7-13-1998

Maximally Disjoint Solutions of the Set Covering Problem

David J. Rader
Rose-Hulman Institute of Technology, rader@rose-hulman.edu

Peter L. Hammer
Rutcor, Rutgers University

MSTR 98-03

Maximally Disjoint Solutions of the Set Covering Problem

Peter L. Hammer and David J. Rader, Jr.

MS TR 98-03

July 13, 1998

Department of Mathematics
Rose-Hulman Institute of Technology
Terre Haute, IN 47803

FAX (812) 877-3198 **Phone: (812) 877-8391**

Maximally Disjoint Solutions of the Set Covering Problem

Peter L. Hammer
RUTCOR, Rutgers University
640 Bartholomew Rd.
Piscataway, NJ 08854-8003
Hammer@RUTCOR.Rutgers.edu

David J. Rader, Jr.
Department of Mathematics
Rose-Hulman Institute of Technology
Terre Haute, IN 47803
David.Rader@Rose-Hulman.edu

Abstract. This paper is concerned with finding two solutions of a set covering problem that have a minimum number of variables in common. We show that this problem is NP-complete, even in the case where we are only interested in completely disjoint solutions. We describe three heuristic methods based on the standard greedy algorithm for set covering problems. Two of these algorithms find the solutions sequentially, while the third finds them simultaneously. A local search method for reducing the overlap of the two given solutions is then described. This method involves the solution of a reduced set covering problem. Finally, extensive computational tests are given demonstrating the nature of these algorithms. These tests are carried out both on randomly generated problems and on problems found in the literature.

In numerous practical situations it is desirable to find multiple and mutually indepedent solutions to the same problem. For example, in network communications, we may want to find several edge-disjoint paths linking two nonadjacent nodes. The existence of such paths reduces the chance that the two nodes cannot communicate, since noncommunication takes place only when all links must have been severed. Another example occurs in staff scheduling, where a manager wants to have several independent teams of workers so that each team has all the resourses needed to complete the job. In combinatorial optimization, typical examples where such situations occur involve finding a set of pairwise disjoint cliques in a graph maximizing the total number of vertices or finding multiple pairwise disjoint edge covers in a graph. This paper is concerned with the problem of finding two solutions to the set covering problem that share a minimum number of variables.

This problem has an important application in the design of tests for evaluating the psychiatric condition of a patient. Psychometric testing is performed by means of large questionaires involving many (sometimes hundreds of) "items" or questions. The answers to these questions are used to determine the condition of the patient. It was noticed, however, that the patient's condition can be identified even if only a subset of the questions is used. There are many subsets of questions providing all the necessary information for classifying a patient. If 0-1 variables are associated to each of the original questions, to indicate whether a question is eliminated from the list or maintained in it, then the acceptable set of questions capable of providing the necessary classifications can be obtained by solving a system of set-covering type inequalities.

Frequently it is important for psychiatrists to administer the same test repeatedly to the same patients. It is advisable in such a situations to minimize the number of overlapping questions appearing in two or more questionaires. This can be achieved by finding several distinct solutions to a system of set-covering type inequalities; the number of questions appearing in any two questionaires is the number of those components of the solution vectors that take the value 1 in both solutions.

Many exact and heuristic algorithms have been proposed for the set covering problem, particularly due to their importance in large aircraft scheduling problems, each capable of solving large instances of the problem. While there is a vast literature on the set covering problem ([1, 4, 6, 8, 12, 16]), it appears that the question of finding two maximally disjoint solutions to a set covering problem has not been previously addressed.

In this paper we are concerned with the problem of finding two solutions to a set covering problem in such a way so that their overlap is minimized. In Section 1, we give a constrained quadratic pseudo-Boolean optimization model for this problem and show that the related decision problem is NP-complete. In Section 2 we develop three heuristic algorithms and a local search procedure for obtaining good approximations to our model. Finally, in Section 3, we describe some computational results obtained for each of these algorithms. These tests are performed both on randomly generated problems and problems obtained from the literature [5].

1 Problem Formulation and Complexity

In this section we will give a formal definition of our problem, along with a formulation of it as a quadratic pseudo-Boolean optimization problem. Finally, we will show that the decision problem is NP-complete, even if we only wish to determine whether there exist two disjoint solutions.

Given the ground set $N = \{1, 2, \ldots, n\}$, the problem we shall consider is the following:

`Overlapping Covers`:

Instance: A collection N_1, $N_2, \ldots,$ N_m of subsets of N and an integer k.

