International Journal of Electronic Commerce Studies
Vol.4, No.1, pp. 33-46, 2013
doi: 10.7903/ijecs.1138

GENERATIONAL MODEL GENETIC ALGORITHM FOR REAL WORLD SET PARTITIONING PROBLEMS

Chi-san Althon Lin
Bay of Plenty Polytechnic, Tauranga, New Zealand
Private Bag 12001, 200 Cameron Road, Tauranga 3134
althon.lin@boppoly.ac.nz

ABSTRACT

This paper proposes a generational model genetic algorithm-based system for solving real-world large scale set partitioning problems (SPP). The SPP is an important combinatorial optimization and has many applications like airline crew scheduling. Two improved genetic algorithm (GA) components are introduced and applied to the generational model GA system that can effectively find feasible solutions for difficult and large scale set partitioning problems. The two components are the grouping crossover operator and a modified local optimizer. The experimental results in this research show that the performance of this GA based system is capable of producing optimal or near-optimal solutions for large scale instances of SPP.

Keywords: Combinatorial Optimization, Set Partitioning Problem, Genetic Algorithm, Crew Scheduling, Grouping Crossover

1. INTRODUCTION

Set Partitioning Problem (SPP) is to find a good partition of a set. In addition, some constraints like fixed or limited number of subset combinations and cost functions are pre-defined for different aims. A feasible solution to SPP "must comply with various hard constraints" and "the objective of the grouping to optimize the cost function defined over the set of all valid groupings" [1]. One of the well-known applications of the SPP is airline crew scheduling[2, 3].

Set partitioning problem is a NP-complete combinatorial problem[4] and has been widely studied and researched in the past[1,5, 6, 7, 8, 9, 10, 11, 12]. Among various methodologies for solving SPP, Genetic algorithms (GA) is one of

the popular approach developed by J. Holland[8] due to its comprehensive capability to search a large solution space[7].

Among GA based approaches, D. Levin[5] and Chu and Beasley[10] focused on real-life set partitioning problems available in OR-library of Beasley[13]. Both systems studied various GA-based operators and techniques such as Parallel GA, fitness evaluation, selection, and local heuristics. Their experimental results showed the successful applications of GA to find optimal or near-optimal solutions of SPP. But the results also showed that even after a large number of calculations both systems struggle to find a feasible solution to some hard test sets like aa01, aa03, and aa05 available in OR-library of Beasley. Additional to GA based framework, a well performing system by Van Krieken[9] also failed to find a feasible solution on the test instance aa01. Hoffman and Padberg[3] commented that the aa01 instance "require significantly more computational efforts" and Borndörfer (1998)[11] described aa01 as a "hard problem".

The purpose of this paper is to design an improved GA system that can find optimal or near-optimal solutions efficiently for hard SPP instances available in the Beasley OR-library.

2. RELATED WORK

2.1 The Set Partitioning Problem

Consider a set $I = \{1, \ldots, m\}$ the set of rows and a set $J = \{1, \ldots, n\}$ the set of columns. Let a cost $c_j > 0$ be associated with every $j \in J$. The set partitioning problem is the problem to find a partitioning with the minimum cost of a given matrix, written as the integer linear program.

$$\text{Minimize} \quad \sum_{j=1}^{n} c_j x_j \tag{1}$$

$$\sum_{j=1}^{n} a_{ij} x_j = 1, \qquad i = 1, \ldots, m \tag{2}$$

$$x_j = 0,1, \qquad j = 1, \ldots, n \tag{3}$$

where

$x_j = 1$ if j is in the partitioning, 0 otherwise
$a_{ij} = 1$ if the column j includes row i, 0 otherwise

2.2 Genetic Algorithms

A genetic algorithm (GA) can be understood as a search algorithm based on the mechanics of natural selection and genetics. It can be applied to different optimization problem[2]. Initially, a set of chromosomes will be generated in the chromosome population pool. Then the generic search proceeds over a number of generations, with each generation represented by a population of current chromosomes.

