J. Opl Res. Soc. Vol. 43, No. 5, pp. 495-505
Printed in Great Britain. All rights reserved

0160-5682/92 \$3.50+0.00
Copyright © 1992 Operational Research Society Ltd

General Purpose Simulated Annealing

DAVID CONNOLLY
London School of Economics

This paper reports on an attempt to write a general purpose simulated annealing algorithm, capable of finding good solutions to problems expressed as pure 0-1 integer linear programs. Computational results are given to support the claim that the resulting program might be a useful addition to the arsenal of techniques for researchers and practitioners wishing to tackle certain types of large scale 0-1 ILPs.

Key words: simulated annealing, combinatorial optimization, integer linear programming

INTRODUCTION

Recently there has been some interest in the use of a technique known as simulated annealing for obtaining 'good' solutions to a wide range of problems. The central idea of this method is that certain uphill steps may be required to prevent a local search scheme from getting stuck in a poor local optimum. In the most common versions of this technique, 'downhill' (i.e. improving) perturbations of the existing solution are always accepted but an 'uphill' step of size δ is only accepted with probability $e^{-\delta/T}$, where T is a parameter known as the temperature. This parameter normally decreases during the search so that uphill steps become less and less likely. This reduction of T is referred to as cooling, and a wide variety of cooling schemes have been suggested, along with alternative acceptance rules and numerous other modifications of the basic scheme.

The interested reader is referred to a comprehensive survey paper by Collins *et al.*[1] They list applications ranging from the Travelling Salesman problem to image processing, from jobshop scheduling to pollution control, from coastguard deployment to DNA mapping and many more. In all of these papers the use of annealing has been problem-specific—a separate program has been needed for each new combinatorial problem tackled. The user must provide rules for moving from one feasible solution to the next and evaluating the change in the objective function. A good automatically-determined cooling scheme helps but adapting annealing to tackle a new problem still involves a non-trivial amount of work. The industrial sponsors of this research, SD-Scicon UK Ltd, were interested in the success of the technique but felt that the need to rewrite a program for each new client limited its usefulness to them. They explained that they would be more interested in a general-purpose algorithm which could be sold 'off the shelf', either as a stand-alone package or as an add-on to their existing optimization software, for example as a 'warm-start' to a branch-and-bound search. Could simulated annealing be generalized in this way? SD-Scicon UK Ltd's strength in the field of integer and linear programming made MPS format the obvious method of specifying a combinatorial optimization problem. In this way existing software could be used to generate input data, (i.e. MGG, the matrix generator package) and their sophisticated ILP code, SCICONIC, could be used as a yardstick against which to measure the performance of the resulting heuristic.

A FORTRAN 77 program, referred to here as GPSIMAN, was written to take an ILP formulation of a combinatorial optimization problem, (involving only 0-1 variables), and perform on it the version of simulated annealing described below.

THE SIMULATED ANNEALING SCHEME

This section attempts to explain the workings of the program GPSIMAN, with the aid of the flow diagrams in Figures 1, 2 and 3. A copy of the FORTRAN source code is available from the author.

Problems are read in as 0-1 Integer Linear Programs, an initial feasible solution is found, (using

Correspondence: D. Connolly, OR Department, SAMS, London School of Economics, Houghton Street, London WC2A 2AE, UK.