Question: Do there exist collections $C_1, C_2 \subseteq N$, such that (a) $N_i \cap C_j \neq \emptyset$ for all $i = 1, \ldots, m$ and $j = 1, 2$, and (b) $|C_1 \cap C_2| \leq k$?

This combinatorial decision problem has an equivalent integer programming formulation. Let A be an $(m \times n)$-matrix where

$$a_{ij} = \begin{cases} 1 & \text{if } j \in S_i \\ 0 & \text{otherwise.} \end{cases}$$

The question then becomes: Do there exist two solutions $\mathbf{x}, \mathbf{y} \in \{0,1\}^n$ such that

$$A\mathbf{x} \geq \mathbf{1}, ::::: A\mathbf{y} \geq \mathbf{1}, ::::: \sum_{i=1}^{n} x_i y_i \leq k?$$

The associated optimization problem can be formulated as an integer program:

$$\min \quad \sum_{i=1}^{n} x_i y_i \tag{1}$$

s.t.

$$A\mathbf{x} \geq \mathbf{1} \tag{2}$$

$$A\mathbf{y} \geq \mathbf{1} \tag{3}$$

$$\mathbf{x}, \mathbf{y} \in \{0,1\}^n, \tag{4}$$

which can be viewed as a quadratic set-covering problem. From now on, we will call solutions $\mathbf{x}, \mathbf{y}$ *covers* if they satisfy (2) and (3), respectively.

Since problem `Overlapping Covers` is very similar to the set-covering problem, it is to be expected that this problem is also NP-complete. Indeed, this is the case. Before we prove this result, for the special case where $k = 0$, we first need a Lemma characterizing whether two non-overlapping solutions can exists. For both proofs, we use the equivalent integer programming formulation of the problem.

Lemma 1 *Given a 0-1 matrix A, there exists two covers $\mathbf{x}, \mathbf{y}$ such that*

$$A\mathbf{x} \geq \mathbf{1}, \quad A\mathbf{y} \geq \mathbf{1}, \quad \mathbf{x}^T\mathbf{y} = 0 \tag{5}$$

if and only if there exists a cover $\mathbf{z}$ such that, for every $1 \leq i \leq m$,

$$1 \leq \sum_{j=1}^{n} a_{ij} z_j \leq \sum_{j=1}^{n} a_{ij} - 1. \tag{6}$$

Proof. Let $\mathbf{x}$ and $\mathbf{y}$ be covers that satisfy (5) and suppose there exists an $i \leq m$ such that

$$\sum_{j=1}^{n} a_{ij} x_j = \sum_{j=1}^{n} a_{ij}.$$

This implies that $x_j = 1$ for all j such that $a_{ij} = 1$. Since $\mathbf{y}$ is a cover, there exists a $j \leq n$ such that $a_{ij} = 1$ and $y_j = 1$. But this contradicts $\mathbf{x}$ and $\mathbf{y}$ satisfying (5). Hence, $\mathbf{x}$ satisfies (6).

Now suppose there exists a cover $\mathbf{y}$ satisfying (6). For each i, let $L(i) = \{j : a_{ij} = 1, y_j = 0\}$. Consider the following constraints:

$$\sum_{j \in L(i)} x_j \geq 1 \qquad i = 1, \ldots, m. \tag{7}$$

Let $\mathbf{x}$ be a solution to (7) with $x_j = 0$ if $j \notin \bigcup_{i=1}^{m} L(i)$; then $\mathbf{x}$ and $\mathbf{y}$ satisfy (5). □

Theorem 1 `Overlapping Covers` *is NP-complete, even if* $k = 0$.

Proof. Consider an instance of the NP-complete problem `Not-All-Equal 3SAT`:

`Not-All-Equal 3SAT`:

Instance: A set U of variables and a collection C of clauses over U such that each $c \in C$ has $|c| = 3$.

Question: Is there a truth assignment of U such that each $c \in C$ has at least one true literal and at least one false literal?

We will construct an instance of `Overlapping Covers` and show, using Lemma 1, that the answer to `Not-All-Equal 3SAT` is YES if and only if the answer to `Overlapping Covers` with $k = 0$ is YES.

