Evaluating the Effectiveness of Mutation Operators on the Behavior of Genetic Algorithms Applied to Non-deterministic Polynomial Problems

Basima Hani F. Hasan
Department of Computer Science
Yarmouk University, 21163 Irbid, Jordan
E-mail: basmah@yu.edu.jo

Moutaz Saleh M. Saleh
Department of Computer Science & Engineering
Qatar University, 2713 Doha, Qatar
E-mail: moutaz.saleh@qu.edu.qa

Keywords: evaluation, genetic, mutation operator, TSP, 0/1 knapsack problem, Shubert function, linear system

Received: April 25, 2011

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 them 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 evaluates the effect of applying well known mutation operators on selected 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, Shubert function, and system of linear equations.

1 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. 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 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 mutation types depending on the representation used are: Flip bit, Boundary, Non-uniform, Inversion, Insertion and Displacement.

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-dimensional Shubert function can be given as:

\[ f(x_i) = \sum_{j=1}^{n} j \cdot \cos((j+1)x_i + j) \]

where \(-10 \leq x_i \leq 10\)

For n-dimension Shubert function, there are \(n^{*}\text{pow}(3, n)\) global solutions. The object function for n-dimension Shubert function can then be given as:

\[ f'(x) = \prod_{i=1}^{n} \sum_{j=1}^{2} j \cdot \cos((j+1)x_i + j) \]

where \(-10 \leq x_i \leq 10, \ i=1, \ldots, n\)

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.

Many problems lend themselves to being solved with systems of linear equations. However, solving systems of linear equations is a common computational problem well known to mathematicians, scientists and engineers. Several algorithms exist for solving this problem. But, when the equations contain interval coefficients, i.e. intervals in which the desired coefficient values are known to lie, the problem may not be solvable in any reasonable sense. In fact, it has been shown that the general problem of solving systems of linear equations with interval coefficients is NP-hard which is extremely difficult to be solved. Hence, 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: Boundary mutation, Uniform random mutation, Non-uniform mutation, Reciprocal mutation, Inversion mutation, Insertion mutation, and Displacement mutation.

2 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 evolitional iterations to reach an optimal solution can be significantly reduced. Liu and Huang [7] proposed a novel genetic algorithm to overcome the deficiencies 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.

The problem of solving systems of linear equations with use of AI based approaches has been studied by many researches for decades. Different definitions for the solution of such problem have been considered and various AI techniques have been successfully developed [15] [16] [17]. Recently, the research in [18] aimed to approximate the exact algebraic solution of this problem through minimizing its cost function. To do so, two different AI approaches were adopted: the neural networks (NN) based approach and the genetic algorithms (GA) based one. The results shows that it will be interesting to combine both GA based and NN based approaches into one single method. GA can be used at the beginning phase for global search, whereas the NN based technique can be used in the final, local search phase to improve the solution obtained by GA.

3 GA Operation

GA is an iterative procedure that consists of a constant population size of individuals which are decoded and evaluated according to a 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. For our research work, we will adopt the GA procedure steps to solve four well known NP problems: Traveling Salesman Problem (TSP), 0/1 Knapsack problem, Shubert function,
and system of linear equations. To do so, Table 1 shows how these steps are applied on the four problems.

![Figure 1: A general GA procedure.](image)

Table 1: Choice of parameters for each GA step.

<table>
<thead>
<tr>
<th>GA Step</th>
<th>TSP</th>
<th>01 Knapsack</th>
<th>Subset Function</th>
<th>System of Linear Equations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encoding</td>
<td>integer</td>
<td>binary</td>
<td>floating point</td>
<td>floating point</td>
</tr>
<tr>
<td>Initial Population</td>
<td>Heuristic (greedy algorithm from different initial guesses)</td>
<td>15 tons</td>
<td>Two genes in each individual</td>
<td>Generated randomly within specified interval</td>
</tr>
<tr>
<td>Parameters</td>
<td>Probability of crossover: 60%</td>
<td>Probability of mutation: 5%</td>
<td>Maximum number of iterations: 5000</td>
<td></td>
</tr>
<tr>
<td>Evaluation Function</td>
<td>total cost of travel between cities</td>
<td>summation of the fitness of the genes selected</td>
<td>smallest value it gets for best</td>
<td>how far the obtained solutions are from the current one</td>
</tr>
<tr>
<td>Selection Operator</td>
<td>roulette wheel</td>
<td>roulette wheel</td>
<td>roulette wheel</td>
<td>roulette wheel</td>
</tr>
<tr>
<td>Crossover Operator</td>
<td>heuristic</td>
<td>single point</td>
<td>single point</td>
<td></td>
</tr>
<tr>
<td>Mutation Operator</td>
<td>Reciprocal</td>
<td>Inversion</td>
<td>Inversion</td>
<td>Uniform</td>
</tr>
<tr>
<td></td>
<td>Inversion</td>
<td>Inversion</td>
<td>Uniform</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Inversion</td>
<td>Inversion</td>
<td>Uniform</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Uniform</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Boundary</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Uniform random</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Non-uniform</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reciprocal</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Boundary</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Uniform random</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Non-uniform</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Inversion</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Inversion</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Uniform</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Insertion</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Displacement</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Uniform</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Uniform</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4 Mutations Operator

Text of the conclusion 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.

