Applying Genetic Algorithm to Set Cover Problem

Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, NY, U.S.A.

November 23, 1998

1 Introduction

The Set Covering Problem (SCP) is NP-complete as it can be transformed to the vertex cover problem [1]. SCP has various application areas such as assigning cabin or cockpit crews to flight legs [2], fire companies to fire houses [3], etc.

Bit string representation has been the common preferred chromosome coding for SCP in the studies so far, since it is suitable for application of generalized operators and techniques. The fitness function for the SCP includes many local optima since a relatively small number of ones are distributed over very long bit strings. Feasible solution regions are separated by a large Hamming Distance, and the objective function values of local optima might differ drastically. This fitness landscape unfortunately traps GA in the attraction points very quickly and a mutation operator is generally unable to push it to another region.

Liepins [4, 5] compared two algorithms: one is based on *VEGA* and the other is based on Pareto-optimality. In the VEGA model, one objective is to minimize the cost while the other objective is to minimize the infeasibility of a solution. In the Pareto model, Goldberg's ranking is used and tried to minimize a combination of cost and infeasibility. Liepins compared 1PX, 2PX, and a greedy crossover operator. Greedy crossover requires the Chavatal's [6] greedy choice on a column set obtained from parents in the crossover step. He reported a slightly better performance in favor of the greedy crossover operator.

Hernandez [7] proposed a divide-and-conquer approach with GA. In his formulation, the problem is broken down into several non-overlapping subproblems, and for each subproblem, a population of solutions is produced using a GA. The fitness value of a chromosome in a subpopulation is evaluated according to its ability to produce a feasible solution when combined with other sub-solutions from other populations. He reported that better performance is obtained compared to other GA approaches.

Back [8] applied an additive penalty term to punish the infeasible solutions. The penalty term is calculated by estimating the total minimum cost to transform an infeasible solution to a feasible one based on greedy choice. He compared 1PX, 2PX, uniform crossover, and fusion crossover [9]. Fusion crossover takes genes from parents with probability proportional to parent's fitness. He reported that without repair (with only penalty functions), GA could not produce comparable results compared to GA meanwhile GA with repairing sometimes beats the greedy solutions. He also reported that the selection of any crossover operator did not affect the final results at all.

Caprara *et al.* [10, 11] describes their CFT algorithm.

1.1 Greedy Algorithm for SCP

This algorithm repeatedly selects columns that cover uncovered nodes with a minimum cost:

- while (there is an uncovered row (flight) in the solution set)
  - pick a column whose *cost_ratio* is minimum and add this to the solution set.

This definition can be stated in various ways. Let U denotes a set of uncovered rows in the solution, C denotes a set of covered rows by column S, then $cost\_ratio(S)$ can be defined as follows.

- Chavatal [6] Definition :

$$cost\_ratio(S) = \frac{cost(S)}{|C|}$$

- Back [8] Definition :

$$cost\_ratio(S) = \frac{cost(S)}{|U \cap C|}$$

  This calculation requires $O(n \cdot m)$ time where n and m denote the number of rows and columns, respectively.

- A problem requiring the SPP [12] model can be relaxed by using the SCP model. SCP is easier than SPP in terms of searching and repairing operations. To obtain better a solution to SPP with SCP relaxation, the issue of over covering should be handled carefully. A solution should attempt to minimize cost and number of over coverings. Therefore

$$cost\_ratio(S) = \frac{cost(S)}{|U \cap C|} \times (|\overline{U} \cap C| + 1)$$

  expresses this objective.

2 Set Cover Problem with GA

A set representation is preferred for chromosomes unlike previous studies [5, 13, 7, 8, 9]. The set contains indices of columns in the current solution. This representation does not require a huge memory because the number of elements in this set will be no more than the number of rows in the SCP.

The Hamming Distance (HD) between two chromosomes, say c_1 and c_2, can be defined as follows:

$$d_H(c_1, c_2) = |P_1 - P_2| + |P_2 - P_1|$$

where P_i denotes the set of column indices in chromosome c_i. Normalized HD can be given as:

$$d_H^N(c_1, c_2) = \frac{d_H(c_1, c_2)}{2 \times NumberOfRows}$$

so that $0 \leq d_H^N(c_1, c_2) \leq 1.0$.

2.1 Initialization

1. Let c[i] denote whether i^{th} row is covered or not. If $c[i] = 1$, then row i is covered. If $c[i] = 0$, then row i is not covered. Let m denote the number of rows. Let *Solution* denote the set of column indices.