For each $u \in U$, construct two variables $x_u, \overline{x}_u$, representing the two literals used in our 3SAT problem. For each $c \in C$, we will represent the literals as z_i. Hence, $z_i = x_u$ or $z_i = \overline{x}_u$ for some variable $u \in U$. Consider the following set covering constraints:

$$\sum_{i \in c} z_i \geq 1, \qquad c \in C \tag{8}$$

$$x_u + \overline{x}_u \geq 1, \qquad u \in U. \tag{9}$$

Suppose the answer to `Not-All-Equal 3SAT` is YES, and let $\mathbf{z}$ be such a truth assignment. It is easy to see that the following conditions must hold:

$$1 \leq \sum_{i \in c} z_i \leq 2, \qquad c \in C$$

$$1 \leq x_u + \overline{x}_u \leq 1, \qquad u \in U.$$

Hence, by Lemma 1, `Overlapping Covers` has a YES answer. If `Not-All-Equal 3SAT` has a NO answer, then, for every truth assignment $\mathbf{z}$, there must exist a $c \in C$ such that

$$\sum_{i \in C} z_i = 3.$$

By Lemma 1, this implies that `Overlapping Covers` has a NO answer. □

2 Heuristics and Local Search Algorithms

In this section we describe three heuristic methods for solving the `Overlapping Covers` problem (1)-(4). Each algorithm is a variant on the standard greedy algorithm for the set covering problem, but with differing methods for obtaining the two needed solutions. Two of the algorithms obtain the two solutions sequentially, using the first solution to obtain the second solution, while the third algorithm obtains the two covers in parallel.

We will also describe a local search algorithm for reducing the overlap of any pair of solutions of the set covering problem. This is achieved by obtaining a cover to a related set covering problem which uses the two given solutions. In this way we make all of our improvements in one step, instead of the traditional local search algorithms that make many small improvements.

Throughout this section, the following definitions are used. Let N be the set of variable indices and M be the set of row indices. Let S_x and S_y be the sets of variables selected for the covers $\mathbf{x}$ and $\mathbf{y}$. A row i is *covered* if there exists a variable j such that j is selected for that cover and $a_{ij} = 1$. We denote the cost of variable j by $cost(j)$ and the number of variables in row i by $size(i)$. At each iteration, let $c(j)$ denote the number of uncovered rows i such that $a_{ij} = 1$, i.e the number of new rows that would be initially covered by variable j if j is selected. In addition, for each row i we let $r(i)$ be the number of variables in row i that have been selected in the current cover.

Since each heuristic described below is a variant of the standard greedy method for set covering problems, we will review that algorithm. At each iteration, the greedy method obtains the list of unselected variables j that maximize the *greedy function* $G(j) = c(j)/cost(j)$. This list is referred to as the candidate list CL. A variable j is then selected randomly from CL. Once a variable has been selected, the value of $c(k)$ is updated for each unselected variable k. The algorithm continues until all rows have been covered. For an analysis of this algorithm, see [8].

2.1 `Twosol` Algorithm

In this section, we describe a heuristic `Twosol` for solving (1)-(4). This algorithm uses the greedy method described above to generate two solutions with minimal overlap.

`Twosol` can be broken down into two main components: first, we find an initial solution that tries to "leave room" in each row for a second solution; second, we find a second solution which uses as many variables as possible that are not used in the first solution. In order to obtain our initial cover S_x, we initially assign each variable j a cost $cost(j) = 1$. At each iteration, we select a variable j from the candidate list CL. We then update the data structures $r(i)$ and $c(j)$ for each row i and variable j. If there exists a row i such that $r(i) = size(i) - 1$, i.e. there is only one unselected variable k remaining in row i, we change the cost of variable k to $cost(k) = M$, for some large number M. This change of cost is done, in the spirit of Lemma 1, in an attempt to generate solutions with no overlap. Because of this change in cost, a variable j with $cost(j) = M$ will only be selected if there are no more unselected variables with unit cost. This procedure is then repeated until all rows have been covered.

Once an initial solution S_x is obtained, we proceed to find a second solution that has minimal overlap with the first. For each variable $j \in S_x$, we set the cost of j to $cost(j) = M$; for all other variables, we assign a unit cost. In this way, we will only have a variable j selected in both solutions if a second solution cannot be obtained without it. This second solution S_y is then obtained using

the standard greedy heuristic for set covering problems.