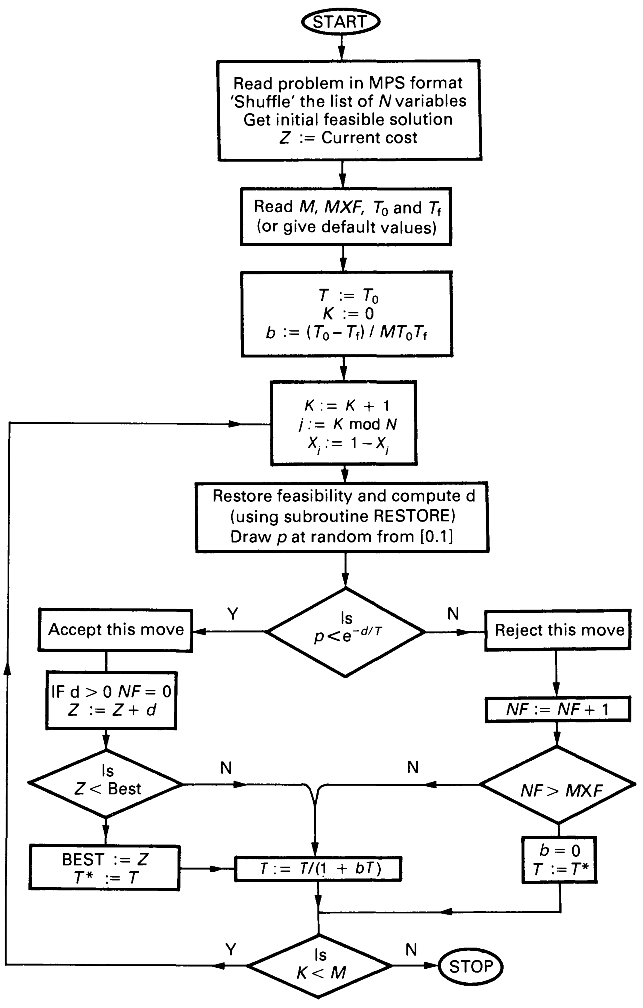

FIG. 1. *Flow chart for GPSIMAN.*

the subroutine RESTORE described below), and any 'free moves' are made by checking each column in turn. (Free moves are variables which can be introduced or removed to reduce the cost of the solution without causing any infeasibility. For example, in a knapsack problem an empty knapsack is the initial feasible solution but a number of objects can be added before the constraint becomes effective and this leads to a better starting solution). The starting point for this checking is chosen at random to prevent undue bias towards the low-numbered columns.

The version of simulated annealing used is based on the Q8–7 scheme developed in Connolly[2], where it was shown to outperform a number of other annealing schemes when applied to a particular combinatorial optimization problem. In this scheme the temperature T is controlled by a Lundy and Mees[4] continuous cooling scheme defined by the recurrence relation $T_{n+1} := T_n/(1 + bT_n)$ where b is chosen so that the temperature falls from an initial value T_0 to a final temperature T_f in M iterations, (see Figure 1). T_0, T_f and M are specified by the user, but default values are supplied by the program and these are used in all computational experiments reported here. If NF, (the number of consecutive uphill rejections), exceeds the parameter MXF, then the cooling is stopped and the temperature is returned to T^*, the value at which the current best solution (BEST), was found and the remainder of the M iterations are performed at this fixed temperature. (MXF is another user-definable parameter). See Connolly[2,3] for further details of this cooling scheme.

A perturbation consists of flipping one of the N neighbour-generating variables X_i from its current value of 0 to 1, or vice versa, restoring feasibility and calculating d, the resultant change

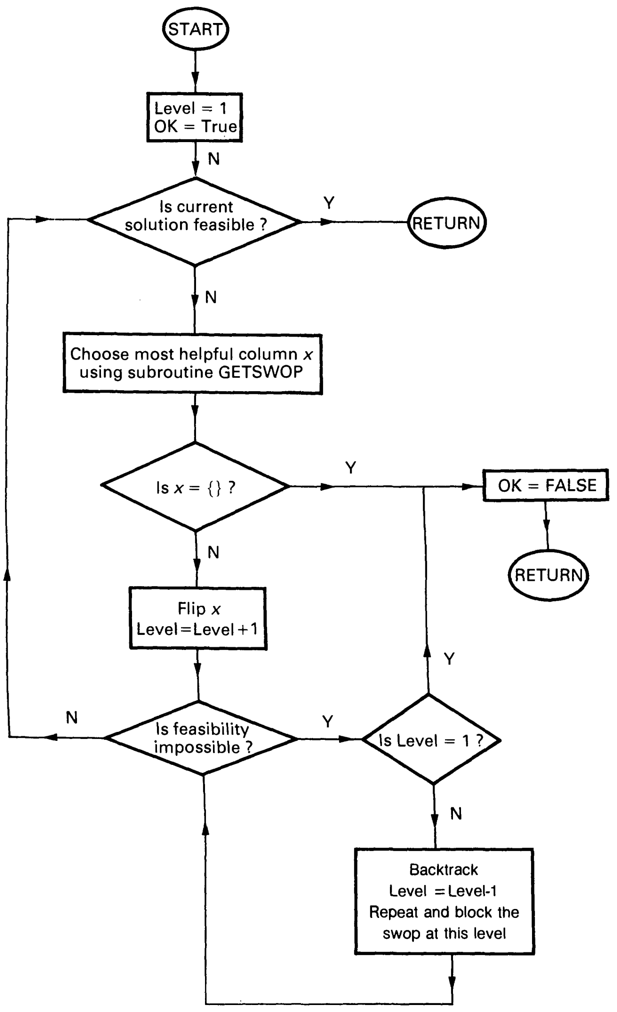
START
Level = 1
OK = True
N
Is current solution feasible ?
Y
RETURN
N
Choose most helpful column x using subroutine GETSWOP
Is $x = \{\}$?
Y
OK = FALSE
RETURN
N
Flip x
Level = Level + 1
Y
N
Is feasibility impossible ?
Y
Is Level = 1 ?
N
Backtrack
Level = Level-1
Repeat and block the swop at this level

