Comparative Study of Mutation Operators on the Behavior of Genetic Algorithms Applied to Non-deterministic Polynomial (NP) Problems

Basima Hani Hasan
Department of Computer Science
Yarmouk University, 21163 Irbid, Jordan
basmah@yu.edu.jo

Moutaz Saleh Mustafa
Department of Computer Science & Eng.
Qatar University, 2713 Doha, Qatar
moutaz.saleh@qu.edu.qa

Abstract—**Genetic Algorithms (GAs) are powerful general-purpose optimization search algorithms based upon the principles of evolution observed in nature. Mutation operator is one of the GA operators that used to produce new chromosomes or modify some features of it depending on some small probability value. The objective of this operator is to prevent falling of all solutions in population into a local optimum of solved problem. This paper applies several mutation methods to different non-deterministic polynomial (NP) hard problems and compares the results. The problems that will be introduced in this paper are: traveling salesman problem (TSP), 0/1 Knapsack problem, and Shubert function.**

Keywords-GA; Mutation; NP; TSP; 0/1 Knapsack problem; Shubert function.

I. INTRODUCTION

GAs are powerful general purpose optimization search algorithms based upon the principles of evolution observed in nature. Even with today's high-powered computers, using an exhaustive search to find the optimal solution for even relatively small problems can be prohibitively expensive. For many problems, genetic algorithms can often find good solutions, near-optimal, in around 100 generations. This can be many times faster than an exhaustive search. A solution to a problem solved by genetic algorithms is evolved. Algorithm is started with a set of solutions, represented by chromosomes, called population. Solutions from one population are taken and used to form a new population. This is motivated by a hope, that the new population will be better than the old one. Solutions which are selected to form new solutions, offspring, are considered according to their fitness; the more suitable they are the more chances they have to reproduce. Then crossover and mutation are applied on them to find new points in the search space.

This paper will use GA to solve NP problems. A decision problem is called an NP problem if particular examples of it can be solved in polynomial time by a nondeterministic process i.e. by generating possible solutions at random guessing. There are many problems for which no polynomial-time algorithm is known, this paper will consider three of them; Traveling Salesman Problem (TSP), 0/1 Knapsack problem, and Shubert function.

Given a finite number of cities along with the cost of travel between each pair of them; TSP try to find the cheapest way of visiting all the cities each of which exactly once before returning back to the starting point. TSP is a problem arises in many applications as its difficulty comes from the fact that for N cities there are N!/2N possible paths. There are several algorithms, which approach this problem. This paper will use GA to solve this problem by applying several types of mutation methods and compare the results. These types, depending on the representation used are: Reciprocal exchange Mutation, Inversion mutation, Insertion mutation, Displacement mutation, Boundary mutation and Uniform random mutation.

The main idea of 0/1 Knapsack problem is how to fill the knapsack with the subset of given items to reach maximum profit without exceeding the knapsack capacity. The knapsack problem arises whenever there is resource allocation with financial constraints. Profit would be the importance of the item, while the cost is the amount of space it occupies in the knapsack. So we need to maximize our profit while minimizing our cost. This problem shows how to deal with constraints. One way to achieve that is based on the application of special repair algorithms to correct any infeasible solutions. This paper will use GA to solve this problem by applying several types of mutation methods and compare the results. These types depending on the representation used are: Flip bit mutation, Boundary mutation, Non uniform mutation, Inversion mutation, Insertion mutation and Displacement mutation.