2.2 GRASP

In this section, we present a randomized version of the `Twosol` heuristic described in Section 2.1. This heuristic is based on the GRASP heuristic for set covering problems that was described in [2, 3, 11, 12]. GRASP algorithms, or "Greedy Randomized Adaptive Search Procedures", have been used to develop effective heuristics for various combinatorial problems (see [13] for references). In general, this procedure chooses a candidate list RCL of possible next iterates using a greedy function, and then randomly chooses the next iterate from this list. Once a solution is obtained, a local search algorithm is employed to improve upon the solution. This algorithm is repeated many times, with the best solution kept throughout. The randomization in the algorithm occurs in the generation of a candidate list. In traditional greedy methods, the candidate list consists of those variables j optimizing a given greedy function. In a GRASP algorithm, the candidate list is expanded to contain those variables that are within a certain user-defined percentage of the optimal greedy function value. This allows a larger variety of variables to be chosen, including those that may not give the greatest gain in the current iteration. The GRASP algorithm is run many times to take advantage of this randomization, with the best solution updated after each run. Since greedy methods typically do not generate optimal solutions, such approaches have typically produced better heuristic values (see for example [12, 13, 16]).

The GRASP algorithm for the overlapping set covering problem incorporates a randomized version of the `Twosol` algorithm described in Section 2.1. Thus we first use a randomized greedy algorithm to obtain each solution separately. Initially, we assign each variable i a unit cost ($cost(i) = 1$). At each step in finding a cover, we initally find $\gamma = \max\{G(j) : j \text{ not selected}\}$. We then create the candidate list

$$RCL = \{j \text{ not selected } : G(j) \geq (1-\alpha)\gamma\},$$

where $0 \leq \alpha \leq 1$ is a parameter describing how large our candidate list is. A variable j is then selected randomly from RCL and placed in S_x. We make certain that an unselected variable i with $cost(i) = 1$ is chosen, if one exists. This ensures that the general idea of our deterministic heuristic is kept. Each $c(j)$ and $r(i)$ is updated for all unselected variables j and rows i. If there exists a row i such that $r(i) = size(i) - 1$, i.e. there is only one unselected variable k remaining in row i, we change the cost of variable k to $cost(k) = M$, for some large number M. This process is repeated until all rows have been covered.

Once a solution is found, we obtain a second solution much like we did in Section 2.1. Initially, we set $cost(j) = M$ for each $j \in S_x$ and set $cost(j) = 1$ for all other variables. In each iteration, we obtain the candidate list RCL as above by first identifying γ and including those indices with greedy function value at least $(1-\alpha)\gamma$. Again, a variable $j \in RCL$ is selected at random and placed in S_y. We then update each $c(k)$ and iterate again until all rows have been covered.

We generate multiple covers in order to take advantage of the randomization process. To do this, we generate, for every "first" solution S_x, a series of "second" solutions S_y, and do our "local search". After each pair of covers is obtained, the best solutions are updated. Typically, one would generate multiple "first" solutions, and for each such solution, generate multiple "second" solutions.

2.3 Generating Solutions Simultaneously

In this section we describe the heuristic `Simul` for obtaining solutions with small overlap. This heuristic differs from the first two in that the two solutions will be constructed simultaneously instead of sequentially. Because of this difference, we need to keep track of two instances of the same problem, one for each solution. These instances will be referred to as instance I_x and instance I_y. For each instance we will use the definitions $cost_x(j)$ ($cost_y(j)$), $c_x(j)$ ($c_y(j)$), and $r_x(i)$ ($r_y(i)$) to refer to the information needed for instance I_x (I_y).

Th `Simul` heuristic is very similar to the greedy heuristic described in Section 2.1. At each iteration, we select a variable from instances I_x and I_y that covers the most uncovered rows at minimum cost. We then update each instance and proceed. In each iteration, we wish to select variables j_x and j_y that (a) have not been selected in the other instance, and (b) are not the last unselected variable in a given row. To do this, we set up two candidate lists for each instance: the "good" list C_g^x (C_g^y) and the "bad" list C_b^x (C_b^y). C_b^x (C_b^y) contains those unselected variables in instance I_x that have been selected in instance I_y (I_x) as well as those variables that are the last remaining unselected variable in some row of instance I_x (I_y). C_g^x (C_g^y) contains all remaining unselected variables of instance I_x (I_y). In order to select the next two variables, we first select, using a greedy approach, the best variable in C_g^x and C_g^y. If either of these lists is empty, we select a variable from the "bad" list. If we have selected the same variable, we choose other variables at random from the same lists that have the same greedy value, if possible; otherwise we make no change in variables. Once the two variables j_x and j_y have been selected, we update the data structures as follows: (a) in instance I_y (I_x), j_x (j_y) is moved from C_g^y (C_g^x) to C_b^y (C_b^x); (b) each $c_x(k)$ and $c_y(k)$ is updated for all unselected variables in I_x and I_y; (c) each $r_x(i)$ and $r_y(i)$ is updated in I_x and I_y. If there exists a row i in instance I_x (I_y) with $r_x(i) = size(i) - 1$ ($r_y(i) = size(i) - 1$), then the remaining free variable in row i is moved from C_g^x (C_g^y) to C_b^x (C_b^y). This process is repeated until at least one of the instances has no remaining uncovered rows. If the other instance has rows still uncovered, we complete this cover by using the standard greedy algorithm, where each variable $j \in C_g$ has $cost(j) = 1$ and each $j \in C_b$ has $cost(j) = M$.

