VOL. 11, NO. 7, APRIL 2016 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

PERFORMANCE EVALUATION OF VARIOUS GENETIC ALGORITHM APPROACHES FOR KNAPSACK PROBLEM

A. Syarif, Aristoteles, A. Dwiastuti and R. Malinda
Department of Computer Science, Faculty of Mathematics and Natural Sciences, The University of Lampung, Indonesia
E-Mail: admi_syarif@yahoo.com

ABSTRACT

Knapsack Problem (KP) is known as one of optimization problems that has taken great interest of researchers. It has been applied for many practical applications. Since it belongs to the class of NP-hard problems, most of researchers reported heuristic methods to solve it. Those include Branch and Bound, Greedy Algorithm, Genetic Algorithm and Dynamic Programming.

In this paper, we focus on the performance evaluation of various Genetic Algorithm (GA) approaches to solve Knapsack Problem. We developed four different GA approaches with different strategies. The first, random penalty GA (rpGA) uses random strategy to generate chromosome and penalty strategy to handle infeasible chromosome. The second, directed penalty GA (dpGA) uses directed strategy to generate chromosome and penalty to handle infeasible chromosome. The third, random repairing GA (rrGA) uses random strategy to generate chromosome and repairing strategy to handle infeasible chromosome. The fourth, directed repairing GA (drGA) uses directed strategy to generate chromosome and repairing strategy to handle infeasible chromosome.

In order to investigate the performance of those algorithms, we have done several numerical experiments by using different size Benchmark test problems given in literature. The effectiveness and the efficiency of the methods are also evaluated by varying GA parameters. Based on our experiments, it is shown that drGA was the best performance to give optimal solution within reasonable computational time.

Keywords: knapsack problem, combinatorial optimization, evaluation strategy, genetic algorithm.

1. INTRODUCTION

Knapsack Problem (KP) is one of well known combinatorial optimization problems. It has taken great interest of researchers in these several decades. It is regarded as grouping items into two classes, those being put into the Knapsack and those being discarded. The objective is to maximize the profit of a subset the chosen of item in a Knapsack. There have been many variations of this problem for different applications [1]. Among them, however, 0-1 KP is the most intensively studied. The reasons for such interests basically derive from three facts: (a) it can be viewed as the simple integer linear programming problem; (b) it appears as a sub problem in many complex problems; (c) it may represent many practical situations [2]. Practical applications of 0-1 KP also can be found in some of our daily life applications such as: the daily diet program where a person must choose some food without surpassing diet limit calories, an optimal investment plan, choosing which stock should be taken, cargo loading, cutting stock, budges control, and financial management [3-4].

KP belongs to the class of NP-Hard problems [5]. The body of literature on the methods for solving KP is large; and, most of them deal with conventional methods including Branch and Bound (BB) and Dynamic Programming [6], Greedy Strategy [7], and heuristic method, such as Repairing Operator Strategy [8].

When using heuristic methods, there are several important issues. Those are including how to generate feasible solution and how to handle infeasible chromosome. Figure-1 shows the mapping of encoding space to the solution space.

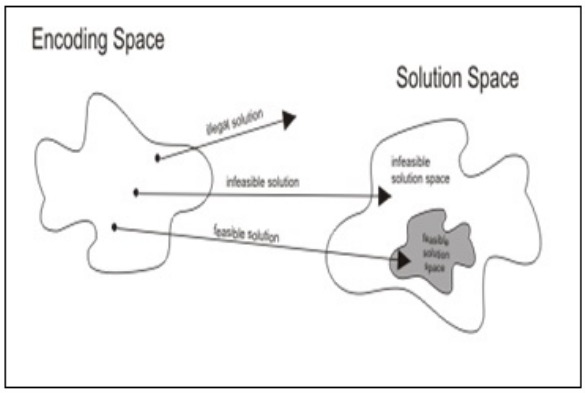


Figure-1. Solution spaces.