FIG. 2. *Flow chart for subroutine RESTORE.*

in the objective function value. An improvement is always accepted but an uphill step is accepted only with probability $e^{-d/T}$. The user decides which variables can be used to initiate neighbourhood moves in this way, with the default being that every variable is neighbour-generating. Variables fixed at a particular value, either initially by the user or subsequently by the program deciding that the alternative is infeasible, are removed from the list of neighbour-generating variables. The list of these variables is 'shuffled' once at the start of an annealing run and thereafter scanned sequentially, corresponding to the 'shuffle-and-search' method developed in Connolly[3]. The author believes that such a systematic search is superior to the random choice of moves often suggested by others. When a column has been flipped to initiate a neighbourhood move, feasibility is restored by the subroutine RESTORE, (see Figure 2). At each stage RESTORE chooses the most 'helpful' column to flip from 0 to 1 or vice versa in order to move towards feasibility. If no such column exists then the algorithm backtracks by undoing the latest move and removing it from the list of possibilities at the level above. The process continues until the current solution is feasible or it becomes impossible to restore feasibility at the top level. In this latter case the neighbourhood move which initiated this sequence of moves is fixed at its original value and removed from the list of neighbour-generating columns.

Most of the subroutine RESTORE consists of simple book-keeping operations, maintaining lists of helpful columns, infeasible rows, forbidden moves and so forth. The only sophisticated part is the subroutine GETSWOP which determines the most helpful column to flip at each stage. This routine is illustrated in Figure 3.

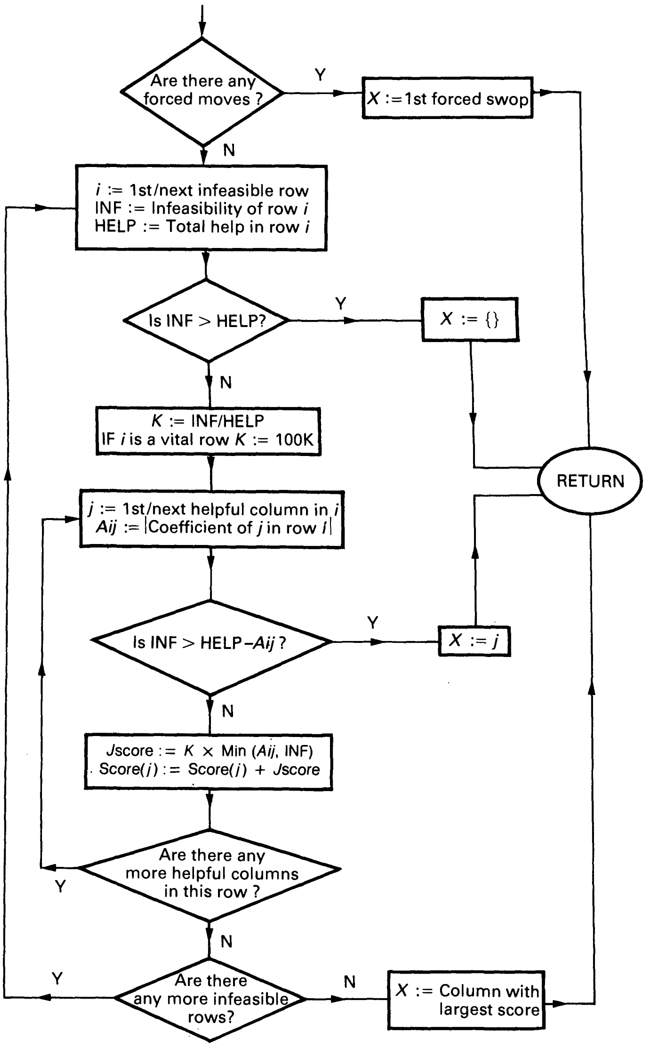
Are there any forced moves ?
Y
N
X := 1st forced swop
i := 1st/next infeasible row
INF := Infeasibility of row i
HELP := Total help in row i
Is INF > HELP?
X := {}
K := INF/HELP
IF i is a vital row K := 100K
RETURN
j := 1st/next helpful column in i
Aij := |Coefficient of j in row i|
Is INF > HELP-Aij ?
X := j
Jscore := K x Min (Aij, INF)
Score(j) := Score(j) + Jscore
Are there any more helpful columns in this row ?
Are there any more infeasible rows?
X := Column with largest score

