Web Application of Traveling Salesman Problem using Genetic Algorithms

Milena N. Karova[1], Julka Petkova[2] and Stoyan P. Penev[3]

***Abstract* – This paper introduces online method for finding a solution to the travelling salesman problem using a genetic algorithm. The travelling salesman problem comes up in different situations in out world. It is a special kind of optimisation problem. There had been many attempts to address this problem using classical methods, such as integer programming and graph theory algorithms with different success. The solution, which this paper offers, includes a genetic algorithm implementation in order to give a maximal approximation of the problem, modifying online a generated solution with genetic operators.**

***Keywords* – genetic algorithms, TSP, travelling salesman problem, optimisation, selection, genetic operator, crossover, mutation.**

I. INTRODUCTION

Genetic Algorithms are stochastic search methods that mimic the metaphor of natural biological evolution [1]. Genetic Algorithms (Gas) operate on a population of potential solutions applying the principal of survival of the fittest to produce better and better approximations to a solution. At each generation, a new set of approximations is created by the process of selecting individuals according to their level of fitness in the problem domain and breeding them together using operators from natural genetics [3]. This process leads to the evolution of populations of individuals that are better suited to their environment than the individuals that they were created from, just as in natural adaptation.

Gas model natural processes, such as selection, recombination, mutation, migration, locality and neighborhood

II. ONLINE SOLUTIONS OF TRAVEL SALESMAN PROBLEM

In the Internet world there are many web sites for solving Travel Salesman Problem (TSP). Usually the solutions are Java applets [4]. It proposes random choice of city's number N (between 1 and 9). The city's distance represents matrix NxN. The algorithm terminates in two away: finding optimal distance or using definite number of iterations.

This algorithm contains two important disadvantages: small city's number and limit population size.

The Ga TSP solving in the University in Ohio offers to user defining city's location on the work area manually [5]. The population size is programming limited. If the city's number is big, the absence of automatically drawing of cities is disadvantage [6].

Generally the following actions are performed to compute the shortest path:

- Fitness function: the sum of distances between all cities
- Chromosomal Representation: sequence of numbers containing a permutation of the first n numbers represented as an array; e.g. 1-2-3-4-6-5
- Selection method: K-tournament selection
- Initialization: Random
- Genetic Model: Generate the next generation from the scratch
- Termination condition: The system is run for N generations and the best (or best k) solution is reported
- Operators: mutation, crossover, copy
- Operator application probabilities: crossover: 0% at generation 1; increase to 95% at generation N; mutation: 95% at generation 1 is reduced to 0% at generation N; copy: fixed at 5%
- Population size PS (e.g. 500)

III. GENETIC PARAMETERS AND GENETIC OPERATORS

A. Population

Population is a combination of chromosomes. In the program to present the population it uses array of 1002 chromosomes. The thousand and first chromosome stores the worst tour. The name of the array is population.

public static TChromosome population = new TChromosome[1001];

For each chromosome it calculates the length that is coded into it, actually this is the fitness of the tour [2]. It is stored in the next array:

[1]Milena N. Karova is with the Department of Computer Science and Technologies, Technical University Varna, Studentska str. 1, 9010 Varna, Bulgaria, E-mail: mkarova@ieee.bg

[2]Julka P.Petkova is with the Department of Computer Science and Technologies, Technical University Varna, Studentska str. 1 9010 Varna, E-mail: jppet@mbox.digsys.bg

[3]Stoyan P. Penev is with the Department of Computer Science and Technologies, Technical University Varna, Studentska str. 1, 9010 Varna, Bulgaria, E-mail: penev@engineer.bg

public static double[] popFitness=new double[1001];

Now it knows that the tour with index i has a fitness popFitness[i].

The maximum number of towns is 32. The current number is stored in the variable townCount. In the same way the number of populations – popCount. In the process of mutation, it uses the coefficient MutInd.

B. Genetic Operators for Recombination

Two of the main problems that occur ware choosing proper methods of crossover and mutation. It has implemented two types of crossover – cycle crossover and a custom one [8]. The user can choose which one to use in the calculation. Let us take a closer look at the Cycle crossover.