Preprocessing test data and Initialize the parameters of GA;
Population initialization;
Evaluate fitness of individual chromosome in the population pool;
  Repeat
    Select parents from the pool;
    Crossover operation to produce children;
    Mutation operation
    Apply Local heuristic to each child chromosome
    Evaluate each child chromosome
    Replace some or all of the population by the children;
  Until a satisfactory solution has been found;

Algorithm 1: Genetic algorithm –based algorithm for set partitioning problem.

In every generation, a new set of partitions or chromosomes is reproduced by a crossover operation. Each child chromosome is evaluated by the fitness function. "Survival of the fittest" decides which chromosome will keep staying in the chromosome population as a parent for next generation. In addition, an occasional new part trigged by mutation operation is tried for good measure.

Basic components of GA-based algorithm for SPP includes Preprocessing procedures, Pool initialization, Fitness evaluation, Parental selection, Crossover, Mutation, and Replacement of chromosomes. The basic steps of a simple GA for SPP are show in Algorithm 1.

Preprocessing procedures

Real life set partitioning problems for airline problems are obtained from OR-library[13]. These instances are normally generated by computers. This can result in a lot of duplicate or redundant items. It is often possible to

reduce the size of the problems by applying some reduction rules in order to improve the performance of a solver. In the past many procedures were reported, including Equal Column, Contained Rows, Clique, Equal Rows, Clique, and Row combinations[3, 9, 10]. The most influential rules this research found are as follows.

> **Column reduction**: if a column *j* contains the same rows with another one or two mutually exclusive columns and the cost of column *j* is bigger than the other or the sum of the two columns, this costly column *j* can be removed.
>
> **Contained Row reduction**: if row *s* contains another row *r*, row *s* as well as all columns that are in row *s* but not in row *r* can be removed.

Population Initialization

Population initialization is for setting up the chromosome in the population pool. The size of population decides the amount of memory used and the computation time. A large population will definitely occupy more memory and will also increase the computation time. Although Chu and Beasley[10] claimed that larger size of population does not offer better quality of solutions, results in Garey and Margaritis[15] showed that a GA performance improves when the population size increased. A random initialization algorithm proposed by Chu and Beasley[10] is described as follows.

---

for k =1 to N do
    set S_k : = ∅; set U := I;
    repeat
        randomly select a row $i \in U$;
        randomly select a column $j \in \alpha_i$ such that $\beta_j \cup (I - U) = \emptyset$;
        if j exists then
            set $S_k \leftarrow S_k + j$; set $U \leftarrow U - i$, $\forall i \in \beta_j$;
        else
            set $U \leftarrow U - i$;
        endif
    until U = ∅;
end for

Algorithm 2: Population initialization proposed by Chu and Beasley.

Fitness function

Fitness function is used for evaluating how fit a chromosome is. The most common approach is to use penalty function[12]. The following formula defined by D. Levin[5] is a basic fitness evaluation for each chromosome S.

$$\text{Fitness} \qquad F(S) = \sum_{j=1}^{n} c_j x_j + \sum_{i=1}^{m} \lambda_i f(S) \tag{4}$$

where $f(S) = 1$ if the chromosome S is feasible, 0 otherwise.

The first summation is the sum of the costs of selected columns in a chromosome. The second summation used to evaluate unfitness of a chromosome can be considered as a penalty for an infeasible chromosome. D. Levin[5] noted that there is no method to calculate the optimal value for λ_i . The definition of λ_i based on D. Levin's empirical choice is the highest cost of columns that contain the row i. Chu and Beasley[10] proposed another approach by separating the single fitness measure into two parts: fitness and unfitness scores to obviate the need for the scaling factors λ_i .

Based on Levin's formula in (4), the fitness evaluation used in this research is modified as follows.

$$F(S) = \sum_{j=1}^{n} c_j x_j + \sum_{i=1}^{m} \lambda_i f_i(S) \tag{5}$$

Where $f_i(S)$ = the absolute value of the number of selected columns that contain the row i, minus one.

