New Genetic Algorithm for the Travelling Salesman Problem

Ayşe Hande Erol
Department of Industrial Engineering
Marmara University
Goztepe 34722, Turkey

Serol Bulkan
Department of Industrial Engineering
Faculty of Engineering
Marmara University
Goztepe 34722, Turkey

Abstract

Given a collection of cities and the cost of travel between each pair of them, the traveling salesman problem is to find the cheapest way of visiting all of the cities and returning to the starting point. The simplicity of the statement of the problem is deceptive, the traveling salesman problem is one of the most intensely studied problems in computational mathematics and yet no effective solution method is known for the general case. This problem is one of the most difficult problems in the NP-hard class, which implies that finding a polynomial time algorithm to solve it is unlikely. There are different methods ranging from exact to heuristic and metaheuristics. It takes a long time to have an optimum solution with exact algorithms, so several heuristics have been proposed for handling near optimum solutions. In this study, a genetic algorithm as a heuristic optimization technique and neighborhood approximation is developed. This new algorithm’s solvability in benchmark and symmetric traveling salesman problem instances is tested problems in the sense that traveling from city X to city Y costs just as much as traveling from Y to X, and results are listed.

Keywords
Heuristic methods; metaheuristic methods; neighborhood approximation; travelling salesman problem

1. Introduction

The Traveling Salesman Problem (TSP) is a well known and important combinatorial optimization problem. The goal is to find the shortest tour that visits each city in a given list exactly once and then returns to the starting city. Formally, the TSP can be stated as follows. The distances between \( n \) cities are stored in a distance matrix \( D \) with elements \( d_{ij} \) where \( i, j = 1, \ldots, n \) and the diagonal elements \( d_{ii} \) are zero. A tour can be represented by a cyclic permutation \( \Pi \) of \( \{1, 2, \ldots, n\} \) where \( \Pi(i) \) represents the city that follows city \( i \) on the tour.

The TSP is then the optimization problem to find a permutation \( \Pi \) that minimizes the length of the tour denoted by (Hahsler and Hornik, 2007):

\[
\sum_{i=1}^{n} d_{i \Pi(i)}
\]

(1)

For this minimization task, the tour length of \((n-1)!\) permutation vectors have to be compared. This results in a problem which is very hard to solve and in fact known to be NP-complete. According to Lenstra and Kan, solving TSPs is an important part of applications in many areas including vehicle routing, computer wiring, machine sequencing and scheduling, frequency assignment in communication networks. Applications in statistical data
analysis include ordering and clustering objects. For example, data analysis applications in psychology ranging from profile smoothing to finding an order in developmental data are presented by Hubert and Baker (Hubert and Baker, 1978).

In this study, a very brief overview of the TSP is given and a GA algorithm is introduced for TSP which provides an infrastructure for handling and solving TSPs. This paper is organized as follows; in Section 2, the objective is stated. In Section 3, recent studies related to different formulations and solution approaches to TSP are summarized. In Section 4, an overview of Genetic Algorithms (GAs) is given. In Section 5, GA implementation details are explained. In the next section, GA application and computational study is presented. And finally, in the last section, the paper is ended with conclusion and future research part.

2. Objectives
In this study, exact algorithms, heuristic and meta-heuristic algorithms which are used to solve NP-hard problems are surveyed. These algorithms’ solution approaches in problem instances are compared and new improved algorithms for TSP in the literature are also surveyed. The main objective is to determine and compare the recent studies and these solution techniques until now and to benchmark the research’s results in literature with new developed genetic algorithm approximation.

3. Literature Review
The TSP is known to be NP-hard. This means that no known algorithm is guaranteed to solve all TSP instances to optimality within reasonable execution time. So in addition to exact solution approaches, a number of heuristics and metaheuristics have been developed to solve problems approximately. Heuristics and metaheuristics trade optimality of the solutions that they output with execution times. They are used to find “good” quality solutions within reasonable execution times (Basu and Ghosh, 2008).

