Solving NP-Complete Problems Using Genetic Algorithms

Bander Hassan Arabi
Research and Development Department,
Deanship of Distance Learning, King Abdulaziz University,
Kingdom of Saudi Arabia, P.O. Box 80254.
barabi@kau.edu.sa

Abstract—Genetic Algorithms are designed to find the accuracy of approximated solutions in order to perform as effectively as possible. This paper presents a new way for genetic algorithm to solve NP-Complete problem. We study genetic algorithm to find an optimal solution for instances of the Traveling Salesman Problem. To overcome this solution, we have to see what is the shortest path that satisfies all of these conditions? We review the paradigms of design of genetic algorithm and demonstrate how they can be applied for the Traveling Salesman Problem. The experiments are presented in which the performance of genetic algorithm is compared to that of Exhaustive Searches.

Keywords-component; NP-Complete Problem; Genetic Algorithm; Traveling Salesman Problem;

I. INTRODUCTION

The Traveling Salesman Problem (TSP) is a classic computer science problem of some complexity, though the nature of the problem is quite simple. A salesman or 'nomadic man' wants to visit a number of scattered cities. He knows the distances between these cities. He must visit every city only once and return to the home to his origination point. What is the shortest path that satisfies all of these conditions? This problem is classified as an NP-Complete problem. Problems of this class, also known as hard problems, can be solved in a period of time that is bounded by a polynomial function about the problem with the assumption of being able to guess perfectly. While there have been many approaches attempted to solve problems of this nature, the application of genetic algorithm theory has had a particular significance.

Genetic Algorithms are designed to find the accuracy of approximated solutions in order to perform as effectively as possible. It is classified as a global heuristics search, meaning that Genetic Algorithms find correct solutions that are not necessarily the best solutions. This requires searching through all possible solutions in a theoretically and practically functional manner that is improvable to find the best solution and yet still does. Genetic Algorithms get their name based on a kind of evolutionary computation in a similar way to evolutionary biology in utilizing inheritance, selection, and crossover. This allows finding the best selection available from a group for a certain design. This selection is performed based on Darwinian theory such that the best features of a solution are passed through successive reproduction so that as this cycle continues, better and better solution-seeds are produced [1].

The goal of this paper is to use of Genetic Algorithms as finding solutions for configurations of the Traveling Salesman Problem. To overcome this task we will solve Traveling Salesman Problem, "symmetric", closed TSP using genetic algorithms. Specifically, two different selection methods were considered: "Roulette selection" and "Rank selection".

This paper consists of four sections. Section 1 discusses the methodology of NP-Complete problems and project's goals Section 2 presents genetic algorithm concepts, including operators, and models. Section 3 illustrates the application of the genetic algorithm. In section 4, we discussed the system that was implemented and section 5 concludes the work.

II. METHODOLOGY

The Traveling Salesman Problem (TSP) is one of the benchmark and oldest problems in computer science and operations research. It can be stated as [2]:

There exist a network with 'n' nodes (or cities) with 'node 1' as 'headquarters' and a computed travel-cost (or distance or travel-time) matrix $C = [c_{ij}]$ of order 'n' comprised of ordered node pairs $(i, j)$. The problem here is to find a least-cost Hamiltonian cycle.

Based on the structure of the cost matrix, TSPs are classified into two groups: symmetric and asymmetric. The TSP is symmetric if $c_{ij} = c_{ji}$ for all $i, j$ and asymmetric otherwise. For an n-city asymmetric TSP, there are $(n-1)!$ possible solutions, one or more of which gives the minimum cost. For an n-city symmetric TSP, there are $(n - 1)! / 2$ possible solutions along with their reverse cyclic permutations having the same total cost. In either case the number of solutions becomes extremely large for even a moderately large 'n', making an exhaustive search impracticable. [2]

There are mainly three reasons why the TSP has attracted the attention of many researchers and remains an active area of research.