2. Let $k = 1, ..., m$

   (a) If $c[i] = 0$, then

      i. Pick a column (c_k) covering row k randomly.

      ii. Add column (c_k) to the *Solution*.

      iii. Update c according to rows covered by column c_k.

Another initialization approach is to start from a random point in the range [1,m] instead of using the same loop and find the solution with random choice.

2.2 Crossover Operation

Let's assume that two parents, P_1 and P_2, will undergo crossover operation:

- $p_1 = \frac{cost(P_1)}{(cost(P_1)+cost(P_2))}$

- $p_2 = \frac{cost(P_2)}{(cost(P_1)+cost(P_2))}$

- $P1\_P2 = P_1 - P_2$

- $P2\_P1 = P_2 - P_1$

- $Solution = P_1 \cap P_2$

- Build *set* by putting columns from $P1\_P2$ with probability p_2 and from $P2\_P1$ with probability p_1.

- while (*not all rows are covered*)

  - randomly pick a column (c) from *set*
  - add c to *Solution*

2.3 Greedy Choice Crossover Operation

- $Solution = P_1 \cap P_2$

- Restore the *Solution* with greedy choice for each uncovered node.

2.4 Random Choice Crossover Operation

- $Solution = P_1 \cap P_2$

- Restore the *Solution* with random choice for each uncovered node.

2.5 Distance Preserving Crossover Operation

Distance Preserving Crossover (DPX) operation produces an offspring such a way that it has equally distant from its both parents. Let's say a crossover operation applied on chromosomes c_1 and c_2, and an offspring (o) is obtained. Then

$$d(c_1, o) = d(c_2, o)$$

should hold for these chromosomes. Since the proposed chromosome structure does not require a fixed length, a modified definition can be used to express the same concept such as.

$$|P_1 \cap P_o| = |P_2 \cap P_o|$$

where P_1, P_2, and P_o denote the set of column indices in chromosomes c_1,c_2 , and o, respectively. Based on DPX concept, two variants of the previously described crossover operation can be given and both of them assure that an offspring has common genes with its parents.

2.5.1 Random DPX Operation

- $Solution = P_1 \cap P_2$
- For each uncovered node (r) use a random column (c) such a way that $c \notin P_1$ and $c \notin P_2$.

2.5.2 Greedy Choice DPX Operation

- $Solution = P_1 \cap P_2$
- For each uncovered node (r) use a greedy choice (c) such that $c \notin P_1$ and $c \notin P_2$.

2.6 Mutation Operation #1

- Pick a random column and delete it from a solution
- Start from a random column index, and walk through all the columns until all rows are covered.

2.7 Mutation Operation #2

- Pick a random column and delete it from a solution
- Fill up the rest of uncovered nodes with greedy choice.

2.8 Mutation Operation #3

Deletion with weighting aims to get rid of the columns that causes overcoverings in the solution. Weight array denotes which rows are covered by how many columns therefore it can apply a pressure to decrease the number of columns in the solution. Selection is done according to these weights so that the probability of deleting a column causing overcovering is increased.

1. Let $w[i]$ denote the number of columns which covers row i.
2. For all i, let $w[i] = w[i] - 1$.
3. Let $sumw = \sum_{i=1}^{m} w[i]$.
4. Generate a random number (r) in range $[1, sumw]$.
5. Let $k = 0$, $sum = w[0]$.
6. $while(\ sum > r\ )\ \{\ k++; sum += w[k];\ \}$
7. Find indices of columns that cover row k.
8. Pick one of them (c_p) randomly and delete.
9. restore the solution by abandoning the deleted column (c_p).

2.9 Fitness Value

There are several different fitness function definitions to minimize several objectives.

2.10 Fitness Value for minimizing cost

$$FitnessValue(S_i) = \frac{1}{Cost_i}$$

where

$$Cost_i = \sum_{c \in S_i} cost(c)$$

2.11 Fitness Value for minimizing cost and number of columns

$$FitnessValue(S_i) = \frac{1}{|S_i| \times Cost_i}$$

where

$$Cost_i = \sum_{c \in S_i} cost(c)$$

2.12 Fitness Value for minimizing cost and number of overlaps

Let $w[i]$ denote the number of columns covering row i. Then the number of overcoverings for the feasible solution (that covers all rows): $NumberOfOvercoverings = \sum_{i=1}^{m}(w[i]-1)$

where m denotes number of rows. Then the overcovering ratio is:

$ratio = \frac{NumberOfOvercoverings}{m}$

$$FitnessValue(S_i) = \frac{1}{(ratio+1)\times Cost_i}$$

where

$$Cost_i = \sum_{c\in S_i} cost(c)$$

Obviously, $FitnessValue(S_i)$ will produce very small values but fitness scaling techniques can be applied to solve this problem. Given fitness function tries to minimize the over covering of rows and the total cost of the solution at the same time.

