Journal of Heuristics, 1: 247–259 (1995)
© 1995 Kluwer Academic Publishers

A Combined Lagrangian, Linear Programming, and Implication Heuristic for Large-Scale Set Partitioning Problems

A. ATAMTÜRK
Georgia Institute of Technology, School of Industrial and Systems Engineering, Atlanta, GA 30332-0205

G.L. NEMHAUSER
Georgia Institute of Technology, School of Industrial and Systems Engineering, Atlanta, GA 30332-0205

M.W.P. SAVELSBERGH
Georgia Institute of Technology, School of Industrial and Systems Engineering, Atlanta, GA 30332-0205

Abstract

Given a finite ground set, a set of subsets, and costs on the subsets, the set partitioning problem is to find a minimum cost partition of the ground set. Many combinatorial optimization problems can be formulated as set partitioning problems. We present an approximation algorithm that produces high-quality solutions in an acceptable amount of computation time. The algorithm is iterative and combines problem size-reduction techniques, such as logical implications derived from feasibility and optimality conditions and reduced cost fixing, with a primal heuristic based on cost perturbations embedded in a Lagrangian dual framework, and cutting planes. Computational experiments illustrate the effectiveness of the approximation algorithm.

Key Words: set partitioning, preprocessing, linear programming, Lagrangian dual

1. Introduction

Given a finite ground set, a set of subsets, and costs on the subsets, the set partitioning problem is to find a minimum cost partition of the ground set. Let A be an $m \times n$ 0-1 matrix with a row for each element in the ground set and a column for every characteristic vector of a feasible subset. Then the set partitioning problem can be formulated as

$$\begin{array}{lll} \min & cx & \\ s.t. & Ax = 1 & \\ & x_j \in \{0, 1\} \quad j = 1, \ldots, n. & \end{array} \qquad (SP)$$

Many real-life problems, such as vehicle routing and airline crew scheduling, can be formulated as set partitioning problems. Since there is a column for each feasible subset and the number of feasible subsets is usually huge, the number of variables is also huge.

The set partitioning problem has been studied extensively (see Balas and Padberg, 1976, for a survey of some of its applications and solution methods). Recently, set partitioning based algorithms have been applied very successfully to airline crew scheduling problems (see, for instance, Marsten and Shepardson, 1981, and Hoffman and Padberg, 1993). In both cases, an LP based branch-and-bound algorithm has been used to solve large-scale instances of the set partitioning problem. Other solution approaches have been developed as well. Fisher and Kedia (1990) present a dual-ascent algorithm. Chu and Beasley (1995) discuss a genetic algorithm, and Wedelin (1995) introduces a Lagrangian dual approach with cost perturbations.

Our goal has been the design and implementation of a fast approximation algorithm that generates provably good solutions. Therefore, our algorithm produces both an upper bound (a feasible solution) and a lower bound on the optimal objective function value.

Our approximation algorithm combines several, largely known, techniques that have been useful either specifically for set partitioning problems or for general mixed 0-1 integer programming problems and applies them iteratively. The synthesis and iterative application of these techniques result in an empirically demonstrable efficient approximation algorithm.

The paper is organized as follows. In Section 2, we describe problem size-reduction techniques based on logical implications derived from feasibility and optimality conditions. In Section 3, we discuss a Lagrangian dual algorithm with an embedded cost perturbation technique that is used to generate feasible solutions. In Section 4, we review reduced cost fixing and indicate its role in integrating the various components into a single iterative algorithm. In Section 5, we present a cut generation technique to improve the LP lower bound. In Section 6, we put the pieces together and present the complete algorithm. In Section 7, we describe the computational experiments that illustrate the efficiency of our iterative approximation algorithm. Finally, in Section 8, we present some conclusions and ideas for future research.

2. Preprocessing techniques

For large-scale instances, preprocessing may substantially reduce the size of the instance and therefore the overall computation time. Our algorithm incorporates three techniques to reduce the size of an instance: removal of duplicate columns, removal of dominated rows, and fixing variables by probing. Each of these techniques will be explained in more detail below. The three techniques are applied iteratively to produce the best possible results. We emphasize that preprocessing is an integral part of our approximation algorithm and may be called many times. See Garfinkel and Nemhauser (1969) for the historical use of preprocessing in the solution of set partitioning problem.