FIG. 3. *Flow chart for subroutine GETSWOP(X).*

GETSWOP works roughly as follows: if no essential move has been detected, for example because it is vital to restoring feasibility of one of the rows, then GETSWOP proceeds to calculate a 'help-score' for each column. This score depends on how much the column helps the currently-infeasible rows and how critical these rows are. The criticality K_i of a row i is measured by the ratio of the size of the infeasibility to the help available in that row, i.e.

$$K_i = \mathrm{INF}_i/\mathrm{HELP}_i \text{ where}$$

$$\mathrm{INF}_i = \begin{cases} \mathrm{Max}(0, \mathrm{LHS}\text{-}\mathrm{RHS}) & \text{if constraint is '}\leqslant\text{'} \\ |\mathrm{RHS}\text{-}\mathrm{LHS}| & \text{if constraint is '='} \\ \mathrm{Max}(0, \mathrm{RHS}\text{-}\mathrm{LHS}) & \text{if constraint is '}\geqslant\text{'} \end{cases} \quad \text{and}$$

$$\mathrm{HELP}_i = \sum_{j \in H(i)} |a_{ij}|$$ where $H(i)$ is the set of columns which can be flipped to reduce the infeasibility of row i.

For example, consider the constraints

$$x_1 + x_2 + x_3 + x_4 + x_5 \geqslant 1 \quad \text{and}$$
$$x_1 \qquad\qquad\qquad + x_5 \geqslant 1$$

where all variables currently have the value zero. In both rows the infeasibility is 1, but the second is the more serious because there are fewer helpful columns available in it than in the first. A large criticality ratio, close to 1, indicates that almost all the helpful columns in this row are going to be needed, while a small value means that plenty of help is still available and this row should not have much influence on the choice of the next column to flip. A ratio greater than 1 implies that it is impossible to restore feasibility to this row and GETSWOP will immediately return the value $x = \{ \}$. Each helpful column j in row i then has its 'help-score' increased by an amount based on the criticality of i and the coefficient of j in i. When all infeasible rows have been examined GETSWOP returns the index of the column with the largest help-score. If two or more columns have the same maximum help-score then the one which will cause the least new infeasibility is chosen. If this test fails to separate them then the objective function values are examined to decide the issue. More will be said about these decision rules later.
(NB The user may initially specify a subset of the constraints as 'vital'. These are given a much higher criticality ratio so that they are restored to feasibility as quickly as possible.)

It should be stressed here that in order for the annealing technique to be successful a large number of feasible solutions must be examined. If feasibility is hard to achieve and/or large numbers of variables must be flipped to get from one solution to the next then the success of GPSIMAN is likely to be extremely limited. Here, attention is restricted to problems where feasibility is easily achieved and maintained and the difficulty is not in finding a feasible solution but in choosing the best from a very large number of them. Note also that to reduce storage and to improve the efficiency of GPSIMAN it was assumed that the coefficient matrix will consist entirely of integers. The test problems described below were chosen with these restrictions in mind.

TEST PROBLEMS

This section describes the test problems which were used to test the effectiveness of GPSIMAN. The ILP formulations of these problems are given in the Appendix.

QAP6 The Nugent *et al.*[5] ($n = 6$) Quadratic Assignment Problem. Assign n objects to n locations to minimize the cost of communication between them (see Connolly[3] for further details). Note that only 36 of the variables need be considered as integer when solving the ILP, and only these are used by GPSIMAN to initiate neighbourhood moves in GPSIMAN. The remainder will automatically take values 0 or 1 in the optimum ILP solution.

Number of constraints = 312
Number of variables = 336
Formulation QAP

QAP8R2 Nugent *et al.*[5] ($n = 8$) Quadratic Assignment Problem with the flow and distance matrices reduced to introduce as many zeros as possible to reduce the number of rows and columns required. (The idea of matrix reduction was suggested by Roucairol[6] and used as a technique for reducing the size of the ILP formulation of the QAP in Connolly[3]). Here only the 64 x-variables are specified as being integer in the ILP formulation and used to initiate neighbourhood moves.

Number of constraints = 458
Number of variables = 506
Formulation QAP

COLOR30 A graph-colouring problem on a random graph G with $N = 30$ vertices; i.e. find the minimum number of colours needed to colour the vertices of G such that no two adjacent vertices have the same colour. The example used here is for a graph with 30 vertices and 74 edges. Eight colours were available.

