A Genetic Algorithm Approach for Set Covering Problems

Wen-Chih Huang , Cheng-Yan Kao+ and Jorng-Tzong Horng*

Department of Computer Science and Information Engineering
National Taiwan University, Taipei, Taiwan
*National Central University, Chungli, Taiwan

+ All correspondences should be sent to the second author.

Abstract

In this paper, we introduce a genetic algorithm approach for set covering problems. Since the set covering problems are constrained optimization problems we utilize a new penalty function to handle the constraints. In addition, we propose a mutation operator which can approach the optima from **both sides** of feasible/infeasible borders. We experiment with our genetic algorithm to solve several instances of computationally difficult set covering problems that arise from computing the 1-width of the incidence matrix of Steiner triple systems. We have found better solutions than the currently best-known solutions for two large test problems.

1 Introduction

Set Covering Problems (SCPs) [5] are difficult zero-one optimization problems. They are often encountered in a wide area of applications such as resource allocation [12] and scheduling [1, 2].

Fulkerson et al. [5] have given two empirically difficult set covering problems arising from Steiner triple systems. Fulkerson et al. suggest that these are good problems for evaluating the computational efficiency of integer programming and set covering algorithms because they have far fewer variables than numerous solved problems in literature; however, experience shows that they are hard to compute and verify.

The β-width of a (0,1)-matrix A is the minimum number of columns that can be selected from A such that all row sums of the resulting submatrix of A are at least β. Here β is an integer parameter ranging from zero to the smallest row sum of the matrix A. The 1-width of A is:

$$\begin{aligned} w(A) &= \min e_n^T x \\ \text{subject to } Ax &\geq e_m , \qquad (1) \\ x &\in \{0,1\}^n, \end{aligned}$$

where e_n is an n-vector of ones and e_m an m-vector of ones. The 1-width is a set covering problem. The incidence matrix A that arises from Steiner triple systems has precisely 3 ones in each row. This matrix is also characterized as follow: for every pair of columns j and k there is exactly one row i for which $a_{ij} = a_{ik} = 1$. The incidence matrix of Steiner triple systems can be constructed by a recursive formulation [5].

Fulkerson, Nemhauser and Trotter [5] discuss computational experience with A_9, A_{15}, A_{27} and A_{45}. They are able to solve A_9 with a cutting plane code after generating 44 cuts, but this cutting plane method is unsuccessful on the three larger problems. In contrast, using an implicit enumeration algorithm similar to one developed by Geoffrion [6], they are able to solve A_{15} and by further inspecting the optimal solution, A_{27} can be solved in reasonable computer time. But several attempts at solving the problem A_{45} **fail**. Table 1 summarizes several instances of the Steiner triple systems, showing that the sizes of optimal covers are known for few of them.

2 Related Works

Karmarkar [8] gives an interior-point approach to solving 0-1 integer programming problems. Such problems, which are NP-complete in general, are converted to **nonconvex** quadratic programs over polytopes. He experiments with the Steiner triple systems using his approach in [9] and produces the best known covers for all test instances. Because of nonconvex programming, some **local optima** may be encountered during solution and the effect caused by the local optima becomes much more serious as the problem size is larger. This fact can be found from his result that his approach takes only about 6 minutes for problem A_{81} while over 56 hours and 226 hours for problems A_{135} and A_{243}

respectively.

Feo and Resende [4] pursue a non-deterministic method for solving the difficult Steiner triple systems. The procedure is based on Chvátal's iterative cost to benefit greedy approach [3]. In order to improve upon Chvátal's heuristic they introduce randomization. This heuristic also provides the best known solutions to all instances attempted in literature. However, our results are better than those of Feo and Resende.

Liepins et al. [10, 11] investigate genetic algorithms for set covering problems with two types of crossover operators in conjunction with three penalty functions and two multi-objective formulations. They do not experiment on the Steiner triple systems. Their results are encouraging and point to the *greedy crossover*, tight upper bounds for cost of completion of covers (as a penalty function $P3$), and Pareto based selection of the gene pool as promising techniques.

3 A GA for Set Covering Problems

Genetic Algorithms (GAs) are search and optimization algorithms based on the mechanics of natural selection and natural genetics. The genetic search proceeds over a number of generations. "Survival of the fittest" provides the pressure for populations to develop increasingly fit individuals. The primary monograph on the topic is Holland's [7] *Adaptation in Natural and Artificial Systems.* Having been established as a valid approach to problems requiring efficient and effective search, genetic algorithms are now finding more widespread applications in business, scientific, and engineering.

