 29% |
| nw19 | 2879 | 40 | 25193 | 22% |
| nw33 | 3068 | 23 | 21704 | 31% |
| nw09 | 3103 | 40 | 20111 | 16% |
| nw07 | 5172 | 36 | 41187 | 22% |
| aa02 | 5198 | 531 | 36359 | 1% |
| nw06 | 6774 | 50 | 61555 | 18% |
| aa04 | 7195 | 426 | 52121 | 2% |
| aa06 | 7292 | 646 | 51728 | 1% |
| kl01 | 7479 | 55 | 56242 | 14% |
| aa05 | 8308 | 801 | 65953 | 1% |
| aa03 | 8627 | 825 | 70806 | 1% |
| nw11 | 8820 | 39 | 57250 | 17% |
| aa01 | 8904 | 823 | 72965 | 1% |
| nw18 | 10757 | 124 | 91028 | 7% |
| us02 | 13635 | 100 | 192716 | 14% |
| nw13 | 16043 | 51 | 104541 | 13% |
| us04 | 28016 | 163 | 297538 | 7% |
| kl02 | 36699 | 71 | 212536 | 8% |
| nw03 | 43749 | 59 | 363939 | 14% |
| nw01 | 51975 | 135 | 410894 | 6% |
| us03 | 85552 | 77 | 1211929 | 18% |
| nw04 | 87482 | 36 | 636666 | 20% |
| nw02 | 87879 | 145 | 721736 | 6% |
| nw17 | 118607 | 61 | 1010039 | 14% |
| nw14 | 123409 | 73 | 904910 | 10% |
| nw16 | 148633 | 139 | 1501820 | 7% |
| nw05 | 288507 | 71 | 2063641 | 10% |
| us01 | 1053137 | 145 | 13636541 | 9% |