2.1. Removal of duplicate columns

In many applications, especially in airline crew scheduling problems, an instance may contain many replicated columns with different objective function coefficients. Since at most one of them can be part of any feasible solution, we can delete all of them except for a column with the minimum cost.

Although removing duplicate columns is a trivial operation, care has to be taken to implement it efficiently. Even though the number of nonzeros per column is usually small, typically less than ten, checking every pair of columns for equivalence may require hours of computation time when the number of columns is in the hundreds of thousands. Hoffman and Padberg (1993) present a more sophisticated approach. They suggest performing a comparison of columns if and only if the sum of the index of the first and last nonzero entry are equal. Observe that this can be implemented efficiently by sorting the columns in order of nonincreasing values of these sums. The best performance is obtained when the least number of columns needs to be compared. Therefore, we have modified Hoffman and Padberg's approach as follows. First, we assign to each row a random number from a large distribution. Second, we assign to each column a number that is equal to the sum of the numbers associated with the rows in which this column has a 1. This scheme assigns the same number to duplicate columns and with very high probability different numbers to different columns. Therefore, the number of actual comparisons that needs to be performed is very small.

2.2. *Removal of dominated rows*

For each element i let $V(i) = \{k : a_{ik} = 1\}$ denote the set of columns in which element i appears. A row i is said to be dominated by a row j if $V(i) \subset V(j)$. Obviously, if row i is dominated by row j, then $x_k = 0$ for all $k \in V(j) \setminus V(i)$ in every feasible solution. Therefore, besides deleting dominated row i, we also fix all variables $x_k = 0$ for all $k \in V(j) \setminus V(i)$.

In order to find dominated rows efficiently, we store the variables appearing in a row in increasing order of their indices so that nondominance can be detected as early as possible when comparing two rows. Even though there are worst case instances where no dominance can be found and one needs to perform ml comparisons, where l is the number of nonzeros in the matrix, in practice this rarely happens. In our experiments, actual computation time seems to depend only on m.

A slightly different situation can also be exploited successfully. Suppose there are two rows that differ only by two variables—say, x_u and x_v—but neither one of the rows dominates the other. It is easy to see that x_u and x_v must have the same value in every feasible solution. Therefore, if there also exists a row in which both of them appear, then they can be fixed at zero. This observation is used during the search for dominated rows to fix additional variables. Note that if there does not exist a row in which both of the variables appear, then we can replace the two variables by a single variable by merging the associated columns and adding the cost coefficients. However, we have not implemented this observation because it cannot be done efficiently relative to the savings it might yield.

2.3. *Fixing variables by probing*

Probing a variable means tentatively setting the variable to one of its bounds and observing if its implications imply infeasibility, in which case the variable can be permanently fixed to the opposite bound. Probing has been used effectively in general mixed 0-1 integer programming (Savelsbergh, 1994). However, to the best of our knowledge, it has not been used successfully, for set partitioning. Probing can be very effective in reducing the size of a set partitioning instance. However, for each probed variable, we need to reduce the system—that is, effectuate *all* implications that can be derived from tentatively setting the variable to one of its bounds—and determine whether it is infeasible. Since this operation may be time consuming, we apply probing judiciously as explained later.

3. Constructing feasible solutions

We have embedded a variation of the Lagrangian dual cost perturbation algorithm of Wedelin (1995) to try to construct primal feasible solutions. In this section, we present an overview of the heuristic and discuss how we have incorporated it into our approximation algorithm.

