Enhancing Simulated Annealing for 0-1 Problems with Linear Programming Preprocessing

David Abramson
School of Computing and Information Technology
Griffith University
Kessels Rd, Nathan, Qld 4111
davida@cit.gu.edu.au

Henry Dang
Department of Computer Systems Engineering
Royal Melbourne Institute of Technology
P.O. Box 2476V,
Melbourne, 3001.
henry@mel.dit.csiro.au

Mohan Krishnamoorthy
Division of Mathametics and Statistics
C.S.I.R.O.
Private Bag #10,
Clayton, Vic 3169.
mohan@mel.dms.csiro.au

Abstract

Simulated annealing (SA) is a heuristic technique which has been successfully applied to a wide range of optimisation problems. Despite its wide applicability, it does have the drawback that the final solution could be suboptimal. Moreover, in problems with enormous solution spaces, the search for good solutions can be long. In contrast, techniques such as branch and bound provide optimal solutions by systematically pruning the search space with the help of information about the problem. In this paper we describe a scheme for combining a relaxed linear program (LP) of a 0-1 optimisation problem with an SA search. This improves the performance of the SA by pruning the search space as it proceeds. Further pruning is achieved by investigating the structural properties of the problem. Also, the lower bound from the relaxed LP provides a measure of the quality of the SA solution. We are not aware of this approach being used with simulated annealing. We illustrate the technique by applying it to the set partitioning problem (SPP) and give some performance results.

1. Introduction

Simulated annealing (SA) is a *Monte Carlo* method, based on the analogy between the aggregation of many particles in a physical system as it is cooled and the optimisation of large combinatorial problem (see, for example, van Laarhoven & Aarts [36]). In the physical system, annealing is the process of heating a solid until it melts, followed by a cooling operation until the substance crystallises. Fast cooling leads to a process called *quenching* in which the crystal which is formed has a metastable amorphous structure (or imperfect lattice), with a very large internal energy. Slow cooling allows the system to reach its equilibrium (lowest energy level) at each decremental temperature. Hence it is possible to reach the zero energy level or ground state. From the optimisation perspective, the search space can be viewed as a physical system in which elements or variables correspond to particles. The atomic bonding force is equivalent to the set of constraints imposed on the problem. The energy of the system is modelled as the penalty cost, so that when the system reaches the ground state, the process arrives at the global optimum configuration.

Starting off at maximum temperature, the metal liquid is cooled. At each temperature, thermodynamic equilibrium is established. The probability of a given state, **E**, having an energy **i** is given by the Boltzmann distribution shown below:

$$\Pr_i(T) = \frac{e^{\frac{-E_i}{kT}}}{Z(T)}$$

where k = Boltzmann constant
T = Temperature
$e^{\frac{-E_i}{kT}}$ = Boltzmann factor

Z(T) **=** Normalisation function = $\sum e^{\frac{-E_j}{kT}} \forall j$

SA is a probabilistic process which is similar to the well known local search descent algorithms [1]. In such algorithms, the local *neighbourhood* of a current solution is recursively searched for improvements (known as *down hill* moves) until none are possible. In contrast to descent algorithms, however, SA allows *up hill* movements that lead to deteriorations in the solution quality in order to escape from local minima. In SA, all down hill movements are accepted because they decrease the system energy. At higher temperatures up hill movements are accepted readily. As the temperature decreases, the chance of going up hill is decreased; up hill movement becomes impossible when the mass is frozen. Given a neighbourhood structure, SA attempts to transform the current solution into one in its neighbourhood. This mechanism is mathematically described as a *Markovian transformation* where the current structure depends only on the previous one (see van Laarhoven & Aarts [36]). To simulate the evolution of the SA algorithm, a *starting temperature* is computed so that almost all transitions from all neighbourhood configurations are accepted. The algorithm iterates by perturbing the current configuration and measuring the change in cost. If the change in cost is negative, the new configuration is automatically accepted. Otherwise the probability of accepting the increase is computed by evaluating the Boltzmann's factor. If the probability is greater than a random number in the interval [0,1], the new configuration is accepted, otherwise, it is rejected. At each temperature, the process is repeated an appropriate number of times, called a *Markov chain length*. The temperature is then decreased according to one of a number of *cooling schedules* (see van Laarhoven & Aarts [1987]). The process is repeated until the system is frozen or no change in cost between a number of consecutive Markov chains is detected.

SA has the advantage of searching for a solution without prior knowledge of the problem. It has been used for solving combinatorial, integer, linear and non-linear problems with very little change to the core algorithm. There are, however, two major disadvantages of SA. First, because all practical cooling schedules are not infinitely long, it gives no guarantee that the solution returned is optimal. Worse still, in many cases the optimal cost is unknown, and thus it is not possible to measure how far the solution is from optimal. Secondly, in problems that have an enormous search space, SA takes no account of problem structure, and thus may consider any point in the search space to be a likely candidate for the solution. The consequence of this is that the search can consume huge amounts of computer time. However, if a reliable lower bound can be determined through some other method, then it is possible to measure the quality of the SA solution. The lower bound can be used to prune sections of the search space, thereby improving the performance of SA. Further, if the problem has a particular structure it may be possible to further reduce the search space by incorporating that knowledge into the core algorithm. In Sections 3 and 5 we introduce such a technique.

2. Using SA to Solve 0-1 Integer Programs

Many combinatorial optimisation problems can be formulated as the minimisation of an objective function C(**x**) where the vector **x**= X_j, j=1, ... ,N, consists of integers which can

only take the values zero or one. The minimisation is usually subject to a number of constraints which link the value of the free variables so that they obey certain limitations. These constraints are held in the matrix **A**. In the linear system, both the objective function and the constraints are linear, and take the following form:

$$\begin{aligned} \min \quad & C(\mathbf{x}) = \sum_{j=0}^{n} c_j x_j \\ \textit{subject to:} \quad & \mathbf{Ax} \le \mathbf{b} \\ & x_j \in \{0,1\} \end{aligned}$$

Problems of the type described above can often have enormous solution spaces. SA, as described above, has the ability to traverse such spaces in search of global minima. In the context of the 0-1 formulation, the search can proceed by randomly toggling the values of the binary variables, whilst using the Boltzmann distribution to determine whether up hill moves are accepted.