Firstly, a large number of real-world problems can be modeled by the TSP. Secondly, it was proved to be an NP-
Complete problem. Thirdly, NP-Complete problems are intractable in the sense that no one has found any provably-efficient way of solving them for large problem sizes. Additionally, NP-complete problems are known to be more or less equivalent to each other, meaning that if one knows how to solve one of them, one could solve others [2]. Methods that provide an exact optimal solution to the problem are called exact methods. An implicit way of solving the TSP is simply to list all feasible solutions, evaluate their objective function values, and pick the best. However, it is obvious that this “exhaustive search” is grossly inefficient and impractical because of the vast number of possible solutions even for a problem of moderate size. Since practical applications require solving larger problems, emphasis has shifted from finding exact, optimal solutions to the TSP to obtaining ‘heuristically-good’ solutions in reasonable time and ‘establishing the degree of goodness’ [2].

III. GENETIC ALGORITHMS

Genetic algorithms are implemented as a computer simulation in which a population of abstract representations (called chromosomes) of candidate solutions (called individuals) to an optimization problem evolve toward better solutions. The evolution usually starts from a population of randomly generated individuals. In each generation, the fitness of every individual in the population is evaluated, multiple individuals are stochastically-selected from the current population (based on their fitness), and modified (recombined and possibly randomly-mutated) to form a new population. The new population is used in the next iteration of the algorithm. Genetic algorithms find application in bioinformatics, computational science, engineering, economics, chemistry, manufacturing, mathematics, physics, and other fields. This chapter presents the concepts of genetic algorithms and their features. These concepts include advantages, models, vocabulary, and operators of genetic algorithms.

There are several advantages of applying GAs to optimization problems. The most important ones include adaptability, robustness, and flexibility.

- Adaptability. GAs do not have many mathematical requirements regarding optimization problems. Due to their evolutionary nature, GAs search through solutions without paying attention to any specific inner-workings of the problem. Thus, GAs can handle any kind of objective function and any kind of constraint (i.e. linear or nonlinear) defined for discrete, continuous, or mixed-search spaces [4][10][11][12][13].

- Robustness. The use of evolutionary operators makes GAs very effective in performing a global search (in probability) while most conventional heuristics usually perform a local search. It has been proven by many studies that GAs are more efficient and more robust in locating optimal solutions and reducing computational effort than other conventional heuristics [4].

- Flexibility. GAs provides great flexibility to hybridize with domain-dependent heuristics in making an efficient implementation for a specific problem. GAs also have disadvantages. They operate very slowly and cannot always find exact solutions though they always find the best solutions. For further details about advantages and disadvantages, see [4].

Terminologies for Genetic Algorithms:

- Chromosome: contains a solution in the form of genes
- Gene: part of a chromosome that contains part of a solution (e.g. "1 6 7 4 3" is a chromosome containing 1, 6, 7, 4 and 3 as its genes)
- Allele: a single value of a gene
- Individual: synonymous with chromosome
- Population: set of chromosomes
- Fitness function: provides the mechanism to evaluate each individual

Termination criterion: criterion checked after each generation to determine whether to continue or stop the search. Common termination criteria are:

- A solution has been found that satisfies minimum criteria.
- A fixed number of generations has been reached.
- An allocated budget (computation time/money) has been reached.
- The highest ranking solution’s fitness is approaching or has reached a plateau such that successive iterations no longer produce better results.
- A manual determination is made to stop the search.
- Any combinations of the above can make a stronger, long-lasting criterion.

Encoding: the way in which a solution is converted into a binary representation system, such as in this case encoding GA chromosomes to the TSP Path and vice versa

Decoding: the way in which a solution is interpreted into human-comprehensible by reversing the encoding [4].