Genetic Algorithm (GA) has been known as one of powerful heuristic methods to find optimal solutions for many hard optimization problems. It was introduced by Holland [9]. Then it has been popularized by some researchers, Gen and Cheng [10-11], Goldberg [12] and Michalewicz [13]. GA starts with an initial set of random or directed candidate solutions called population satisfying boundary and/or system constraints to the problem. GA also works with certain parameters, does searching process with a group of candidate solution and uses information from objective function [10]. There are many advantages

of using GA. One of them is its flexible to be combined with other methods.

In our previous works, we also have done intensive research works to implement GA approaches for solving various logistic problems [14-17]. Our results show that GA is very effective and efficient. However, its performances depend on some basic components: genetic representation, way to create a population, evaluation strategy and the method for handling infeasible chromosome, Genetic operators (crossover, mutation, selection, etc) and GA parameters (population size, crossover and mutation probabilities) [18].

In this research, we focused on the performance evaluation of various GA approaches to solve KP. We develop various GAs with different strategies especially for the methods to generate and evaluate of chromosome. The first, random penalty GA (rpGA) uses random strategy for generating chromosome and penalty strategy for handling infeasible chromosome. The second, directed penalty GA (dpGA) uses directed strategy to generate chromosome and penalty strategy to handle infeasible chromosome. The third, random repairing GA (rrGA) uses random strategy to generate chromosome and repairing strategy to handle infeasible chromosome. The fourth, directed repairing GA (drGA) uses directed strategy to generate chromosome and repairing strategy to handle infeasible chromosome. The performances of those algorithms are evaluated by comparing the results with the known optimal solutions of Benchmark test problems in literature [19]. We also verify the efficiency of the methods by varying the GA parameters.

The rest of this paper is organized as follows: In the next Section, the mathematical formulation of this problem is given. The design of our algorithm including the chromosome representation and the GA process is described in Section 3. In Section 4, Numerical experiments and comparative results of algorithms are presented to demonstrate the effectiveness and efficiency of the methods. Finally, some concluding remarks are given in Section 5.

2. MATHEMATICAL MODEL

Knapsack Problem (0-1 KP) is a problem of choosing the subsets of the n items such that corresponding profit sum is maximized without having the total weight to exceed the Knapsack capacity c .

The mathematical model of KP is given as follows:

$$\textbf{max}: \ z = \sum_{i=1}^{n} p_i x_i \tag{1}$$

$$\textbf{s.t} \ : \sum_{i=1}^{n} w_i x_i \le c \tag{2}$$

$$x_i \in \{0,1\}, \ i = 1,2,..., n$$

with

p_i = profit of item *i.*

w_i = weight of item *i.*

c = maximum capacity of Knapsack.

In the above model, x_i does a binary variable equal to 1 if item *i* should be included in the Knapsack, 0 otherwise.

The equation 1 represents the objective function to be maximized and the equation (2) is the capacity constraint.

3. DESIGN OF ALGORITHM

One of the important and the influential components to the GA performance is the way on how the initial of chromosome formed. The most commonly used technique to generate the initial chromosome is with greedy method. Here, genes are generated randomly. For combinatorial optimization problems, however, constraint function will make population not always feasible. It can be all feasible, half feasible half infeasible, even all infeasible. To control this state, GA has two strategies: repairing and penalty strategies. Repairing strategy include the procedure to repair an infeasible solution until the solution is feasible. Penalty strategy gives penalty to decrease or increase the fitness value so the infeasible chromosome isn't chosen.

a) Chromosome representation and initialization

How to represent chromosome is the first step of implementing GA. Chromosome representation is data structure to represent solution candidates. There are many ways to represent chromosome depend on the problems. For 0-1 KP, we use binary representation. One chromosome has some genes. Here, the number of genes matches with the number of item. The value 1 of genes shows that item include to the Knapsack. The illustration of the chromosome representation for this GA is given in the following Figure-2.