λ_i = the highest cost of columns that contain the row i .

This penalty part is only applied to the over or under-coved rows in a chromosome.

Parental Selection

Parental selection is for assigning reproductive opportunities to each individual in the population based on their relative fitness. Roulette-wheel selection is a popular method for parental selection. The effect of roulette wheel selection is to return a randomly selected parent, but each parent's chance of being selected is directly proportional to its fitness.

Chu and Beasley proposed a matching selection method that is to select parents would result in an improvement in feasibility without sabotaging solution quality[10]. In their selection process, the first parent is selected based on its fitness. The second one is then selected according to the highest compatibility score calculated by the number of unions minus the number of intersections between two parents. If the first parent is feasible, the second parent is selected based on their fitness.

Crossover operation

In nature, crossover occurs when two parents exchanges parts of their corresponding chromosomes. In a genetic algorithm, crossover recombines the genetic material in two part chromosomes to create two children. There are several crossover operators proposed in the past including one-point, two-point, and uniform. Both D. Levine's[5] and Chu's[10] used the uniform operator described as follows.

1. Two parents are selected, and two children are produced according to a randomly generated mask.
2. For each bit position on the parents, if the corresponding mask bit value is false, the first parent will contribute its value to the first child in the responding position, otherwise, the second parent will contribute to the first child.

Syswerda[14] noted that the one-point and two-point operators are special cases of uniform crossover. One of drawbacks from the above bit-type conventional crossover operators is the possibility of "reinventing the wheel" by preparing crossing columns that both parents do or do not have. Instead of bit string, Falkenauer introduced his grouping genetic algorithm on SPP[1]. The purpose of the grouping crossover is to evade the possibility of producing invalid or too bad children. This operation briefly described as follows can assure more productive valid operations.

1. Select at random two crossing points in each of the two parents.
2. Duplicate the first parent.
3. Inject or add the columns of the crossing points from the second parent into the first parent.
4. Eliminate or cut the columns of the first parent that contains the same rows with the newly embedded columns from the second parent.
5. Repeat adding the columns of the crossing points from the first duplicate parent and the cutting procedure to the second parent.

Let
S = the set of columns in a solution,
U = the set of uncovered rows,
α_i = the set of columns that cover row i,
w_i = the number of columns that cover row i , $i \in I$ in S.
β_j = the set of rows covered by column j.

Initialize $w_i := | \alpha_i \cap S |$, $\forall i \in I$;
set $T := S$; /* T is a dummy set */
repeat /* Drop procedure */
    randomly select a column j , $j \in T$ and set $T \leftarrow T - j$;
    if $w_i \geq 2$, for any $i \in \beta_j$ then
        set $S \leftarrow S - j$; set $w_i \leftarrow w_i - 1$; $\forall i \in \beta_j$;
    end if
until $T = \emptyset$;

initialize $U := \{ i \mid w_i = 0 , \forall i \in I \}$;
set $V := U$; /* V is a dummy set */
repeat /* Add procedure */
    randomly select a row $i \in V$ and set $V \leftarrow V - i$;
    search for the column $j \in \alpha_i$ that satisfies $\beta_j \subseteq U$, and minimizes $c_j / | \beta_j |$;
    if j exists then
        set $S \leftarrow S + j$;
        set $w_i \leftarrow w_i + 1$, $\forall i \in \beta_j$;
        set $U \leftarrow U - \beta_j$ and $V \leftarrow V - \beta_j$;
    end if
until $V = \emptyset$;

Algorithm 3: Add and Cut heuristic proposed by Chu and Beasley.

Mutation operation

Mutation is needed due to the overzealousness of reproduction and crossover operation that leads to premature convergence[10]. Normally, in order to prevent disturbing the current structure of a chromosome, mutation rate should be set in a low probability. In a GA system, mutation is the random alteration of the value of a chromosome. It simply means to invert a specific column in a chromosome.