However, the basic SA algorithm described in the introduction takes no account of constraints when it performs the Markovian tranformation. There are two main techniques that can be used to incorporate these constraints. First, they can be enforced when each local move is made, and thus only feasible moves are generated. This approach is described in [14], and is very effective when feasibility can be restored easily. It has the advantage that all partial solutions are feasible, and thus the search can stop at any time with a feasible solution. However, it has the disadvantage that it is difficult to generate moves when the problem is tightly constrained. In fact, finding an initial feasible solution may be very hard. The second approach is to relax the constraints and incorporate them into the objective function by using a penalty function. Thus, constraints which are violated force the cost to rise. When the cost is minimised, the original objective is minimised and the constraints are satisfied. This approach has the advantage that the move generator can produce any change without concern for the constraints. Unlike the previous scheme, the search can proceed through infeasible regions in the hope that a feasible, and lower cost, region lies beyond. However, it has the disadvantage that the new objective function is highly non-linear even if the original problem was linear. Thus, it is possible for the search to stop in a local minimum, which may not even be inside the feasible region. Regardless of which generation strategy is used, the basic form of the algorithm is as follows:

```
Generate initial configuration called x
Compute cost of initial configuration C(x)
Compute initial temperature T0
Set T=T0
    While not frozen and C(x) non optimal do begin
        Repeat markov_chain_length times begin
            i = random(0,n-1)
            Set x' = x ⊕ 2^i
            If using first generation scheme,
                then restore feasibility of x'
            ΔC = C(x) - C(x')
            If ΔC ≤ 0 or e^(-ΔC/T) > unif_rand(0,1) then set x = x'
        end
        T = cool(T)
    end
```

where:

a ⊕ b returns the exclusive-or of a and b

random(a,b) returns a random uniformly distributed integer between a and b

unif_rand(a,b) returns a random uniformly distributed real between a and b

cool (T) returns a new value for the temperature after applying a cooling operation to T. In the experiments reported in this paper we use a reheating scheme in which the temperature is reduced and then raised to escape from local minima.

As described in the introduction, SA takes no account of the problem structure and simply randomly makes moves in the search space. There is nothing to prevent the search from cycling or generating moves into regions which cannot be in the final solution. In the next section we describe a technique which allows the algorithm to reduce the size of the search by setting certain values in the x vector to zero. In this way they can be removed from the search.

3. Preprocessing 0-1 IPs Using LP Lower Bounds

Consider the 0-1 integer programming problem,

$$P_{0-1}: Z_{0-1} = \min \quad Cx \tag{1}$$

$$st: Ax \le b \tag{2}$$

$$X_j \in \{0,1\} \tag{3}$$

Here, C is an n-vector, A is an $m \times n$ matrix and b is an m-vector. The problem $P_{0\text{-}1}$ has an optimal solution value, $Z_{0\text{-}1}$.

A relaxation of $P_{0\text{-}1}$ is a problem in which some of the difficult constraints have been either ignored or simplified. In particular, the LP relaxation of $P_{0\text{-}1}$ is one wherein constraint (3) is simplified.

$$P_{0-1}: Z_{0-1} = \min \quad Cx \tag{4}$$

$$st: Ax \le b \tag{5}$$

$$x \le 1 \tag{6}$$

$$x \ge 0 \tag{7}$$

$LP_{0\text{-}1}$ is an LP relaxation of $P_{0\text{-}1}$. It follows that $Z_{LP} \le Z_{0\text{-}1}$. In other words, the optimal solution to $LP_{0\text{-}1}$ forms a lower bound to the solution of $P_{0\text{-}1}$ This lower bound is quite useful in several respects. First, it can be used to measure the quality of the upper bounds. Let Z_{UB} be the upper bound obtained through the SA heuristic. The optimal solution to the $P_{0\text{-}1}$ is such that $Z_{LP} \le Z_{0\text{-}1} \le Z_{UB}$. The *gap* between Z_{LP} and Z_{UB} can, therefore, be used to gauge the effectiveness of the upper bounding algorithm. Second, the lower bound can be used in conjunction with the upper bound to perform preprocessing operations. Given a formulation of a problem, preprocessing refers to operations that improve or simplify it by tightening bounds, fixing values of variables, and so on.

Consider the problem $LP_{0\text{-}1}$ and let x^{LP} be the optimal solution vector. Let $x_B^{LP} \subset x^{LP}$ be the set of basic variables and let $x_J^{LP} \subset x^{LP}$ be the set of non basic variables in the LP optimal solution x^{LP}. For simplicity, let us assume that the current optimal basic feasible solution is non-degenerate (that is, for the m constraint LP, the current optimal solution has m variables assuming positive values). For any nonbasic variable $X_k \in x_J^{LP}$, we can define the reduced cost, C_k, as the amount by which the objective function coefficient of X_k must be improved (decreased) before X_k could assume a positive value in the optimal solution. Another interpretation of C_k, the reduced cost of a nonbasic variable $X_k \in x_J^{LP}$, is that it is the amount by which the value of Z_{LP} will change (increase, in the case of a minimisation problem) if we increase the value of X_k by 1 from its value of 0 (while all the other nonbasic

variables remain equal to zero), thereby introducing it into the basis. By definition, the reduced cost of basic variables is 0.

In 0-1 integer programming problems, the reduced cost information provides us with a measure to evaluate an increase by 1 (the upper bound) in the activity level of nonbasic variables from their value of 0 in x^{LP}. Thus, the introduction of a nonbasic variable, $X_k \in x_J^{LP}$ in the basis has the effect of an increase in the lower bound from Z_{LP} to $Z_{LP}+C_{k'}$. We can use this information to eliminate variables from the problems and hence, reduce its size.

For all $X_k \in x_J^{LP}$, if $Z_{LP}+C_{k'}>Z_{UB}$, X_k can never be a part of any optimal solution to $P_{0\text{-}1}$ and hence, can be eliminated from the problem. This is because we know that the optimal solution to $P_{0\text{-}1}$ is such that $Z_{LP} \le Z_{0\text{-}1} \le Z_{UB}$.

In this manner, all nonbasic variables in x_{LP} can be considered in turn as candidates for elimination. This process often results in significant reductions in problem size. We will illustrate this in Section 6.

4. The Set Partitioning Problem

Let M = {1, ..., m} and N = {1, ... n} denote the set of rows and columns, respectively, in an mXn matrix, A. Further, assume that the elements of the matrix A take values of either 0 or 1. Let a_{ij} denote the i,jth element of A. Associated with each column j, of the matrix A is a cost C_j.

Then, the set partitioning problem (SPP) on matrix A can be formulated as follows:

$$P_{SPP}: Z_{SPP} = \min \sum_{j=1}^{n} C_j X_j \tag{8}$$

$$s.t. \sum_{j=1}^{n} a_{ij} X_j = 1, \forall i \in M \tag{9}$$

$$X_j \in \{0,1\} \tag{10}$$

A well known application of the SPP is the airline crew scheduling problem (CSP). The goal is to minimise crew costs while satisfying constraints on work/duty content. Given an airline's flight schedule (arrivals and departures at all destinations, including aircraft types), *pairings* are initially formed. Each pairing is a sequence of flight `legs' which begin and end at a base location, known as *home-base*. Pairings are governed by union rules, aviation rules and operational rules (length of stay away from home-base), and so on. Each pairing also has a cost associated with it. Given an exhaustive set of legal pairings, each with an associated cost, the problem of finding the best set of pairings such that each flight leg is covered once is the SPP, given by (8), (9), and (10).