Let $L(\lambda)$, for a given λ, denote the value of the Lagrangian relaxation of SP obtained by dualizing all equality constraints—that is,

$$L(\lambda) = \lambda 1 + \min_{x \in \{0,1\}} (c - \lambda A)x,$$

and let the associated Lagrangian dual be given by

$$\max_{\lambda \in R^m} L(\lambda). \tag{LD}$$

Let $\bar{c} = c - \lambda A$. Observe that if LD has an optimal solution (x^*, λ^*) such that x^* is feasible to SP and for all j either $\bar{c}_j < 0$ or $\bar{c}_j > 0$, then x^* is the unique optimal solution to the Lagrangian relaxation and also an optimal solution to SP. Observe that since LD gives the same bound as the LP relaxation of SP, this situation will only occur if x^* is the unique optimal solution to the LP relaxation that is complementary to λ.

Wedelin proposed to solve LD by an iterative coordinate search method. Let e_i be the ith unit vector and $\lambda \in R^m$. Then the step-size problem for λ and direction e_i can be stated as

$$\max_{\alpha \in R} L(\lambda + \alpha e_i) = \lambda 1 + \max_{\alpha \in R} (\alpha + \min_{x \in \{0,1\}} (\bar{c} - \alpha a_i)x),$$

where a_i denotes the ith row of A. Since a_i is a 0-1 vector, $L(\lambda + \alpha e_i)$ is stationary for $r^- \leq \alpha \leq r^+$, where r^- and r^+ are the smallest and second smallest reduced costs of variables with nonzero coefficients in row a_i. Since L is a concave function, we conclude that $\alpha^* \in [r^-, r^+]$ is an optimal step size. Hence, we can iteratively solve LD by starting from an arbitrary λ_0 and by solving a step size problem along each coordinate direction. In each direction e_i, the step length α^* is set to $(r^+ + r^-)/2$. Therefore, if $V(i)$ denotes the set of variables appearing in row i, then $\bar{c}_j$ for $j \in V(i)$ is updated by

$$\bar{c}_j \leftarrow \bar{c}_j - \frac{r^+ + r^-}{2}.$$

Since our objective is not necessarily to solve the Lagrangian dual but to find a good feasible solution to SP, we perturb c at each iteration of the coordinate search to move $\bar{c}_j$ away from 0. That is, we try to perturb c in such a way that when LD is solved to optimality the reduced costs $\bar{c}$ of the associated Lagrangian relaxation are such that either $\bar{c}_j < 0$ or $\bar{c}_j > 0$ for all j.

The perturbation of the objective function is introduced by using two different values for α in the step-size problem. More precisely, $\bar{c}_j$ for $j \in V(i)$ is updated by

$$\bar{c}_j \leftarrow \begin{cases} \bar{c}_j - \dfrac{r^+ + r^-}{2} - \dfrac{K(r^+ - r^-)}{1 - K} - \delta, & \bar{c}_j \leq r^- \\[2ex] \bar{c}_j - \dfrac{r^+ + r^-}{2} - \dfrac{K(r^+ - r^-)}{1 - K} + \delta, & \bar{c}_j \geq r^+, \end{cases}$$

where K is a perturbation parameter chosen from the interval [0, 1] and δ is a small constant to force the reduced costs to be nonzero. We observed that the solution quality is highly dependent on the choice of K and that small values of K usually result in higher quality solutions.

Although time consuming, the heuristic tends to give better solutions than an LP-based diving heuristic, which works as follows. In each step, an LP is solved, integer valued variables are fixed to their values, and a fractional variable is fixed to one of its bounds. This step is iterated until either an integral solution is found or the LP becomes infeasible.

Wedelin proposed the cost perturbation heuristic as an alternative to LP based algorithms for solving large-scale set partitioning problems. However, we have effectively incorporated it into an LP-based algorithm. Since iterative coordinate search methods may stall for functions that are not differentiable everywhere, the heuristic may stall at the nondifferentiable points of the piecewise linear concave function L. We reduce the chance of stalling by choosing an initial λ given by the dual LP solution.