First of all it fits perfectly to the way our tour is represented in the chromosome. For example if our tour is

Tour = 1234

This means that it goes from city 1 to city 2 to city 3 to city 4.Unlike other methods of crossover here it does not pick a crossover point at all. It chooses the first gene from one of the parent chromosome. If our parents are

parent1 = 12345678
parent2 = 85213647

say it picks 1 from parent 1,

*child = 1*******

It must pick every element from one of the parents and place it in the position it was previously in. Since the first position is occupied by 1, the number 8 from parent2 cannot go there. So it must now pick the 8 from parent1.

*child = 1*******8*

This forces us to put the 7 in position 7 and 4 in position 4, as in parent1.

*child = 1**4**78*

Since the same set of position is occupied by 1,4,7,8 in parent1 and parent2, it finishes by filling in the blank positions with the elements of those positions in v2. Thus

child 1 = 15243678

and it get child2 from the complement of child1.

This type of crossover ensures that each new created chromosome is legal. A chromosome is legal if it is constructed according to the requirements of the salesman problem. In this crossover notice that it is possible for us to end up with the offspring being the same as the parents. This is not a problem since it will usually occur if parents have high fitness, in which case, it could still be a good chance.

In the program it uses function TestCrossOver:

Public static TChromosome TestCrossOver (int indPar1, int indPar20);

where indPar1 and indPar2 are the parent's chromosomes.

If it want to solve this problem or other like not getting trapped in a local optimum we could use mutation. Due to the randomness of the process it will occasionally have chromosomes near a local optimum but none near the global optimum. Therefore the chromosomes near the local optimum will be chosen to crossover because they will have the better fitness and there will be very little chance of hiding the global optimum. So mutation is a completely random way of getting to possible solutions that would otherwise not be found.

Mutation is performed after crossover. In presented algorithm, there are 3 kinds of mutation: transposition, inversion and changing city's position. The mutation index (MutInd) must decide weather to perform mutation on this child chromosome or not. It then chooses a point to mutate and switch that point. For instance, in our example it had

child = 12345678

If we choose the mutation point to be gene three and 7, the child would become

child = 12745638

It simply switched the places of genes 3 and 7. Another mutation that takes place is inverting a subtour in our child chromosome. Let us have the chromosome

child = 12345678

and choose the same mutation points 3 and 7.The subtour between these tow point is switched in reverse order

child = 12765438

After the mutation process the program makes a strict verification of the chromosome. If it is not legal then the chromosome is ignored.

In the program it uses function Mutate:

Public static TChromosome Mutate (TChromosome ch);

The idea of the traveling salesman problem is to find a tour of a given number of cities, visiting each city once and returning to the starting city where the length of this tour is minimized.

C. Fitness Function

The Purpose of the fitness function is to decide if a chromosome is good how good it is [4]. In the traveling salesman problem the criteria for good chromosome is it's

length. The longer the tour that is coded, the better the chromosome is. Calculation takes place during the creation of the chromosomes. Each chromosome is created and then its fitness function is calculated. The length of the chromosome is measured in pixels by the scheme of the tour.

$$fitness\_chromosome = \sum_{i=1}^{towncount} t_i ,$$

where towncount is the number of cities in population, et ti is the distance between two cities.

D. Basic Functions used in this application

In this product were used 10 basic functions and procedures in order to create a completely working program:

-DrawCity -ShowCities -DrawChromosome
-GenerateTownSet -CreateTown -GetClick
-Mutate -ClenUp -Sort
-CreatePopulation -TestCrossOver

IV. Interface and Test Results

The interface [Fig.1] is Web application, showing the current result at the moment they are calculated (the best and the worst chromosome). This screen offers determining cities, population, stop criteria (iteration number), crossover mode, mutation mode and mutation rate (coefficient of mutation). In this order tone of the stop criteria is that the user can terminate the calculations if he finds a feasible solution. The screen [Fig.2] shows the worst chromosome and the Fig. 3 – the best solution that is the chromosome with the smallest distance between cities.