Given a solution vector x, a flight leg is said to be *overcovered* if more than one pairing covers it; it is said to be *undercovered* if it is not covered by any pairing; it is exactly covered if it is covered by exactly one pairing.

Each row of the matrix A = $[a_{ij}]$ represents a flight leg and the 0-1 variables X_j indicate which of the pairings j (j=1,....n) are selected in the optimal solution. The matrix A is constructed such that:

$$a_{ij} = \begin{cases} \textbf{\textit{1, if flight leg i is covered by pairing j}} \\ \textbf{\textit{0, otherwise}} \end{cases}$$

CSPs are normally very large problems. For example, for problems involving just 500 flight legs, in excess of a million feasible pairings could easily result. This then results in a 0-1 integer programming problem with more than one million variables and 500 constraints.

Operating costs for the flying crew of some major airlines are in excess of $1.3 billion each year [3]. Moreover, personnel costs are often the second largest item, after fuel, of the operating costs for most airlines. Hence, even small improvements in existing operations can result in huge savings. Thus, good solution methods for these large and difficult 0-1 problems are essential.

Most algorithms for the CSP consist of two steps. The first is the generation of a set of feasible pairings and the second is the solution of the resulting SPP. In the second stage, the SPP can either be solved exactly or by heuristic methods. In this paper, we develop an approximate algorithm for SPP using SA, while constantly attempting to reduce the problem size.

Like the CSP, most other applications of the SPP also arise in the transportation area: scheduling of trucks [7, 6], scheduling of ships [11, 18], scheduling of airline fleets [26], airline crew scheduling [30, 28, 29]. Detailed surveys on the SPP, including its applications, can be found in [15, 20, 5, 6].

Specific algorithms for solving the SPP have been proposed in [4, 20, 34, 27, 31, 33]. The most effective of these, for a wide class of problems, appears to be the one developed by Marsten [27].

Most algorithms relax the binary condition on X_j, in P_{SPP}, thereby obtaining the LP relaxation problems $P_{LP\text{-}SPP}$, with the constraints $0 \leq X_j \leq 1$ instead of constraint (10). The non-integral X_j's are resolved through a branch-and-bround tree-search. In initial implementations of branch and bound algorithms, the choice of branch variables and constraints were not sophisticated: The solution space is searched by setting variable values to 0 and 1 alternately, thereby generating sub-problems and partial solutions. However, a more sophisticated branching scheme was applied by Marsten [27], in which branching decisions are based on groups of variables, through which the status of individual variables is determined automatically. This approach works particularly well for SPPs with sparse matrices (like that in the CSP).

However, in all of these algorithms, the solution of many LPs is the critical factor affecting computational efficiency of the resulting algorithm. This is because the LP relaxation of P_{SPP}, being highly degenerate in most instances, is difficult to solve. Hence, although solutions to $P_{LP\text{-}SPP}$ are normally integral and although $Z_{LP\text{-}SPP}$ is, in practice, a tight lower bound on Z_{SPP}, the branch and bound algorithm that uses the LP relaxation is often computationally expensive. In a recent paper, Hoffman & Padberg [21] solve CSPs using what they call a *branch and cut* algorithm. This has proved quite effective even for large problems.

In this paper we demonstrate the use of SA to solve the set partitioning problem. The method will be discussed in the following section.

5. Using SA to Solve the Set Partitioning Problem

In Section 2 we described a general SA algorithm capable of solving general 0-1 problems. In this section we discuss some of the features of the solution which are particular to the set partitioning problem. These are the internal data structures, the structural properties which can be exploited, the cost function structure adopted, the choice of the move operator and the cooling function.

Our choice of problem (the SPP) is not intended to demonstrate an algorithm which is more effective at solving the SPP than the specific algorithms in the literature that have been reviewed in Section 4. Rather, it provides a test problem which is widely recognized as

difficult. Further, our SA algorithm makes use of the technique of lower bounding of the objective function as used in the branch and bound techniques used for solving SPPs. Thus we demonstrate that this scheme has value in simulated annealing.

5.1 Data Structures

A sparse matrix representation is employed to denote the characteristic of the data structure. Each pairing is uniquely defined by its own features such as cost, number of flights, LP reduced cost, etc. Each pairing also points to a chain of non-zero flight leg elements which are the flight legs that it covers. Pairings are packed into a one dimensional array for the ease of indexing.

A hash table is generated at the beginning of the run. It uses binary representations of flights and pairings to make the process of swapping columns easier.

A solution array is created to hold the column indices which are currently in the solution *i.e.*, j such that $X_j=1$. Also we maintain a flight counter to keep track of the current number of flight legs in the solution, and the number of flight legs that cover each row, after problem reduction is completed. The latter helps the second pruning scheme using the structural properties of the SPP.

5.2 Structural Properties

Apart from the problem reduction scheme (mentioned in Section 3) using the LP relaxation, the second pruning scheme can be derived based on the SPP structural property. Let J be the set of pairings associated with nonzero entries in row i. If one of the pairings, $k \in J$ is known to be in the solution, then every pairing $l \in J$, $l \neq q$ k must be zero in all feasible zero-one solutions.

This structural property can be incorporated into the problem by maintaining the number of flight legs remaining for each row, i. If one of the flight counters gets down to 1, the column containing that flight leg must be in the solution. We call this column an *indispensable* column. Let us assume that column j is indispensable. Assume further that column j also covers flight leg l. Clearly, since j must be in the optimal solution, flight l is also automatically covered. Furthermore, all other columns that cover l (other than j) can be removed from the problem as they become *superfluous.* This process is carried out till no other superfluous columns can be identified.

This results in a very low number of columns at the end of the SA for many of the test problems we used. The number of columns remaining is negligible even to perform an exhaustive search to prove the global optimality.

5.3 Cost Function Structure

The relaxation of the constraint (9) results in a second type of cost, called *penalty cost*, beside the actual operating cost, C_j,which acts on the rows of the problem. The latter operates on the columns. Let the *set cost* be the sum of all the row penalty costs. The penalty cost is positive when the SA solution violates some of the row sum constraints. Clearly, when an intermediate SA solution is such that, all rows are covered exactly once, the penalty cost for all rows is zero. The set cost for this solution is also zero. If a row is under covered or over covered, we impose a penalty depending on the amount of violation. The profile of the penalty cost curve we have used is shown in Figure 1.

The profile of the penalty cost has to be chosen carefully. If the cost for deviation from a row sum of 1 is too high, then it will be more difficult for a new configuration to be generated. However, if the penalty for deviation is too low, infeasible solutions will be readily admitted and it is likely that the final solution will be infeasible.

Moreover, to prevent excessive over covering of a flight, a large penalty cost is assigned to rows that are covered by more than MaxCount (MC) pairings. Although the introduction of such a penalty (set) cost makes the original linear cost function nonlinear, it removes the constraints in the problem.