4. Reduced cost fixing

Reduced cost fixing is a well-known technique used in LP-based branch-and-bound algorithms for mixed 0-1 programs. It fixes nonbasic variables based on implications derived from optimality conditions. Since the upper bounds on the variables in the LP relaxation of SP are redundant, they are not explicitly included in the LP relaxation. Therefore, every variable at its upper bound is basic and every nonbasic variable is at its lower bound. Hence $\bar{c}_j \geq 0$ for all nonbasic variables in an optimal LP solution. Let z_{LP} be the current LP value and z_H be the value of the current best primal feasible solution. If $z_{LP} + \bar{c}_j > z_H$, then x_j must be at its lower bound in every optimal solution. Hence we can fix x_j to 0.

The success of reduced cost fixing strongly depends on the quality of the LP relaxation and the quality of the best primal feasible solution. The closer the two bounds are, the more variables may be fixed. Reduced cost fixing complements probing in the sense that it is based on optimality whereas probing is based on feasibility. When some variables are fixed by reduced cost fixing, it may be possible to remove more dominated rows and to fix more variables by probing. These two observations illustrate the interactions among the techniques embedded in our approximation algorithm and motivate the iterative application of these techniques.

5. Clique inequalities

A clique C in a graph is a set of nodes with the property that each pair of nodes in the set is connected by an edge. The last technique embedded in our approximation algorithm strengthens the LP relaxation by adding clique inequalities of the form $\Sigma_{j \in C} x_j \leq 1$. Better linear programming bounds may lead to more variables being fixed by reduced cost fixing, which in turn may lead to more variables being fixed by probing, better performance of the Lagrangian heuristic, and so on. Clique inequalities are facet defining for the set packing relaxation of SP (Padberg, 1973) and therefore valid for SP. We build a conflict graph for the variables with fractional values in the current LP solution. That is, we introduce a node for each variable with fractional value and an edge between two nodes if the variables associated with these nodes share a common row in the coefficient matrix. Then if C is a clique in the conflict graph, $\Sigma_{j \in C} x_j \leq 1$ is a valid inequality. In order to find cliques in the conflict graph, we have implemented a greedy type search algorithm. Once a violated inequality is found, we look for larger cliques that contain the violated clique to obtain a stronger inequality.

6. The approximation algorithm

The approximation algorithm combines the techniques discussed in the previous sections and applies them iteratively. This synthesis and the iterative application of the techniques yield an efficient and effective approximation algorithm.

A flowchart of the approximation algorithm is given in Figure 1.

The approximation algorithm has two integrated loops, a first loop and a second loop. There are two reasons for placing clique generation in the second loop. It prevents the LP from getting too large and clique generation can be very time consuming, especially the search for a larger clique containing the violated clique.

In the first loop, we start by removing any duplicate columns from the formulation. In our computational experiments, we found that for the large instances removing duplicate columns can reduce the size by more than 50 percent.

Next, we search for and delete dominated rows. Since removing dominated rows and fixing variables may lead to new dominated rows, we iterate our search for dominated rows until none exists.

Next, we try to fix variables by probing. Since fixing variables by probing is computationally intensive, we do not do this the very first iteration because the number of active variables is still large (usually the number of active variables drops significantly after reduced cost fixing has been applied). For the same reason, we only make two passes through the

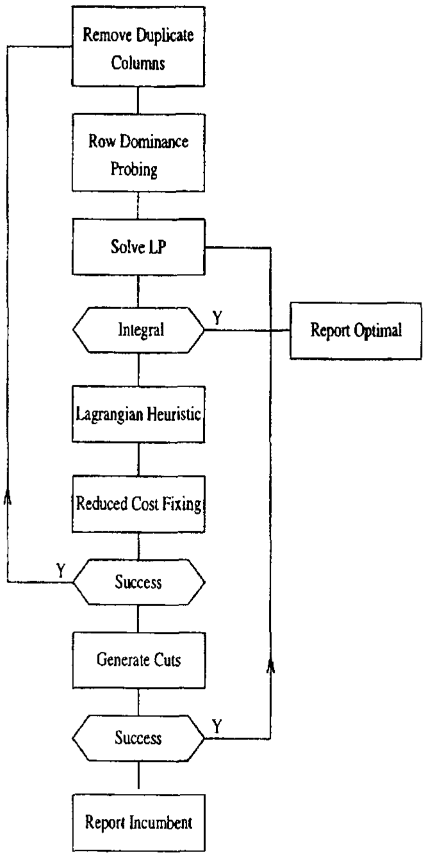

Figure 1. Flowchart of the algorithm.

variables, even though this means we may not fix as many variables as possible. Computational experiments have shown that two passes give the best balance between efficiency and effectiveness. Note that probing would have fixed all the variables that were fixed during the deletion of dominated rows. However, deleting dominated rows is computationally far less expensive than fixing variables by probing.

