Journal of the Operational Research Society (1996) 47, 702–709 0160-5682/96 $12.00

A Genetic Algorithm for the Set Covering Problem

K. S. AL-SULTAN[1], M. F. HUSSAIN[2] and J. S. NIZAMI[3]

[1] Systems Engineering Department, [2] Data Processing Center and [3] Mechanical Engineering Department, King Fahd University of Petroleum and Minerals

In this paper, the set covering problem (SCP) is considered. Several algorithms have been suggested in the literature for solving it. We propose a new algorithm for solving the SCP which is based on the genetic technique. This algorithm has been implemented and tested on various standard and randomly generated test problems. Preliminary results are encouraging, and are better than the existing heuristics for the problem.

Key words: Chvatal's algorithm, genetic algorithm, implicit enumeration, Lagrangian heuristic, NP-complete problem, set-covering problem, 0–1 integer programs

INTRODUCTION

In this paper, we consider the set covering problem (SCP), which is the problem of covering the rows of a matrix A by a subset of its columns at minimum cost. The problem is a special class zero-one integer program, and is considered a classical NP-complete problem[1]. The SCP has applications in assembly line balancing[2], facilities location[3], information retrieval[4] and airline crew scheduling[5]. Other applications include bus driver scheduling[6], tanker routing and switching theory[7]. Further applications are given in refs 8–10.

In this paper, a new algorithm based on the genetic technique[11,12] is developed for the SCP. The algorithm is tested on several randomly generated problems of low sparsity, and standard test problems from the literature and its performance is compared with Chvatal's algorithm[13], the Lagrangian heuristic of Beasley, and the exact solution obtained by the implicit enumeration approach[14].

The paper is organized in the following way. First, the SCP is described. Then, a brief description of the genetic algorithm and its implementation to solve the set covering problem are summarized. We then present a statement, followed by a pseudo code of the new algorithm. Finally, we present the implementation of the new algorithm and a comparison with some of the available algorithms.

STATEMENT OF THE SET COVERING PROBLEM AND LITERATURE SURVEY

The SCP is a zero-one integer program which can be mathematically defined as:

$$\text{Minimize} \quad cx \tag{1}$$

$$\text{Subject to} \quad Ax \geqslant e \quad x \in \{0, 1\}^n, \tag{2}$$

where c is a real vector of length n, and A is a $(m \times n)$ matrix of zeros and ones, and e is an array, and $x = (x_j), j = 1, 2, \ldots, n$ where

$$x_j = \begin{cases} 1 & \text{if column } j \text{ is the solution} \\ 0 & \text{otherwise.} \end{cases}$$

The SCP is very cumbersome to solve, especially when the sparsity of the matrix A increases, since the set covering problem is NP-complete[1,15].

Various approaches have been suggested in the literature to solve the SCP. Some of the approaches are exact, and include branch and bound[16], cutting planes[17], a recent algorithm by

Correspondence: K. S. Al-Sultan, Systems Engineering Department, King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia

Beasley[18] and a dual heuristic approach[19]. Though these approaches solve the SCP optimally they take a lot of time and, therefore, they cannot be applied to large-scale problems if the solution is needed in real time. Other techniques are heuristic, and though they provide only approximate solutions, they are computationally efficient. This makes them better candidates for solving large-scale problems, as they obtain near optimal solutions in real time. Heuristics for the SCP can be found in refs 13 and 20–22. A variety of SCPs that are publicly available are detailed in refs 23 and 24.

THE GENETIC TECHNIQUE AND ITS IMPLEMENTATION

Genetic algorithms are basically search techniques. They emulate the natural process of evolution by progressing towards the optimum. These algorithms have been used extensively in solving various optimization problems[11,12].

In a genetic approach, at any given instant of time, a population of possible solutions is generated. Their number is arbitrarily chosen, but depends in some way on the nature and size of the problem. It may also depend on the memory size of the computer on which it is being implemented. Each element of the population is called a chromosome, which is a combination of symbols, known as genes. Chromosomes are possible solutions to the problem. In each generation, the best chromosomes are selected using the Roulette principle, to act as new parents (an explanation of the Roulette principle follows). Three genetic operators known as Reproduction, Crossover and Mutation are applied to these parents to generate new offspring. These new offspring inherit good qualities from the parents. The process of generating new offspring is continued until there is no further improvement in the offspring.