General Genetic Algorithm Model

The block diagram of the general GA model is depicted in Figure 1. First, the initial population is generated randomly. Next, the selection operator is applied to select chromosomes based on their fitness values. The higher the fitness value of a chromosome, the more likely its selection becomes. The selection of chromosomes continues until the mating pool size is equal to the population size. Then, crossover is applied over the chromosomes in the mating pool to produce offspring. Finally, the mutation operator is applied to produce the next generation. The termination criteria are then evaluated regarding the new generation. If no criterion is met, the next generation replaces the population and the process is repeated.
IV. DEVELOPMENT

In the program implemented for this work, the Traveling Salesman Problem is represented as a matrix of distances with resolution SIZE x SIZE, where SIZE is equal to the number of cities in the corresponding TSP. In this pathlen matrix, each element i,j contains the distance between cities i and j. Because the TSP is symmetric here, pathlen is also symmetric in that \( P_{ij} = P_{ji} \). Obviously, the primary diagonal of this matrix consists only of 0 because the distance from a city to itself is 0. This is immaterial because the constraints of the problem forbid double-visiting a city, even from itself. All other elements are generated randomly and contain integer values in the range \((1:20)\). In generating the matrix, the diagonal is first initialized to 0. Only \( n*(n-1)/2 \) random numbers are generated due to the symmetry of the matrix. Table I. Shows an example instance of the matrix.

<table>
<thead>
<tr>
<th>j=0</th>
<th>i=0</th>
<th>i=1</th>
<th>i=2</th>
<th>i=3</th>
<th>i=4</th>
<th>i=5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>17</td>
<td>19</td>
<td>3</td>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>3</td>
<td>10</td>
<td>12</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>16</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>16</td>
<td>0</td>
<td>3</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>12</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>13</td>
<td>6</td>
<td>11</td>
<td>18</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

For solving the TSP with a GA, the genes of chromosomes are represented with integer values. Each integer denotes a city. The number of chromosomes is equal to the number of cities. In the program, each chromosome is a single-dimensional array of integers. In the previous example, the chromosome size is equal to 6 genes. Each gene takes on values between 1 and 6, inclusively. This algorithm uses these global values:

- \( \text{SIZE} \) - size of the problem (count of cities, size of each chromosome)
- \( \text{path} \) - array that contains the current path (solution, permutation)
- \( \text{dirs} \) - array that contains directions for each number in the permutation

for each chromosome \( \text{path}[i], i=0,1...,\text{POPUL_SIZE} \) in population do:
  initialize chromosome's genes with -1 value
  \( \text{path}[i][j]=-1, j=0,1,...,\text{SIZE} \);
  for each \( j=0 \) to \( \text{SIZE}-1 \) do:
    select random value for \( \text{pos} \) in range \([0; \text{SIZE}-j]\)
    for each \( k=0 \) to \( \text{SIZE}-1 \) do:
      if \( \text{path}[i][k]=-1 \) then:
        if \( \text{pos} == 0 \) then:
          \( \text{path}[i][k]=j; \)
        else
          \( \text{pos}=\text{pos}-1; \)
      end
    end
  end
end

for each chromosome \( \text{path}[i], i=0,1...,\text{POPUL_SIZE} \) in population do:
  Reset sum to zero: \( \text{sum}=0; \)
  for each \( j=0 \) to \( \text{SIZE}-1 \) do:
    \( a=\text{path}[i][j]; \)
    \( b=\text{path}[i][j+1]; \)
    \( \text{sum}=\text{sum}+\text{pathlen}[a][b]; \)
    \( a=0; \)
    \( b=\text{SIZE}-1; \)
    \( \text{sum}=\text{sum}+\text{pathlen}[a][b]; \)
    \( f[i]=\text{sum}; \)
end

The fitness function used for the TSP can be modeled as:

\[
 f = \sum_{i=1}^{n} W_{i \mod n + 1} 
\]

Where: \( i = 1 \) gene (city) per chromosome, \( n \) is the number of chromosomes (paths), \( W \) is the matrix of distances between cities.

V. EXPERIMENTS, RESULTS AND ANALYSIS

In order to investigate the comparative-effectiveness of comparable algorithms and identify the most effective parameters for the implemented genetic algorithms, it was necessary to perform several test runs with different initial conditions.