After the application of the preprocessing techniques, we solve the LP relaxation of the remaining problem. Computational experiments have shown that a dual simplex algorithm with steepest edge pricing performs best. If the solution to the LP relaxation is integral, the algorithm stops with a proof of optimality. If the solution is fractional, then we try to construct a feasible solution. In the first iteration, we use a diving heuristic. In all subsequent iterations, we use the Lagrangian dual heuristic. The quality of the primal feasible solutions produced by the diving heuristic is not as good as those produced by the Lagrangian dual heuristic, but these solutions are obtained much faster, especially for large instances. If the new solution is better than the incumbent, then it is accepted as the new incumbent; otherwise, it is rejected.

Next, we apply reduced cost fixing using the current linear programming solution and the current best primal feasible solution. If any variables are fixed, we return to removal of dominated rows.

In our computational experiments, we have seen that iterating preprocessing, solving LP, constructing a primal solution, and reduced cost fixing is very effective in reducing the size of the instance. This is very important because studies have indicated that the LP relaxations of large-scale instances are highly fractional whereas the LP relaxations of small instances have relatively few fractional variables, which makes them easier to solve (Hoffman and Padberg, 1993). We have also observed that the solution quality of the primal heuristics tends to be better for smaller sizes.

In many places, some fine tuning may improve the performance of the approximation algorithm. The Lagrangian dual-cost perturbation heuristic is very sensitive to the choice of K. Other possible enhancements are to decide whether or not to fix variables by probing based on a threshold value (that is, only fix variables by probing if the number of active variables is less than a given threshold), to decide which heuristic to use based on a threshold value (that is, use the Lagrangian dual cost perturbation heuristic if the number of active variables is less than a given threshold and otherwise use the diving heuristic), and to limit the number of variables that we consider when we look for larger cliques after a violated clique inequality has been found.

7. Computational results

The approximation algorithm has been implemented using MINTO version 2.0. MINTO is a software system that solves mixed-integer linear programs by a branch-and-bound algorithm with linear programming relaxations. It also provides automatic constraint classification, preprocessing, primal heuristics, and constraint generation. Moreover, the user can enrich the basic algorithm by providing a variety of specialized application routines that can customize MINTO to achieve maximum efficiency for a problem class. An overview of MINTO, discussing the design philosophy and general features, can be found in Nemhauser, Savelsbergh, and Sigismondi (1994), and a detailed description of the customization options can be found in Savelsbergh and Nemhauser (1994). Here we only use MINTO to solve the root node problem—that is, no branching is done.

In order to test the efficiency of our approximation algorithm, we have compared its performance to existing algorithms discussed in the literature. For most of our computational experiments, we have used the Hoffman-Padberg (HP) data set (available via anonymous ftp at happy.gmu.edu in the directory /pub/acs). This data set consists of forty-eight real-life airline crew scheduling problems. Problem us01 of the data set, which has 145 rows and 1,053,137 columns, could not be run due to lack of memory on our machine.

All runs were done on an IBM RS6000 Model 590 workstation. Hoffman and Padberg used an IBM RS6000 Model 550 workstation, except for the two largest instances, which were run on a single processor of a CONVEX C-220 because it had more memory. Since the model 590 is about twice as fast as the model 550, whenever we report times for HP's algorithm, we divide the time by 2 to reflect the speed difference. Since we do not know how the IBM RS6000 Model 590 compares to the CONVEX-220, we do not report times for HP's algorithm for the two largest instances.