A detailed description of the Roulette principle and genetic operators for the SCP is given below.

THE ROULETTE PRINCIPLE

The Roulette principle is basically a process by which a good parent (solution) is assigned a higher probability of selection from a total solution pool than a bad parent. This is done by allocating weights to each solution in such a way that good solutions have a higher selection probability.

Implementaton of the Roulette principle to the set covering problem

Let us assume $X = (x_1, \ldots, x_t, \ldots, x_r)$ to be the pool of parent solutions, where x_t is the tth parent solution, and r is the size of the parent population. Let $\boldsymbol{C} = (c_1, \ldots, c_t, \ldots, c_r)$ be the cost vector, where c_t is the cost associated with the solution x_t, defined as

$$c_t = \begin{cases} cx_t & \text{if } x_t \text{ is feasible} \\ cx_t + M & \text{if } x_t \text{ is infeasible} \end{cases} \tag{3}$$

where M is the penalty imposed on any infeasible solution. A probability vector $\boldsymbol{P} = (p_1, \ldots, p_t, \ldots, p_r)$ is constructed as follows:

Assume, without loss of generality, that c is an ascending order (i.e. $c_i \leqslant c_j$, for $i \leqslant j$). Let

$$\Delta_i = \left(c_r + 1 + \frac{1}{r}\right) - c_i, \quad 1 \leqslant i \leqslant r$$

$$P_t = \frac{\sum_{k=1}^{t} \Delta^k}{\sum_{k=1}^{r} \Delta^k}, \quad 1 \leqslant t \leqslant r.$$

A random number $v \sim u(0, 1)$ is generated, where $u(0, 1)$ is the uniform distribution between 0 and 1. A corresponding cut s is obtained such that $p_{s-1} \leqslant v < p_s$, where $p_0 = 0$. This makes the solution x_s the most favourable parent.

REPRODUCTION

Reproduction is the main genetic process in which two parents are selected from the total parent solution pool using the Roulette principle. From these parents an offspring is generated which inherits qualities from both parents.

Implementation of reproduction to the set covering problem

Let $x = (x_i)$ and $y = (y_i)$ be two solutions, and let s be the cut generated such that $1 \leqslant s \leqslant n$. The reproduced solution is defined as $x^R = (x_i^R)$, where

$$x_i^R = \begin{cases} x_i & 1 \leqslant i \leqslant s \\ y_i \quad \text{when } x_i = y_{i-s} & s < i \leqslant n \\ y_{i-s} \quad \text{when } x_i \neq y_{i-s} & s < i \leqslant n. \end{cases} \tag{4}$$

CROSSOVER

The crossover operator usually operates on two genes of a parent solution, and generates an offspring. Since it is an inheritance mechanism, the offspring generated inherits some characteristics of the parent.

Implementation of crossover to the set covering problem

Let $x = (x_i)$ be the current solution. Generate two cuts s, t randomly such that $1 \leqslant s, t \leqslant n$, and $s \neq t$. The crossover solution is defined as $x^c = (x_i^c)$, where

$$x_i^c = \begin{cases} x_i & i \neq s, t \\ x_t & i = s \\ x_s & i = t. \end{cases} \tag{5}$$

MUTATION

Mutation normally brings random changes in a parent solution to generate an offspring. Mutation is very important, because crossover and reproduction by themselves do not always produce good offspring.

Implementation of mutation to the set covering problem

Let $x = (x_i)$ be the current solution. Generate randomly a cut s such that $1 \leqslant s \leqslant n$, the mutation solution is defined as $x^M = (x_i^M)$ where

$$x_i^M = \begin{cases} x_i & i \neq s \\ (x_i + 1) \bmod 2 & i = s. \end{cases} \tag{6}$$

APPROACH

Given a set covering problem, a random pool of parent solutions of size r is generated. The cost for each solution of the parent pool is calculated. An additional penalty is added to the cost of infeasible solutions as given in Equation (3). High probabilities are assigned to good solutions, using the Roulette principle. Offspring are then generated based on the genetic operators, reproduction, crossover and mutation. Note that the sizes of the parent pool and offspring pool are taken to be the same. The choice of genetic operator to be used is made through probabilistic methods, by associating a weight with each operator. It is known[11] that by varying these probabilities the convergence of the genetic algorithm is affected. The selection of parents to generate offspring is done via the Roulette principle.