Number of constraints = 630

Number of variables = 248
Formulation COLOUR

DP16164 This is an extension of a brainteaser in which the aim was to place as many crosses as possible on a 5×5 grid, subject to the condition that no more than two crosses appeared in any horizontal, vertical or diagonal line. This was generalized to an $m \times n$ grid with no more than λ crosses in a line. The results given here are for $m = n = 16$ and $\lambda = 4$.

Number of constraints = 78
Number of variables = 256
Formulation DP

KS100 A 100-item knapsack problem with random 'weights' and profits, chosen uniformly from [1,2,..., 100]. The right-hand side of the single constraint = 1000.

Number of constraints = 1
Number of variables = 100
Formulation KNAPSACK

STELLA1 A real-life problem borrowed from a colleague[7] in which the object is to assign processes to processors to maximize the level of communication within each processor subject to various restrictions on permitted combinations.

Number of constraints = 698
Number of variables = 66
Formulation STELLA

STELLA2 As in STELLA1 above, using a different data set.

Number of constraints = 697
Number of variables = 66
Formulation STELLA

TT10R3 A small version of a seminar scheduling problem in which the object is to arrange four periods of seminars using three rooms to maximize the satisfaction of the first ten participants whose choices are as described by Eglese and Rand[8]. The formulation is as suggested by Eglese and Rand[8] except that we maximize satisfaction, rather than minimizing dis-satisfaction. These objectives are equivalent. Note that only 68 variables need to be treated as integer by SCICONIC. The remainder will automatically take values 0 or 1 in any optimal solution. All variables are treated as neighbour-generating in GPSIMAN.

Number of constraints = 282
Number of variables = 268
Formulation SEMINARS

TT20R5 As above, now satisfying the first 20 participants using five rooms. Again only 68 of the variables are treated as integers by SCICONIC.

Number of constraints = 492
Number of variables = 468
Formulation SEMINARS

RESULTS

The commercial ILP package SCICONIC (Version 1.4) was run on each of the nine test problems described above and the best score found in 1 CPU hour recorded. GPSIMAN was then run 20 times on each of the test problems and the best score, average score, success rate and average CPU time per start are recorded in Table 1. Both programs were run on a VAX 11/785. (NB A trial was

TABLE 1. *Comparison of SCICONIC and GPSIMAN Solutions and CPU times*

		SCICONIC		GPSIMAN (20 starts)				
Problem	Best known	Best found	CPU(secs)	Best score	No of opts	Average score	Success rate	CPU per start
QAP6	86↓	86	39.4	86	20	86.00	1.00	24.5
QAPRN8	214↓	214	2974.8	214	20	214.00	1.00	43.1
DP16164	64↑	64	200.2	64	20	64.00	1.00	62.7
KS100	2138↑	2138	57.4	2138	14	2136.85	0.70	13.0
COLOR30	4↓	4	1226.3	4	18	4.10	0.90	95.6
STEL1	132↑	132	1881.2	132	5	127.25	0.25	99.0
STEL2	157↑	146†	2978.6	157	9	149.45	0.75	95.6
TT10R3	1320↑*	1260†	2184.1	1320	9	1295.00	0.95	44.5
TT20R5	2760↑*	2730†	2033.2	2760	1	2711.00	0.30	111.1

↑ and ↓ denote maximization and minimization respectively
* denotes that best known solution may not be optimal
† denotes that optimum was not found in 1 CPU hour.

considered a success if it achieved a score at least as good as that found by SCICONIC.)

In all cases GPSIMAN regularly finds solutions at least as good as (and in the case of the last three problems, even better than) those found by SCICONIC, and yet never requires more than a few minutes running time per start. Figure 4 compares the expected CPU time required by GPSIMAN to achieve a success with that needed by SCICONIC and highlights the effectiveness of this simulated annealing approach. Note that the SCICONIC times are only those required to find these solutions. Proof of their optimality, (or otherwise), requires even more effort. In fact, only three out of the nine searches are actually completed in the allotted 1 hour CPU time; (QAP6, DP16164 and KS100). Note also that the optimum solutions found by GPSIMAN are often found quite early in the search so that the time quoted to perform the default number of iterations M, ($= 20 \times N$, where N is the number of neighbour-generating variables), is an over-estimate of the time taken to find these solutions. This is especially true of the smaller problems.