In order to apply GAs to a particular problem, we first need to select an internal string representation for the solution space. Set covering problems seem to have a highly desirable string representation, namely, binary strings of length N in which the j-th bit represents whether the j-th set P_j is in the cover or not. Further implementation of our approach is described below.

Selection Strategy

The purpose of selection in a genetic algorithm is to give more reproductive chances to those population members that are better fit. Our mechanism for selective pressure is the linear function described by Whitley [15].

$$\begin{aligned}\text{index} \;=\; & \mathit{POPULATION\_SIZE} \times \\ & (\mathit{bias} - \text{sqrt}(\mathit{bias} \times \mathit{bias} - 4.0(\mathit{bias} - 1) \times \text{random}())) \\ & /2.0/(\mathit{bias} - 1).\end{aligned} \tag{2}$$

where the function random() returns a random fraction between 0 and 1. A bias of 1.5 implies that the top ranked individual in the population is 1.5 times more likely to reproduce (on one reproductive cycle) than the median individual in the population.

Crossover Operator

Instead of Liepins' greedy crossover we adopt the uniform crossover operator proposed by Syswerda [14]. Syswerda suggests that the probability of 1's in the mask string be 0.5. But we find that probability of 0.6 results in better performance for our algorithms. This probability is fixed for all trials.

Penalty Function

Richardson et al. [13] suggest that the penalty function approach is the most suitable approach for constrained optimization problems such as set covering problems. Their illustration also convinces us that the infeasible solutions should provide information and not just be thrown away, especially the "next-door neighbors" of the optima in Hamming space. In view of this, we define a new penalty function $P3'$ as follows:

Let A be the incidence matrix of the original set covering problem with columns p_j and associated costs c_j.

$P3'$:

If S is a cover, then cost $= \sum_{j \in S} c_j$.

If S fails to be a cover, then

1. **Set S' to S. Set *total_cost* to 0.**
2. **Strike each column, say p^1, in S' and the rows covered by p^1 from A. Add the cost associated with p^1 to the *total_cost*. Let this new matrix be A' and set A to A'.**
3. **For the unused columns and uncovered rows, calculate the cost-ratios (cost / *number_of_rows_uncovered* $= c_j$ / *number_of_rows_uncovered*).**
4. **Append to S' the column, say p^2, with the least cost-ratio (break ties randomly). Add the cost associated with p^2 to the *total_cost*.**
5. **Strike column p^2 and the rows covered by p^2 from A. Let this new matrix be A'.**

6. If S' is a cover, return *total_cost* as the cost of S. Otherwise set A to A' and go to step 3.

We note that $P3'$ is more complex than $P3$ and proceeds in a column-by-column manner instead of the row-by-row manner of $P3$. This column-by-column manner can estimate the Hamming distance from the optimum more accurately. Thus $P3'$ is able to collect more information from the neighborhood of the optimal solutions.

Mutation Operator

Richardson et al. [13] not only establish some guidelines for penalty function, they also suggest that *good search should approach the optima from both sides of the feasible/infeasible border*. But they do not explain clearly how to *approach* the optima.

To achieve this, we propose a mutation operator. Our mutation operator first randomly selects a bit, say b_j, from a chromosome and then mutates b_j based on some mutation rate. Whether the selected bit is mutated or not also depends upon the feasibility of the chromosome to be applied on. Our mutation operator is explained below. Note that we assume each cost c_j associated with each column j is non-negative (≥ 0). Note that there are two mutation rates used with our mutation operator.

- If the chromosome is **feasible**, then
  - if $b_j = 1$ then mutate b_j based on *mutation_rate* = 0.85.
  - otherwise (i.e., $b_j = 0$), mutate b_j based on *mutation_rate* = 0.1.
- If the chromosome is **infeasible**, then
  - if $b_j = 0$ then mutate b_j based on *mutation_rate* = 0.85.
  - otherwise (i.e., $b_j = 1$), mutate b_j based on *mutation_rate* = 0.1.