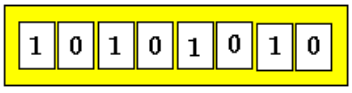


Figure-2. Representation chromosomes.

i. Random strategy

To generate the chromosome in the initial population, we use two strategies. The first is called as random strategy or greedy strategy that generates each gene in the chromosome for the initial population randomly. With this situation, however, it is possible to generate some infeasible chromosome in population. The procedures of random strategy are given as follows:

Procedure: random strategy
Initialization =
fix(2* rand(pop_size,genes));

ii. Directed strategy

The second is called directed strategy. In this strategy, we include the procedure that can guarantee the feasibility of the chromosome as follows:

Procedure: Directed Strategy

```
begin
init_pop = zeros(pop_size,genes);
  for j = 1 : pop_size
   cap_now = capacity;
   for k = 1 : genes
    index = fix((rand*genes)+1);
    if init_pop[j,index] == 0 &&
    weight[index] <= cap_now then
      init_pop[j,index] = 1;
      cap_now = cap_now - weight[index];
      if cap_now <= 0 then
        break;
      end
    end
   end
  end
end
```

b) Genetic Operations

i. Crossover

Crossover is known as the most important recombination operator in GA. In this paper, we adopt one-point crossover by determining a cross point randomly to make the chromosome into two parts, the left side and the right side. Then, the left side of Parent 1 will cross with the right side of Parent 2.

```
begin
%procedure of choosing chromosome for crossover
 k = 0; i=1;
 while ( k < pop_size )
  r[k]→ random number [0,1];
  if r[k] <= prob_crossover then
   Choose chromosome[k] for crossover as
   parent[i];
  end
  k = k + 1; i = i + 1;
 end
   Generate random number (p) of crossover point;
   Exchange parent[i] from gene 1 to p with
   parent[i+1] from gene p+1 to genes;
end
```

The illustration of the one-point crossover is given as follows:

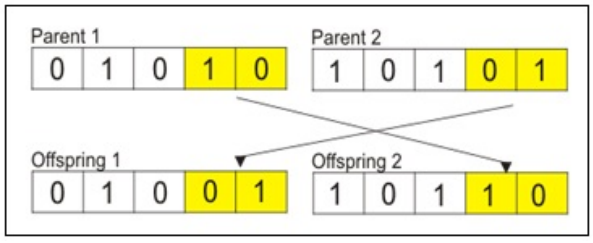


Figure-3. One-point crossover operation.

ii. Mutation

Mutation is usually used to prevent premature loss of information. It is usually done by exchanging the information within a chromosome. Here, we adopt flip mutation by modifying the value of gene whether it is 0 and will become 1, and vice versa.

Procedure: Flip Mutation

```
begin
k = 0;
while (k < genes)
  if (random[k] <= prob_mutation) then
    Choose genes[k] for mutation
    if genes[k] == 0 then
           Replace genes[k] = 1;
    else
           Replace genes[k] = 0;
    end
  end
 end
k = k + 1;
end
```

The illustration of the flip mutation is given as follows:

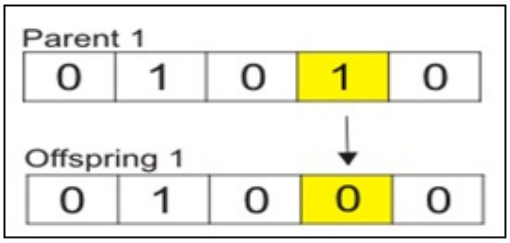


Figure-4. Flip mutation operations.

c) Evaluation strategy

As in nature, it is necessary to provide driving mechanism for better individuals to survive. Evaluation is to associate each chromosome with a fitness value that shows how good it is based on its achievement of the objective function. Since crossover and mutation operations would also generate infeasible offspring, we have to check the feasibility of the offspring. To handle such infeasible chromosome, there are two ways which is commonly used. Those are repairing strategy and penalty strategy.