The environment and technologies utilized should be noted as:
- The programs were implemented using the C++ and C# languages.
- For the testing, we used this software/hardware
configuration:

- OS: Windows 7
- CPU: Intel Core 2 2.1 GHz
- RAM: 4 Gb

The following parameters are initially controllable:
- Size: the number of cities (an increased size indicates a harder problem with a longer solution search time)
- Population size: controls the size of the population in GA
- N: count of generations for GA (used for termination condition)
- Selection method: control whether ‘roulette’ or ‘rank’ will be used for selection
- Mutation probability: controls how frequently mutations will be applied

Where it was possible, the same graph (distance matrix) was used consistently for each test cycles to reduce the effects of randomization on the conclusions. It was determined that any bias in the random numbers generated for the utilized graph that might lead to favor being given to any particular algorithm or set of parameters was a fair tradeoff given the potentially-overpowering effects of random-number selection influence. Keeping the same initial conditions allows reasonably fair testing using different parameters.

In executing the tests, an array of distance matrices was first created, initialized, and stored. These matrices were utilized throughout several tests to yield a decent cross-section of results. Because of the non-deterministic nature of the program's execution, time counters were instrumented throughout the code to measure the time needed for the evolutionary process, especially for an increased maximum number of generations.

**Test 1**

Compare Exhaustive Search Method “ESM” and Genetic Algorithms “GA” for the best path and time to solution.

**Background:**

From theory, it is known that an ESM algorithm always finds the best solution, but that it is applicable only for a small number of cities due to having a huge algorithm complexity for a larger number of cities. On the other hand, Genetic Algorithms provide close to optimal solutions with much less algorithm complexity. In this test, the complexity of each algorithm is indicated along with the time necessary to find solutions. The primary variable of concern here is the number of cities. The path results of each algorithm are also compared for effectiveness of solution. This test indicates whether ESM or GA is more effective for a given set of conditions for solving the TSP, primarily based on the number of cities visited.

- Initial conditions:
- Size: 8, 9, 10, 11, 12
- Population size: 20
- N (for GA only): 50

Test results are provided in Table II. and Figures 2. and 3.

**TABLE II. WORKING TIME AND BEST RESULT FOR ESM & GA ALGORITHMS**

<table>
<thead>
<tr>
<th>Count of towns</th>
<th>ESM work time, sec</th>
<th>ESM result</th>
<th>GA work time, sec</th>
<th>GA result</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0.01</td>
<td>75</td>
<td>0.52</td>
<td>84</td>
</tr>
<tr>
<td>9</td>
<td>0.11</td>
<td>82</td>
<td>0.55</td>
<td>82</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>102</td>
<td>0.54</td>
<td>112</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>105</td>
<td>0.58</td>
<td>111</td>
</tr>
<tr>
<td>12</td>
<td>135</td>
<td>127</td>
<td>0.57</td>
<td>145</td>
</tr>
</tbody>
</table>

**Analysis.**

Based upon the test results, when the number of cities is kept relatively small, the ESM algorithm works well in comparison to the GA in that it always finds the best available solution in a shorter time than the GA. However, increasing the size of the problem changed the outcome dramatically. A small increase in the number of towns from 10 to 12 increased the execution time of the ESM algorithm by 100 fold. On the other hand, the increase in time of the GA was negligible. In terms of algorithm complexity, ESM can be classified as having exponential complexity with time of working increasing proportional to $2^n$. GA can be classified as having a close to linear complexity with time of working increasing proportional to $n$. 

![Figure 2. Time of calculation for ESM & GA](image-url)
Test 2
In this test, the ESM and GA were compared with an attempt at evaluating the lower and upper ranges of the GA for the TSP application.

Background:
The previous test indicated that ESM performs more effectively than GA for small problem sizes, but unacceptably for larger problem sizes of more than 20 due to unacceptable working time. For problem sizes in the range of [10:15], it is possible to use either method. In this case, it can be interesting to know what kind of response can be expected from GA and whether additional working time should be granted for a better solution. This requires understanding the kinds of results from GA. What are best and worst results that can be expected when using GA and what result will be most probable?