Our genetic algorithms use the generational replacement. The best solution is copied into the next generation and replaces the worst solution. For all trials the population size is 80 and the crossover rate is 1.0. There are two generation sizes used, 250 and 400, depending upon the quality of the results.

4 Results and Comparisons

All empirical experiments are implemented in C and the tests are carried out on SUN SPARC Station 2. The C compiler is used to compile the codes with -O2 optimization level. The computational experiment tests the genetic algorithms on five set covering problems that arise from Steiner triple systems: A_{27}, A_{45}, A_{81}, A_{135} and A_{243}. These test problems are obtained in the same way as in [4] and [9].

We compare our genetic algorithm with the greedy genetics of Liepins et al. Their main implementation issues include **greedy crossover, penalty function** $P3$, and **no mutation**. Four genetic algorithms were investigated here to compare relative performance: (1) greedy_p3: greedy crossover with penalty function $P3$, (2) greedy_p3p: greedy crossover with penalty function $P3'$, (3) uniform_p3: uniform crossover with penalty function $P3$, and (4) uniform_p3p: uniform crossover with penalty function $P3'$. All do not use mutation. The selection strategy used is the linear selection mechanism instead of the Pareto based selection used by Leipins et al.

Table 2 to Table 5 summarize the computational results of the above four genetic formulations on problem A_{27} whose optimal cover size is 18. The first row of these tables (i.e., bias) is the *bias* value in equation (2) and takes the values 1.2, 1.4, 1.6, 1.8 and 2.0. The second row (i.e., stability) is the proportion of optima found; i.e., stability $\frac{m}{n}$ means that the program meets optima m times in n runs. The third row (i.e., least generations) dedicates the smallest number of generations when the best solution is found. The fourth row shows the least amount of time required to identify the best solution. Below each table there is also shown how many generations the corresponding formulation runs and the average total time it takes. From these we can compare the relative efficiency of all formulations.

From Table 2 to Table 5 we find that uniform crossover is about 10 times faster than greedy crossover and can produce more *stable* results than the greedy crossover; that is, uniform crossover can find optima more quickly and easily than greedy crossover. In fact the greedy crossover is a little better than the ill-performed Chvátal's heuristic mentioned in [4].

We add our proposed mutation into uniform_p3p, resulting in a more complex genetic algorithm—mut_uniform_p3p.

Table 6 summarizes the computational results of mut_uniform_p3p on problems A_{27} to A_{243}. The third column denotes the best cover size that mut_uniform_p3p finds corresponding to each bias value. Please notice that the fourth column (i.e., stability) is the proportion of cases when the optimal or best known covers described in Table 1 are found. The fifth column dedicates the smallest number of generations when the best cover is found. The sixth row shows the least amount of time in which the best solution is identified. Here we set generation size to 400 for problem A_{81} and 250 for the others.

One important fact found in Table 6 is that mut_uniform_p3p finds a cover size 104 for A_{135}. This cover size is better than the currently best known solution 105. Here the stability includes the cases when cover size is 104 or 105. But it's a pity that only one such case for 104 occurs in the 10 runs when bias is 1.6 or 1.8. Also note that in Table 6 mut_uniform_p3p finds a cover size 203 for A_{243}. This cover size is also better than the currently best known solution 204. The stability includes the cases when cover size is 203 or 204. But it's a pity that only one such case for 203 occurs in the 10 runs when bias is 1.8 or 2.0. Our best solutions for A_{135} and A_{243} are shown in the Appendix.

5 Discussions

$P3'$ does occasionally have difficulty finding solutions on smaller bias values. We have found several ignored rows (i.e., denoted by $\times$) in Table 6, indicating that using penalty function $P3'$ may produce a fittest (least-cost) solution which is **not** feasible at all. However we find that this occurrs only on smaller bias values and $P3'$ consistently finds lower cost solutions than $P3$. This fact suggests that more accurate estimates of the completion cost make better penalties.

In Table 6 we also find that when solving problem A_{27}, mut_uniform_p3p produces more stable results than uniform_p3p in Table 5. That is, our proposed mutation operation can increase the stability of uniform_p3p. It is the same for larger problems.

From Table 6 we discover that **mut_uniform_p3p found cover sizes which are smaller than the currently best known sizes for both problems A_{135} and A_{243}.** Compared with the results of Karmarkar's interior-point approach, our genetic algorithm is not as seriously affected by the local optima. The quality of solution found is highly related to the *bias* value. For example, when solving problem A_{243}, it takes over 8 hours to find cover size 204 on bias 1.6 while less than 4 hours to find cover size 203 on bias 1.8 and 2.0.