4.1 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.

![Figure 2: Boundary mutation.](image)

- **Displacement**: a sub tour and a place to be inserted in it are selected randomly as shown in Figure 3.

![Figure 3: Displacement mutation.](image)

- **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.
4.2 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.

![Figure 4: Flip mutation.](image)

- **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.

![Figure 5: Non-uniform mutation.](image)

If \(s_1^t = <v_1, ..., v_m>\) is a chromosome (t is the generation number) and the element \(v_k\) was selected for this mutation, the result is a vector \(s_{1}^{t+1} = <v_1', ..., v_k', ..., v_m'>\), where:

\[
v_k' = \text{mutate}(v_k, \text{pos})
\]

\(\text{LB and UB are lower and upper domain bounds of the variable } v_k\). In addition, the following function is used:

\[
y(t, y) = (1 - r(1 - T)b) \sum_{j=1}^{n} j \cdot \cos[(j + 1)x_i + j]
\]

where \(-10 \leq x_i \leq 10\) \(i = 1, 2\)

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 \(s_0 = <v_1, ..., v_m>\) is a chromosome, \(t\) is the generation number, and the element \(v_k\) was selected for this mutation, the result is a vector \(s_{1}^{t+1} = <v_1', ..., v_k', ..., v_m'>\), where,

\[
\begin{align*}
v_k' &= v_k + \Delta (t, UB-v_k) & \text{if a random digit is 0} \\
v_k' &= v_k - \Delta (t, v_k-LB) & \text{if a random digit is 1}
\end{align*}
\]

4.3 Applying Mutation on Shubert Function

for best practices, floating point representation is commonly used for encoding Shubert function with two genes, variables, in each individual. Accordingly, a 2-}

dimension Shubert function is adopted. This function has 760 local minima, 18 of which are global minima with value -186.73067. Its object function is:

\[
f(x_i, x_j) = \sum_{j=1}^{n} j \cdot \cos[(j + 1)x_i + j]
\]

4.4 Applying Mutation on Linear System Equation

The floating-point representation is used for encoding individuals in such system with every gene represents the value of one variable. The individuals of the initial population can be generated randomly within a specified interval for each variable, and the evaluation function is used to indicate how far the obtained solution from the correct one. For the general system of linear equations:

\[
a_{11}X_1 + a_{12}X_2 + ... + a_{1n}X_n = b_1 \\
a_{21}X_1 + a_{22}X_2 + ... + a_{2n}X_n = b_2 \\
\vdots \\
a_{n1}X_1 + a_{n2}X_2 + ... + a_{nn}X_n = b_n
\]

suppose that the vector \(V = (v_1, v_2, ..., v_n)\) represents a solution. Then the evaluation function will be:
1.0
\[ \sum_{j=1}^{n} \left( \sum_{i=1}^{n} a_{ij} x_j \right) - b_i \]

Eventually, the following mutation types will be applied for this linear equation system: Boundary, Uniform random, Non-uniform, Reciprocal, Inversion, Insertion, and Displacement.

### 5 Test and Results

After running the program 50 times for each type of mutation, we get the results represented in tables 2, 3, 4, and 5 for TSP, 0/1 Knapsack, Shubert function, and linear equation system problems respectively.

#### Table 2: Output per mutation type applied to TSP.

<table>
<thead>
<tr>
<th>Reciprocal</th>
<th>Inversion</th>
<th>Insertion</th>
<th>Displacement</th>
<th>Boundary</th>
<th>Uniform</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>1811.8</td>
<td>1709.9</td>
<td>1712.8</td>
<td>1967.5</td>
<td>2168.9</td>
</tr>
<tr>
<td>Std Dev</td>
<td>429.7</td>
<td>301.5</td>
<td>193.3</td>
<td>270.0</td>
<td>470.5</td>
</tr>
</tbody>
</table>

#### Table 3: Output per mutation type applied to 0/1 Knapsack.

<table>
<thead>
<tr>
<th>Flip Bit</th>
<th>Boundary</th>
<th>Non-Uniform</th>
<th>Inversion</th>
<th>Insertion</th>
<th>Displacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>1514.8</td>
<td>1500.2</td>
<td>1515.9</td>
<td>1533.6</td>
<td>1520.1</td>
</tr>
<tr>
<td>Std Dev</td>
<td>78.7</td>
<td>70.7</td>
<td>67.6</td>
<td>74.7</td>
<td>78.7</td>
</tr>
</tbody>
</table>

#### Table 4: Output per mutation type applied to Shubert function (x1, x2).

<table>
<thead>
<tr>
<th>Boundary</th>
<th>Uniform</th>
<th>Non-Uniform</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>1002.2</td>
<td>2078.5</td>
</tr>
<tr>
<td>Std Dev</td>
<td>1.7</td>
<td>7.3</td>
</tr>
</tbody>
</table>

