SOUTHEAST EUROPE JOURNAL OF SOFT COMPUTING
Available online at www.scjournal.com.ba

UML modeling for traveling salesman problem based on genetic algorithms

Muzafer Saračević [a], Sead Mašović [b], Šemsudin Plojović [c]

[a] Faculty of Science and Mathematics,University of Niš, Serbia, muzafers@gmail.com

[b] Faculty of Science and Mathematics,University of Niš, Serbia, sekinp@gmail.com

[c] Department for Computer Sciences, University of Novi Pazar, Serbia, s.plojovic@uninp.edu.rs

***Abstract* — The major purpose of this paper is to present a way of solving problems through so-called visual planning and programming using object-oriented concepts. This paper describes the process of UML modeling for solving the traveling salesman problem using one of the metaheuristic-genetic algorithms. The analysis and problem solving in this way has many advantages just because it provides a clear definition of requirements and specific plan that we will later use to create specific applications. This is a good way to resolve because the UML describes the source code, models help to visualize the system as it is or what it should be and allow you to determine the structure and behavior of the system. Static and dynamic diagrams implemented in developing tools for modeling, as well as a description of specific applications and testing are mentioned. With this approach we describe modeling tool that can be used in the development of specific solutions and a way of establishing explicit links between concepts and execution code.**

Keywords — Genetic algorithms, Optimization, TSP, NP-complete problems

INTRODUCTION

In this paper, we described the traveling salesman problem (TSP), which belongs to the NP class - difficult problems (Garey, 1979) which does not recognize a polynomial algorithm (Cormen, 2001 and Eberle 2008). We talked about modeling in the UML (*Unified Modeling Language*). Heuristics used to solve this problem are genetic algorithms which belong to the modern meta-heuristics. In the end we specified a particular application that illustrates how these algorithms work. The application is capable of setting the necessary parameters and visual display of the optimal

solution with the diagram for the calculation of solutions improvement.

Models document the decisions that we were making and provide samples to guide us during the construction of the system. Traveling salesman problem is a known problem in the theory and practice. We can find attempts for solving this problem in the distribution, collection or other ground handling of the transport network (Applegate, 1998). While trying to solve the problem we are striving to find the shortest route that starts at a certain point, goes through all the other points and ends at the initial one. The things that we optimize may not only be the distance. Those can be travel expenses, travel duration or some other variable. In the traveling salesman problem there might be a stricter requirement to pass through each point exactly once. It is then we are talking about the classic traveling salesman problem.

In a genetic algorithm, a population of strings which encode candidate solutions to an optimization problem, evolves toward better solutions. Traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible. The evolution usually starts from a population of randomly generated individuals and happens in generations. 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 then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. If the algorithm has terminated due to a maximum number of generations, a satisfactory solution may or may not have been reached.

In order to solve the above problem, there are other methods that give good results. In works (Pilat 2002, Ngassa 2007, Baran 2005) a method of ant colony optimization (*ACO*) is described. ACO is an algorithm which, by its mode, falls into the category of evolutionary algorithms. Mode of ant finding food, as a part of a large colony, is trying to be imitated in the appropriate algorithm. In paper (Salajic 2001), heuristics such as genetic algorithms and simulated tempering are listed. Simulated tempering uses stochastic approach of leading the search. This method allows the search to continue in the direction of neighboring solutions, although the objective function in this way gives worse results. The paper (Thamilselvan 2009) refers to the so-called tabu search. Tabu Search (*TS-Tabu Search*) is a method of local search in combinatorial optimization, which shows very good results in solving vehicle routing problem (VRP). The paper (Kratica 2000) gives parallel genetic algorithms for solving some NP-complete problems, while the paper (Saiko 2005) gives a concrete implementation in the Java language for the TSP that is to be solved using genetic algorithms.

GENETIC ALGORITHMS AND TRAVELING SALESMAN PROBLEM

First, we will mention some of the main features of graphs. Graph **G=(T,L)** is an arranged pair **(T,L)** where T is the set of vertices (nodes) and L system of arcs (links) of the graph. Route is a series of vertices connected by arches. If the route has all the arcs different it is called a chain, and if all the vertices are different then it is a simple chain. Simple closed chain is called a cycle. We will look at the traveling salesman problem, by solving a concrete example, through a fully non-oriented graph. This means that we chose the stricter requirement, to pass through each node exactly once. Graph in which it is possible to construct a cycle that goes through each node is called a Hamiltonian graph.