Finding the exact solution to a TSP with \( n \) cities requires to check \((n-1)!\) possible tours. To evaluate all possible tours is infeasible for even small TSP instances. Held and Karp presented the dynamic programming to find the optimal tour in 1962 (Hahsler and Hornik, 2007).

A different method, which can deal with larger instances, uses a relaxation of the linear programming problem and iteratively tightens the relaxation till a solution is found. This general method for solving linear programming problems with complex and large inequality systems is called cutting plane method and was introduced by Dantzig, Fulkerson, and Johnson in 1954. If no further cutting planes can be found or the improvement in the objective function due to adding cuts gets very small, the problem is branched into two sub-problems which can be minimized separately. Branching is done iteratively which leads to a binary tree of subproblems. Each sub-problem is either solved without further branching or is found to be irrelevant because its relaxed version already produces a longer path than a solution of another sub-problem. This method is called branch-and-cut which is a variation of the well known branch-and-bound procedure (Hahsler and Hornik, 2007).

The development of computational methods to solve the TSP is an active field of research, and Applegate et. al. proposed a comprehensive review about solving TSP. These methods can be classified into two broad categories, exact algorithms which are guaranteed to output optimal tours, and heuristics which generate good quality tours within reasonable execution time. The former category includes cutting plane algorithms proposed by Dantzig et. al. in 1954; Grötschel and Padburg in 1985; Hong in 1972, branch and bound algorithms proposed by Balas in 1965; Held and Karp in 1970; Lin in 1965, branch and cut algorithms proposed by Hong in 1972; Crowder and Padberg in 1980; Grötschel and Holland in 1991; Padberg and Rinaldi in 1991 among other techniques. The latter include several construction heuristics such as the nearest neighbor heuristic, the nearest, farthest, and cheapest insertion heuristics, Christofides' heuristic, as well as improvement heuristics such as local search proposed by Lin and Kernighan in 1973, tabu search proposed by Fiechter in 1990; Gendreau et. al. in 1994; Knox, in 1994; Potvin et. al. in 1996; Tsubakitani and Evans in 1998, simulated annealing proposed by Cerny in 1985, genetic algorithms proposed by Nguyen in 2004, and swarm algorithms proposed by Goldberg et. al. in 2008; Wang et. al. in 2003 (Ghosh and Basu, 2011).
4. Genetic Algorithms

Genetic Algorithms (GAs) receive their name from an intuitive explanation of the manner in which they behave. This explanation is based on Darwin’s theory of natural selection. GA stores a set of solutions and then work to replace these solutions with better ones based on some fitness criterion, usually the objective function value (Burkard and Çela, 1996).

GAs represent a powerful and robust approach for developing heuristics for large-scale combinatorial optimization problems. The motivation underlying GAs can be expressed as follows: Evolution is remarkably successful in developing complex and well-adapted species through relatively simple evolutionary mechanisms. A natural question is the following: What ideas can we adapt from our understanding of evolution theory so as to solve problems in other domains? This fundamental question has many different answers because of the richness of evolutionary phenomenon. Holland provides the first answer to this question by developing GAs (Ahuja, et. al., 2000).

GAs are a specific type of evolutionary algorithms. Evolutionary algorithms are population-based, adaptive search algorithms designed to attack optimization problems. They are inspired by models of natural evolution of species and use the principle of natural selection which favors individuals that are more adapted to a specific environment for survival and further evolution. Each individual in an evolutionary algorithm typically represents a solution with an associated fitness value. The three main operators used are selection, mutation, and recombination. Selection prefers fitter individuals to be chosen for the next generation and for the application of the mutation and recombination operator. Mutation is a unary operator that introduces random modifications to an individual. Recombination combines the genetic material of two individuals, also called parents, by means of a crossover operator to generate new individuals, called offsprings (Stützle, 1998).

The three main algorithmic developments within the field of evolutionary algorithms are genetic algorithms, evolution strategies and evolutionary programming. These algorithms are developed independently and, although these algorithms initially are proposed in the sixties and seventies, only in the beginning of the nineties the researchers become aware of the common underlying principles of these approaches (Stützle, 1998).