This test utilized two different problem sizes, 8 and 12, which correspond to approximately equivalent ESM and GA working times and working times where ESM begins to significantly exceed GA, respectively. A couple of tests were performed for each problem size with different distance matrices to get a more robust range of results. For each distance matrix, only one execution of ESM was performed because of the constancy of its results. However, because the GA has a factor of randomness, it was necessary to perform several executions for each distance matrix to mitigate the effects of randomness. For 100 GA executions per distance matrix, the best path of ESM and the best, average, and worst path of GA were recorded.

Initial conditions:
- Size: 8, 12
- Population size: 20
- N (for GA only): 50
- Test results are provided in Table III.

Analysis.
The tests results indicated that GA provides a solution that is 15-20% worse than the best solution, which was founded by the ESM algorithm. The worst solution found by GA was 70% worse than the ESM result. Thus, it can be decided that if the price of a worse solution is not very high and accepting a nearly double time of travel is worth the risk with a more-probable outcome being only an 18% increase, GA might be the better choice. However, if either of those conditions do not hold, ESM would likely be necessary even with a much longer working time. In time-critical situations, the tradeoffs would need to be carefully weighed. If a choice of algorithm is necessary in a safety-critical situation, such as might be needed in an aircraft, the time needed to find an ideal, exhaustively-searched solution might result in the worst outcome of all: doing nothing.

Test 3
This test was created to compare the effectiveness of different selection operations.

Background:
While previous tests focused on the effects of problem parameter variation, this test focused on changing the actual algorithm used for selection in the GA. The Roulette Wheel and Rank methods were investigated for effectiveness in the TSP application by utilizing several instances of the distance matrix for each size of problem in the set of {10, 50, 100}.

- Initial conditions:
  - Size: 10, 50, 100
  - Population size: 50
  - N (for GA only): 50

Test results are provided in Table IV. and Figure 4.

<table>
<thead>
<tr>
<th>Count of towns</th>
<th>ESM best result</th>
<th>GA worse result</th>
<th>GA best result</th>
<th>GA average result</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>85</td>
<td>141</td>
<td>92</td>
<td>98</td>
</tr>
<tr>
<td>12</td>
<td>111</td>
<td>187</td>
<td>123</td>
<td>132</td>
</tr>
</tbody>
</table>

TABLE III. WORKING TIME AND BEST RESULT FOR ESM & GA ALGORITHMS

<table>
<thead>
<tr>
<th>Count of towns</th>
<th>GA with Roulette Wheel selection</th>
<th>GA with Rank selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>145</td>
<td>163</td>
</tr>
<tr>
<td>50</td>
<td>491</td>
<td>549</td>
</tr>
<tr>
<td>100</td>
<td>1238</td>
<td>1372</td>
</tr>
</tbody>
</table>
Analysis.

The test results indicated that the Roulette Wheel method typically provided better results than the Rank method regardless of the size of problem presented. It is hypothesized that the superior results are due to a relatively large population size (50) and a relatively narrow spread of fitness values. In fact, all fitness function values were within a factor of 10 of one another. The case of inferiority for the Roulette Wheel never appeared, which is when there is a wide spread of fitness values. This seems to indicate the usage of the Roulette Wheel method in solving the TSP.

VI. CONCLUSION

This work dealt with the Traveling Salesman Problem (TSP) and its solution using Genetic Algorithms (GAs). The TSP is an important problem that has NP-complexity and a number of real-world applications, including most logistics optimizations, finding the best path for Internet traffic, and the manufacture of printed circuit boards. Based on the results generated through several tests, it was proven that GA can be used for solving the TSP in effective time and with acceptable results. One of the primary reasons to utilize GA is that it is easily scalable and distributable.

Experimentation showed that an Exhaustive Search Method can be used when the size of a problem is small, but that when the problem size increases to 12 or more, GA provides a much more acceptable working time. In terms of the selection method used for GA, the Roulette Wheel method was proven with some definiteness to be more effective than the Rank method for solving the TSP by more than 10%.

REFERENCES