A Genetic Algorithm for a Traveling Salesman Problem

Milena N. Karova[1] Vassil J. Smarkov[2] Stoyan P. Penev[3]

Abstract - **This paper introduces a flexible method for finding a solution to the traveling salesman problem using a genetic algorithm. The traveling salesman problem comes up in different situations in out world. It is a special kind of optimization 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 a generated solution with genetic operators.**

Keywords: **genetic algorithm, TSP, traveling salesman problem, constraints, optimization, approximation, selection, genetic operator, crossover**

I. Introduction

The genetic algorithms are an optimization technique based on natural evolution. They include the survival of the fittest idea info a search algorithm which provides a method of searching which does not need to explore every possible solution in the feasible region to obtain a good result. Genetic algorithms are based on the natural process of evolution. In natureq the fittest individuals are most likely to survive and mate; therefore the next generation should be fitter and healthier because they were bred from healthy parents. This same idea is applied to a problem by first ‘guessing’ solutions and then combining the fittest solution to create a new generation of solutions which should be better than the previous generation. We also include a random mutation element to account for the occasional mishap in nature.

The genetic algorithm process consists of the following steps:

1. Encoding
2. Evaluation
3. Crossover
4. Mutation
5. Decoding

A suitable encoding is found for the solution to our problem so that each possible solution has unique encoding and the encoding is some form of a string. The initial population is then selected, usually at random though alternative techniques using heuristics have also been proposed. The fitness of each individual in the population is then computed that is, how well the individual fits the problem and whether it is near the optimum compared to the other individuals in the population.

[1]Milena N. Karova is with the department of Computer Science, Studentska 1, Technical University Varna Email: mkarova@ieee.bg

[2]Vassil J. Smarkov is with the department of Computer Science, Studentska 1, Technical University Varna Email: smarkov@ieee.bg

[3]Stoyan Penev is student, department of Computer Science, Technical University Varna Email penev@engineer.bg

This fitness is used to find the individual’s probability of crossover. If an individual has a high probability (which indicates that it is significantly closer the optimum than the rest of its generation) then it is more likely to be chosen to crossover. Crossover is where the two individuals are recombined to create new individuals which are copied into the new generation. Next mutation occurs. Some individuals are chosen randomly to be mutated and then a mutation point is randomly chosen. The character in the corresponding position of the string is changed. Once this is done, a new generation has been formed and the process is repeated until some stopping criteria have been reached. At this point the individual who is closest to the optimum is decoded and the process is complete.

II. Constraints

The genetic algorithms are used for solving complex problems such as NP-hard problems. They can be used in teaching different machines such as robots and simple evolving programs. They can also be used in creating pictures and music.

One of the great advantages of the genetic algorithm is the hing level of parallelism because of whish the programs are very easy to apply. Once created, a genetic algorithm with small changes can be adjusted to solve a completely different problem. The choice of coding technique and calculation of fitness function can be very complicated and difficult.

The main advantage of the genetic algorithms is that they can found a feasible solution for a very short time.

Two of the main problems that occur was choosing proper methods of crossover and mutation. We have implemented two types of crossover – cycle crossover and a custom one. 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 we go from city 1 to city 2 to city 3 to city 4.Unlike other methods of crossover here we do not pick a crossover point at all. We choose the first gene from one of the parent chromosome. If our parents are

parent1 = 12345678

parent2 = 85213647

say we pick 1 from parent 1,

child = 1*******

We 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 we 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, we finish by filling in the blank positions with the elements of those positions in v2. Thus

child 1 = 15243678

and we 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.

If we 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 we 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. The mutation index (MutInd) must decide weather to perform mutation on this child chromosome or not. We then choose a point to mutate and switch that point. For instance, in our example we had

child = 12345678

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

child = 12745638

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

child = 12345678

And choose the same mutation points 3 and 7.The sub tour 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.

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.

The product finds a solution to the traveling salesman problem. For this purpose we use cities, chromosomes and populations. Each city is represented by an object of class TCity. The declaration is:

```
TCity=class( TObject )
    Name:String;
    x,y: Integer;
  end;
```

Each city is situated on coordinates (x, y) on the map. In the working process a defined number of cities are being created. Then the program solves the traveling salesman problem for these cities. The combination of cities is stores into an object collection citylist.t consists of objects of class TCity.

A tour or a chromosome represents a succession of indexes of cities. An index of object city in the collection citylist is the same as the index of the city. A tour contains variable of type TChromosome.

```
TChromosome=array[0 .. 32 ] of Integer;
```

This is an array of 32 elements, each one has an integer value. Let M is a variable of type TChromosome:

```
Var M:TChromosome;
```

Then the tour starts from city with index M[1], continues to city with index M[2]. The maximum number of cities is 32, which is the length of the array.