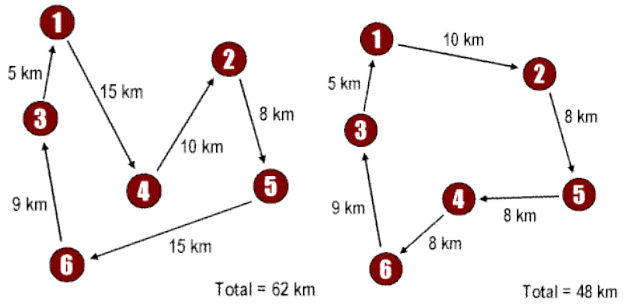

Figure 1: Finding the shortest path

So, for the given set of cities and the travel cost between each pair of the cities, the solution of traveling salesman problem has to find the cheapest route to visit all cities exactly once and in the end return to the starting city. The problem can be extended by adding different limitations. This is how time-dependent traveling salesman problem was created, which with a minimum length of route, takes into account the time needed to perform the travel and any periods of time during which one part of the job has to be done (Held 1970). Along to this definition, there is also an asymmetric traveling salesman problem in which the time between city A and city B is not the same as the time between city B and city A (Gutin 2002).

For solving the traveling salesman problem in recent times a number of heuristic algorithms has been developed (Nilsson 2003). The first group consists of algorithms for the

construction route. Using this algorithm and based on the known distances between each node, a route of a salesman is constructed. The second group of heuristic algorithms is made of those that serve to improve the existing route. With their application we are improving the initial route (Lin 1973). Heuristic algorithms for the construction of the route of a salesman which are most commonly used are algorithms of the nearest neighbor, nearest insertion, random insertion, and Christofides's heuristic algorithm and so on (Litlle 1963). The most important methods that we can specify to improve the solutions are the genetic algorithms, tabu search, simulated tempering, k-optimal heuristics (2-OPT, 3-OPT, k-OPT). The first three methods are listed in contemporary metaheuristics. A typical genetic algorithm requires:

- a genetic representation of the solution domain,
- a fitness function to evaluate the solution domain.

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.

Practical application of genetic algorithms get another plus, because in practice it is not necessary to find an optimal path, but it's good enough to find a decent route and one that is around optimum (Bonovska 2006). With all this it is possible to combine genetic algorithms with methods of local optimum search in order to convert a particular individual to its optimum which is potentially global as well (Johnson 1995). Such algorithms are called hybrid. An example of hybridity achieved here is use of 2opt method for an operator of mutation (White 2004) .

There are many possible ways to show this, for example, **(0,1,3,5,4,6,2,7)**. In this case, there are 8 cities to visit and the sequence of visit is given in the order of numbers. Another way to show this is a matrix display. The matrix is a type of $n \times n$ and it is actually the adjacency matrix in the graph that is obtained by connecting all cities. If there is a way out of town A to town B then there is 1 in the place of M[A,B], instead of number 0. In the present modeling and the application we have provided the following combination of operators:

- *the natural selection (elimination),*
- *2opt methods (mutation) and*
- *Greedy Subtour crossover (crossing).*

Operators of selection

In this particular application we used the so-called *natural selection*. From the initial population we eliminate $R = M \times P_e / 100$ individuals. Individuals are eliminated so as to preserve the diversity of the population or eliminate similar individuals. At the beginning, the entire population is sorted out according to the fitness function.

SOURCE CODE FOR SELECTION METHOD:

```
public static void selection() {
    int Vr_popul[] = new int[nM];
    double d[][] = new double[nM][nB];
    double e[] = new double[nM];
    for (int i=0; i<nM; ++i){
      for (int l=0; l<nB; ++l)
       d[i][l] = c[i][l];
      e[i] = f[i]; Vr_popul[i] = i;
    }
    shuffle(Vr_popul); int k = 0;
    for (int i=0; i<nA; ++i) {
      if (e[Vr_popul[k]] < e[Vr_popul[k+1]]){
       for (int l=0; l<nB; ++l)
              c[i][l]=d[Vr_popul[k]][l];
        f[i] = e[Vr_popul[k]];
      }
      else {
       for (int l=0; l<nB; ++l)
       c[i][l]=d[Vr_popul[k+1]][l];
        f[i] = e[Vr_popul[k+1]];
      }
      k += 2; }
}
```

After that we compare the similarity of fitness values of neighboring individuals and if their difference is less than predefined small positive real number ε, we eliminate one of the n - th. This is repeated until the number of eliminated individuals is less than R. If after this procedure the number of individuals that we eliminated is still less than R, then we eliminate individuals with worse fitness value function. The way selection operator works is realized through the UML activity diagrams [Appendix].