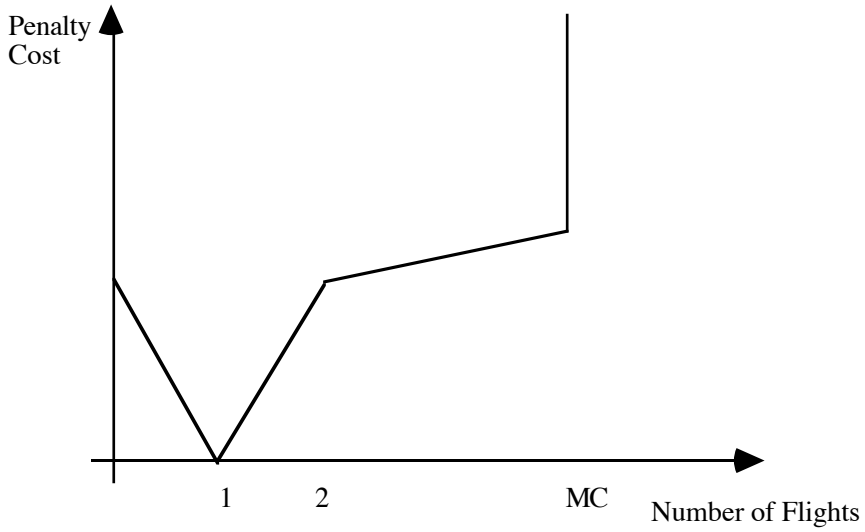

Figure 1: Penalty cost profile for the SPP

5.4 The Move and Swap Operators

Given an initial configuration, a transition is performed within the current configuration's neighbourhood. This is achieved by randomly choosing a variable. We take it out of the solution if it is currently in it or insert it in the solution if it is not. The change in cost is then calculated. If it is accepted by the SA rules, the configuration is updated. Otherwise the old configuration is retained.

With a finite cooling schedule, the SA algorithm drives the system to a local optimum solution. The gap between this local optimum and the global optimum depends on the number of permutations performed at each temperature (or the Markov chain length) and the cooling schedule. Faster cooling and short Markov chains lead to poorer final solutions. If the system is trapped in local minima at low temperatures, the chance of escaping from it is difficult, unless the neighbourhood configuration is extended. This is achieved by either increasing the system energy (described in the next Section) or by a more sophisticated move.

We use a sophisticated K-R opt method for swapping pairings [17]. Under this method, a group of pairings, of size K which are currently in the solution is randomly chosen. The union of the flights that are covered by this group is computed. The set of flights covered by the chosen group of pairings is denoted as `UNISET'. An attempt is made to swap this group with another group of pairings of size R (not necessarily having the same number of pairings) which are not in the current configuration. The new group is chosen such that no flight is covered more than once. Moreover, the union of their flight legs is the same as UNISET.

In the case where R is less than K in the K-R opt procedure, a move operator is also performed in conjunction with these swaps in order to prevent the progressive lack of pairings in the solution as the SA algorithm proceeds.

5.5 The Cooling Schedule

A cooling schedule consists of: (i) the starting temperature. (ii) the cooling factor. (iii) the length of the Markov chain. (iv) the process termination condition.

We use a starting temperature based on the work of White [37], who proposed that the starting temperature should be one standard deviation around C_{mean}, the mean cost, based

on his configuration density function. We use a cooling factor of 0.9. This is based on work by Kirkpatrick [23, 24] who propose that the temperature decrement rule of should be controlled by a constant cooling factor which is small but close to 1. This rule was used by, for example, Johnson, et al [22], Bonomi and Lutton [8,9,10], Leong & Liu [25], Morgenstern & Sharpiro [32], Sechen & Sangiovanni-Vincentelli [35]. For simplicity, we used a constant Markov chain length for all the problems we tested our SA on.

The difference between our cooling schedule and those used eariler in the literature is the use of a *temperature reset* procedure which increases the temperature if we detect that the system is trapped in local minima. We say a local minimum is reached if two consecutive Markov chains provide no change in the solution (either uphill or downhill). Furthermore, we say a local minimum is reached if there is a constant fluctuation around a point (present in cases of cycling). At such points, we reset the system temperature according to:

$$T_{new} = P \times C(x^b) + Q$$

where, T_{new} = new reset temperature when a local minimum is detected.
x^b = the current best solution
$C(x^b)$ = the cost of the current best solution
P and Q = constants

Using the above temperature reset scheme, the temperature will be increased up to the point at which the rate of change of expected cost with respect to temperature is at its maximum. In the physical analogy, this rate is referred to as the specific heat of the mass (see [36] pages 41-49). A clear exposition of this variation in our cooling schedule and the mathematical justification for it is provided in [2]

6. Computational Results

In this section we present results of our SA algorithm. The test problems are taken from data used in Hoffman & Padberg [21]. Table 1 presents the results of using the lower bound pruning technique (in combination with the SA). Table 2 shows the results achieved using basic SA. Table 3 shows the results from incorporating the structural properties into the algorithm.

Each problem originally consists of a number of rows and columns. After pre-processing it is possible to remove redundant rows and columns using a technique described in Hoffman & Padberg [21]. In this manner the problem size is reduced. As the SA algorithm proceeds it attempts to reduce the problem size further by using the reduced cost values in conjuction with the LP lower bound,as described in Section 3. Figure 2 shows a plot of the reduced costs plus the lower bound versus the column number for Test Problem #12. It shows that providing the SA can produce a reasonably tight upper bound, a large portion of the columns can be pruned through the technique described in Section 3.

In Table 1 we show the number of columns at the beginning of the SA run and the number remaining at the end. It can be seen that in many cases if the LP lower bounds are continually used in conjunction with the SA upper bounds, the number of columns in the problem (and hence the search space) is reduced dramatically. Table 1 also presents the known optimal cost for each of the test problems, together with the best and average costs returned by the SA (from 6 independent runs). The average times are also presented, as well as the amount of time required to preprocess the data and solve the relaxed LP. In the final column we indicate whether the SA best cost is optimal, and if not, we give the fraction of optimal. 31 out of the 37 tests returned an optimal result.

In order to demonstrate that the performance of the basic SA algorithm is improved by using the problem reduction scheme described in Section 3, we ran the same problems without employing the problem reduction tests. Results of this experiment are provided in Table 2. We report the final cost achieved after *the same amount of time as when the pruning scheme*

was used, and also after a significantly larger time. We have chosen the first 29 test problems, all of which solved in Table 1. The results show that the pruning scheme *consistently* produces a higher quality solution in the same amount of time, and that after allocating significantly more time to the basic SA that the quality of the solution had not improved very much. This is due to the random nature of the basic SA algorithm and its inability to restrict itself to fruitful areas of search.