3 Results

The tested GA can be described as follows:

- Calculates diversity according to HD in the population and randomly initialize the population when diversity measure drops below the threshold (0.1 is used).
- Mutation randomly deletes a column and repairs by introducing new columns ($p_m = 0.2$).
- Distance Preserving Greedy3 Restore Crossover is used ($p_c = 0.6$).
- Steady State Genetic Algorithm (SSGA) is used with 20 population size and 100 generation.

Data Set	Number of Columns	Data Set	Number of Columns
d74_20	200	d74g_20	74
d102_20	931	d102g_20	204
d133_20	2714	d133g_20	266
d217_20	5575	d217g_20	434

Table 1: dX_Y denotes that it has X number of flights and columns are generated such a way that none of them exceeds Y hours.

References

[1] A.V. Aho, J.E. Hopcoft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison Wesley, 1974.

[2] J. Rubin. A technique for the solution of massive set covering problems, with application to airline crew scheduling. *Transportation Science*, (7):34–48, 1973.

[3] W. Walker. Using the Set-Covering Problem to Assign Fire Companies to Fire Houses. *Operations Research*, 22(22):275–277, 1974.

[4] D. E. Brown and C.C. White. *Operations Research and Artificial Intelligence: The Integration of Problem-Solving Strategies.* Kluwer Academic Publishers, Boston, 1990.

[5] G. Liepins, M. Hillard, J.T. Richardson, and M.R. Palmer. *Genetic Algorithms Application to Set Covering and Traveling Salesman Problems*, chapter in [4], pages 29–57.

[6] V. Chavatal. A Greedy Heuristic for the Set Covering Problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[7] L.F.G. Hernandez. Evolutionary Divide and Conquer for the Set-Covering Problem. Technical report, 1995.

[8] T. Back, M. Schutz, and S. Khuri. A Comparative Study of a Penalty Functions, a Repair Heuristic, and Stochastic Operators with the Set-Covering Problem. In J.-M. Alliot, E. Lutton, E. Ronald, M. Schoenauer, and D. Snyers, editors, *Artificial Evolution, European Conference, AE 95*, pages 320–332. Springer-Verlag, 1995.

[9] J.E. Beasley and P.C. Chu. A Genetic Algorithm for the Set Covering Problem. *European Journal of Operational Research*, 94:392–404, 1996.

[10] A. Caprara, M. Fischetti, and P. Toth. A heuristic method for the set covering problem. Technical Report DEIS OR-95-8, University of Bologna, Italy, Available at http://www-or.deis.unibo.it/abstract.html, 1995.

[11] A. Caprara, M. Fischetti, and P. Toth. Algorithms for the set covering problem. Technical Report DEIS OR-98-3, University of Bologna, Italy, Available at http://www-or.deis.unibo.it/abstract.html, 1998.

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

[13] D. Levine. A Genetic Algorithm for the Set Partition Problem. In S. Forrest, editor, *Proc. of Fifth International Conference on Genetic Algorithms*, pages 481–487, San Mateo , California, 1993. Morgan Kaufmann.

[14] P.C. Chu and J.E. Beasley. Constraint Handling in Genetic Algorithms: the Set Partitioning Problem. Technical report, Imperial College, UK, 1997.

	Chavatal [6]	Back [8]	GA#1 [14]	Greedy#3
Cost	13016	12826	20516	12826
Number of Columns	34	33	24	33
Number of Overlaps	10	6	0	2
Time (secs)	< 1	8	1504	14

Table 2: Results for d74_20 data set.

	Chavatal [6]	Back [8]	GA#1 [14]	Greedy#3
Cost	22131	18285	20701	18285
Number of Columns	63	49	44	49
Number of Overlaps	43	11	16	11
Time (secs)	2	89	4020	162

Table 3: Results for d102_20 data set.

	Chavatal [6]	Back [8]	GA#1 [14]	Greedy#3
Cost	32349	24378	34186	24566
Number of Columns	89	64	52	64
Number of Overlaps	90	30	78	19
Time (secs)	5	537	10624	968

Table 4: Results for d133_20 data set.

	Chavatal [6]	Back [8]	GA#1 [14]	Greedy#3
Cost	58320	47482	85912	47705
Number of Columns	149	114	138	116
Number of Overlaps	121	45	264	42
Time (secs)	15	2422	31520	4676

Table 5: Results for d217_20 data set.