i. Repairing strategy

For the repairing strategy, we include the procedure to repair infeasible chromosome. The procedure will choose item with the small ratio between profit and weight. It is done as follows:

```
begin
 for i = 1: pop_size
  %check total weight of chromosome i, whether
  it's exceed capacity or not
  if total weight of chromosome i > capacity
     then
      //it means chromosome is infeasible
      Select gene with smallest ratio;
      Change the value of gene become 0;
      Recalculate the total weight chromosome;
  end
 end
end
```

ii. Penalty strategy

The Penalty strategy uses penalty function for the chromosome that doesn't require constraint function. Since this problem is maximization, the penalty function should decrease the total profit of item. With this situation, the chromosome that has low total profit will not be included into the next generation. In this research, we adapted Olsen's penalty function, as follows [20]:

$$penalty = p_i * (dist / diff)$$

where p_i represent the profit of the *i*-th individual before the penalty is applied, dist (*distance*) refers to the difference between the maximum weight allowed (*W*) for a feasible solution and the actual weight of an individual solution (w_i) and where *TW* is the total of all weights..

$$dist = |w_i - W|$$
$$diff = \min(W, |TW - W|)$$

It can be noticed that the penalty is calculated by using ratio *dist/diff* and the profit, p_i. Thus, penalty will increases as the profit increases.

```
begin
total_allWeight = sum(weight);
for i = 1 to n
  if tot_weight_chrom[i] > capacity then
     dist = |tot_weight_chrom - capacity|;
     diff = min (capacity,|total_allWeight -
            capacity |);
     penalty = 1 - (dist/diff);
     if  penalty <= 0 then
         penalty = 0.00001;
     end
     fit_of_string_i= fit_of_string_i *
     penalty;
  end
 end
end
```

d) Selection

Selection is also one of important steps on GA. It will choose chromosome that will pass through the next generation. For selection method, we use combination of roulette wheel and elitism method. Roulette wheel method gives probability value at each chromosome. The chromosome with higher objective value will have more probability to be chosen for next generation. Elitism will maintain the good chromosome. Thus, the best chromosome will be included for the next generation [9].

e) Overall algorithm

Overall procedure: UGA_for_Knapsack
t = 0;
Generate *P(t)*
- (Random Strategy or Directed strategy);
Evaluate chromosome *P(t);*
while (**not** stopping condition) **do**
begin
t = *t* + 1;
GA Operations (Crossover and Mutation);
Evaluate (Check Feasibility) Offspring Chromosome
- (Penalty strategy or Repairing);
Select chromosome for next generation;
end
end

4. NUMERICAL EXPERIMENTS

For the numerical experiments, we develop four kinds of GA approaches with different strategy. Those approaches were implemented in MatLab R2009a version and run on PC with processor Intel-Core i3. For the experiments, we use nine different size test problems that their optimal solution has been known. Those test problems are taken from a set of standard test problems given in the literature [19]. Table-1 shows the information of test problem and its optimal solutions.

Table-1. Test problems.

No.	Test Problems*	*n*	Optimum
1	p02	5	51
2	p03	6	150
3	p04	7	107
4	p05	8	900
5	p01	10	309
6	dataset 20_1000	20	4129
7	p08	24	13549094
8	Small	40	1149
9	Medium	100	1173

*http://kpacking.googlecode.com/svn/trunk/.

For these experiments, we set the GA parameters as follows: *pC* = 0.4, *pM* = 0.2, *pop_size* = 100 and *max_gen* = 500. For each test problem, we run the algorithm ten times. We noted the best results given by each algorithm in the experiment. The following Table-2 shows the comparative average objective value given by those four algorithms.

Table-2. Comparative computational results.

No.	Test Problem	*n*	rpGA	dpGA	rrGA	drGA
1	p02	5	51	51	51	51
2	p03	6	150	150	150	150
3	p04	7	107	107	107	107
4	p05	8	900	900	900	900
5	p01	10	309	309	309	309
6	dataset20_1000	20	4129	4129	4129	4129
7	p08	24	13549094	13549094	13549094	13549094
8	small	40	infeasible	1149	1149	1149
9	medium	100	infeasible	1157	1173	1173

*after 10 times running program

Table-3. Comparative errors of various pC and pM.