After the generation of r offspring, the best r solutions (in terms of cost) from $(2r)$ parents and offspring are retained. These retained solutions are considered as new parents. The parents thus obtained are used to generate new offspring until there is no reduction in the objective function for a given number of iterations.

PSEUDOCODE OF THE PROPOSED SET COVERING ALGORITHM

Step 0: Initialization step

m = number of rows
n = number of columns
r = number of parent solutions
r = number of child solutions
M = penalty associated with the infeasible solution
spar = sparsity of the $\boldsymbol{A}$ matrix
p_1 = probability for reproduction
p_2 = probability for crossover
p_3 = probability for mutation (note: $p_1 + p_2 + p_3 = 1$)
lc = lower bound on cost coefficients
uc = upper bound on cost coefficients
$Niter$ = number of iterations for termination when there is no improvement
$term$ = logic variable for termination
ncs = counter for the child solution

Step 1: Generation step generate randomly

$\boldsymbol{A}$ = coefficient matrix with sparsity 'spar', with
$\boldsymbol{C}$ = cost vector using the bounds lc and uc
compute the cost for each parent

Step 2: Main algorithm

Do while term = false
$ncs = 1$
$Iter = 0$
Do while $ncs \neq r$
Choose $v \sim u(0, 1)$ where $u(0, 1)$ is a uniform random number between 0 and 1.

Scheme selection

If $0 \leqslant v < p_1$ then *scheme = reproduction*
$p_1 < v \leqslant p_1 + p_2$ then *scheme = crossover*
$p_1 + p_2 < v \leqslant 1$ then *scheme = mutation.*

Parent selection

```
If scheme = reproduction
        Do i = 1 to 2
        Call Roulette
        End Do
        x and y are selected
Else
        Call Roulette
        x is selected
End If
If scheme = reproduction
        Call reproduction (x, y, z)
Else If scheme = crossover
        Call crossover (x, z)
Else If scheme = mutation
        Call mutation (x, z)
```

End If
Offspring z is generated
$ncs = ncs + 1$
End (end while ncs)
Compute the cost of r offspring solutions
Choose the best r new parents (based on the cost) from the old and newly generated offspring solutions.
If $c(1) = c(r)$ then
$Iter = Iter + 1$
Else
Term = False
$Iter = 0$
End If
If $Iter = Niter$ then
Term = True
End If
End While.

RESULTS

The proposed genetic algorithm was implemented and tested on a 386 PC, running at 33 MHz speed with a coprocessor. It was applied to the following:
(1) randomly generated problems of sparsity 15% and cost vector with elements generated from a uniform distribution between 1 and 10;
(2) test problems obtained from electronic mail[24];
(3) randomly generated problems from the scheme proposed in Balas and Ho[17].

All these problems have been tested using the genetic algorithm, the Lagrangian heuristic of Beasley[20] and the implicit enumeration algorithm[16]. Examples of the results obtained for the randomly generated problems (set 1 above) are presented in Table 1. In this table, results for only one problem of each size are shown. However, 10 problems have actually been solved and results are not shown in the table to save space. Our genetic algorithm (GA) has obtained results that are

TABLE 1. *Comparison between the genetic algorithm (GA), Lagrangian heuristic (LH) and the implicit enumeration algorithm (IE) for random problems*

	Size		IE		LH		GA	
Problem	Rows	Columns	obj. fn.	Time (sec)	obj. fn.	Time (sec)	obj. fn.	Time (sec)
1.1	5	10	65	25.1[a]	67	6.2[a]	65	6.2[a]
2.1	5	20	87	27.2	87	6.9	87	6.9
3.1	10	20	86	28.4	86	7.3	86	7.3
4.1	10	30	92	31.2	92	8.9	92	8.9
5.1	20	30	94	35.6	94	9.6	94	9.5
6.1	20	40	121	37.8	122	10.2	121	8.6
7.1	20	50	146	41.3	147	11.1	146	9.3
8.1	30	40	118	46.5	118	11.1	118	10.2
9.1	30	50	141	57.3	141	11.8	141	9.8
10.1	30	60	171	64.4	171	12.1	171	11.2
11.1	40	50	143	58.9	143	14.6	143	13.3
12.1	40	60	187	67.8	187	16.0	187	15.4
13.1	40	70	213	73.2	214	18.2	213	18.2
14.1	50	60	165	71.1	165	18.6	165	18.5
15.1	50	70	280	74.5	280	18.9	280	18.7
16.1	50	80	245	75.2	245	20.1	245	19.6
17.1	50	90	265	76.9	265	20.6	265	20.5
18.1	50	100	287	79.3	287	21.0	287	21.0
19.1	60	70	198	78.9	198	22.5	198	22.5
20.1	60	80	223	79.6	224	22.9	223	22.7