In Shubert function the target is to benchmark global optimization methods. The formulation of the global optimization problem is to find the absolute minimum for a given function over the allowed range of its variables. So, Shubert function is used as an indicator for the efficiency of these methods. The Shubert function could be 1, 2, 3… or N dimension. The object function for one-dimension Shubert function can be given as:

$$f(x_1) = \sum_{j=1}^{5} j \cdot \cos[(j+1)x_1 + j]$$
$$\text{where} \quad -10 \leq x_1 \leq 10$$

For N-dimension Shubert function, there are N*pow(3,N) global solutions. The object function for N-dimension Shubert function can then be given as:

IEEE
computer
society

$$f(\mathbf{x}) = \prod_{i=1}^{n} \sum_{j=1}^{5} j \cdot \cos[(j+1)x_i + j]$$
$$\text{where} \quad -10 \le x_i \le 10, \ i = 1, \cdots, n$$

GAs are general purpose search algorithms, so we can use them for searching the global minima of the Shubert function. This paper will use GA to solve this problem by several types of mutation methods and compare the results. These types depending on the representation used are: Boundary mutation, Uniform random mutation and Non-uniform mutation.

II. RELATED WORK

In recent years, the genetic algorithms for solving NP hard problems have achieved great results [1] [2] [3] [4]. In particular, the Traveling Salesman Problem (TSP) has been receiving continuous and growing attention in artificial intelligence, computational mathematics and optimization. For instance, the work in [5] proposed an improved GA to solve TSP through adopting an untwist operator which can unite the knots of route effectively, so it can shorten the length of route and quicken the convergent speed. In [6], a new selection strategy is incorporated into the conventional genetic algorithm to improve the performance of genetic algorithm in solving TSP. The results show that the number of evolutional iterations to reach an optimal solution can be significantly reduced. Liu and Huang [7] proposed a novel genetic algorithm to overcome the defections of slow convergence of traditional GA. The algorithm creates crossover and mutation by merging two kinds of heuristics. Simulation results indicated that it can get high-quality solution while consume less running time.

The 0/1 Knapsack Problem is a well known NP hard problem [8] [9] [10] [11] as it appears in many real life world with different application. In [12] an evolutionary genetic algorithm for solving multi objective 0/1 Knapsack Problem is introduced. Experimental outcome show that the proposed algorithm outperforms the existing evolutionary approach. In addition, the work in [13] proposed a genetic algorithm using greedy approach to solve this problem. The experiments prove the feasibility and validity of the algorithm.

Shubert function optimization problem has also been studied in literature. Based on the principle of free energy minimization of thermodynamics, a new thermodynamics evolutionary algorithm (TDEA) for solving Shubert function optimization problem has been proposed in [14]. The results show that thermodynamics evolutionary algorithm is of potential to obtain global optimum or more accurate solutions than other evolutionary methods.

III. GA OPERATION

A GA is an iterative procedure that consists of a constant size population of individuals. These individuals are decoded and evaluated according to fitness function. To form a new population, individuals are selected according to their fitness. A crossover and mutation are then applied on them to find new points in the search space. Figure 1 shows the iterative procedure of a general GA.

```
Generate random population of n chromosomes
Loop (until best solution)
    Evaluate fitness f(x) of each chromosome
    Loop (until new population is completed)
        Select 2 parent chromosome
        Crossover the parents to form offspring
        Mutate new offspring at each locus
        Place new offspring in a new population
    End Loop
    Adopt newly generated population
    Test for best solution
End Loop
```

Figure 1. A General GA Procedure

For our research work, we will adopt the GA procedure steps to solve three well known NP problems: Traveling Salesman Problem (TSP), 0/1 Knapsack problem and Shubert function. To do so, Table 1 shows how these steps are applied on the three problems.

Table 1. Applying GA steps on TSP, 0/1 Knapsack & Shubert Function

GA Step	TSP	0/1 Knapsack	Shubert function
Encoding	integer	binary	floating point
Initial Population	Heuristic (greedy algorithm from different 17 cities)	15 items	Two genes in each individual
Parameters	Population size=50 Probability of crossover=80% Probability of mutation=5% Maximum number of Generations=5000		
Evaluation Function	Total cost of travel between cities	summation of the profits of the items selected	smallest value we get is the best
Selection Operator	roulette wheel	roulette wheel	roulette wheel
Crossover Operator	Heuristic	Single point	Single point
Mutation Operator	• Reciprocal • Inversion • Insertion • Displacement • Boundary • Uniform	• Flip bit • Inversion • Insertion • Displacement • Boundary • Non uniform	• Boundary • Uniform random • Non uniform