Example. Consider the following set covering constraints:

$$\begin{aligned} w_1 + w_2 + w_3 &\geq 1 \\ w_1 + w_2 + w_5 &\geq 1 \\ w_1 + w_3 + w_5 &\geq 1 \\ w_1 + w_4 &\geq 1 \\ w_2 + w_3 + w_5 &\geq 1 \\ w_2 + w_4 &\geq 1 \\ w_3 + w_4 + w_5 &\geq 1. \end{aligned}$$

At the start of `Simul` we have the following data:

$$\begin{aligned} C_g^x,\ C_g^y &= \{1,2,3,4,5\} \\ C_b^x,\ C_b^y &= \emptyset \\ c_x(1) = c_y(1) &= 4 \end{aligned}$$

$$\begin{aligned} c_x(2) = c_y(2) &= 4 \\ c_x(3) = c_y(3) &= 4 \\ c_x(4) = c_y(4) &= 3 \\ c_x(5) = c_y(5) &= 4. \end{aligned}$$

Iteration 1: We choose the following variables at random from C_g^x and C_g^y: $j_x = 1$, $j_y = 2$. This leads to the following updated instances and data (we list only uncovered rows in each instance):

$$\begin{array}{rc} I_x: & w_2 + w_3 + w_5 \geq 1 \\ & w_2 + w_4 \geq 1 \\ & w_3 + w_4 + w_5 \geq 1 \end{array} \qquad \begin{array}{rc} I_y: & w_1 + w_3 + w_5 \geq 1 \\ & w_1 + w_4 \geq 1 \\ & w_3 + w_4 + w_5 \geq 1 \end{array}$$

$$C_g^x = \{3,5\} \qquad C_g^y = \{3,5\}$$
$$C_b^x = \{2,4\} \qquad C_b^y = \{1,4\}.$$

Note that $j = 4$ is in both C_b^x and C_b^y since it is the last remaining unselected variable in a row of instances I_x and I_y.

Iteration 2: At the start of this iteration we have $c_x(3) = c_x(5) = 2$ and $c_y(3) = c_y(5) = 2$. Choosing at random the variables $j_x = 3$, $j_y = 5$, we get the following instances:

$$I_x: \quad w_2 + w_4 \geq 1 \qquad\qquad I_y: \quad w_1 + w_4 \geq 1$$

$$C_g^x = \emptyset \qquad C_g^y = \emptyset$$
$$C_b^x = \{2,4,5\} \qquad C_b^y = \{1,3,4\}.$$

Iteration 3: We now have $c_x(2) = c_x(4) = 1$, $c_x(5) = 0$, $c_y(1) = c_y(4) = 1$ and $c_y(5) = 0$. We now randomly choose the variables $j_x = 2$, $j_y = 1$ to obtain covers $S_x = \{1,2,3\}$ and $S_y = \{1,2,5\}$.

Let us remark here that we can implement a GRASP version of `Simul` similar to the GRASP version of `Twosol`. In this version, we expand the candidate list in each instance I_x and I_y by using some predefined parameter α. Here we only include variables within the same list; for example, if $C_g^x \neq \emptyset$, then $RCL \subseteq C_g^x$. This way, we only choose those variables that would give an overlap if it is necessary to cover every row.

2.4 Local Search Method

In this section we describe the local search method used after two solutions have been found using any of the heuristics given in Sections 2.1, 2.2, and 2.3. This procedure tries to find all improvements at once by solving an additional set covering problem. Our local search procedure changes the cover S_x in the following way. Let S be the solution of the "local search" set covering problem and let $\hat{S}$ be those variables in S_x that are not in S_y. Then our revised cover $\hat{S}_x$ becomes $S \cup \hat{S}$.