No	Test Problem	Directed Penalty								Directed Repairing*							
		pC = 0.4				pC = 0.6				pC = 0.4				pC = 0.6			
		pM															
		0.2	0.4	0.6	0.8	0.2	0.4	0.6	0.8	0.2	0.4	0.6	0.8	0.2	0.4	0.6	0.8
1	p02	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	p03	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	p04	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	p05	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	p01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	dataset20_1000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	p08	1,25	1,26	1,41	0,44	0,89	0,65	0,61	0,60	0	0	0	0	0	0	0	0
8	small	1,13	0,96	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	medium	2,39	0,42	0,26	0	0,09	0	0,09	0	0	0	0	0	0	0	0	0

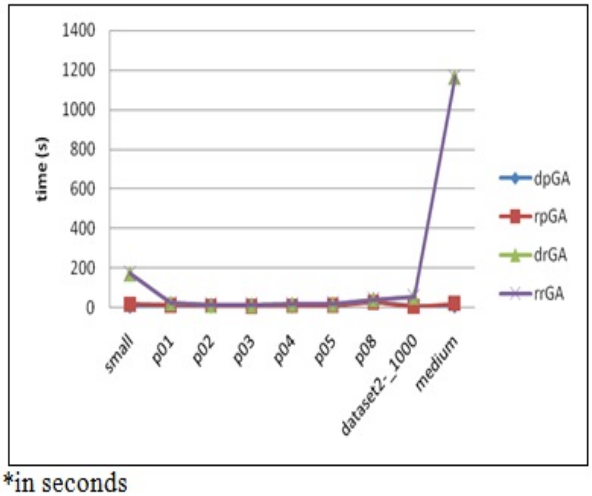
*in percent (%)

From the above results, we can see that directed GA combined with both penalty and repairing strategy can give optimal/near-optimal solutions for all of the problems. On the other hand, for relatively large size or the strict constraint problem, however, we can also notice that random strategy GA would possible to give infeasible solutions. The reason is because random strategy GA cannot generate feasible chromosome in the initial population. The difficulty of generating feasible chromosome occurs for the problem with most of item weights almost the same as its capacities. Thus, the difference between weight of each item and Knapsack capacity is very small.

In order to compare the performance of penalty and repairing strategy GA, we also run the algorithms by varying GA parameters. The results are given in Table 3. The percent errors in this Table are computed by:

$$Error = \frac{(Optimal - Objective)}{Optimal} \times 100\ \%$$

Those error values happen if there is difference between optimal value and objective value. Then it shows that GA with directed strategy combined with repairing strategy (drGA) can give optimal solutions all of the time. It is also shown that the algorithms are sensitive with the variations of GA parameters. Thus, by varying GA parameters (population size, crossover and mutation probabilities), we can improve the quality of solutions.



*in seconds

Figure-5. Comparative computational time for all algorithms.

Finally, in order to see the efficiency of the methods, we also compare the average computational time (ACT) of those algorithms. The results of these experiments are shown in Figure-5. It can be seen that GA would solve KP within reasonable time. For hard constraint KP, however, the repairing strategy GA would give more computational time. This is due to more time consuming to repair infeasible offspring resulted by crossover and mutation operations.

5. CONCLUSIONS

In this paper, we report our results on the performance evaluation of various GA approaches. Those differ in the way to generate the initial population to solve KP and the way to handle infeasible chromosome. Our results show that directed strategy combined with repairing strategy GA could give good quality solutions all the time and error = 0%. Moreover, for some specific problems, we found that random generated GA could not generate feasible solution on small and medium test problems. It is also shown repairing strategy would need more time consuming. Future works may address for hybridizing some of evaluation strategies.