Operators of mutation

We used the so-called *2opt method* for the operator of mutation. This method is one of the most popular methods of local search in algorithms which solve the traveling salesman problem. The algorithm is described as follows: "Imagine the way from city A to city B and from city C to city D. Then we check whether AB + CD> AC + BD. If it is, then it comes to the replacement as shown in Figure 1. This is repeated as long as it possible to shorten the tour.“ The way this operator works is realized through the UML activity diagram [Appendix].

SOURCE CODE FOR MUTATION METHOD:

```
public static void mutation() {
    Random rVrednost = new Random();
    double r[] = new double[nB];
    for (int i=0; i<ne; ++i) {
      for (int l=0; l<nB; ++l) r[l] = c[i][l];
      int mb = (int)(nB*pMutate+1);
```

```
      for (int j=0; j<mb; ++j){
       int ib =
       (int)(nB*rVrednost.nextDouble());
         r[ib] = rVrednost.nextDouble(); }
      double e = cena(r);
      if (e<f[i]){
        for (int l=0; l<nB; ++l)
       c[i][l] = r[l]; f[i] = e; }
}
    int mmax = (int)((nM-ne)*nB*pMutate+1);
    for (int i=0; i<mmax; ++i) {
       int ig = (int)((nM-ne)*rVrednost.
       nextDouble()+ne);
      int ib =
      (int)(nB*rVrednost.nextDouble());
      c[ig][ib] = rVrednost.nextDouble(); }
}
```

Operators of crossing

We used the GSX (greedy subtour crossover) for the intersection operator which tries to take the longest subset of the cities from both parents. In this way the genetic material of the parents is best preserved. It actually means the following, if there are two chromosomes so that both contain the subsets the optimal tour, with this crossing we can very quickly get to the junction of those parts which of course leads to faster convergence of the problem.

SOURCE CODE FOR CROSSING METHOD:

```
public static void crossing() {
    Random rVrednost = new Random();
    int k = 0;
    for (int i=nA; i<nA+nA/2; ++i) {
       int nx = 1 + (int)(nB*rVrednost.
       nextDouble());
      for (int l=0; l<nx; ++l){
             c[i][l] = c[k][l];
             c[i+nA/2][l] = c[k+1][l]; }
      for (int l=nx; l<nB; ++l){
             c[i][l] = c[k+1][l];
             c[i+nA/2][l] = c[k][l]; }
      k += 2; }
}
```

In particular, this method is to take two randomly selected points, and replace their places, but provided only if the mutation route is less longer than the one before the mutation. The way this operator works is realized through the UML activity diagrams and sequence [Appendix].

The process of finding solutions and improving the initial solution using the GA

If, for example, we form 26 points (mark them with **A,B,C,...,Z**), also mention **x** and **y** coordinates. First,we determine the starting point (point A, for example), which is the focus of points. Then we calculate all the distances from the starting point to other points. In this way, we're forming the so-called distance matrix. Then there is a minimum for each line separately, or at least we're looking for the least remote point from point A, then the least remote point from point B and so on. The exception is if the point C is at least distant point from point A and again the least distant point from the point F. In this case we're seeking for the second minimum, therefore, we omit the used point (specifically in this example we omit the point C, because there is already a relation A to C, and in order to respect the condition that each city is visited only once, except of course the starting point from which all starts and ends). Then we form the route. The length of route at the departure point is 0, and then we add to it to the distance from point A and the point at least distant from it. Then, from that point (which is closest to the first), we look for at least distant point to the second point, and we of course leave out the used point A. And so on. So we look for the points that are free and never return to the previous one. And so on until the end which means to the last point that is of course connected to the first one. Sum all these distances is the initial solution.