Test No	No. of R x C	No of (R x C) After Pre-Processing	No. of C After SA	Known Opt. Cost	SA Best Cost	SA Aver. Cost	SA Aver. Time (Secs)	Preprocessing Time Secs)	LP Solve Time (Secs)	Optimal or % away from optimal
1	17 x 197	17 x 177	17	11307	11307	11307	27.6	0.2	0.6	Yes
2	19 x 294	18 x 251	25	14877	14877	14877	33.9	0.4	1	Yes
3	19 x 404	19 x 336	18	10809	10809	10809	32.3	0.7	1.4	Yes
4	24 x 434	21 x 352	20	35894	35894	35894	324.9	1.0	1.9	Yes
5	31 x 467	29 x 405	30	67743	67743	67743	68.9	1.1	2	Yes
6	25 x 577	25 x 421	16	7408	7408	7408	66.9	2.0	2.1	Yes
7	23 x 619	23 x 521	23	6984	6984	6984	344.8	1.3	2.5	Yes
8	27 x 626	25 x 451	27	14118	14118	15338	1276.6	1.4	1.9	Yes
9	25 x 677	25 x 565	20	10080	10080	10080	68.4	2.3	2.6	Yes
10	22 x 685	22 x 536	34	16812	16812	16812	118.7	1.7	3.5	Yes
11	19 x 711	18 x 423	57	12534	12534	12534	312.6	2.5	2	Yes
12	19 x 770	19 x 639	19	10068	10068	10068	30.8	3.5	2.8	Yes
13	23 x 771	21 x 468	22	6796	6796	6796	78.0	2.2	2.3	Yes
14	24 x 853	21 x 655	22	68271	68271	68271	971.0	2.7	3	Yes
15	20 x 899	20 x 718	21	10488	10488	10488	64.3	3.1	3.9	Yes
16	18 x 1072	17 x 982	21	8904	8904	8904	509.2	2.4	4.4	Yes
17	23 x 1079	23 x 795	39	7656	7656	7656	152.8	4.4	4.6	Yes
18	18 x 1210	18 x 582	18	8298	8298	8298	18.1	6.4	3	Yes
19	20 x 1217	20 x 844	21	5960	5960	5960	155.2	3.9	5	Yes
20	23 x 1220	21 x 723	25	5558	5558	5558	19.3	10.8	5.1	Yes
21	22 x 1355	22 x 817	15	9933	9933	9933	69.0	6.9	4.2	Yes
22	19 x 1366	19 x 926	57	6314	6314	6314	77.1	5.7	4.6	Yes
23	23 x 1709	23 x 1191	18	7216	7216	7216	150.5	9.1	5.9	Yes
24	20 x 1783	20 x 1244	108	7314	7314	7314	328.1	11.4	12	Yes
25	18 x 2504	18 x 2034	38	4274	4274	4274	306.0	15.3	15.4	Yes
26	26 x 2653	26 x 1877	36	3942	3942	3942	111.0	27.2	11.8	Yes
27	26 x 2662	26 x 1728	36	8038	8038	8038	155.9	20.2	13	Yes
28	40 x 2879	32 x 2134	21	10898	10898	10898	608.3	23.2	16	Yes
29	23 x 3068	23 x 2308	39	6678	6678	6678	1404.4	28.7	14.6	Yes
30	40 x 3103	38 x 2301	734	67760	69006	69993	5064.3	22.4	25.8	2%
31	36 x 5172	34 x 3105	20	5476	5476	7358	7987.1	67.4	20.7	Yes
32	531 x 5198	360 x 3846	3846	30497	39627	41354	9182.5	137.9	894.6	30%
33	50 x 6774	38 x 5956	81	7810	7810	8778	12226.8	129.6	93	Yes
34	646 x 7292	492 x 5898	5892	27040	40183	41624	10499.8	296.6	2904.7	49%
35	55 x 7479	47 x 5915	5915	1086	1292	1427	7546.4	153.2	132.9	19%
36	100 x 13635	45 x 5766	5766	5965	8469	9511	10379.9	762.2	96.4	42%
37	51 x 16043	50 x 10903	10899	50146	58028	59987	8623.1	994.2	146.3	16%

R refers to the number of rows and C refers to the number of columns in the problem.

Table 1: Results of using lower bound pruning technique

There are a few data sets in the table for which our algorithm has not returned optimal results, namely 30, 32 and 34 through 37. Also, 33 has failed to return the optimal solution for all 6 experiments. In these cases the SA was not able to produce a sufficiently tight upper bound, and the number of columns could not be reduced during the run. The reason why the SA has not produced a good bound can be found by examining the shape of the reduced cost curve

for each problem. Figure 2 shows the reduced cost curve for problem 12, which is actually typical of the first 29. This shows the value of the reduced costs for each column plotted against the column number. It shows that there are only a relatively small number of columns below the optimal cost value, and the shape of the curve means that a reasonable upper bound will prune quite a large number of columns. For example, the optimal cost for Test 12 is 10068. There are only about 20 columns which have reduced costs below that value. Moreover, a cost of 13000 only has a few hundred columns below it, which is substantially smaller than the original 770.

Figure 3 shows the curve for problem 33, which did not return an optimal solution for all 6 runs. Here the curve has a very different shape. The upperbound has to be very good before many columns are pruned from the search space. Figure 4 shows the curve for problem 35, which failed to solve for any of the 6 runs. The steps in the trace indicate that there are many columns with the same cost, but possible each with different coverage of the flights. Also, the curve is very steep like the one in Figure 2, thus a good upper bound is required in order to prune the search.

Thus, we can conclude that SA with pruning is effective only when the relaxed LP produces a sufficiently tight lower bound and when the slope of the reduced cost curve is not too steep. One would expect exactly the same behaviour from a branch and bound algorithm which relied on the same lower bound in order to prune nodes of the search tree.

Table 3 shows the effect of including the structural properties to constantly prune the search space within an SA algorithm. As in Table 1, the figures in Table 3 are averaged over 6 independent runs.