[a] This time is the average for the 10 problems solved in this class.

better than the Lagrangian heuristic (LH) in 53 out of 200 problems (26.5%), and obtained results that are equal to those of the LH in the remaining 147 problems. Our algorithm is also slightly more efficient than the LH in terms of execution time. The GA's times range from 85% to 100% of the LH's times. In comparison with the exact implicit enumeration algorithm (IE), the GA obtained the exact results in 188 out of 200 problems (94%), and for the other 12 problems it obtained results that are close to the exact solution. It is also more efficient than the IE algorithm, and the GA's times range from 17.1% and 36.8% of the times of the IE algorithm.

TABLE 2. *Comparison between the genetic algorithm (GA), Lagrangian heuristic (LH) and the implicit enumeration algorithm (IE) for standard problems*

Problem	Size		IE	LH	GA
	Rows	Columns			
A.1	300	3000	253 (1)	255 (3)	254 (5)
			312 (2)	104 (4)	63 (6)
A.2	300	3000	252	256	252
			322	131	67
A.3	300	3000	232	234	234
			369	127	64
A.4	300	3000	234	235	234
			355	134	76
A.5	300	3000	236	237	237
			371	142	73
B.1	300	3000	69	70	70
			348	138	71
B.2	300	3000	76	77	76
			398	144	69
B.3	300	3000	80	80	80
			377	133	79
B.4	300	4000	79	80	80
			387	165	94
B.5	300	4000	72	72	72
			397	161	98
C.1	400	4000	227	230	227
			434	195	121
C.2	400	4000	219	223	222
			421	193	119
C.3	400	4000	243	251	243
			465	194	125
C.4	400	4000	219	224	219
			461	203	129
C.5	400	4000	215	217	216
			443	197	132
D.1	400	4000	60	61	60
			473	234	196
D.2	400	4000	66	68	68
			462	225	178
D.3	400	4000	72	75	75
			512	265	192
D.4	400	4000	62	64	63
			523	254	235
D.5	400	4000	61	62	62
			498	272	243
T1	1000	5000	[a]	64	63
				1059	931
T2	1000	8000	[a]	59	57
				2312	1978
T3	1000	12000	[a]	73	69
				3564	3121

(1) Objective function value obtained by the implicit enumeration algorithm.
(2) Time in seconds taken by implicit enumeration algorithm.
(3) Objective function value obtained by the Lagrangian heuristic.
(4) Time in seconds taken by the Lagrangian heuristic.
(5) Objective function value obtained by the genetic algorithm.
(6) Time in seconds taken by the genetic algorithm.
[a] Results could not be obtained.

Results for the standard test problems (set 2 above) are reported in Table 2. Clearly, the GA compares favourably with the LH both in terms of time and quality. In 11 out of 20 problems, the GA obtained results that are better than those obtained by the LH, while it obtained the same results as the LH in the remaining nine problems. Execution times of the GA are always less than those of the LH by a factor ranging from 48% to 92.5%.

Comparing the GA with IE algorithm, the GA obtained the exact solution in nine out of 20 problems. In the remaining 11 problems, the GA results are very close to the exact solutions, with the worst results obtained for problems D3 being only 4.2% from the optimal solution. The GA execution times are much less than those of IE, by a factor ranging from 17.5% to 48.8%.

Three large problems of set 3 above were tested, and the results are shown in Table 2 as problems T1, T2 and T3. It is clear from the table that the GA's results are better than those of the LH, both in terms of quality of the solution and in terms of time. The IE results of the algorithm could not be obtained because of the large size of the problems.