For the remaining forty-seven problems in the data set, our approximation algorithm found an optimal solution. In fact, it also proved optimality! Since the smaller problems in the data set are very easy, we present results only for the 17 largest problems.

In Tables 1 and 2, we show the number of times the embedded techniques in our approximation algorithm as well as in the HP algorithm have been applied. Note that we only count the number of times the Lagrangian dual-cost perturbation heuristic has been applied—that is, we do not count the diving heuristic which is used only once.

Except for the problems with an integral LP solution after initial preprocessing, the number of LP calls for our approximation algorithm is much smaller than HPs. This indicates the effectiveness of our aggressive preprocessing and the Lagrangian dual heuristic. Furthermore, the HP algorithm has called its constraint generator ninety-seven times, generated 915 cuts, and required branching for three instances. Our approximation algorithm called the cut generator only twice, generated only twenty-six cuts, and proved optimality in all cases. The small number of constraint generation calls in our approximation algorithm shows that almost all of the problems were solved to optimality in the first loop of the algorithm. Also, adding only twenty-six cuts shows that we have been very successful in keeping the size of the active LP small.

In Table 3, we present the CPU times of our approximation algorithm and compare them to the solution times for the HP algorithm. Furthermore, we break down the CPU time over the different components of our algorithm. Note that the component times do not add up to the total CPU time. There are several reasons for that. First, the time used by the diving heuristic is not accounted for, which may be substantial for larger problems since it

Table 1. Number of calls.

Columns	Rows	RDom Calls	Probe Calls	LP Calls	Heuristic Calls	CG Calls	Number of Cuts
5172	36	1	0	1	0	0	0
5198	531	4	0	1	0	0	0
6774	50	9	4	3	1	0	0
7292	646	21	6	5	3	1	8
7479	55	14	8	5	3	0	0
8308	801	21	4	3	1	1	18
8627	825	20	6	4	2	0	0
8820	39	3	1	2	0	0	0
10757	124	7	2	2	0	0	0
13635	100	2	0	1	0	0	0
16043	51	7	3	3	1	0	0
28016	163	7	2	2	0	0	0
36699	71	9	4	3	1	0	0
85552	77	3	0	1	0	0	0
118607	61	9	4	3	1	0	0
123409	73	1	0	1	0	0	0
148633	139	1	0	1	0	0	0

Table 2. Performance measures from Hoffman and Padberg Table 3.

Columns	Rows	LP Calls	CG Calls	Number of Cuts	Number of Nodes
5172	36	1	0	0	0
5198	531	1	0	0	0
6774	50	10	3	33	0
7292	646	12	3	74	0
7479	55	32	20	229	2
8308	801	53	42	345	4
8627	825	12	2	37	0
8820	39	4	1	3	0
10757	124	3	1	15	0
13635	100	1	0	0	0
16043	51	5	2	4	0
28016	163	2	0	0	0
36699	71	10	8	127	0
85552	77	1	0	0	0
118607	61	43	15	48	4
123409	73	1	0	0	0
148633	139	1	0	0	0

Table 3. Computation times (seconds).

Columns	Rows	RmDup Time	RDom Time	Probe Time	LP Time	Heuristic Time	CG Time	Total CPU	HP CPU
5172	36	0.23	0.01	0.00	0.43	0.00	0.00	1.48	0.37
5198	531	0.06	0.44	0.00	3.02	0.00	0.00	4.14	5.08
6774	50	0.13	0.11	7.48	1.15	2.38	0.00	13.55	5.21
7292	646	0.10	1.44	1.31	10.44	10.68	0.07	51.49	18.65
7479	55	0.13	0.28	0.88	1.41	2.03	0.00	5.61	17.70
8308	801	0.10	2.61	8.19	17.28	15.05	0.90	53.93	107.65
8627	825	0.10	1.63	0.71	19.27	1.72	0.00	35.56	24.21
8820	39	0.26	0.02	0.00	0.80	0.00	0.00	2.62	1.03
10757	124	0.33	0.14	0.88	3.42	0.00	0.00	7.96	31.25
13635	100	0.60	0.86	0.00	2.24	0.00	0.00	5.80	2.39
16043	51	0.59	0.04	0.02	1.58	0.16	0.00	5.51	2.15
28016	163	2.47	1.91	0.02	2.62	0.00	0.00	12.15	5.60
36699	71	2.11	0.60	3.33	7.71	3.46	0.00	34.78	67.19
85552	77	11.47	5.45	0.00	10.26	0.00	0.00	42.55	10.13
118607	61	7.20	0.19	0.78	20.53	0.23	0.00	64.05	43.76
123409	73	5.04	0.10	0.00	16.02	0.00	0.00	33.03	—
148633	139	4.03	3.59	0.00	78.06	0.00	0.00	105.91	—