A standard representation of the solution is as an array of bits. Arrays of other types and structures can be used in essentially the same way. The main property that makes these genetic representations convenient is that their parts are easily aligned due to their fixed size, which facilitates simple crossover operations. Variable length representations may also be used, but crossover implementation is more complex in this case. Tree-like representations are explored in genetic programming and graph-form representations are explored in evolutionary programming (Obitko, 2011).

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 selected according to their fitness - the more suitable they are to reproduce. This is repeated until some condition (for example number of populations or improvement of the best solution) is satisfied (Obitko, 2011).

In Figure 1, an algorithmic skeleton for the application of genetic local search algorithms is given. p denotes the population. A set of new individuals p' is generated in the function Recombination applying some crossover operator. After some individuals of the population are mutated, local search is applied to the newly generated solutions represented in p' and p''. In the last step the new population is determined by the Selection function (Stützle, 1998).

GAs imitate the process of evolution on an optimization problem. Each feasible solution of a problem is treated as an individual whose fitness is governed by the corresponding objective function value (Ahuja, et. al., 2000).

GA is a randomized search methodology having its roots in the natural selection process. Initially the neighborhood search operators (crossover and mutation) are applied to the preliminary set of solutions to acquire generation of new solutions. Solutions are chosen randomly from the existing set of solutions where the selection probability and the solution’s objective function value are proportional to each other and eventually the aforesaid operators are
applied on the chosen solutions. GAs have aided in the successful implementation of solutions for a wide variety of combinatorial problems (Radhakrishnan, et. al., 2009).

The robustness of the GAs as search techniques can be theoretically and empirically proved. The artificial individual is the basic element of a GA. An artificial individual consists of a chromosome and a fitness value, similar to a natural individual. The individual’s likelihood for survival and mating is determined by the fitness function. In accordance with the Darwin’s principle, individuals superior to their competitors, are more likely to promote their genes to the next generations (Radhakrishnan, et. al., 2009).

![Figure 1: The scheme of the genetic algorithms based on the new replacement strategy (Ji, et.al., 2006)](image)

In GAs, Radhakrishnan et. al. encode a set of parameters mapped into a potential solution, named chromosome, to the optimization problem. The population of candidate solutions can be obtained through the process of selection, recombination and mutation performed in an iterative manner. Chromosomes refer to the random population of encoded candidate solutions with which the GAs initiate. Then the set (a population) of possible solutions (called chromosomes) can be generated (Radhakrishnan, et. al., 2009).

A function assigns a degree of fitness to each chromosome in every generation in order to use the best individual during the evolutionary process (Radhakrishnan, et. al., 2009). The fitness function is defined over the genetic representation and measures the quality of the represented solution (Obitko, 2010). In accordance to the objective, the fitness function evaluates the individuals. Each chromosome is evaluated using a fitness function and a fitness value is assigned. Then, three different operators—selection, crossover and mutation—are applied to update the population. A generation refers to an iteration of these three operators. The promising areas of the search space are focused in the selection step. The selection process typically keeps solutions with high fitness values in the population and rejects individuals of low quality. Hence, this provides a means for the chromosomes with better fitness to form the Mating Pool (MP) (Radhakrishnan, et. al., 2009).

After the process of selection, the crossover can be performed. In the crossover operation, two new children are formed by exchanging the genetic information between two parent chromosomes (C1 and C2 which are selected from the selection process). A crossover point is chosen at random by the crossover operator. At this point, two parent chromosomes break and then exchange the chromosome parts after that point. Consequently, the partial features of two chromosomes are combined to generate two off springs. The chromosome cloning takes place when a pair of chromosomes does not cross over, thus creating off springs that are exact copies of each parent. The ultimate step in each generation is the mutation of individuals through the alteration of parts of their genes. Mutation alters a portion of a chromosome and thus institutes variability into the population of the subsequent generation.