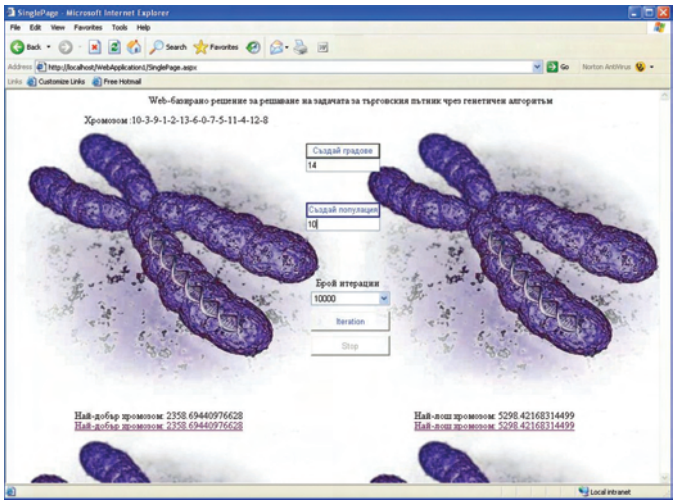

Fig. 1. The main page

Fig. 4 shows that algorithm needs the minimum iterations to obtain optimal solution (minimized cutie's tour) using mutation rate 3,5%.

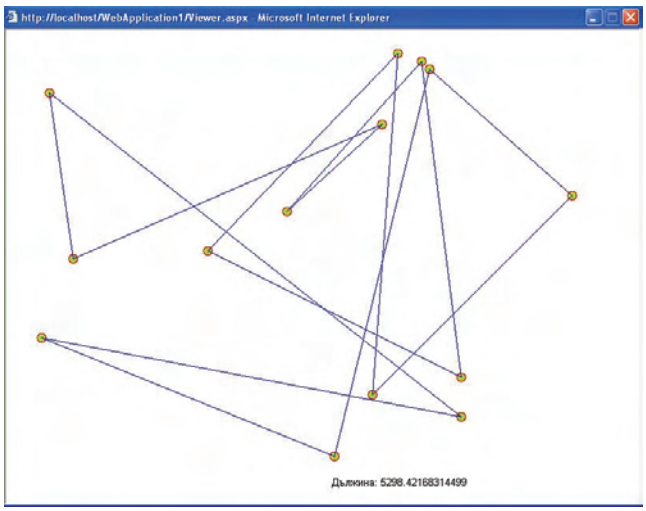

Fig. 2. The worst chromosome

The algorithm finds a good solution when there are 30 cities, when the coefficients of inversion and transposition have lower values. When there are less cities, coefficient's influence is smaller. When the distance between cities is constant, the dependences are the same [Fig.5, Fig.6].