CONCLUSIONS

The results above show that GPSIMAN is an effective heuristic technique for quite a wide range of combinatorial optimization problems, certainly when compared with an ILP branch-and-bound package. It should be noted here that the problems SD-Scicon UK Ltd encounters in the 'real' world are not usually pure 0–1 problems, tending instead to be mixed integer linear programs (MIP) with a significant continuous component, and it is in tackling these that SCICONIC's real strengths lie. Since most of the logic used to restore feasibility at each iteration in GPSIMAN is based on the assumption that every variable must be either 0 or 1, it would require considerable extra effort to extend GPSIMAN to tackle MIPs and there is no guarantee that such problems could be tackled as effectively by such an approach. It should also be noted that special purpose algorithms either exist or could easily be constructed to solve any of the above problems much more effectively. For example, Quadratic Assignment problems much larger than $n = 8$ are tackled successfully in Connolly[2,3] and elsewhere, dynamic programming and other approaches easily solve large knapsack problems optimally, Eglese and Rand[8] use simulated annealing to tackle much larger versions of the seminar scheduling problem and graph-colouring heuristics abound. However, as was pointed out in the introduction, the aim here was to produce a general purpose algorithm which could solve a range of combinatorial optimization problems without *a priori* knowledge of the problem structure, (other than the ILP formulation). As such, GPSIMAN would seem to be a success.

It should be noted again here that all the test problems above satisfy the condition that attaining and maintaining feasibility is relatively easy, and that this condition is critical to the performance of GPSIMAN. For example, if, in the COLOR30 problem, the number of available colours is reduced from eight to five, then the running time rises from 96 seconds per start to 134 seconds, despite the reduction in the size of the model.

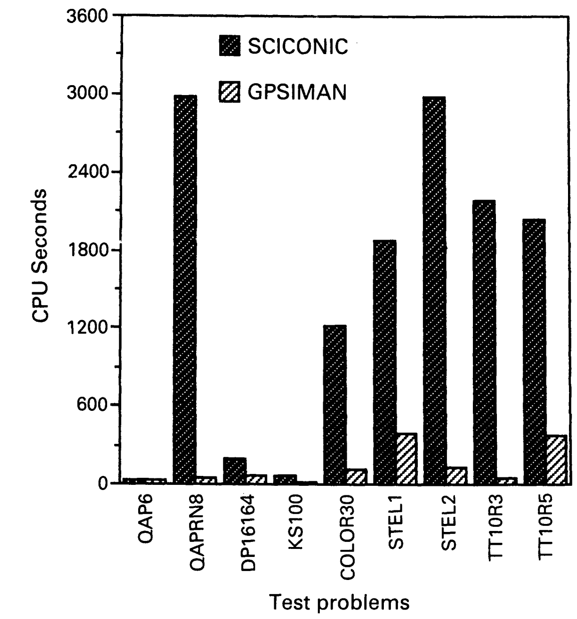

FIG. 4. *Comparison of expected CPU times.*

An even more drastic increase in running time occurs if only the minimum four colours are available for use. This is simply because GPSIMAN finds it harder to move from one solution to the next. Roughly speaking, the subroutine RESTORE should cope with problems where the sequence of moves needed to restore feasibility after flipping any variable is fairly obvious to the user. If no such obvious neighbourhoods exist then GPSIMAN is unlikely to perform well. One solution to this problem may be to use a penalty-function approach, whereby some or all of the constraints are relaxed by introducing artificial slack variables which incur a large penalty cost. The optimization process would then ensure that none of these artificial variables appeared in the optimum solution. Unfortunately the existing algorithm would probably perform poorly in these circumstances. Recall that, when restoring feasibility, GPSIMAN uses the helpfulness of each column as the main criterion. The objective function is used only to settle any ties that occur.

When artificial variables are present the current scheme would therefore simply use those to restore feasibility in the new relaxed problem and will tend to ignore the genuine variables which might help reduce the penalty costs. If this penalty-function approach is to be useful then the objective function must play a more significant role in the subroutine GETSWOP, so that artificial slack variables are not chosen so enthusiastically. However, care is required because attaching too much significance to the cost of a variable may lead to GPSIMAN missing the obvious route back to feasibility and lead to a long series of cheap but ineffectual moves. More work is planned by the author in an attempt to find a satisfactory trade-off between cost and 'helpfulness' in GETSWOP so that harder problems can be tackled successfully by GPSIMAN.