For improving the initial solution, we used genetic algorithms. The steps for the application of genetic algorithms are as follows. At each step (iteration - *t*) we create a population of individuals that represent potential solutions ***P(t)*.** Each individual represents a potential solution to the problem and for each one we calculate the benefit (fitness function that determines whether one solution is better than others, it is calculated for each individual and depends on the problem that is solved) and select the best individuals for the next generation and the bad ones disappear. Crossing combines the genetic material of two parents in order to obtain superior successor (we also choose a random number r and if it is smaller than the crossing probability ***(Pc)*** than the crossing is performed. Mutation is done on a small fraction of the population to avoid instability of the procedure (we choose a random number r and if it is smaller than the probability of mutation ***(*** P_m ***)*** we change the genes.

THE PROCESS MODELING IN UML

UML standard that is applied to the object-oriented approach provides appropriate views of the system, so that are in all terms system can be described from a static (structural) and dynamic aspect. It is used to design software which needs a plan, offers the possibility of visualization in multiple dimensions and levels of detail and is suitable for upgrading old systems. It is quite certain, that such action will simplify the process of obtaining a solution.

This paper presents modeling first through the static and then dynamic diagrams. This is a good way to resolve because the UML describes the source code, models help to visualize the system as it is or what it should be and allow you to determine the structure and behavior of the system. Models document the decisions that we were making and provide solutions to guide us during the construction of the system and specific applications.

The already mentioned diagram 1 of cases of usage is to represent the functional requirements that the system needs to fulfill. It is composed of a single actor and the cases of usage arising from the description of genetic algorithms (initialization, evaluation, selection, crossing and mutation). All cases of use in the appropriate context, and there is interaction between them.

Activity diagram 2 is actions that are performed, namely, this diagram gives a general view of the activities that are further decomposed. Then the following diagrams represent the decomposition of the activities mentioned to sub-activities containing concrete actions (Diagram 3 and 4). Diagram 3 gives a visual representation of how the process of selection is calculated, so-called, fitness function and probability for each member individually, while in the diagram 4 we stated how the use of the obtained values from previous activity and selection procedure of members is based on their probability.

The sequence diagram 5 presents the so-called GSX intersection operator and the way it works. This diagram presents mutual interaction among objects (parent, child, methods GSX) and which generally represents a series of exchanges of messages between classes, with the sequence and time course of sending and receiving messages clearly marked. Diagrams 6 and 7 represent the actions that are carried out within the activity of "crossing" and describe GSX operator from another aspect. In the previous sequence diagram, the order and the interaction between objects that participate in these activities are clearly indicated, and these diagrams show that the emphasis is on the actions and their implementation and the conditions that exist. Diagram 8 is decomposition of "mutations" activities and represents the actions performed within this operator.

DESCRIPTION OF THE APPLICATION, TESTING AND RESULTS

The application we have provided is designed in the DELPHI software package, and is based on previously described UML diagrams. The design of this application is mentioned in appendix [Figure 2] for the traveling salesman problem, which contains the relevant sections (panels).

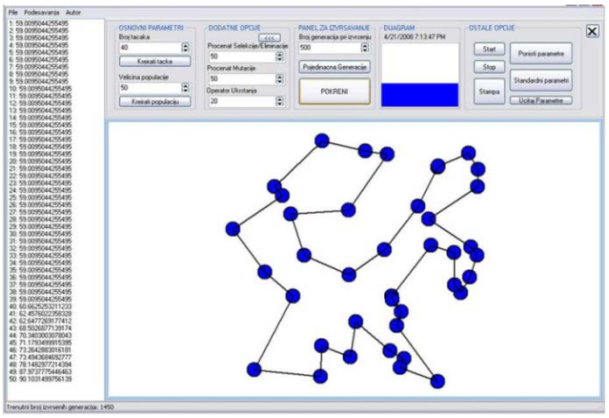

Figure 2: Application TSP

The largest part of the application contains a panel where the nodes are displayed. In the upper part of the application, there is a part of the required parameters input [Figure 3] where there following options are given:

- input of number of points,
- number of population,
- the percentage of selection, mutation, crossing,
- number of generations during the execution,
- individual performance review and
- button for the total number of generations.