Suppose we are given two covers S_x and S_y. Let $OV = S_x \cap S_y$ be those variables in both covers and let $F = N - (S_x \cup S_y)$ be those variables in neither cover. The variables in our "local search" set covering problem are $\hat{N} = OV \cup F$. The set of rows for this new problem consists of all those rows $i \in M$ that contain a variable $j \in OV$ and no variable in $S_x - OV$. In other words, we are

trying to cover with variables in F the rows that are only covered in solution S_x by variables in OV. Each variable $j \in OV$ is assigned $cost(j) = M$, while each $j \in F$ is assigned $cost(j) = 1$. Once this new problem is defined, the standard greedy algorithm is used to obtain a new solution S, and then $S_x = S \cup \hat{S}$.

Example. Consider the example given in Section 2.3. The two covers given there were $S_x = \{1, 2, 3\}$ and $S_y = \{1, 2, 5\}$. Our "local search" set covering problem then consists of the variables $OV \cup F = \{1, 2\} \cup \{4\}$ and rows

$$\begin{aligned} w_1 + w_2 &\geq 1 \\ w_1 + w_4 &\geq 1 \\ w_2 + w_4 &\geq 1. \end{aligned}$$

Note that any row which contains variable 3 and either of variables 1 or 2 (i.e, rows 1,3, and 5) are not included in this new problem since $j = 3$ covers these rows already and will be in the revised cover $\hat{S}_x$. The standard greedy algorithm will give one of two solutions: $S = \{1, 4\}$ or $S = \{2, 4\}$. Choosing the first one, without loss of generality, yields the revised cover $\hat{S}_x = \{1, 3, 4\}$ which only overlaps S_y in one variable. It is not too difficult to see that this is the best we can do with these rows.

3 Computational Results

In this section we present the results of computational experiments used to examine the bahavior of the algorithms given in this paper. We implemented six versions of our algorithms: `Twosol`, GRASP algorithms with $\alpha = 0.25$ and $\alpha = 0.5$, `Simul`, and GRASP versions of `Simul` with $\alpha = 0.25$ and $\alpha = 0.5$. Our algorithms were written in C++ and were run on a Silicon Graphics MIPs 10000 with 192M memory and 175 Mhz clock speed. The GRASP versions of `Twosol` generated 25 "first" solutions and, for each such solution, 20 "second" solutions, for a total of 500 solutions, while the GRASP versions of `Simul` generated 500 pairs of solutions. The algorithms were tested on several types of problems, some from the literature and others randomly generated. The problems from the literature come from [1, 4, 12, 16] and are available on the internet ([5]).

The first set of problems from the literature are problem sets 4-6 from [1] and sets A-E from [4]. Most of these problems were originally generated to test weighted set covering problems. Here, we disregard the costs of the columns. Table 1 gives the details on these problems. Note that, for these problems, we have more columns than rows.

For the problems in each of the above problem sets our algorithms found two disjoint solutions. This does not not seems surprising; in randomly generated problems, we should expect that, if there are many more variables than constraints, there are more choices for covers, and hence, at least two disjoint solutions.

The next set of problems we consider comes from [12]. These problems arise in computing the 1-width of incidence matrices of Steiner triple systems. The β-width of a $(0, 1)$-matrix A is the minimum number of columns that can be selected from A such that all row sums of the resulting submatrix of A are at least β. The incidence matrices A that arise from Steiner triple systems have precicely 3 ones per row. These problems are considered computationally difficult for the set covering problem. Details for these problems are given in Table 2. For several of the test problems

Problem Set	Rows	Columns	Density (%)	Number of Problems
4	200	1000	2	10
5	200	2000	2	10
6	200	1000	5	5
A	300	3000	2	5
B	300	3000	5	5
C	400	4000	2	5
D	400	4000	5	5
E	50	500	20	5

Table 1: Details of Problem Sets 4-6 and A-E

Problem	Rows	Columns	Best Known Cover	Bound on Overlap
A_9	12	9	5*	1
A_{15}	35	15	9*	3
A_{27}	116	27	18*	9
A_{45}	330	45	30*	15
A_{81}	1080	81	61	41
A_{243}	9801	243	204	165

Table 2: Details of Steiner Triple Problems (* optimal)

of this type (A_9, A_{15}, A_{27}, A_{45}), the optimal cover size is known. Since this size is more than half of the number of variables, a lower bound on the minimal overlapping of two solutions can be derived. For all other problems, we only have the best known size of the cover; hence, any "lower bound" derived from them may not actually be a bound. We list each of these bounds in Table 2.

The test results for these problems are given in Table 3. Note that, as can be expected, the GRASP versions of `Twosol` and `Simul` performed better than the deterministic versions. Also, the GRASP versions of `Twosol` found optimal solutions for A_9, A_{15}, and A_{27}.