Finally, we compared the GA with Chvatal's algorithm[13]. For reasons of space limitations, the results of this comparison are not shown. However, the GA compares favourably with Chvatal's algorithm both in terms of time and quality.

Our results are preliminary only, and better results may be obtained by designing more efficient genetic operators.

CONCLUSIONS

In this study, a new algorithm based on the genetic technique has been developed for the set covering problem. The algorithm has been implemented and tested on several standard and randomly generated test problems. The results compare favourably with existing algorithms, both in terms of time and quality of the solution. They also show that the algorithm is a very robust and efficient approach for the set covering problem.

Acknowledgements—The authors are grateful to two referees whose comments have improved this paper and also wish to acknowledge the support provided by King Fahd University of Petroleum and Minerals in carrying out this research.

REFERENCES

1. M. R. Garey and D. S. Johnson (1979) *Computers and Interactability: A Guide to the Theory of NP-completeness.* Freeman, San Francisco, CA, USA.
2. M. E. Salverson (1955) The assembly line balancing problem. *J. Indus. Engng* **6**, 18–25.
3. W. Walker (1974) Applications of the set covering problem to the assignment of ladder trucks to fire houses. *Opns Res.* **22**, 275–277.
4. R. H. Day (1965) On optimal extracting from a multiple file data storage system: an application of integer programming. *Opns Res.* **13**, 482–494.
5. E. K. Baker, L. D. Bodin, W. F. Finnegan and R. J. Ponder (1976) Efficient heuristic solutions to an airline crew scheduling problem. *A.I.I.E. Trans* **11**, 79–85.
6. A. Paias and J. Paixao (1993) State space relaxation for set covering problem related to bus driver scheduling. *Eur. J. Opl Res.* **71**, 303–316.
7. W. V. Quine (1955) A way to simplify truth functions. *Am. Math. Monthly* **62**, 627–631.
8. E. Balas and M. W. Padberg (1979) Set partitioning – a survey. In *Combinatorial Optimization* (N. Christofides, A. Mingozzi, P. Toth and C. Sandi, Eds). pp 151–210. John Wiley, New York.
9. R. S. Granfinkel and G. Nemhauser (1972) *Integer Programming.* John Wiley, New York.
10. K. G. Murty (1976) *Linear and Combinatorial Programming.* John Wiley, New York.
11. D. Goldberg (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, MA.
12. D. Lawrence (Ed.) (1991) *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York.
13. V. Chvatal (1979) A greedy heuristic for the set covering problem. *Math. Opns Res.* **4**(3), 233–235.
14. H. A. Taha (1982) *Operations Research: An Introduction.* Third Edition. Macmillan, New York.
15. N. Christofides and S. Korman (1975) A computational survey of methods for the set covering problem. *Mgmt Sci.* **21**, 591–599.
16. C. E. Lemke, H. M. Salkin and K. Spielberg (1971) Set covering by single branch enumeration with linear programming subproblems. *Opns Res.* **19**, 988–1022.
17. E. Balas and A. Ho (1980) Set covering algorithms using cutting planes, heuristics and subgradient optimization: a computational study. *Math. Programming* **12**, 37–60.
18. J. E. Beasley (1987) An algorithm for the set covering problem. *Eur. J. Opl Res.* **31**, 85–93.
19. M. L. Fisher and D. Kedia (1990) Optimal solution of set covering/partitioning problems using dual heuristics. *Mgmt Sci.* **36**, 674–688.

20. J. E. Beasley (1990) A Lagrangian heuristic for the set covering problem. *Naval Res. Log.* **37**, 151–164.
21. J. E. Beasley and K. Jornsten (1992) Enhancing an algorithm for set covering problems. *Eur. J. Opl Res.* **58**, 293–300.
22. L. A. N. Lorena and F. Belo Lopes (1994) A surrogate heuristic for set covering problems. *Eur. J. Opl Res.* **79**, 138–150.
23. E. El-Darzi and G. Mitra (1990) Set covering and set partitioning: a collection of test problems. *Omega* **18**, 195–201.
24. J. E. Beasley (1990) OR-Library: distributing test problems by electronic mail. *J. Opl Res. Soc.* **41**, 1069–1072.

Received June 1994; accepted October 1995 after two revisions