6 Conclusions

In this paper, we have introduced a genetic algorithm approach for set covering problems. We have described in detail the procedure for generating a new penalty function which generates better results. In addition we demonstrate that uniform crossover is more efficient than the greedy crossover. We also propose a mutation operator that accelerates the convergence to the optimal solutions.

To illustrate the effectiveness of our resulting algorithm, mut_uniform_p3p, we apply it to solve several instances of computationally difficult set covering problems. We have found optimal covers for two instances, A_{27} and A_{45}, with known optimal solutions and the best known covers for instances varying in size from 81 variables and 1080 constraints to 243 variables and 9801 constraints, i.e., A_{81}, A_{135} and A_{243}, while taking much less time than the Karmarkar approach experimented on the same set of test problems. In addition our genetic algorithm can find better solutions than the currently best known solutions for the two larger problems. Earlier best approaches, including mathematical and stochastic algorithms, for these two problems were always trapped into a worse local optimum. This means that the ability of *exploration and exploitation* of genetic algorithms is much better than those of traditional optimization algorithms.

Appendix: Our Best Solutions for A_{135} and A_{243}

Best Solution of A_{135}:

$x_j = 0$, if $j \in \{$ 3, 13, 14, 17, 23, 28, 34, 36, 48, 52, 53, 62, 66, 68, 70, 75, 76, 78, 80, 81, 94, 97, 101, 104, 105, 106, 118, 120, 121, 125, 134 $\}$, optimal cover size=104.

Best Solution of A_{243}:

$x_j = 0$, if $j \in \{$ 5, 7, 9, 12, 13, 39, 41, 48, 52, 59, 60, 63, 75, 79, 81, 89, 93, 99, 101, 102, 111, 119, 126, 129, 136, 143, 155, 194, 196, 200, 203, 209, 221, 223, 225, 227, 234, 235, 239, 240 $\}$, optimal cover size=203.

References

[1] Arabeyre, J. P., Fearnley, J., Steiger, F. C. and Teather, W., "The airline crew scheduling problem: a survey", *Transportation Science* 3, pp.140-168, 1969.

[2] Aubin, J., "Scheduling Ambulances", *Interfaces* 22, pp.1-10, 1992.

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

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

[5] Fulkerson, D. R., Nemhauser, G. L., and Trotter, Jr., L. E., "Two computationally difficult set covering problems that arise in computing the 1-width of incidence matrices of Steiner triple systems", *Mathematical Programming Study* 2, pp.72-81, 1974.

[6] Geoffrion, A. M., "An improved implicit enumeration approach for integer programming", *Operations Research* 17, pp.437-454, 1969.

[7] Holland, J. H., *Adaption in Natural and Artificial Systems*, Ann Arbor: The University of Michigan Press, 1975.

[8] Karmarkar, N., "An interior-point approach to NP-complete Problems—Part I", *Contemporary Mathematics* 114, Lagarias, J. C. and Todd, M. J., editors, American Mathematical Society, pp.297-308, 1990.

[9] Karmarkar, N., Resende, M. G. C. and Ramakrishnan, K. G., "An interior point algorithm to solve computationally difficult set covering problems", *Mathematical Programming* 52, pp.597-618, 1991.

[10] Liepins, G. E., Hilliard, M. R., Palmer, M. and Morrow, M., "Greedy genetics", in Grefenstette J. J., editor, *Genetic Algorithms and Their Applications: Proceedings of the 2nd International Conference on Genetic Algorithms*, July 1987.

[11] Liepins, G. E., Hilliard, M. R., Richardson, J. and Palmer, M., "Genetic algorithms applications to set covereing and traveling salesman problems", in Brown (ed), *Operations Research and Artificial Intelligence: The Integration of Problem-Solving Strategies*, Kluwer Academic Publishers, 1990.

[12] Revelle, C. D., Marks, D. and Liebman, J. C., "An analysis of private and public sector facilities location models", *Management Science* 16, pp.692-707, 1970.

[13] Richardson, J. T., Palmer, M. R., Liepins, G. and Hilliard, M., "Some guidelines for genetic algorithms with penalty functions", in Schaffer D., editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, June 1989.