III. Population

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

```
Population: array [0 1001] of TChromosome;
```

For each chromosome we calculate the length that is coded into it, actually this is the fitness of the tour. It is stored in the next array:

```
popFitness: array[ 0 .. 1001] of Real;
```

Now we know 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, we use the coefficient MutInd.

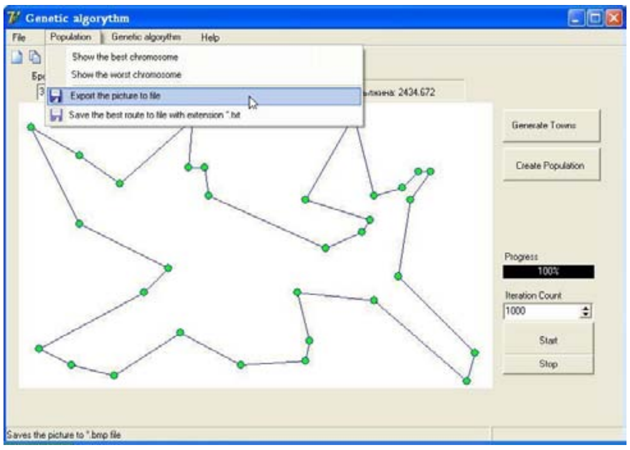
IV. Basic Functions

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

2.1 DrawCity
2.2 ShowCities
2.3 DrawChromosome
2.4 GenerateTownSet
2.5 CreateTown
2.6 GetClick
2.7 Mutate
2.8 CleaUp
2.9 SaveToText
2.10.Sort
2.11.Image1MouseMove
2.12.seTownCountChange
2.13.TownKeyPress
2.14 CreatePopulation
2.15 TestCrossOver
2.16 CrossOver

V. Interface

The interface [Fig.1] is Windows Forms oriented, showing the current result at the moment they are calculated. In this order one of the stop criteria is that the user can terminate the calculations if he found a feasible solution



2434.672

Progress 100%

Iteration Count 1000

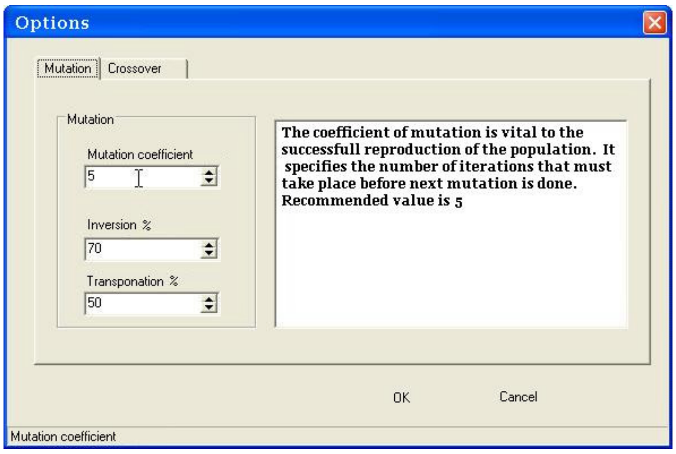
Saves the picture to *.bmp file

Fig. 1. Program Interface

The options screen [Fig.2] allows us to change the parameters of the genetic algorithm. We can change the mutation, inversion, and transposition coefficients.

VI. 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



Mutation

Mutation coefficient 5

Inversion % 70

Transponation % 50

The coefficient of mutation is vital to the successfull reproduction of the population. It specifies the number of iterations that must take place before next mutation is done. Recommended value is 5

Mutation coefficient

Fig. 2. Options Screen

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. L., "Genetic Algorithms in Search, Optimization, and Machine Learning", Addison-Wesley, 1989

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

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

[4] Paechter B., Rankin R., Cumming A., "Timetabling the Classes of an Entire University with an Evolutionary Algorithm", Napier University, Edinburgh, Scotland.

[5] Michalewicz Z., Janikow C., "Handling constraints in genetic algorithms" *In Proceeding of the 4th International Conference in Gas*. Morgan Kauffman, 1991

[6] Michalewicz Z, Genetic Algorithms+ Data Structures = Evolution Programs, Springer Verlag, 1992.

[7] Holland, John H., "Adaption in Natural and artificial systems", the Mit Press, 1992

[8] Spears, W. M. and DeJong K., "An analysis of multi-point crossover", *Foundations of Genetic Algorithms*, pages 301-315, Morgan Kaufmann, 1999.

[9] Syswerda, G., "Uniform crossover in genetic algorithms", *Procedings of the Third International Conference on Genetic Algorithms*, pages 2-9, 1995