#### Table 5: Output per mutation type applied to Linear Equation System.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Levels</td>
<td>6</td>
</tr>
<tr>
<td>Population Size</td>
<td>300</td>
</tr>
<tr>
<td>Observed J-T Statistic</td>
<td>25605.500</td>
</tr>
<tr>
<td>Mean J-T Statistic</td>
<td>18750.000</td>
</tr>
<tr>
<td>Std. Deviation of J-T Statistic</td>
<td>855.730</td>
</tr>
<tr>
<td>Std. J-T Statistic</td>
<td>8.011</td>
</tr>
<tr>
<td>Asymp. Sig. (2-tailed)</td>
<td>0.000</td>
</tr>
</tbody>
</table>

#### Table 6: Results of Jonckheere-Terpsta Test on TSP.

#### Table 7: Results of Jonckheere-Terpsta Test on 0/1 Knapsack.

#### Table 8: Results of Jonckheere-Terpsta Test on Shubert function (x1, x2).

To evaluate the effectiveness of each mutation type applied to the linear equation system, the following two hypotheses are considered:

**First Hypothesis (H0):**

\[ M_{\text{nonUniform}} = M_{\text{Boundary}} = M_{\text{Reciprocal}} = M_{\text{Displacement}} = M_{\text{inversion}} = M_{\text{Insertion}} \]

**Second Hypothesis (H1):**

\[ M_{\text{nonUniform}} \leq M_{\text{Boundary}} \leq M_{\text{Reciprocal}} \leq M_{\text{Displacement}} \leq M_{\text{inversion}} \leq M_{\text{Insertion}} \]

After applying the Jonckheere-Terpstra test for TSP, we get the results shown in Table 6. 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.

To evaluate the effectiveness of each mutation type applied to 0/1 Knapsack, the following two hypotheses are considered:

**First Hypothesis (H0):**

\[ M_{\text{Boundary}} = M_{\text{FlipBit}} = M_{\text{Non-Uniform}} = M_{\text{Insertion}} \]

**Second Hypothesis (H1):**

\[ M_{\text{Boundary}} \leq M_{\text{FlipBit}} \leq M_{\text{Non-Uniform}} \leq M_{\text{Insertion}} \leq M_{\text{Displacement}} \leq M_{\text{inversion}} \]

After applying the Jonckheere-Terpstra test, we get the results shown in Table 7. Since \( P = 0.016 \) which is > 0.01, we reject H1 and accept H0.

To evaluate the effectiveness of each mutation type applied to Shubert, the following two hypotheses are considered:

**First Hypothesis (H0):**

\[ M_{\text{Boundary}} = M_{\text{Non-Uniform}} = M_{\text{Uniform}} \]

**Second Hypothesis (H1):**

\[ M_{\text{Boundary}} \leq M_{\text{Non-Uniform}} \leq M_{\text{Uniform}} \]

After applying the Jonckheere-Terpstra test, we get the results shown in Table 8. Accordingly, since \( P = 0.000 \) which is < 0.01, we reject H0 and accept H1.

To evaluate the effectiveness of each mutation type applied to the linear equation system, the following two hypotheses are considered:

**First Hypothesis (H0):**

\[ M_{\text{nonUniform}} = M_{\text{Boundary}} = M_{\text{Reciprocal}} = M_{\text{Displacement}} = M_{\text{inversion}} = M_{\text{Insertion}} \]

**Second Hypothesis (H1):**

\[ M_{\text{nonUniform}} \leq M_{\text{Boundary}} \leq M_{\text{Reciprocal}} \leq M_{\text{Displacement}} \leq M_{\text{inversion}} \leq M_{\text{Insertion}} \]


Second Hypothesis (H1): $M_{\text{nonUniform}} \leq M_{\text{Boundary}} \leq M_{\text{displacement}} \leq M_{\text{reciprocal}} \leq M_{\text{inversion}} \leq M_{\text{insertion}} \leq M_{\text{uniform}}$

After applying the Jonckheere-Terpsta test, we get the results shown in Table 9. Accordingly, since $P = 0.000$ which is < 0.01, we reject H0 and accept H1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Levels</td>
<td>7</td>
</tr>
<tr>
<td>Population Size</td>
<td>350</td>
</tr>
<tr>
<td>Observed J-T Statistic</td>
<td>38576.000</td>
</tr>
<tr>
<td>Mean J-T Statistic</td>
<td>26250.000</td>
</tr>
<tr>
<td>Std. Deviation of J-T Statistic</td>
<td>1082.035</td>
</tr>
<tr>
<td>Std. J-T Statistic</td>
<td>11.392</td>
</tr>
<tr>
<td>Asymp. Sig. (2-tailed)</td>
<td>0.000</td>
</tr>
</tbody>
</table>

Table 9: Results of Jonckheere-Terpsta Test on Linear Equation System

6 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 Shubert Function, and linear equation system. Mutation is one of the important operators of genetic algorithms since the type of mutation used often has great effects on the results. The research study shows that insertion mutation is the best suite for TSP, Boundary and non-uniform mutations are the best to use for Shubert function and linear equation system, but for 0/1 knapsack problem 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 mutation operators.

References