involves solving linear programs. Second, not all the techniques operate on the same internal data structures, which means that data conversions are taking place. Finally, the approximation algorithm is developed using MINTO—that is, not directly on top of the LP solver. Although this greatly reduces development time, it adds some overhead. Table 3 shows that the CPU times for both algorithms are comparable. The approximation algorithm performs better on those instances where the HP algorithm requires a lot of cut generation, which are typically the harder ones. Observe that solving the LP relaxation is the most time consuming component in almost all of the problems. We can also see that removal of duplicate columns is done very efficiently in our algorithm. This is evident especially for the two largest instances with 123,409 and 148,633 variables, where preprocessing times are 5.14 and 7.62 seconds, respectively. Probing time is usually only a small percentage of the total time, which indicates that this potentially time consuming component is properly integrated in the algorithm.

We tested our algorithm on some other problems as well. Two of the problems, data01 and data02, come from a commercial airline company. The next two problems air04 and air05 are from MIPLIB (Bixby, Boyd, and Indovina, 1992). These problems are the hardest ones because we either could not find the optimal solution or could not prove optimality. In Table 4, we present the initial sizes of these problems and their sizes when our approximation algorithm terminated.

Tables 5 and 6 are similar to Tables 1 and 3. Air04 and air05 have been solved by HP. They report that proving optimality required 4.01 hours and 38.7 hours of CPU time on an IBM RS6000 Model 550, respectively. For air04, we observe that the Lagrangian dual heuristic is called forty-five times and this has been the most time consuming routine. For air05, Hoffman and Padberg indicate that feasible solutions 26,707 and 26,453 were found within 1,821 and 3,612 seconds, respectively. (Again, these numbers are divided by 2 to compensate for machine differences.) Our approximation algorithm found the solution of 26,458 in 344 seconds.

In Table 7 we summarize the performance of our algorithm on the hard problems. The second column gives the value of the LP relaxation of the remaining problem at the end of the algorithm. The third and fourth columns give the optimal and the heuristic objective function values, respectively. Even though we have found optimal solutions for data01 and data02, we have not been able to prove their optimality. However, we can guarantee that the heuristic solutions are within only 0.180 percent and 0.057 percent of the optimal solutions, respectively. The solution we found for air04 is very close to optimal with a 0.887 percent quality guarantee. The optimal solution value for air05 has a quality guarantee of 1.846 percent.

Table 4. Remaining problem size.

Problem	Rows	Columns	Remaining Rows	Remaining Columns
data01	144	74081	59	242
data02	174	369568	44	125
air04	823	8904	519	3613
air05	426	7195	327	4508

Table 5. Number of calls.

Problem	RDom Calls	Probe Calls	LP Calls	Heuristic Calls	CG Calls	Number of Cuts
data01	31	19	16	15	4	14
data02	28	21	20	19	8	24
air04	80	57	45	45	2	55
air05	28	20	16	16	3	47

Table 6. Computation times (seconds).

Problem	RDom Time	Probe Time	LP Time	Heuristic Time	CG Time	Total CPU
data01	1.45	20.88	33.94	27.76	0.14	265.73
data02	3.77	1.30	256.14	24.34	0.38	725.80
air04	16.02	295.14	304.23	1813.65	7.00	2625.33
air05	1.78	157.40	92.92	323.85	29.23	668.81

Table 7. Performance of the algorithm.