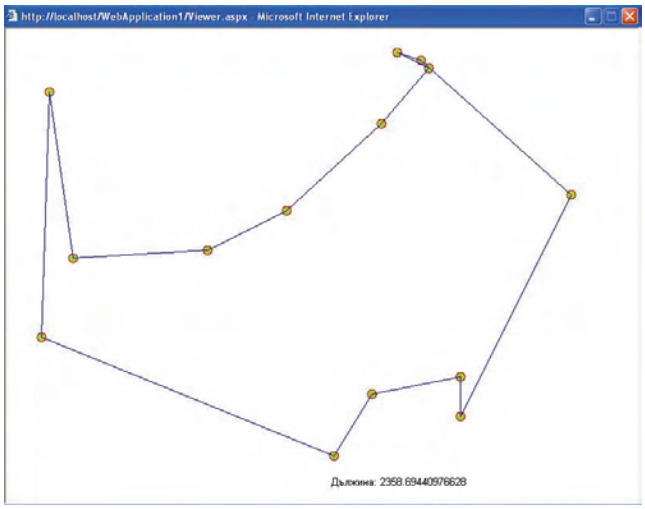

Fig. 3. The Best chromosome

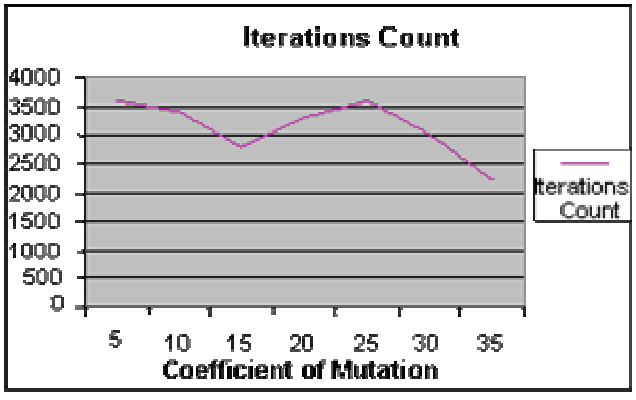

Fig.4 Coefficient of Mutation

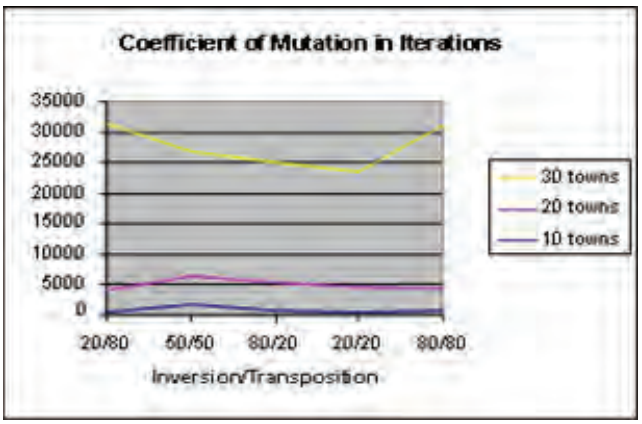

Fig. 5 Inversion/ Transposition

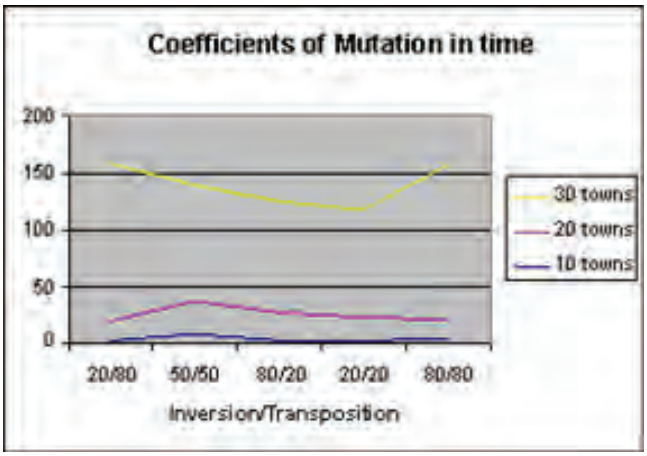

Fig. 6. Mutation Coefficients in time

V. Conclusion

Genetic algorithms appear to find good solutions for the traveling salesman problem, however it depends very much on the way the problem is encoded and which crossover and mutation methods are used. It seems that the methods that use heuristic information or encode the edges of the tour perform the best and give good indications for future work in this area.

Overall, it seems that genetic algorithms have proved suitable for solving the traveling salesman problem. As yet, genetic algorithms have not found a better solution to the traveling salesman problem that is already known, but many of the already known best solutions have been found by some genetic algorithm methods also.

It seems that the biggest problem with the genetic algorithm devised for the traveling salesman problem is that it is difficult to maintain structure from the parent chromosomes and still end up with a legal tour in the child chromosomes. Perhaps a better crossover or mutation routine that retains structure from the parent chromosomes would give a better solution that we have already found for some traveling salesman problems.

References

[1] Goldberg D, "Web Courses", http://www.engr.uiuc.edu/OCEE, 2000.

[2] Engebretsen L., Karpinski M. Approximation hardness of TSP with bounded metrics, Proceedengs of 28th ICALP, LNCS 2076, Springer 2001

[3] Mitchell M., "An Introduction to Genetic Algorithms", Massachusetts Institute of Technology, 1996

[4] http://students.ceid.upatras.gr/~papagel/project/tspprobl.htm

[5] http://www.personal.kent.edu/~rmuhamma/Algorithms/AproxAlgor/TSP/tsp.htm

[6] http://home.planet.nl/~onno.waalewijn/tspx.html