*Table 2: Results solver*

| Problem | Optimum | Lower | Upper | Time lower | Time total | Time Cplex | Nodes |
|---|---|---|---|---|---|---|---|
| nw41 | 11307 | 11307.00 | 11307 | 0.000 | 0.000 | 0.020 | 0 |
| nw32 | 14877 | 14569.96 | 14877 | 0.016 | 0.030 | 0.030 | 6 |
| nw40 | 10809 | 10657.06 | 10848 | 0.015 | 0.015 | 0.020 | 3 |
| nw08 | 35894 | 35894.00 | 35894 | 0.030 | 0.030 | 0.030 | 0 |
| nw15 | 67743 | 67743.00 | 67743 | 0.000 | 0.000 | 0.110 | 0 |
| nw21 | 7408 | 7408.00 | 7408 | 0.016 | 0.016 | 0.030 | 0 |
| nw22 | 6984 | 6984.00 | 6984 | 0.000 | 0.000 | 0.030 | 0 |
| nw12 | 14118 | 14118.00 | 14118 | 0.000 | 0.000 | 0.030 | 0 |
| nw39 | 10080 | 9868.50 | 10410 | 0.015 | 0.015 | 0.060 | 6 |
| nw20 | 16812 | 16624.72 | 16965 | 0.060 | 0.060 | 0.060 | 11 |
| nw23 | 12534 | 12317.00 | 12534 | 0.031 | 0.031 | 0.060 | 14 |
| nw37 | 10068 | 10068.00 | 10068 | 0.000 | 0.000 | 0.030 | 0 |
| nw26 | 6796 | 6796.00 | 6796 | 0.000 | 0.000 | 0.050 | 0 |
| nw10 | 68271 | 68271.00 | 68271 | 0.015 | 0.109 | 0.050 | 0 |
| nw34 | 10488 | 10453.50 | 10488 | 0.015 | 0.015 | 0.050 | 4 |
| heart | 180 | 179.54 | inf | 0.546 | 0.610 | 95.330 | 853 |
| nw43 | 8904 | 8904.00 | 8904 | 0.000 | 0.015 | 0.050 | 0 |
| nw42 | 7656 | 7484.94 | 7832 | 0.047 | 0.063 | 0.090 | 21 |
| delta | 126 | 126.00 | inf | 0.313 | 0.359 | 2.000 | 1981 |
| nw28 | 8298 | 8298.00 | 8298 | 0.000 | 0.000 | 0.060 | 0 |
| nw25 | 5960 | 5852.00 | 5960 | 0.032 | 0.032 | 0.080 | 5 |
| nw38 | 5558 | 5552.00 | 5630 | 0.047 | 0.062 | 0.080 | 8 |
| nw27 | 9933 | 9933.00 | 9933 | 0.000 | 0.000 | 0.060 | 0 |
| nw24 | 6314 | 6314.00 | 6314 | 0.000 | 0.000 | 0.080 | 0 |
| nw35 | 7216 | 7216.00 | 7216 | 0.016 | 0.016 | 0.060 | 0 |
| nw36 | 7314 | 7259.96 | 7328 | 0.078 | 0.109 | 0.280 | 27 |
| Bill's snowflake | 34 | 11.96 | inf | 6.297 | 17.719 | 94.300 | 42734 |
| Exotic fives | 12 | 11.93 | inf | 0.922 | 0.969 | 73.980 | 47 |
| meteor | 60 | 60.00 | inf | 0.405 | 0.453 | 15.560 | 286 |
| nw29 | 4274 | 4189.80 | 4344 | 0.093 | 0.093 | 0.160 | 12 |
| nw30 | 3942 | 3942.00 | 3942 | 0.016 | 0.016 | 0.160 | 0 |
| nw31 | 8038 | 7980.00 | 8046 | 0.110 | 0.110 | 0.190 | 11 |
| nw19 | 10898 | 10898.00 | 10898 | 0.016 | 0.032 | 0.130 | 0 |
| nw33 | 6678 | 6678.00 | 6678 | 0.000 | 0.031 | 0.160 | 0 |
| nw09 | 67760 | 67760.00 | 67760 | 0.016 | 0.313 | 0.130 | 0 |
| nw07 | 5476 | 5476.00 | 5476 | 0.016 | 0.031 | 0.200 | 0 |
| aa02 | 30494 | 30494.00 | 30494 | 0.844 | 1.297 | 0.510 | 0 |
| nw06 | 7810 | 7639.72 | 8706 | 0.468 | 0.562 | 1.020 | 100 |
| aa06 | 27040 | 26973.26 | 27129 | 1.751 | 5.812 | 3.160 | 286049 |
| kl01 | 1086 | 1083.45 | 1087 | 0.172 | 0.234 | 0.940 | 489 |
| aa05 | 53839 | 53721.42 | 53949 | 1.703 | 5.954 | 5.360 | 366307 |
| aa03 | 49649 | 49607.10 | 49649 | 1.468 | 3.156 | 3.020 | 2125 |
| nw11 | 116256 | 112403.86 | 116256 | 0.314 | 1.156 | 0.410 | 214073 |
| nw18 | 340160 | 329099.16 | 342998 | 1.469 | 4.000 | 1.530 | 236132 |
| us02 | 5965 | 5965.00 | 5965 | 0.141 | 0.453 | 0.760 | 0 |
| nw13 | 50146 | 50131.67 | 50206 | 0.751 | 1.078 | 0.810 | 66 |
| us04 | 17854 | 17722.04 | 17854 | 0.359 | 1.031 | 1.450 | 79 |
| kl02 | 219 | 215.05 | 219 | 3.390 | 4.093 | 4.130 | 119380 |
| nw03 | 24492 | 24447.00 | 24759 | 1.468 | 1.984 | 3.890 | 45 |
| nw01 | 114852 | 114852.00 | 114852 | 2.657 | 4.187 | 3.340 | 0 |
| us03 | 5338 | 5338.00 | 5338 | 0.406 | 3.750 | 5.640 | 0 |
| nw04 | 16862 | 16310.18 | 17264 | 3.610 | 5.140 | 15.050 | 9115 |
| nw02 | 105444 | 105444.00 | 105444 | 1.781 | 4.187 | 6.520 | 0 |
| nw17 | 11115 | 10874.03 | 11481 | 1.126 | 2.078 | 13.130 | 157 |
| nw14 | 61844 | 61844.00 | 61844 | 2.687 | 7.547 | 7.590 | 0 |
| nw16 | 1181590 | 1181590.00 | 1181590 | 2.625 | 10.594 | 18.990 | 0 |
| nw05 | 132878 | 132878.00 | 132878 | 6.469 | 12.656 | 19.280 | 0 |
| us01 | 10036 | 9960.58 | 10056 | 14.844 | 86.000 | 226.340 | 386 |

*Table 3: Comparison solver with Cplex*

| | |
|---|---|
| Number of instances | 58 |
| Number of instances time solver <= time Cplex | 48 |
| Total time solver | 188.303 |
| Total time Cplex | 626.670 |
| Time benefit solver | 438.367 |
| Percentage time benefit solver | 70% |
| Minimum time benefit solver | -2.652 |
| Maximum time benefit solver | 140.340 |