The third set of problems, the CYC set, is derived from an old question of P. Erdös ([9]): what is the minimum number of edges of a hypercube that can be chosen so that every cycle of 4 edges contains at least one chosen edge? This question can be viewed as a set covering problem, with the hypercube edges corresponding to the variables and the 4-cycles to a constraint. We consider problems generated from hypercubes of dimensions $d = 6, 7, 8, 9, 10$. Details of these problems are given in Table 4.

Test results for this problem set are given in Table 5. For these problems, we see that the deterministic algorithms perform much better than their randomized versions, and that, for the larger problems, the more you increase the amount of randomization, the worse the algorithm performs. This may be due to the structure of the problem, but we cannot be certain. Otherwise, note that the deterministic algorithms give optimal or near optimal solutions in every problem.

		GRASP			Simul - GRASP	
Problem	Twosol	$\alpha = 0.25$	$\alpha = 0.5$	Simul	$\alpha = 0.25$	$\alpha = 0.5$
A_9	1	1	1	1	1	1
A_{15}	4	3	3	4	3	3
A_{27}	11	9	9	11	10	10
A_{45}	18	18	18	21	18	19
A_{81}	48	45	45	49	45	44
A_{281}	174	167	168	171	168	169

Table 3: Results on Steiner Triple Problems

Problem	Rows	Columns	Density
CYC.6	240	192	2.1
CYC.7	672	448	0.9
CYC.8	1792	1024	0.4
CYC.9	4608	2304	0.2
CYC.10	11520	5120	0.08

Table 4: Details on CYC Problems

		GRASP			Simul - GRASP	
Problem	Twosol	$\alpha = 0.25$	$\alpha = 0.5$	Simul	$\alpha = 0.25$	$\alpha = 0.5$
CYC.6	0	0	0	0	0	0
CYC.7	0	0	0	0	0	0
CYC.8	0	0	2	0	1	7
CYC.9	0	3	11	1	13	35
CYC.10	1	25	76	14	69	141

Table 5: Results on CYC Problems

				GRASP			Simul - GRASP	
n	m	Row Density	Twosol	$\alpha = 0.25$	$\alpha = 0.5$	Simul	$\alpha = 0.25$	$\alpha = 0.5$
50	500	3.26	20.92	20.8	21.12	21.12	**20.52**	21.16
	1000	3.28	29.4	**28.72**	29.28	29.4	28.88	29.44
	1250	3.30	31.68	31.48	31.88	31.84	**31.36**	31.8
TOTALS			82	81	82.28	82.48	**80.76**	82.4
75	750	4.14	22.32	21.92	22.8	22.24	**21.8**	22.76
	1500	4.18	33.76	**33.6**	34.84	34.28	34.16	34.56
	1875	4.17	**37.64**	37.68	38.68	38.28	37.84	38.76
TOTALS			93.72	**93.2**	96.32	94.8	93.8	96.08
100	1000	5.26	16.8	16.72	17.76	16.8	**16.52**	17.73
	2000	5.26	**31.48**	31.88	33.52	32.16	32.08	33.28
	2500	5.25	**36.36**	36.76	38.38	37.44	37.16	38.2
TOTALS			**84.64**	85.36	89.66	86.4	85.76	89.21

Table 6: Results on Randomly Generated Problems

The last set of problems we consider are randomly generated. We generated random data sets with $n = 50$, 75, and 100, and, for each n, we took $m = 10n, 20n, 25n$. These were done because of our experience in the Steiner Triples and CYC problems sets, where there were more constraints than variables. The matrices are generated with density approximately 5% so that each row has at least 2 entries. For each n and m, we generated 25 test problems. Results of the test runs on these problems are given in Table 6. We also include the average number of variables per row in the problems. In Table 6, we indicate the best average solutions for each problem set. Versions of the algorithms `Twosol` and `Simul` generate the best solutions for roughly half of the problems. Also, there does not appear to be much difference between the best version of `Twosol` and the best version of `Simul`. In addition, note that as we increase the randomization of the algorithms, the solutions tend to get worse. This also occurred in the CYC problem set.

We will next describe the effect of the "local search" procedure we implemented. We use the Steiner Triple problems described in Table 2, the CYC problems described in Table 4, and the randomly generated problems where $m > n$. Test results for all problems are given in Table 7. We give the percentage of improvement over the initial solution of each algorithm. We note that the `Twosol` algorithm and its GRASP version with $\alpha = 0.25$ generate solutions that are almost local, and hence our "local search" yields little improvement. Also, we see great improvement in the `Simul` algorithms. Our procedure in general gives modest improvements when added to each of the described algorithms.