1714
Mutation, a rarity in nature, denotes the alteration in the gene and assists us in avoiding loss of genetic diversity. Its chief intent is to ensure that the search algorithm is not bound on a local optimum (Radhakrishnan, et. al., 2009).

5. Genetic Algorithm Implementation Details
A standard representation of the solution is as an array of bits. Arrays of other types and structures can be used in essentially the same way. The main property that makes these genetic representations convenient is that their parts are easily aligned due to their fixed size, which facilitates simple crossover operations. Variable length representations may also be used, but crossover implementation is more complex in this case. Tree-like representations are explored in genetic programming and graph-form representations are explored in evolutionary programming. The fitness function is defined over the genetic representation and measures the quality of the represented solution.

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 selected according to their fitness - the more suitable they are the more chances they have to reproduce. This is repeated until some condition (for example number of populations or improvement of the best solution) is satisfied (Obitko, 2011).

6. Genetic Algorithm Application and Computational Study

6.1 Initial Requirements
The GA algorithm is developed on Eclipse Classic 3.5.2 and JCreator LE 4.50 by using Java Development Kit 1.6 and ran on a computer with Intel ® Core(TM) 2 Duo 1.83 Ghz CPU and 1 GB - RAM.

6.2 Input Files
This program can be examined in three important parts; input files, program files, and output file. Input files are the problem sets taken from the TSP Library on the web address http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/. Symmetric TSP instances are used in this study.

The distances between \( n \) cities are stored in a distance matrix (D) with elements \( d_{ij} \) where \( i, j = 1, \ldots, n \) and the diagonal elements \( d_{ii} \) are zero. A tour can be represented by a cyclic permutation \( \Pi \) of \( \{1, 2, \ldots, n\} \) where \( \Pi(i) \) represents the city that follows city \( i \) on the tour. As an illustration of TSP instance, it is shown the form of the problem (ulysses16.tsp) as an example in Figure 2.

The TSP is then the optimization problem to find a permutation \( \Pi \) that minimizes the length of the tour. The goal is to find the shortest tour that visits each city in a given list exactly once and then returns to the starting city.

6.3 Genetic Algorithm Application

Creating population
Based on the size of problem \( (n) \), chromosomes in the initial population are created by randomly assigning \( n \) cities to \( n \) position in a permutation. Population size is set to \( n*10 \).

Crossover
Cycle Crossover is chosen as crossover operator in the algorithm. After two parents are chosen according to roulette wheel selection technique crossover points are determined with random number. In every generation, every chromosome is paired up with another chromosome randomly to be used in crossover operation with a probability of 0.97.

According to crossover points, children sets are created for child1 as follows; between two crossover points, the assignments located in the same place as in the parent 1, then the program checks parent 2 assignment sequences from the second crossover points to other ones one by one with locating one city only one place. According to the cost of each different assignment, if one assignments cost is less than the other assignments, the city is assigned to the location which has minimum cost until every city is assigned to different locations. For child2, the same procedure is used on parent 1 assignment sequences. \( n*10 \) children are created after crossover operation.
NAME: ulysses16.tsp
TYPE: TSP
COMMENT: Odyssey of Ulysses (Groetschel/Padberg)
DIMENSION: 16
EDGE_WEIGHT_TYPE: GEO
DISPLAY_DATA_TYPE: COORD_DISPLAY

NODE_COORD_SECTION
1 38.24 20.42
2 39.57 26.15
3 40.56 25.32
4 36.26 23.12
5 33.48 10.54
6 37.56 12.19
7 38.42 13.11
8 37.52 20.44
9 41.23 9.10
10 41.17 13.05
11 36.08 -5.21
12 38.47 15.13
13 38.15 15.35
14 37.51 15.17
15 35.49 14.32
16 39.36 19.56
EOF

Figure 2: The form of TSP instance (ulysses16.tsp)
(Source: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/)

Mutation
Two cities change their position with each other, randomly. Two-Opt Mutation Operator is used in the algorithm.