IV. MUTATION OPERATOR

From biology view mutation is any change of DNA material that can be reproduced. From computer science view mutation is a genetic operator that follows crossover operator. It usually acts on only one individual chosen based on a probability or fitness function. One or more genetic components of the individual are scanned. And this component is modified based on some user-definable probability or condition. Without mutation, offspring chromosomes would be limited to only the genes available within the initial population. Mutation should be able to introduce new genetic material as well as modify existing one. With these new gene values, the genetic algorithm may be able to arrive at better solution than was previously possible. Mutation operator prevents premature convergence to local optima by randomly sampling new points in the search space. There are many types of mutation and these types depend on the representation itself.

A. *Applying Mutation on TSP:* since integer representation is the best representation for TSP, the following mutation types were applied to this problem:

- *Reciprocal*: two cities are exchanged, swapped, after they are selected randomly.
- *Inversion:* two cut points are selected randomly, and then the sub tour between them is inverted.
- *Insertion*: a city and a place to be inserted in it are selected randomly.
- *Boundary:* a city is chosen and replaced randomly with either the upper or lower bound for that city. The chromosome is then searched for the upper or lower bound and that city is replaced with the bound as shown in Figure 2.

```
current = population[i].gene[j];
x = rand()%1000/1000.0;
if (x < PMUTATION)
  {
 //to decide upper or lower boundary
  p = rand()%1000/1000.0;
   if (p < 0.5)
    //replace with lower boundary
      {
      currentpop = i;
      index = search(lbound,currentpop);
      if (index != -1)
        swap(&current,
             &population[i].gene[index]);
       population[i].gene[j]=current;
      }
   else
    //replace with upper boundary
     {
      currentpop = i;
      index = search(ubound,currentpop);
      if (index != -1)
        swap(&current,
             &population[i].gene[index]);
     population[i].gene[j]=current;
      }
```

Figure 2. Boundary Mutation

- *Displacement:* a sub tour and a place to be inserted in it are selected randomly as shown in Figure 3.
- *Uniform:* a city selected for mutation is replaced with a uniform random value between the user-specified upper and lower bounds for that city. Then the chromosome is searched for the uniform random value found and replaces it with that city.

```
CityNum = ( rand() % CitiesNum);
InsertPos = (rand() % CitiesNum);
do
  //length of the tour
  length1 = (rand()%CitiesNum+1);
  while ((InsertPos+length1-1>=CitiesNum)||
         (CityNum+length1-1>=CitiesNum));
  p = rand()%1000/1000.0;
  if (p < PMUTATION)
     {w=0;
      for (int h = CityNum;
           h<=(CityNum+length1-1);h++)
      {w = w + 1;
        SubList[w]=population[i].gene[h];
      }
     if (InsertPos > CityNum)
       {
       for (int j = CityNum + 1; j <=
            InsertPos; j++){
             population[i].gene[j-1] =
             population[i].gene[j+length11];
              w=0;
             j=j-1;}
       for (int jj = j; jj<(j+length1); jj++){
             w = w+1;
             population[i].gene[jj]=SubList[w];
             }
      }
     if (InsertPos <CityNum)
      {
      for (int j=CityNum - 1; j>=InsertPos; j--)
       {
        population[i].gene[j+length1] =
         population[i].gene[j];
        w=0;
        j=j+1;
        for (int jj = j; jj<(j+length1);jj++)
          {w = w+1;
           population[i].gene[jj]=SubList[w];
           }
       }
     }
   }
```

Figure 3. Displacement Mutation

B. *Applying Mutation on 0/1 Knapsack:* since binary representation is the best representation for 0/1 Knapsack problem, the following mutation types were applied to this problem:

- *Flip bit*: an item is selected randomly and its value is inverted from 0, selected, to 1, unselected, or vice versa as shown in Figure 4.