Another area of possible future research is to attempt to use GPSIMAN to aid a branch-and-bound search. Obviously a good heuristic solution will provide a cut-off which should help limit the search, as well as satisfying the impatient user who wants a good solution quickly. However, more sophisticated interaction is also possible. For example, promising branches of a search tree could be explored quickly, using the ability to fix variables in GPSIMAN. Alternatively, GPSIMAN could be allowed to roam freely and the distribution of the best solutions it finds used to aid the branching strategy, not least by improving the estimation of degradation likely to occur when forcing fractional solutions towards integrality. Another area where GPSIMAN could prove useful is in cutting plane approaches which often need good solutions to a variety of 0–1 ILPs which arise when determining the most effective cuts[9]. The results above suggest that in these and other ways GPSIMAN could prove to be a very useful addition to the arsenal of techniques available to practitioners wishing to tackle large scale 0–1 ILPs.

Acknowledgement—This research was jointly sponsored by SD-Scicon UK Ltd and the Department of Education of Northern Ireland under a CASE studentship agreement.

APPENDIX

ILP Formulations

This appendix contains the ILP formulations of the test problems used here. Copies of the actual data sets used are available from the author.

QAP

$$\text{Minimize} \sum_{i=1}^{n} \sum_{a=1}^{n} B_{ia}x_{ia} + \sum_{i=1}^{n} \sum_{a=1}^{n} \sum_{j=1}^{n} \sum_{b=1}^{n} C_{iajb}y_{iajb}$$

subject to:

$$\sum_{i=1}^{n} x_{ia} = 1 \qquad \forall a$$

$$\sum_{a=1}^{n} x_{ia} = 1 \qquad \forall i$$

$$x_{ia} + x_{jb} \leqslant y_{iajb} + 1 \quad \forall i < j,\ \forall a \neq b,\ C_{iajb} \neq 0 \quad (\dagger)$$

$$x_{ia} \in \{0, 1\} \qquad \forall i\ \forall a$$

$$0 \leqslant y_{iajb} \leqslant 1 \qquad \forall i\ \forall a\ \forall j > i\ \forall b \neq a$$

Variable x_{ia} will be 1 if object i is assigned to location a and this will incur the direct cost B_{ia}. If object j is assigned to location b then variable y_{iajb} must take the value 1, by constraint (†), and this will incur cost C_{iajb}, corresponding to the communication cost between objects i and j. Note that if C_{iajb} is equal to zero then the variable y_{iajb} and the corresponding constraint (†) defining it are dropped from the formulation. This is easily achieved using the 'NOT IF' facility in SD-Scicon UK Ltd's MGG matrix generator package. Note also that only the 36 x-variables need be specified as 0–1 variables when solving the ILP, and only they are used to initiate neighbourhood moves in GPSIMAN. The remainder will automatically take values 0 or 1.

COLOUR

$$\text{Minimize} \sum_{k=1}^{K} c_k \text{ subject to}$$

$$\sum_{k=1}^{K} x_{ik} = 1 \qquad i = 1, 2, \ldots N$$

$$(\dagger) \qquad x_{ik} + x_{jk} \leqslant 1 \qquad \forall k\ \forall i,j \text{ s.t. } \exists \text{ an edge } i \rightarrow j \text{ in } G$$

$$(*) \qquad \sum_{i=1}^{N} x_{ik} \leqslant Nc_k \qquad k = 1, 2, \ldots, K$$

$$c_k, x_{ik} \in \{0, 1\} \qquad \forall i\ \forall k$$

x_{ik} will be 1 if vertex i is given colour k. Constraints (†) ensure that no two adjacent vertices are given the same colour and constraints (*) ensure that c_k will be 1 if colour k is used for any of the vertices.

DP

$$\text{Maximize} \sum_{i=1}^{m} \sum_{j=1}^{n} \delta_{ij}$$

subject to

$$\sum_{(i,j)\in L_k} \delta_{ij} \leqslant \lambda$$

$k = 1, \ldots, 3m + 3n - 4\lambda - 2$ where L_k denotes the subset of variables corresponding to each of possible lines containing more than λ squares of the grid

$$\delta_{ij} \in \{0, 1\}\ \forall i\ \forall j$$

($\delta_{ij} = 1$ if there is a cross in row i and column j of the grid)

KNAPSACK

$$\text{Maximize} \sum_{i=1}^{N} P_i x_i$$

$$\text{subject to} \sum_{i=1}^{N} W_i x_i \leqslant W_{\max}$$

$$x_i \in \{0, 1\}\ \forall\ i$$

($x_i = 1$ if object i is included in the knapsack)