Local optimizer

Pure GA can hardly find an optimum or a feasible solution on large scale instances without the help of a local optimizer or heuristic. The child solutions produced by the crossover operator are likely to be infeasible. An infeasible child has some rows under-coved and /or over-covered. The existing heuristics based on GA to solve SPP are Unit heuristic, which was issued by Parker and Rardin in 1988[16], Row heuristic by D. Levine[5], and the other is a Drop and Add heuristic designed by Chu and Beasley[10]. This heuristic consists of dropping and adding steps presented in Algorithm 3.

The Add and Drop heuristic outperforms the other two heuristics according to Chu and Beasley's experimental results[10]. This heuristic contains a hidden premature convergence when adding the best column in a chromosome individually. In order to prevent this situation, this paper proposes a modified Add procedure shown in Algorithm 4.

Initialize $w_i := | \alpha_i \cap S |$, $\forall i \in I$;
set $T := S$; /* T is a dummy set */
repeat /* Drop procedure */
    randomly select a column j , $j \in T$ and set $T \leftarrow T - j$;
    if $w_i \geq 2$, for any $i \in \beta_j$ then
        set $S \leftarrow S - j$; set $w_i \leftarrow w_i - 1$; $\forall i \in \beta_j$;
    end if
until $T = \emptyset$;
initialize $U := \{ i \mid w_i = 0 , \forall i \in I \}$;
set $V := U$; /* V is a dummy set */
repeat /* Modified Add procedure */
    randomly select a row $i \in V$ and set $V \leftarrow V - i$;
    search for the column $j \in \alpha_i$ that satisfies $\beta_j \subseteq U$;
    if j exists then
        set $S \leftarrow S + j$;
        set $w_i \leftarrow w_i + 1$, $\forall i \in \beta_j$;
        set $U \leftarrow U - \beta_j$ and $V \leftarrow V - \beta_j$;
    end if
until $V = \emptyset$;

Algorithm 4: The modified Add procedure

The only difference between these two algorithms is that the modified one does not have entire search for the best column when adding individual column in a chromosome. Obviously skipping the entire search can help to evade the pitfall of local optimization and execute faster.

Population replacement

Population replacement is used to make sure that the reproduced children will be fitter than parents. There are two models of replacement: the generational replacement and steady state models. The generational replacement model replaces the entire population, and the steady state replacement model replaces an individual parent once a new child is generated[17]. Based on the steady state model, Chu and Beasley[10] proposed the Ranking replacement method to insert newly children into the population. Their method is to divide the population into four mutually exclusive subgroups and replace an individual in the population based on its fitness and unfitness scores. Garey and Margaritis[15] concluded that by using the generational replacement model the best selected parents of the current generation will participate in the reproduction of offspring compared to the steady-state one.

Elitism as another enhancement is a small percentage of the best chromosomes that survive in the population. This way can make sure the best chromosomes staying in the population for next generation.

3. METHODOLOGY

The GA components used in this paper are described as follows.

Preprocessing procedures: Column reduction and Contained Row reduction is performed before GA approach. This helps to reduce the size of the problems that leads to better performance and less computational execution time.

Population initialization: Algorithm 2 from Chu and Beasley is used as the population initialization. Population sizes vary during the experiments as indicated.

Fitness function: The modified evaluation function in formula (5) is the fitness evaluation in the paper.

Parental selection: Roulette-wheel selection with Chu and Beasley's matching selection method is applied as the parental selection,

Crossover operator: The grouping crossover operation discussed in Section 2.2 is used in the best system.

Mutation: a static mutation rate, 3% is used for all experiments.

Local optimizer: the modified Add and Drop heuristic originated from Chu and Beasley is used as a local optimizer for the best GA system.

Population replacement: the generational replacement model with the 5% elitism strategy is used in the system. Additionally, there are no duplicate offspring survive in the population pool for next generation when replacing population.

4. EXPERIMENTAL RESULTS

4.1 Test Instances

Chu and Beasley[10] reported that the genetic algorithm-based approach is capable of producing high-quality solutions on a large set of real-life set partitioning problems available in OR-library of Beasley[13]. This paper focuses on the four test instances that both GA-based systems by Levin[5] and Chu[10] do not find any feasible solutions except NW14 used for developing this system.