9

```
x = (rand()%ItemsNum);
p = rand()%1000/1000.0;
if (p < PMUTATION)
     population[i].gene[x] = 1-
     population[i].gene[x];
```

Figure 4. Flip bit Mutation

- *Inversion*: items between two randomly chosen points in the individual are reversed in order.
- *Insertion*: an item is taken at random and inserted randomly into another position in the sequence.
- *Displacement:* A randomly selected section of the individual is moved as a block to another location in the individual.
- *Boundary:* an item is selected randomly and its value is replaced randomly either by the upper bound (1) or the lower bound (0).
- Non-uniform: This type increases the probability that the amount of the mutation will be close to 0 as the generation number increases. This mutation operator keeps the population from stagnating in the early stages of the evolution then allows the genetic algorithm to fine tune the solution in the later stages of evolution as shown in Figure 5. If s_v^t = $<v_1,…,v_m>$ is a chromosome (t is the generation number) and the element v_k was selected for this mutation, the result is a vector s_v^{t+1} = $<v_1,…,v_k',…,v_m>$, where v_k'= mutate(v_k , $\bigtriangledown$(t,n)), where n is the number of bits per one element of a chromosome, mutate(v_k, pos) means mutate the k-th value element on pos bit, and

$$\bigtriangledown (t,n) = \begin{cases} \lfloor \bigtriangleup (t,n) \rfloor & \text{if a random digit is 0} \\ \lceil \bigtriangleup (t,n) \rceil & \text{if a random digit is 1} \end{cases}$$

```
x = rand()%1000/1000.0;
if (x < PMUTATION)
    {
     p = rand()%2;
     y = rand()%1000/1000.0;
     if (p == 0){
          pos=floor(ItemsNum*(1-pow(y,pow(1-
               generation/MAXGENS,5))));
          population[i].gene[pos]=1-
          population[i].gene[pos] ;
       }
     else if (p == 1){
          pos=ceil(ItemsNum*(1-pow(y,pow(1-
               generation/MAXGENS,5))));
          population[i].gene[pos]=1-
          population[i].gene[pos];
       }
    }
```

Figure 5. Non-uniform Mutation

C. *Applying Mutation on Shubert Function:*, the floating point representation is used for encoding this problem with two genes, variables, in each individual. That means 2-dimension Shubert function is used. This function has 760 local minima, 18 of which are global minima with value -186.73067. Its object function is

$$f(x_1,x_2) = \sum_{j=1}^{5} j \cdot \cos[(j+1)x_1 + j]$$

$$\bullet \sum_{j=1}^{5} j \cdot \cos[(j+1)x_2 + j]$$

$$\text{where} \quad -10 \le x_i \le 10 \quad i = 1,2$$

Since this is a two dimension function, with only two variables, the following mutation types are used:

- *Boundary:* a gene is selected and replaced randomly by the upper (10) or lower (-10) bound.
- *Uniform random*: a gene is selected randomly and replaced by a random number from the interval of [-10.0 ... 10.0].
- *Non-uniform*: If s_v^t= $<v_1,…, v_m>$ is a chromosome, t is the generation number, and the element v_k was selected for this mutation, the result is a vector s_v^{t+1} = $<v_1,…,v_k',…,v_m>$, where

$$v_k'= \begin{cases} v_k + \bigtriangleup (t, UB\text{-}v_k) & \text{if a random digit is 0} \\ v_k - \bigtriangleup (t, v_k\text{-}LB) & \text{if a random digit is 1} \end{cases}$$

and LB and UB are lower and upper domain bounds of the variable v_k. In addition, the following function is used:

$$\bigtriangleup (t, y) = y \cdot (1\text{-}r (1\text{-}t/T) b)$$

where r is a random number from[0..1], T is the maximum generation number, and b is a system parameter determining the degree of dependency of the iteration number .

V.   TEST AND RESULTS

After running the program with a population size of N=50 for each type of mutation, we get the results represented in tables 2, 3, and 4 for TSP, 0/1 Knapsack, and Shubert function problems respectively.