Test No	R x C	Known Opt Cost	SA Best Cost	SA Average Cost	SA Average Time	Optimal or % away from optimal	Test No	R x C	Known Opt Cost	SA Best Cost	SA Average Cost	SA Average Time	Optimal or % away from optimal
1	17x197	11307	12474	13301	27.2	10%	16	18x1072	8904	9586	10205	509.6	8%
			11307	11328	2619.8	Yes				9372	9618	4458.2	5%
2	19x294	14877	15339	15668	34.5	3%	17	23x1079	7656	7786	8937	184.2	2%
			14877	14908	2378.8	Yes				7656	7656	1586.2	Yes
3	19x404	10809	11052	11483	33.2	2%	18	18x1210	8298	8298	8967	14.6	Yes
			10809	10809	1367.9	Yes				8298	8298	38.3	Yes
4	24x434	35894	36890	40622	323.3	3%	19	20x1217	5960	7484	8013	150.2	26%
			36478	36583	4137.0	2%				6592	6751	4475.2	11%
5	31x467	67743	67743	68481	69.6	Yes	20	23x1220	5558	5976	6362	20.6	8%
			67743	67743	822.3	Yes				5558	5558	1067.6	Yes
6	25x577	7408	7888	10019	68.6	6%	21	22x1355	9933	11049	11545	71.7	11%
			7408	7449	4017.1	Yes				9933	10020	2630.3	Yes
7	23x619	6984	9211	9211	338.6	32%	22	19x1366	6314	7460	□7887	79.1	18%
			6984	7505	3716.6	Yes				6396	6480	4660.4	1%
8	27x626	14118	15302	15715	1278.6	8%	23	23x1709	7216	9140	9543	153.0	27%
			15086	15479	4155.7	7%				7718	8010	4710.7	7%
9	25x677	10080	10722	12410	68.4	6%	24	20x1783	7314	7442	7459	333.8	2%
			10080	10135	2130.7	Yes				7322	7367	4678.8	Yes
10	22x685	16812	17388	18316	117.4	3%	25	18x2504	4274	4926	5086	316.3	15%
			16812	16838	1646.8	Yes				4328	4456	4955.3	1%
11	19x711	12534	13492	13492	304.2	8%	26	26x2653	3942	5458	6009	118.5	38%
			12762	12762	3635.0	2%				3942	□4188	3468.3	Yes
12	19x770	10068	10233	11949	31.0	2%	27	26x2662	8038	8038	9306	158.8	Yes
			10068	10096	1507.6	Yes				8038	8038	1043.0	Yes
13	23x771	6796	7642	8191	77.1	12%	28	40x2879	10898	12148	12752	612.5	11%
			6796	6852	3955.3	Yes				11678	11923	5012.3	7%
14	24x853	68271	70392	74611	974.7	3%	29	23x3068	6678	8148	9190	1407.5	22%
			68520	69363	4331.9	Yes				7274	7505	5112.0	9%
15	20x899	10488	11289	13051	60.8	8%							
			10488	10636	3760.5	Yes							