Problem Set			GRASP			Simul-GRASP	
n	m	Twosol	$\alpha = 0.25$	$\alpha = 0.5$	Simul	$\alpha = 0.25$	$\alpha = 0.5$
Steiner Triples		0.0	1.62	3.56	4.81	4.67	5.75
CYC		0.0	0.0	1.30	61.11	60.80	51.04
50	500	0.19	2.07	4.69	14.63	13.49	10.34
	1000	0.14	1.78	5.06	7.78	7.32	5.28
	1250	0.25	0.88	1.60	9.03	3.57	3.28
75	750	0.36	2.49	5.00	18.83	22.03	16.32
	1500	0.47	0.94	3.86	14.73	10.48	8.67
	1875	0.11	0.53	4.07	12.12	10.50	7.01
100	1000	0.71	3.69	9.39	27.21	29.88	30.88
	2000	0.38	2.57	5.10	18.95	19.07	16.21
	2500	0.66	1.71	4.34	14.29	14.85	11.90

Table 7: "Local Search" test results (in percentages)

4 Conclusions

In this paper we described two basic algorithms and randomized versions of them for the problem of determining two maximally disjoint solutions to a set covering problem. These algorithms were tested on problems both randomly generated and from the literature, yielding good approximate solutions in each case.

As stated in the introduction, it is often desirable to have more than two solutions to a set covering problem that are pairwise maximally disjoint. The `Twosol` algorithm and its GRASP version can be extended to provide solutions for this case. The major change that would be needed is adjusting the cost of a variable when it has been chosen by other solutions. We may, after finding k solutions, make the cost of variable i be $k_i * M$, where k_i is the number of solutions that have $x_i = 1$. Unfortunately, the `Simul` algorithm does not have such an easy transition to handle this problem.

References

[1] Balas, E. and Ho. A., "Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study", *Math. Programming Study*, 12 (1980), pp. 37-60.

[2] Bard, J.F. and Feo, T.A., "Minimizing the acquisition cost of flexible manufacturing equipment", Technical Report, Operations Research Group, Department of Mechanical Engineering, The University of Texas at Austin, Austin, TX 78712, 1987.

[3] Bard, J.F. and Feo, T.A., "Operations sequencing in discrete parts manufacturing", *Management Science*, 35(1989), pp. 249-255.

[4] Beasley, J.E., "An algorithm for set covering problem", *European Journal of Operational Research*, 31 (1987), pp. 85-93.

[5] Beasley, J.E., "OR-library: Distributing test problems by electronic mail", *J. Oper. Res. Soc.*, 41(1990), pp. 1069-1072. See also WWW site: http://mgcmga.ma.ic.ac.uk/info.html.

[6] Beasley, J.E. and Chu, P.C., "A genetic algorithm for the set covering problem", To appear in Euro. J. Operational Research, 1995.

[7] Brass, P., Harborth, H., and Nienborg, H., "On the maximum number of edges in a C_4-free subgraph", *J. Graph Theory*, 19 (1995), pp. 17-23.

[8] Chvàtal, V., "A greedy heuristic for the set covering problem", *Mathematics of Operations Research*, 4(1979), pp. 233-235.

[9] Erdös, P., "On a combinatorial problem I", *Nordisk Mat. Tidskrift*, 11 (1963), pp. 5-10.

[10] Erdös, P., "On some of my favorite problems in graph theory and block designs", *Le Matematische*, 45 (1990), pp. 61-74.

[11] Feo, T.A. and Bard, J.F., "A network approach to flight scheduling and maintenance base planning", Technical Report, Operations Research Group, Department of Mechanical Engineering, The University of Texas at Austin, Austin, TX 78712, 1987.

[12] Feo, T.A. and Resende, M.G.C., "A probabilistic heuristic for a computationally difficult set covering problem", *Operations Research Letters*, 8 (1989), pp. 67-71.

[13] Feo, T.A. and Resende, M.G.C., "Greedy randomized adaptive search procedures", *Journal of Global Optimization*, 6(1995), pp. 109-133.

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

[15] Goldberg, M.K. and Russell, H.C., "Toward computing $m(4)$", *Ars Combinatoria*, 3 (1995), p. 139-148.

[16] Grossman, T. and Wool, A., "Computational Experience with Approximation Algorithms for the Set Covering Problem", manuscript, April 1996.

[17] Harborth, H. and Nienborg, H., "Maximum number of edges in a six-cube without four-cycles", To appear in bull. Inst. Combin. Appl., 1995.