Table 2. Sample Output per Mutation Type Applied to TSP

	Reciprocal	Inversion	Insertion	Displace	Boundary	Uniform
Mean	1818.8	1791.9	1718.2	1967.5	2168.9	2071.6
Std Div	429.7	301.5	138.3	370.0	470.5	473.8

Table 3. Sample Output per Mutation Type Applied to 0/1 Knapsack

	Flip Bit	Boundary	Non-Uniform	Inversion	Insertion	Displace
Mean	1514.8	1508.2	1519.9	1538.6	1520.1	1530.0
Std Div	17.6	6.7	27.6	90.1	28.7	47.0

Table 4. Sample Output per Mutation Type Applied to Shubert $f(x_1, x_2)$

.

	Boundary	Uniform	Non-Uniform
Mean	1002.2	2078.5	1002.88
Std Div	1.7	7.3	4.0

Now, for evaluating the effectiveness of each mutation type applied to TSP, two hypotheses are considered:
First Hypothesis (H0): $M_{insertion} = M_{inversion} = M_{reciprocal} = M_{displacement} = M_{uniform} = M_{boundary}$
Second Hypothesis (H1): $M_{insertion} <= M_{inversion} <= M_{reciprocal} <= M_{displacement} <= M_{uniform} <= M_{boundary}$
where M_X in is the mean of the number of generations required to reach the desired solution using the X mutation.

In H0 we assume that all types of mutation are having the same mean M, but in H1 we assume that there is a difference between them such that the mean in the first type is less than, better, the mean in the second type and so on. Now, we apply the Jonckheere-Terpstra test of the statistical package SPSS on the collected data for TSP to either accept or reject the above hypotheses. The results in Table 5 give some statistics. The very important part of these results is the value of P (in the last line). Since P = 0.000 which is < 0.01, we reject H0 and accept H1. This means that the proposed order of mutation types in H1 is correct.

Table 5. Results of Jonckheere-Terpsta Test on TSP

Parameter	Value
Number of Levels	6
Population Siz	300
Observed J-T Statistic	25605.500
Mean J-T Statistic	18750.000
Std. Deviation of J-T Statistic	855.730
Std. J-T Statistic	8.011
Asymp. Sig. (2-tailed)	0.000

For evaluating the effectiveness of each mutation type applied to 0/1 Knapsack, two hypotheses are considered:
First Hypothesis (H0): $M_{boundary} = M_{FlipBit} = M_{nonUniform} = M_{insertion} = M_{displacement} = M_{inversion}$
Second Hypothesis (H1): $M_{boundary} <= M_{FlipBit} <= M_{nonUniform} <= M_{insertion} <= M_{displacement} <= M_{inversion}$

After applying the Jonckheere-Terpstra test, we get the results shown in Table 6. Since P = 0.016 which is > 0.01, we reject H1 and accept H0.

Table 6. Results of Jonckheere-Terpsta Test on 0/1 Knapsack

Parameter	Value
Number of Levels	6
Population Siz	300
Observed J-T Statistic	20801.500
Mean J-T Statistic	18750.000
Std. Deviation of J-T Statistic	854.717
Std. J-T Statistic	2.400
Asymp. Sig. (2-tailed)	0.016

Finally, to evaluate the effectiveness of each mutation type applied to Shubert Function, the following two hypotheses are considered:
First Hypothesis (H0): $M_{boundary} = M_{Non-Uniformt} = M_{Uniform}$
Second Hypothesis (H1): $M_{boundary} <= M_{Non-Uniform} <= M_{Uniform}$
After applying the Jonckheere-Terpstra test, we get the results shown in Table 7. Accordingly, since P = 0.000 which is < 0.01, we reject H0 and accept H1.