STELLA

$$\text{Maximize} \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} P_{ij}\, x_{ij}$$

subject to

$$\left.\begin{array}{ll} x_{ij} - x_{ik} + x_{jk} \leqslant 1 & 1 \leqslant i < j < k \leqslant N \\ x_{ij} - x_{ik} + x_{jk} \leqslant 1 & \quad '' \quad '' \quad '' \\ -x_{ij} + x_{ik} + x_{jk} \leqslant 1 & \quad '' \quad '' \quad '' \end{array}\right\} \qquad (\dagger)$$

$$\sum_{i=1}^{k-1} \alpha_i x_{ik} + \sum_{j=k+1}^{N} \alpha_j x_{kj} \leqslant 300 - \alpha_k \quad k = 1, \ldots, N \qquad (*)$$

$$\sum_{i=1}^{k-1} \beta_i x_{ik} + \sum_{j=k+1}^{N} \beta_j x_{kj} \leqslant 350 - \beta_k \quad k = 1, \ldots, N \qquad (\oplus)$$

$$\sum_{i=1}^{k-1} \gamma_i x_{ik} + \sum_{j=k+1}^{N} \gamma_j x_{kj} \leqslant 70 - \gamma_k \quad k = 1, \ldots, N \qquad (\otimes)$$

$$x_{ij} \in \{0, 1\} \qquad 1 \leqslant i < j \leqslant N$$

x_{ij} will be 1 if and only if process i is assigned to the same processor as process j. Constraints ($\dagger$) ensures that x_{ij}, x_{ik} and x_{jk} are logically consistent, while constraints (*), ($\oplus$) and ($\otimes$) are concerned with code storage, data storage and processor utilization restrictions respectively. (α_i, β_i and γ_i are data items corresponding to the storage requirements of process i. See Sofianopoulou[7] for further details.) Note that the N constraints regarding processor utilization referred to in Sofianopoulou[7] were actually multiplied by 100 to obtain the integer version given in ($\otimes$).

SEMINARS

Let x_{ijk} be 1 if and only if the ith person's jth period is spent at seminar k and let $y_{jk} = 1$ if and only if seminar k is held in period j. Assign a profit P_{ik} to person i attending seminar k. (Here $P_{ik} = 50, 40, 30, 20, 10$ according to whether the kth seminar was the ith person's 1st, 2nd, 3rd, or 4th reserve choice, as specified in Eglese and Rand[8]). Then the problem is as follows:

$$\text{Maximize} \sum_{i=1}^{N} \sum_{j=1}^{p} \sum_{k=1}^{s} P_{ik} \, x_{ijk}$$

where N is the number of participants, p is the number of periods and s is the number of seminars, subject to

$$\sum_{k=1}^{s} y_{jk} \leqslant R \; \forall j \text{ where } R \text{ is the number of rooms}$$

$$\sum_{i=1}^{N} x_{ijk} \leqslant N y_{jk} \forall j \; \forall k$$

$$\sum_{k=1}^{s} x_{ijk} = 1 \qquad \forall i \; \forall j \text{ and}$$

$$\sum_{j=1}^{p} x_{ijk} \leqslant 1 \qquad \forall i \; \forall k$$

$$0 \leqslant x_{ijk} \leqslant 1 \quad \forall i \; \forall j \; \forall k$$

$$y_{jk} \in \{0, 1\} \quad \forall j \; \forall k$$

Note that only the y need be specified as being 0–1 variables since the x will automatically take values 0 or 1 in any optimal solution.

REFERENCES

1. N. E. Collins, R. W. Eglese and B. L. Golden (1988) Simulated annealing—an annotated bibliography. *Am. J. Math. Mgmt Sci.* **8**, 209-308.
2. D. T. Connolly (1990) An improved simulated annealing scheme for the QAP. *Eur. J. Opl Res.* **46**, 93-100.
3. D. T. Connolly (1989) A comparison of solution techniques for the quadratic assignment problem and the development of a general purpose simulated annealing algorithm. PhD thesis, London School of Economics.
4. M. Lundy and A. Mees (1986) Convergence of an annealing algorithm. *Math. Prog.* **34**, 111-124.
5. C. E. Nugent, R. E. Vollman and J. Ruml (1968) An experimental comparison of techniques for the assignment of facilities to locations. *Opns Res.* **16**, 150-173.
6. C. Roucairol (1979) A branch-and-bound method for the QAP: the reduction method. Presented at the *3rd European Congress on OR*, Amsterdam 1979.
7. S. C. Sofianopoulou (1992) The process allocation problem: a survey of the application of graph-theoretic and integer programming approaches. *J. Opl Res. Soc.* **43**, 407-413.
8. R. W. Eglese and G. K. Rand (1987) Conference seminar timetabling. *J. Opl Res. Soc.* **38**, 591-598.
9. G. L. Nemhauser and L. A. Wolsey (1988) *Integer and Combinatorial Optimization.* Wiley, New York.