Creating new population
After a generation of GA, a new population to be used as parent population in the next generation is created as follows: Firstly, the parent population and the child population are ordered in nonincreasing fitness value. Then, first 25% is copied from parent population to new population. Then, another 25% is copied from child population as long as they are not already in new population. After that, the remaining 50% of chromosomes are copied randomly from child population in order to span solution space (Erol, 2010). Roulette wheel technique is used for reproduction and survival.

Calculating fitness value
Given a set of \( n \) nodes and distances for each pair of nodes, finding a roundtrip of minimal total length visiting each node exactly once is fitness value (defined as total cost).

The parameters (like control parameters – maximum iteration, population size- and probabilities – mutation and crossover) for GA application are:
- Maximum iteration is 1000,
- Population size is 100,
- Mutation probability is taken 0.20,
- Crossover probability is taken 0.97.

6.4 Output Files and Results
Each TSP instance is solved ten times and the results are recorded.
7. Conclusions and Future Research

In this study, new proposed GA is applied to TSP which is one of the most difficult problems in NP-hard class. Cycle Crossover is used instead of classic two-point crossover technique for this algorithm in the application.

As a summary of results; the minimum and maximum errors as performance measures with respect to the best known solutions for test problem sets in TSPLIB is shown in Table 1. Twelve problem instances are solved by Java programming language and best solutions of ten runs, the best known solutions and average CPU times (in seconds) are listed in Table 1. The proposed GA solved eleven of twelve problem instances optimally.

Table 1: The results for problem instances

<table>
<thead>
<tr>
<th>Problem name</th>
<th>Performance over Best Known Solution</th>
<th>Best solutions of application</th>
<th>Best known solutions</th>
<th>Average CPU times (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min</td>
<td>Max</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ulysses16.tsp</td>
<td>0</td>
<td>0,007</td>
<td>6859</td>
<td>6859*</td>
</tr>
<tr>
<td>ulysses22.tsp</td>
<td>0</td>
<td>0,040</td>
<td>7013</td>
<td>7013*</td>
</tr>
<tr>
<td>fri26.tsp</td>
<td>0,024</td>
<td>0,088</td>
<td>960</td>
<td>937*</td>
</tr>
<tr>
<td>bays29.tsp</td>
<td>0</td>
<td>0,063</td>
<td>2020</td>
<td>2020*</td>
</tr>
<tr>
<td>bayg29.tsp</td>
<td>0</td>
<td>0,087</td>
<td>1610</td>
<td>1610*</td>
</tr>
<tr>
<td>att48.tsp</td>
<td>0</td>
<td>0,075</td>
<td>10648</td>
<td>10648*</td>
</tr>
<tr>
<td>hk48.tsp</td>
<td>0</td>
<td>0,069</td>
<td>11461</td>
<td>11461*</td>
</tr>
<tr>
<td>berlin52.tsp</td>
<td>0</td>
<td>0,098</td>
<td>7542</td>
<td>7542*</td>
</tr>
<tr>
<td>burma14.tsp</td>
<td>0</td>
<td>0,075</td>
<td>3323</td>
<td>3323*</td>
</tr>
<tr>
<td>eil51.tsp</td>
<td>0</td>
<td>0,045</td>
<td>426</td>
<td>426*</td>
</tr>
<tr>
<td>eil76.tsp</td>
<td>0</td>
<td>0,098</td>
<td>538</td>
<td>538*</td>
</tr>
<tr>
<td>eil101.tsp</td>
<td>0</td>
<td>0,099</td>
<td>629</td>
<td>629*</td>
</tr>
</tbody>
</table>

*Best known solution

The improvement of proposed algorithm is shown in Figure 3 for one of the problem instances (ulysses22.tsp).
According to the analysis of the solution techniques, this study can be a basic resource for researches. This proposed GA application will also be an essential framework for other researches. After this project, our aim is to develop a new solution technique with a hybrid genetic algorithm. As a future research, all symmetric and asymmetric TSP problem sets will be solved by this algorithm. Especially for asymmetric problem instances, this new solution technique can give best results if parameters are fine tuned.

References