Table 7. Results of Jonckheere-Terpsta Test on Shubert $f(x_1, x_2)$

Parameter	Value
Number of Levels	6
Population Siz	300
Observed J-T Statistic	20801.500
Mean J-T Statistic	18750.000
Std. Deviation of J-T Statistic	854.717
Std. J-T Statistic	2.400
Asymp. Sig. (2-tailed)	0.016

VI. CONCLUSIONS AND FUTURE WORK

Genetic algorithms are an effective way to solve many problems especially NP-hard problem. In this paper, genetic algorithms were used to solve TSP, 0/1-Knapsack problem and Shubert Function. Mutation is one of the important operators of genetic algorithms since the type of mutation used often has great effects on the results. The study shows that insertion mutation is the best suite for TSP, Boundary and non-uniform mutations are the best to use for Shubert Function problem, but for 0/1 knapsack problem all mutation types used gave nearly the same result. For future work, other NP problems can be solved with genetic algorithms, and new mutations can be obtained by combining two or more types of mutations.

REFERENCES

[1] M. Fangfang, and L. Han, "An Algorithm in Solving the TSP Based on the Improved Genetic Algorithm," 1st IEEE International Conference on Information Science and Engineering (ICISE), 2009, pp. 106-108.

[2] Y.Yi, and Q. Fang, "The improved hybrid genetic algorithm for solving TSP based on Handel-C", 3rd IEEE International Conference on Advanced Computer Theory and Engineering (ICACTE), vol. 3, 2010, pp. 330-333.

[3] Z. Tao, "TSP Problem Solution Based on Improved Genetic Algorithm", 4th IEEE International Conference on Natural Computation, vol. 1, 2008, pp. 686-690.

[4] T. Hong, W. Lin, S. Liu, and J. Lin, "Experimental analysis of dynamic migration intervals on 0/1 knapsack problems", IEEE Congress on Evolutionary Computation, 2007, pp. 1163-1167.

[5] L. Wang, J. Zhang, and H. Li, "An Improved Genetic Algorithm for TSP", IEEE International Conference on Machine Learning and Cybernetics, vol. 2, 2007, pp. 925-928

[6] J. Lu, N.Fang, D. Shao, and C. Liu, "An Improved Immune-Genetic Algorithm for the Traveling Salesman Problem", 3rd IEEE International Conference on Natural Computation, 2007, pp. 297-301.

[7] Y. Liu, and J. Huang, "A Novel Genetic Algorithm and Its Application in TSP ", IEEE IFIP International Conference on Network and Parallel Computing, 2008, pp. 263-266.

[8] H. Ishibuchi, and K. Narukawa, "Performance evaluation of simple multiobjective genetic local search algorithms on multiobjective 0/1 knapsack problems", IEEE Congress on Evolutionary Computation, vol. 1, 2004, pp. 441-448.

[9] D.S. Vianna, and J.E.C. Arroyo, "A GRASP algorithm for the multi-objective knapsack problem", 24th IEEE International Conference of the Chilean Computer Science Society, 2004, pp. 69-75.

[10] H.H. Yang, S.W. Wang, H.T. Ko, and J.C. Lin, "A novel approach for crossover based on attribute reduction - a case of 0/1 knapsack problem", IEEE International Conference on Industrial Engineering and Engineering Management, 2009, pp. 1733-1737.

[11] C.L. Mumford, "Comparing representations & recombination operators for the multi-objective 0/1 knapsack problem", The IEEE 2003 Congress on Evolutionary Computation, vol. 2, 2003, pp. 854-861.

[12] S.N. Mohanty, and R. Satapathy, "An evolutionary multiobjective genetic algorithm to solve 0/1 Knapsack Problem", 2nd IEEE International Conference on Computer Science and Information Technology, 2009, pp. 397-399.

[13] S. Kaystha, and S. Agarwal, "Greedy genetic algorithm to Bounded Knapsack Problem", 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT), vol. 6, 2010, pp. 301-305.

[14] W. Xuan, W. Shao-song, X. Li, "Solving Shubert Function Optimization Problem by Using Thermodynamics Evolutionary Algorithm", IEEE International Conference on Biomedical Engineering and Computer Science (ICBECS), 2010, pp. 1-4.