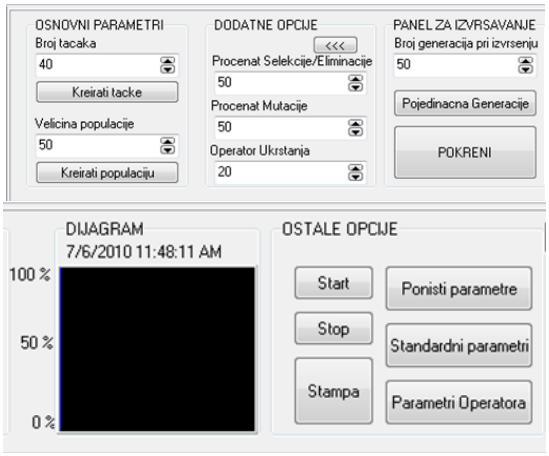

Figure 3: Panels

We have also mentioned a diagram of a graphical representation of relations between the initial lengths of the route and obtained optimum length of the route since the last generation. At the bottom of the application there is a status bar displaying the current number of generation executed. In the left part of the application [Figure 4] there is a panel that displays the times for each route (path length).

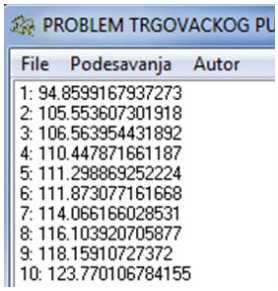

Figure 4: Scroll population

The picture [Figure 5], shows how the application works, i.e. while being tested in a few generations, we can see how we get the optimal solution. We input the following parameters:

- *number of points = 40,*
- *number of population = 50,*
- *number of generations = 50,*
- *parameters that are specified for GA operators 50, 50, 20*

and we can see in the picture shown below the initial solution, solution during the performance and in the end the optimal solution.

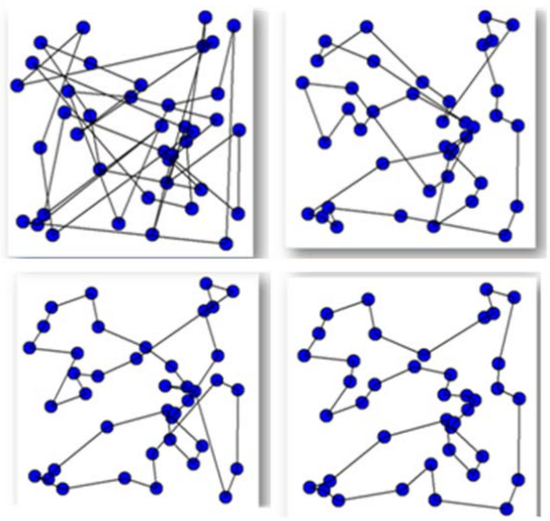

Figure 5: The process of finding the optimal solution of TSP application

Here are the results table for additional testing:

TEST RESULTS - THE FIRST PART OF THE TABLE

number of points	number of generations	S	M	C	random path (A)	optimal path (B)
50	1000	50	50	20	299.92	64.47
50	1000	80	80	50	276.24	60.33
100	1500	50	50	20	583.41	108.25
500	2000	50	50	20	2620.13	894.01
500	2000	80	80	20	2798.21	990.87
750	2500	50	50	20	4078.13	1396.1
900	3000	30	30	20	4917.41	2369.3
1000	3000	50	50	20	5319.07	2097.1

TEST RESULTS - THE SECOND PART OF THE TABLE

number of points	shortened for (A-B)	%	the length performance of TSP
50	235.45	*78.50*	0:21
50	215.91	*78.16*	0:22
100	475.16	*81.44*	0:41
500	1726.12	*65.88*	6:48
500	1807.34	*64.59*	6:56
750	2682.09	*65.77*	8:51
900	2548.05	*51.82*	11:35
1000	3221.93	*60.57*	15:02

In the second picture [Figure 6], we specified the comparison between the initial and optimal solution and we can see that the length of time decreased to 73.04% (from 234.69 to 53.05).