Table 1. SPP test problems used in this paper

		Original		PREP			
Problem	Optimum	Rows	Columns	Rows	Columns	% row reduced	% column reduced
NW01	114852	135	51975	135	30148	0.00%	42.00%
NW14	61844	73	123409	73	6163	0.00%	95.00%
AA01	56138	823	8904	620	7529	24.67%	15.44%
AA03	49649	825	8627	566	6732	31.39%	21.97%
AA05	53839	801	8308	545	6214	31.96%	25.20%

Table 2. Performance results for different replacement strategies on NW14

Size of Population	Simple Replacement	No duplicate Offspring Replacement
60	66678	62432
100	66478	**61844**
160	69132	62428
200	69248	62282

Table 1 lists all the problems, optimum, rows, and columns of the original datasets and the related instances after the preprocessing procedures. The two preprocessing procedures can reduce 20 to 95 percents of columns and up to about 32 percents of rows on different test problems. All the following experimental results are based on the preprocessed test problems.

4.2 Computational Results

Table 2 shows the results of performance for different replacement strategies on NW14 when used with the modified Add and Cut heuristic and the grouping crossover operator until 10000 different offspring were generated. The strategy of "no duplicate offspring in the population" produces better quality of the best solutions by different population sizes compared to the ones by the simple replacement that does not avoid duplication in the population.

Table 3. Results for different crossover operators on NW14

Heuristic	Crossover	Generation used	% of different offspring per generation	Best solution	% from the optimum
A&C	One-point	1050	9.52%	82946	34.12%
	Two-point	1887	5.30%	78402	26.77%
	Uniform	367	27.25%	80090	29.50%
	Grouping	236	42.37%	71780	16.07%
Mod A&C	Grouping	228	43.86%	**61844**	0.00%

Table 3 shows the results of performance on NW14 using different operators with 100 chromosomes in the population until different 10000 offspring were generated. "A& C" denotes the Add and Cut heuristic and "Mod A & C", the modified Add and Cut heuristic. This table demonstrates that the one-point and two-point operators require more generations to produce different offspring. Both average percentages of different generation produced per generation are much less than the one by the uniform and grouping crossovers. The quality of their best solutions is about 26 % away from the optimum. The one-point crossover comes to the poorest performance operator compared to the others.

The row of the uniform crossover with the Add and Cut heuristic shows that the uniform crossover can easily find different offspring but obtain a poorer solution on this instance compared to the two-point operator. The row in the grouping crossover shows its efficiency of finding different

offspring and the best quality of the solution among four operators in conjunction with the Add and Cut heuristic. The last row of Table 3 shows that the modified Add and Cut heuristic with the grouping crossover is the best combination for searching different offspring and finding the best solution.

Table 4. Results for different population size on NW14

Size of Population	Best solution	% from the optimum
20	62362	0.84%
40	62284	0.71%
60	62432	0.95%
80	62374	0.86%
100	**61844**	0.00%
120	**61844**	0.00%
140	62284	0.71%
160	62428	0.94%
180	62982	1.84%
200	62282	0.71%
220	62336	0.80%
240	62980	1.84%
260	61850	0.01%
280	62498	1.06%
300	62442	0.97%

Table 4 summarizes the performance results for different population sizes used with the modified Add and Cut heuristic and the grouping crossover until 10000 different offspring were generated. The results show that there is no guaranty that the larger population size can offer better quality of solutions in this random-based system.

Table 5 shows the results on different test problems until 100000 different offspring were generated. The size of pollution used was 100. This table also displays the best solutions found by Chu and Beasley[10] and D. Levin's systems. The "X" means no feasible solutions found, and the "?" means no experimental figures found in the research paper. It is obvious that the proposed GA-based system successfully produces near-optimal solution on large scale SPP instances, AA01, AA03, and AA05. But the best solution of NW01 is far from the optimum compared to those from the AA group.