Table 2: Results using normal SA without the lower bound pruning technique

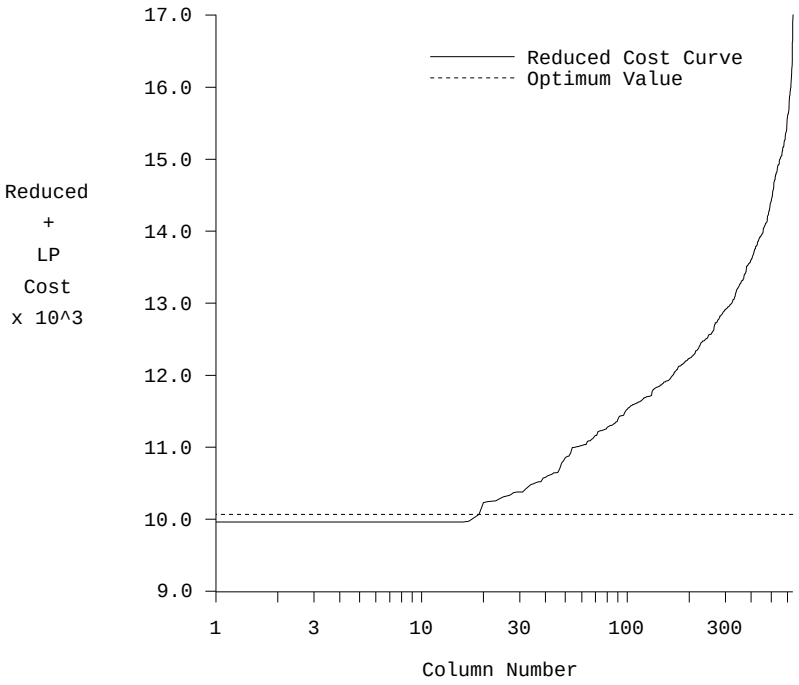

Figure 2 Reduced costs for Problem #12

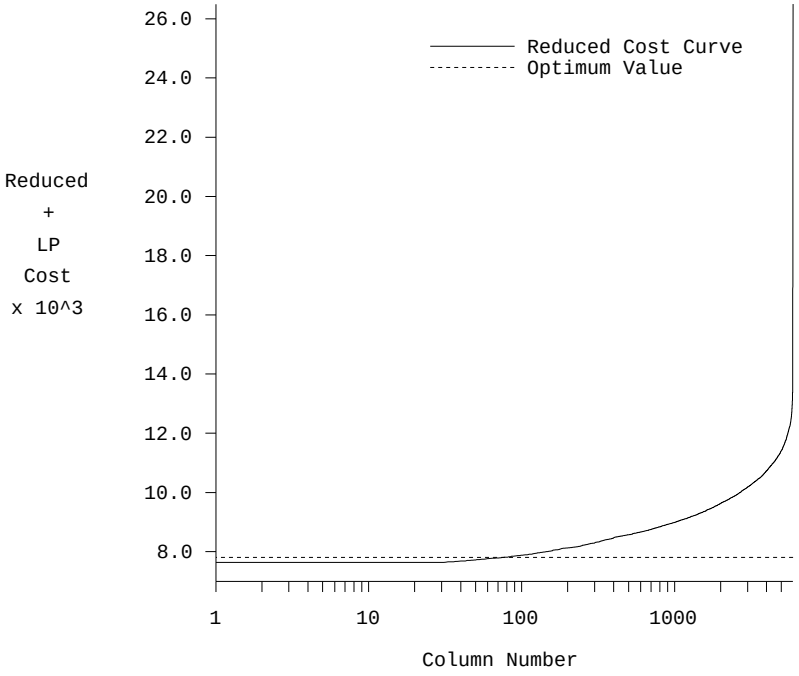

Figure 3: Reduced costs for Problem #33

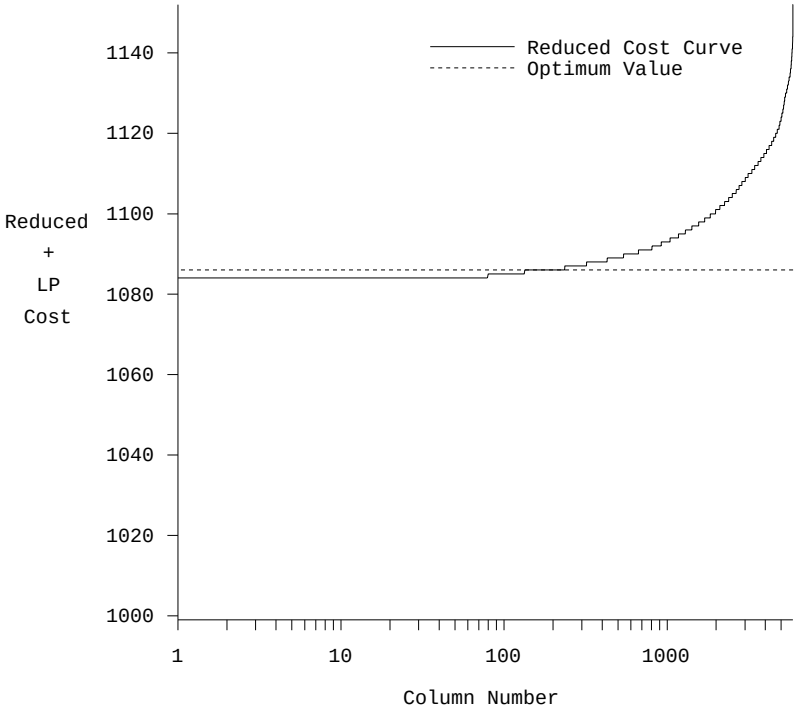

Figure 4: Reduced costs for Problem #35

Test No	RxC	No.of C After SA	Known Opt. Cost	SA Best Cost	SA Aver. Cost	SA Aver. Time (Secs)	Optimal or % away from optimal	Same Soln as Table 1?	Fraction of Time of Table 1	Fraction of C After SA of Table 1
1	17x197	15	11307	11307	11307	26.3	Yes	Yes	95%	88%
2	19x294	25	14877	14877	14877	49.1	Yes	Yes	145%	100%
3	19x404	18	10809	10809	10809	33	Yes	Yes	102%	100%
4	24x434	19	35894	35894	35894	336	Yes	Yes	103%	95%
5	31x467	29	67743	67743	67743	68.8	Yes	Yes	100%	97%
6	25x577	16	7408	7408	7408	75.6	Yes	Yes	113%	100%
7	23x619	22	6984	6984	6984	369.9	Yes	Yes	107%	96%
8	27x626	21	14118	14118	15338	1255.4	Yes	Yes	98%	78%
9	25x677	15	10080	10080	10080	72.4	Yes	Yes	106%	75%
10	22x685	34	16812	16812	16812	116	Yes	Yes	98%	100%
11	19x711	55	12534	12534	12534	304.9	Yes	Yes	98%	96%
12	19x770	11	10068	10068	10068	37.2	Yes	Yes	121%	58%
13	23x771	22	6796	6796	6796	69.8	Yes	Yes	89%	100%
14	24x853	22	68271	68271	68271	970.6	Yes	Yes	100%	100%
15	20x899	15	10488	10488	10488	65.4	Yes	Yes	102%	71%
16	18x1072	20	8904	8904	8904	518.3	Yes	Yes	102%	95%
17	23x1079	39	7656	7656	7656	154.7	Yes	Yes	101%	100%
18	18x1210	18	8298	8298	8298	18.1	Yes	Yes	100%	100%
19	20x1217	21	5960	5960	5960	162.8	Yes	Yes	105%	100%
20	23x1220	25	5558	5558	5558	21.7	Yes	Yes	112%	100%
21	22x1355	15	9933	9933	9933	53.9	Yes	Yes	78%	100%
22	19x1366	57	6314	6314	6314	79.5	Yes	Yes	103%	100%
23	23x1709	18	7216	7216	7216	171.7	Yes	Yes	114%	100%
24	20x1783	108	7314	7314	7314	174.5	Yes	Yes	53%	100%
25	18x2504	38	4274	4274	4274	310.5	Yes	Yes	101%	100%
26	26x2653	36	3942	3942	3942	110.6	Yes	Yes	100%	100%
27	26x2662	36	8038	8038	8038	166	Yes	Yes	106%	100%
28	40x2879	21	10898	10898	10898	637.4	Yes	Yes	105%	100%
29	23x3068	35	6678	6678	6678	787	Yes	Yes	56%	90%
30	40x3103	734	67760	69006	69993	5022.1	2%	Yes	99%	100%
31	36x5172	9	5476	5476	7358	7909.8	Yes	Yes	99%	45%

Table 3: Using Structural properties to fix variables

At first inspection, the results in Table 3 indicate that the use of the problems' structural properties does not improve the performance of our SA algorithm with search-space pruning based on LP lower bounds. In almost all cases the time to solution is actually slower than those reported in Table 1. However, the advantage of using structural properties to prune the search space is that, at the end of the SA run, we are often left with a very few variables in the problem. Hence, even an exhaustive search to prove optimality takes very little computational effort.

7. Conclusions

In this paper, we have presented a simulated annealing algorithm for the SPP, the underlying problem in airline crew scheduling problems. We have tested it on some practical--sized problems from the literature and the results indicate that the algorithm is quite powerful. Using clever pruning techniques, we are able to reduce the search space (by a large amount. The improved algorithm always out performs the basic simulated annealing algorithm, even when it cannot solve the problem). This method also enables us to comment on the quality of the final solution obtained. For some problems of a specific nature however, the technique does not find reasonable solutions. Further investigations need to be carried out to deal with these complications.

Linear programming preprocessing has been used previously in conjunction with SA, although in a different manner to that which we adopt in this paper. Cattrysse et al [12, 13] have applied a variable reduction procedure that fixes some variables X_{ij} to 1 while solving the generalized assignment problem (GAP). This enables a reduction in problem size. The reduction procedure (or, Fixing procedure) is based on the addition of valid inequalities (or facets) of the knapsack polytope. The procedure begins by solving an LP relaxation of the GAP. Based on the solution, the formulation is strengthened by the addition of a valid inequality for every original knapsack constraint that is violated. The new formulation is resolved to generate more facets and a better bound. The procedure is repeated till no further valid inequalities can be found. At the end of this procedure, all variables which have $X_{ij} = 1$ can be removed from the problem and the formulation can be adjusted to reflect this knowledge. The reduced problem is then solved using an SA procedure. The benifits of incorporating the above preprocessing approach for the GAP are increased speed and the ability to solve larger problems.

The difference between the above and our approach is that our approach is more general in its applicability. No specialized knowledge of the problem is required (to generate a family of valid inequalities, for example). Moreover, our procedure is applied continuously, throughout the SA execution. In that sense, the search space is continually reduced, not merely at the start of the procedure.

We have not fully investigated the scheme where only feasible moves are allowed in the algorithm. However, limited initial results seem to indicate that this scheme works better than the one in which infeasible moves are accepted with a resulting penalty. Under this scheme, the time to perform each move appears to be much longer than the current times. However, the overall time for the SA to complete appears to be quicker. Moreover, the quality of the final solution is also better. This particular aspect requires further investigation.

Acknowledgements

Henry Dang is supported by an Australian Postgraduate Research Award (Industry) and Computer Techniques P/L. We also acknowledge the constructive comments on the paper provided by Kurt Brinschwitz and Kate Smith, both of DMS-CSIRO.