[14] Syswerda, G., "Uniform crossover in genetic algorithms", in Schaffer D., editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, June 1989.

[15] Whitley, D., "The *GENITOR* algorithm and selection pressure: why rank-based allocation of reproductive trials is best", in Schaffer D., editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, June 1989.

Table 1: Several Instances of Steiner Triple Systems

Problem	variables/constraints	Best known cover	Optimal?
A_{27}	27/116	18	yes
A_{45}	45/330	30	yes
A_{81}	81/1080	61	unknown
A_{135}	135/3015	105	unknown
A_{243}	243/9801	204	unknown

Table 2: Result of greedy_p3: A_{27}

bias	1.2	1.4	1.6	1.8	2.0
stability	$\frac{0}{15}$	$\frac{0}{15}$	$\frac{3}{15}$	$\frac{0}{15}$	$\frac{1}{15}$
least generations	×	×	1	×	1
time to find cover	×	×	1.01s	×	1.01s

250 generations $\doteq$ 253 seconds

Table 3: Result of greedy_p3p: A_{27}

bias	1.2	1.4	1.6	1.8	2.0
stability	$\frac{1}{15}$	$\frac{0}{15}$	$\frac{1}{15}$	$\frac{1}{15}$	$\frac{1}{15}$
least generations	195	×	1	1	1
time to find cover	3m 58.03s	×	1.22s	1.01s	1.01s

250 generations $\doteq$ 278 seconds

Table 4: Result of uniform_p3: A_{27}

bias	1.2	1.4	1.6	1.8	2.0
stability	$\frac{10}{15}$	$\frac{13}{15}$	$\frac{10}{15}$	$\frac{9}{15}$	$\frac{12}{15}$
least generations	7	3	4	9	4
time to find cover	0.70s	0.30s	0.40s	0.91s	0.40s

250 generations $\doteq$ 25 seconds

Table 5: Result of uniform_p3p: A_{27}

bias	1.2	1.4	1.6	1.8	2.0
stability	$\frac{9}{15}$	$\frac{8}{15}$	$\frac{12}{15}$	$\frac{13}{15}$	$\frac{13}{15}$
least generations	6	1	13	10	8
time to find cover	0.66s	0.10s	1.32s	1.00s	0.80s

250 generations $\doteq$ 27 seconds

Table 6: Summary of results of mut_uniform_p3p

Problem	bias	size of best cover found	stability	least generation to find best cover	least time to find best cover
A_{27}	1.2	18	$\frac{10}{10}$	2	0.24s
	1.4	18	$\frac{10}{10}$	3	0.47s
	1.6	18	$\frac{10}{10}$	4	0.62s
	1.8	18	$\frac{10}{10}$	10	1.29s
	2.0	18	$\frac{10}{10}$	15	1.84s
A_{45}	1.2	30	$\frac{1}{10}$	124	71.89s
	1.4	30	$\frac{6}{10}$	91	51.08s
	1.6	30	$\frac{2}{10}$	60	33.09s
	1.8	30	$\frac{6}{10}$	46	24.82s
	2.0	30	$\frac{2}{10}$	39	20.61s
A_{81}	1.2	61	$\frac{1}{15}$	273	16m 33.47s
	1.4	61	$\frac{9}{15}$	142	8m 8.68s
	1.6	61	$\frac{3}{15}$	83	4m 38.27s
	1.8	61	$\frac{2}{15}$	63	3m 26.61s
	2.0	61	$\frac{3}{15}$	73	4m 13.40s
A_{135}	1.2	×	×	×	×
	1.4	105	$\frac{1}{10}$	239	1h 34m 41.73s
	1.6	**104**	$\frac{6}{10}$	144	51m 39.08s
	1.8	**104**	$\frac{4}{10}$	126	49m 47.11s
	2.0	105	$\frac{6}{10}$	57	20m 7.73s
A_{243}	1.2	×	×	×	×
	1.4	205	$\frac{0}{10}$	202	
	1.6	204	$\frac{2}{10}$	194	8h 17m 31.63s
	1.8	**203**	$\frac{6}{10}$	93	3h 55m 26.47s
	2.0	**203**	$\frac{4}{10}$	89	3h 29m 14.57s