Problem	z_{LP}	z_{OPT}	z_H	% Grnt	% Actl
data01	311.74	312.30	312.30	0.180	0.000
data02	270.51	370.72	370.72	0.057	0.000
air04	55642.96	56137	56138	0.887	0.002
air05	25973.82	26374	26458	1.864	0.318

8. Final remarks

We have several ideas to improve and extend the algorithm that still need to be investigated. Here we discuss some of them briefly.

Many classes of valid inequalities for the set partitioning problem are known. Since we only use clique inequalities, the performance may be increased by incorporating other inequalities, such as odd-hole inequalities.

The Lagrangian dual heuristic requires an initial dual solution. In the approximation algorithm, the initial dual solution is taken to be the dual solution of the LP relaxation. Since we use a simplex algorithm to solve the active linear program, we obtain an extreme point solution. A better initial dual solution may be obtained if the active linear program is solved with an interior point algorithm and the solution is in the interior of the optimal face. The convergence of the Lagrangian dual heuristic itself may be improved by the use of exact penalty methods instead of iterative coordinate search methods.

Very large set partitioning problems need to be solved by column generation (Barnhart et al., 1994). It will be challenging to incorporate techniques such as preprocessing in a column generation algorithm.

Finally, the Lagrangian dual heuristic with cost perturbations only works for 0-1 matrices. Since it is very effective, it is important to investigate whether it can be extended to handle more general matrices.

Acknowledgments

This research was supported by U.S. Army Research Office DAAH04-94-G-0017 and NSF Grant No. DDM-9115768.

References

Balas, E., and M. Padberg. (1976). "Set Partitioning: A Survey." *SIAM Review* 18, 710–760.

Barnhart, C., E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, and P.H. Vance. (1994). "Branch-and-Price: Column Generation for Solving Integer Programs." Technical Report COC-94-03, Computational Optimization Center, Georgia Institute of Technology, Atlanta, Georgia.

Bixby, R.E., E.A. Boyd, and R. Indovina. (1992). "MIPLIB: A Test Set of Mixed-Integer Programming Problems." *SIAM News* 25, 16.

Chu, P.C., and J.E. Beasley. (1995). "A Genetic Algorithm for the Set Partitioning Problem." Technical report, The Management School, Imperial College, London SW7 2AZ, England, April.

Fisher, M., and P. Kedia. (1990). "Optimal Solution of Set Covering / Partitioning Problems Using Dual Heuristics." *Management Science* 36, 674-688.

Garfinkel, R.S., and G.L. Nemhauser. (1969). "The Set-Partitioning Problem: Set Covering with Equality Constraints." *Operations Research* 17, 848–856.

Hoffman, K., and M. Padberg. (1993). "Solving Airline Crew-Scheduling Problems by Branch-and-Cut." *Management Science* 39, 667–682.

Marsten, R.E., and F. Shepardson. (1981). "Exact Solution of Crew Problems Using the Set Partitioning Mode: Recent Successful Applications." *Networks* 11, 165–177.

Nemhauser, G.L., M.W.P. Savelsbergh, and G.S. Sigismondi. (1994). "MINTO: A Mixed Integer Optimizer." *Operations Research Letters* 15, 47–58.

Padberg, M. (1973). "On the Facial Structure of Set Packing Polyhedra." *Mathematical Programming* 5, 199–215.

Savelsbergh, M.W.P. (1994). "Preprocessing and Probing Techniques for Mixed Integer Programming Problems." *ORSA Journal on Computing* 6, 445–454.

Savelsbergh, M.W.P., and G.L. Nemhauser. (1994). "Functional Description of MINTO, a Mixed Integer Optimizer." Technical Report COC-91-03C, Computational Optimization Center, Georgia Institute of Technology, Atlanta, Georgia.

Wedelin, D. (1995). "An Algorithm for Large Scale 0-1 Integer Programming with Application to Airline Crew Scheduling." *Annals of Operations Research* 57, 283-301.