ACKNOWLEDGEMENTS

This research was supported by Scientific Research Grant from Directorate of Higher Education, the Ministry of Education, and Culture, Republic Indonesia. The Grant-in-Aid for "Hibah Bersaing" Scientific Research (2015)

REFERENCES

[1] Martello, S. and Toth P. 1990. Knapsack Problems, Algorithms and Computer Implementations. John Willey and Sons.

[2] Sarac, T. and Anagun, A. S. 2006. Optimization of Performance of Genetic Algorithm for 0-1 Knapsack Problem Using Taguchi Method. Proceedings of ICCSA, (M. Gavrilova *et al* Editors). Springer-Verlag, Berlin.

[3] Pisinger, D. 1995. Algorithm for Knapsack Problem, PhD Thesis. The University of Copenhagen, Universitetsparken 1, DK-2100 Copenhagen, Denmark.

[4] Gupta, M. 2013. A Fast and Efficient Genetic Algorithm to Solve 0-1 Knapsack Problem. International Journal of Digital Application and Contemporary Research. Vol. 1, No. 6.

[5] Pisinger, D. 2005. Where Are The Hard Knapsack Probems?. Computer and Operation Research. Vol. 32, No. 9. pp. 2271-2284.

[6] Martello, S., Pisinger, D. and Toth P. 2000. New Trends in Exact Algorithms for the 0-1 Knapsack Problem. European Journal of Operational Research. pp. 325-332.

[7] Zhou, J., Huang, T., Pang, F. and Liu, Y. 2009. Genetic Algorithm based on Greedy Strategy in the 0-1 Knapsack Problem. Proceedings of the Third International Conference of Evolutionary Computing. pp. 105-107

[8] Garg, M. L. and Gupta, S. 2009. An Improved Genetic Algorithm Based on Adaptive Repair Operator for Solving the Knapsack Problem. Journal of Computer Science, Vol. 5, No. 8. pp. 544-547.

[9] Holland, J. 1992. Adaptation in Natural and Artificial Systems. University of Michigan Press, 1975 and MIT Press.

[10] Gen, M. and Cheng, R. 1997. Genetic Algorithms and Engineering Design. John Wiley & Sons, New York, USA.

[11] Gen, M. and Cheng, R. 2000. Genetic Algorithms and Engineering Optimization. John Wiley & Sons, New York, USA.

[12] Goldberg, D. 1989. Genetic Algorithm in Search, Optimization and Machine Learning. Reading, MA: Addison-Wesley.

[13] Michalewicz, Z. 1994. Genetic Algorithms + Data Structure = Evolution Program. Springer-Verlag, New York.

[14] Syarif, A. and Gen, M. 2003. Solving Exclusionary Side Constrained Transportation Problem by Using A Hybrid Spanning Tree-based Genetic Algorithm. Journal of Intelligent Manufacturing, Vol. 14 (3/4). pp. 389-399.

[15] Syarif, A. and Gen, M. 2003. Double Spanning Tree-Based Genetic Algorithm for Two Stage Transportation Problem. The International Journal of Knowledge-based Engineering Systems. Vol. 7, No. 4. pp. 214-221.

[16] Syarif, A., Yun, Y.S. and Gen, M. 2002. Study on Multi-stage Logistics Chain Network: A Spanning Tree-based Genetic Algorithm Approach. International Journal of Computer and Industrial Engineering. Vol. 43, No. 1-2. pp. 299-314.

[17] Syarif, A., Wamiliana and Yasir, W. 2008. Evaluasi Kinerja Metode-Metode Heuristik untuk Penyelesaian Traveling Salesman Problem. Jurnal Sains MIPA (In Indonesian Language). Vol. 14, No. 1.

[18] Syarif, A. 2014. Algoritma Genetika: Teori dan Aplikasi, 2nd Ed. PT Graha Ilmu, (in Indonesian Language).

[19] HUhttp://kpacking.googlecode.com/svn/trunk/UH downloaded on January 23rd. 2015.

Olsen, A. L. 1994. Penalty Functions and the Knapsack Problem. Proceedings of the First IEEE Conference on IEEE. pp. 554-558.