Table 5. Summary of performance comparison for different GA-based systems

Problem	Optimal	This System		Best solution time(sec)	Chu and Beasley	D. Levin
		Best	% from the optimum			
NW01	114852	131007	14.07%	307500	X	?
NW14	56138	61844	0.00%	1980	62262	?
AA01	56138	57528	2.48%	4004	X	X
AA03	49649	50747	2.21%	3969	X	?
AA05	53839	55160	2.45%	2081	X	X

5. CONCLUSION AND FUTURE WORK

This paper applies the grouping crossover operation and the modified local optimizer to the generational model genetic algorithm system. Experimental results demonstrate that the modified system has successfully found near-optimal solutions on large scale of real-life set partitioning problems. Future research is to improve the system and/or add some other GA components to find optimal or near-optimal solutions for the NW01 problem.

6. REFERENCES

[1] E. Falkenauer, A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2(1), p5-30, 1996. http://dx.doi.org/10.1007/BF00226291.

[2] J. Arabeyre, J. Fearnley, F. Steiger, and W. Teather, The airline crew scheduling problem: A survey. *Transportation Science*, 3(2), p140-163, 1969. http://dx.doi.org/10.1287/trsc.3.2.140.

[3] K.L. Hoffman, and M. Padberg, Solving airline crew scheduling problems by branch-and-cut. *Management Science*, 39(6), p657-682, 1993. http://dx.doi.org/10.1287/mnsc.39.6.657.

[4] J.G. Digalakis, and D.S. Johnson, *Computer and intractability – A guide to the theory of NP-completeness*. San Francisco, USA: W.H. Freeman.

[5] D. Levine, A parallel genetic algorithm for the set partitioning problem. *PhD thesis, Illinois Institute of Technology, Department of Computer Science*, 1994.

[6] D.E. Goldberg, *Genetic algorithms in search optimization and machine learning*. New York: Addison-Wesley Publishing Co., 1989.

[7] J. Chen, Y. Lin, and L. Chen, A relation-based genetic algorithm for partitioning problems with applications. *Lecture Notes in Computer Science*, 4570, p217-226, 2007. http://dx.doi.org/10.1007/978-3-540-73325-6_22.

[8] J. Holland, *Adaptation in natural and artificial systems*. Ann Arbor: The University of Michigan Press, 1975.

[9] M.G.C. Van Krieken, H.A. Flueuren, and M.J.P. Peeters, A lagrangean relaxation based algorithm for solving set partitioning problems, *CentER Discussion*, Paper No. 2004-44, 1989.

[10] P.C. Chu, and J.E. Beasley, Constraint handling in genetic algorithms: The set partitioning problem. *Journal of Heuristics*, 4(4), p323-357, 1998.

[11] R. Borndörfer, Aspects of set packing, Partitioning, and Covering. *PhD thesis, TU Berlin*, 1998.

[12] S.M. Shahriar Nirjon, Study on GA based solutions to set partitioning problem and proposed new approach. *Daffodil International University of Science and Technology*, 1(1), 2006. Retrieved September 28, 2012, from http://www.daffodilvarsity.edu.bd/diujst/is-1-v1.php.

[13] J.E. Beasley, OR-Library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11), p1069-1072, 1990.

[14] G. Syswerda, Uniform crossover in genetic algorithms, In J.D. Schaffer (Ed.), *Proceeding of the 3rd International Conference on Genetic Algorithms* (p2-9). George Mason University, Fairfax, Virginia, USA: Morgan Kaufmann, 1989.

[15] M.R. Garey, and K.G. Margaritis, On benchmarking functions for genetic algorithms. *International Journal of Computer Mathematics*, 77(4), p481-506, 2000.

[16] R. Parker, and R. Rardin, *Discrete optimization*. San Diego: Academic Press, 1988.

[17] M. Lozano, F. Herrera, and J.R. Cano, Replacement strategies to maintain preserve useful diversity in steady-state genetic algorithms. *Inf Sci*, 178(23), p4421-4433, 2008.