References

[1] Aarts.E and Korst. J., (1989), "Simulated Annealing and Boltzmann Machines", John Wiley, Chichester.

[2] Abramson, D.A. , Dang, H. and Krisnamoorthy, M., (1994), *Cooling Schedules for Simulated Annealing Based Scheduling Algorithms*, Proceeding of the 17th Australian Computer Science Conference, Univ of Caterbury, Christschurch, NZ, Jan 1994.

[3] Anbil, R., Gelman, E., Patty, B., and Tanga, R., (1991), *Recent Advances in Crew-Pairing Optimization at Armerican Airlines*, Interfaces, 21, 62--74.

[4] Balas, E., (1975), *On the Set Covering Problem II. An Algorithm for Set Partitioning*, Oper. Res, 23, 1, 74-90.

[5] Balas, E., and Padberg, M.W., (1974), *Set partitioning*, MSRR No. 351, CMU.

[6] Balas, E., and Padberg, M.W., (1976), *Set Partitioning: A Survey*, SIAM Review, 18, 710--760.

[7] Balinski, M.L., and Quandt, R.E., (1964), *On an Integer Program for a Delivery Problem*, Oper. Res., 12, 300-304.

[8] Bonomi, E. and Lutton, J. L., (1984a), *The N-city Travelling Salesman Problem: Statistical Mechanics and the Metropolis Algorithm,* SIAM Rev., 26, 551-568.

[9] Bonomi, E. and Lutton, J. L., (1984b), *The Asymptotic Behaviour of Quadratic Sum Assignment Problems: A Statistical Mechanics Approach*, Euro. J. of Oper. Res.

[10] Bonomi, E. and Lutton, J. L., (1986), *Simulated Annealing Algorithm For the Minimum Weighted Perfect Euclidean Matching Problem*, R.A.I.R.O. Recherche operationelle, 20, 177-1282.

[11] Brown, G.B., and Graves, G.W., and Ronen, D., (1987), *Scheduling Ocean Transportation of Crude Oil*, Management Sci., 33, 3, 335-346.

[12] Cattrysse.D.G, Set partitioning approaches to combinatorial optimization problems, PhD Thesis, Catholique Universiteit Leuven, Department Wertuigkunde, Appeling Industrieel Beleid, Belgium.

[13] Cattrysse.D.G, Van Wassenhowe.L.N, A survey of algorithms for the generalized assignment problem, European Journal of Operational research, 60, pp 260-272, (1992).

[14] Connolly, D. (1992), *General Purpose Simulated Annealing*, J Opl. Res. Soc., 43, 5, 495-505.

[15] Christofides, N. et. al., (1979), *Combinatorial Optimization*, Wiley

[16] Cullen, F.H., Jarvis, J.J., Ratliff, H.D., (1981), *Set Partitioning Based Heuristics for Interactive Routing*, Networks, 11, 125-143.

[17] Fisher, M.L., and Kedia, P., (1990), *Optimal Solution of Set Covering/Partitioning Problems using Dual Heuristics*, Management Sci., 36, 6, 674-688.

[18] Fisher, M.L., and Rosenwein, M.B., (1989), *An Interactive Optimization System for Bulk Cargo Ship Scheduling*, Naval Res. Logist. Quart., 36, 27-42.

[19] Garfinkel, R.S., (1972), *Optimal Set Covering: A Survey*, in A. Geoffrion (Ed.), Perspectives on Optimization, Addison-Wesley, Reading, MA.

[20] Garfinkel, R.S., and Nemhauser, G.L., (1969), *The Set Partitioning Problem: Set Covering with Equality Constraints*, Oper. Res., 17, 848-856.

[21] Hoffman, K.L., and Padberg, M., (1992), *Solving Airline Crew-Scheduling Problems by Branch-and-Cut*, Technical Report, George Mason University.

[22] Johnson, D.S., Aragon, C.R., McGeoch, L.A., and Schevon, C., (1986), *Optimization by Simulated Annealing: an Experimental Evaluation*, List of Abstracts, Workshop on Statistical Physics in Engineering and Biology, Yorktown Heights.

[23] Kirpatrick, S., Gelatt Jr., C.D., Vecchi, M.P., (1982), *Optimization By Simulated Annealing*, IBM Research Report RC 9355.

[24] Kirpatrick, S., Gelatt Jr., C.D., Vecchi, M.P., (1983), *Optimization by Simulated Annealing*, Science, 220, 671-680.

[25] Leong, H.W., and Liu, C.L., (1985), *Permutation Channel Routing*, Proc. IEEE Int., Conference on Computer Design, Port Chester, 579-584.

[26] Levin, A., (1969), *Fleet Routing and Scheduling Problems for Air Transportation Systems*, Ph.D. dissertation, MIT.

[27] Marsten, R.E., (1974), *An Algorithm for Large Set Partitioning Problems*, Management Sci., 20, 779--787.

[28] Marsten, R.E., Muller, M.R., and Killion, C.L., (1979), *Crew Planning at Flying Tiger: A Successful Application of Integer Programming*, Management Sci., 25, 12, 1175-1183.

[29] Marsten, R.E., and Shepardson, R., (1981), *Exact Solution of Crew Scheduling Problems using the Set Partitionning Model: Recent Successful Applications*, Networks, 11, 2, 165-177.

[30] McCloskey, J.F., and Hanssman, F., (1957), *An Analysis of Stewardess Requirements and Scheduling for a Major Airline*, Naval Res. Logist. Quart., 4, 183-192.

[31] Nemhauser, G.L., Trotter, L.E., and Nauss, R.M., (1974), *Set Partitioning and Chain Decomposition*, Management Sci., 20, 22, 1413-1423.

[32] Morgenstern, C.A., and Shapiro, H.D., (1986), *Chromatic Number Approximation Using Simulated Annealing*, Department of Computer Science, The University of Mexico, Albuquerque, Technical Report No. CS86-1}.

[33] Pierce, J.F., (1968), *Application of Combinatorial Programming Algorithms for a Class of All Zero-One Integer Programming Problems*, Management Sci., 15, 191-209.

[34] Pierce, J.F., and Lasky, J.S., (1973), *Improved Combinatorial Programming Algorithms for a Class of All Zero-One Integer Programming Problems*, Management Sci., 19, 528-543.

[35] Sechen, C., and Sangiovanni-Vincentelli, A.L., (1985), *The Timber Wolf Placement and Routing Package*, IEEE J. Solid State Circuits, SC-20, 510-522.

[36] van Laarhoven.P.J.M., and Aarts.E.H.L., (1987), *Simulated Annealing: Theory and Applications*, Reidel, The Netherlands.

[37] White, S.R., (1984), *Concepts of Scale in Simulated Annealing*, Proc. IEEE Int. Conference on Computer Design,Port Chester, 646-651.