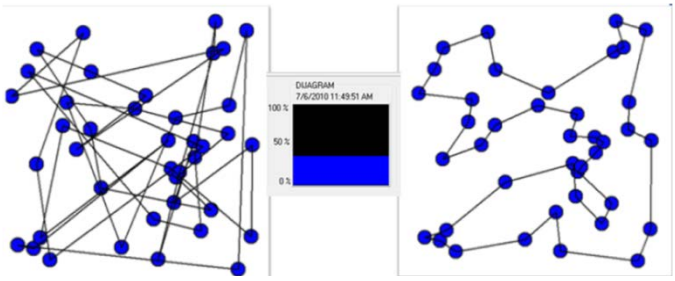

Figure 6: The results of this testing (ratio A,B)

CONCLUSION

The application of genetic algorithms is very wide, they are actually only a principle, idea or policy to solve a problem in a different way than traditional methods, because it's all up to the user to decide on whether to develop his own algorithm, or will he try to adapt his problem to some existing algorithm that solves a similar class of problems. Also, we see that genetic algorithms are useful for those classes of problems that cannot be solved in classical ways. Although the speed is not at the top, by the size of the area they search through they are far better than any other method. It is very well seen in the example of the traveling salesman problem.

We have listed UML modeling as specific for these problems, and the main aim is to present detailed procedure and the functioning of genetic algorithms and a way of solving problems by applying them. Finally, practical work of this application has shown that one of the most important things for successful work of this algorithm is choosing the correct genetic operators and parameters that will determine the behavior of these operators. If they are correctly set, the algorithm gives fantastic results, but if the choice of these operators is not advised, algorithm will end up working in a local optimum, closer or further from the true optimum, depending on the parameters.

REFERENCES

M. Garey, D. Johnson (1979), "Computers and Intractability: A Guide to the Theory of NP-Completeness". *W.H. Freeman*.

T. Cormen, C. Leiserson, R. Rivest, C. Stein (2001), Introduction to Algorithms, *MIT Press and McGraw-Hill*, second edition.

M. Pilat, T. White (2002), Using genetic algorithms to optimize ACS-TSP, *Lecture Notes in Computer Science*, v. 2463, pp. 282-287.

J. Ngassa, J. Kierkegaard (2007), ACO and TSP, *Roskilde University, Bachelor of Computer Sciences*, 2nd module.

I.Salajic, J.Nikolic, M. Žoljom (2001), TSP – Problem trgovackog punika na potpunom grafu, primjenom genetskog algoritma (GA) i algoritma simuliranog kaljenja (SA).

R.Thamilselvan, P. Balasubramanie (2009), A Genetic Algorithm with a Tabu Search for Travelling Salesman Problem. *Int. Journal of R.T. in Engineering*, Vol. 1, pp. 607-610.

J. Kratica (2000), Paralelizacija genetskih algoritama za rešavanje nekih np - kompletnih problema, *Doktorska disertacija*, Beograd.

D. Saiko (2005), Traveling Salesman Problem , Java Genetic Algorithm Solution.

B. Barán, O. Gomez (2005), Omicron ACO. A New Ant Colony Optimization Algorithm, *CLEI Electron. J. 8* (1).

G. Gutin, A.Punnen (2002), The traveling salesman problem and its variations, Kluwer, Dordrecht.

S. Eberle (2008), A Polynomial Algorithm for a NP-hard to Solve Optimization Problem, Dissertation der Fakultät für Physik der Ludwig-Maximilians-Universität München.

C. Nilsson (2003), Heuristics for the traveling salesman problem. *Tech. Report*, Linköping University, Sweden.

M. Held, R. Karp (1970), The Traveling Salesman Problem and Minimum Spanning Trees, *Operations Research* 18.

J. Little, K. Murty, D. Sweeney, C. Karel (1963). An algorithm for the traveling salesman problem. *Operations research*, 11 (6), pp. 972-989.

C. White, G. Yen (2004), A Hybrid Evolutionary Algorithm for Traveling Salesman Problem, Evolutionary Computation, 2004. CEC2004. Congress on, Vol.2, pp. 1473 – 1478.

P. Borovska (2006)., Solving the Travelling Salesman Problem in Parallel by Genetic Algorithm on Multicomputer Cluster, *International Conference on Computer Systems*.

Applegate, D., Bixby, R., Chvátal, V., Cook, W (1998), On the solution of traveling salesman problems. Documenta Mathematica Extra Volume (Proceedings of the International Congress of Mathematicians), pp. 645-656.

D. Johnson, L. McGeoch (1995), The traveling salesman problem: a case study in local optimization.

S. Lin, B. Kernighan (1973), An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2), pp. 498-516.

Graph Theory, http://en.wikipedia.org/wiki/Graph_theory

Traveling salesman problem: http://www.tsp.gatech.edu/

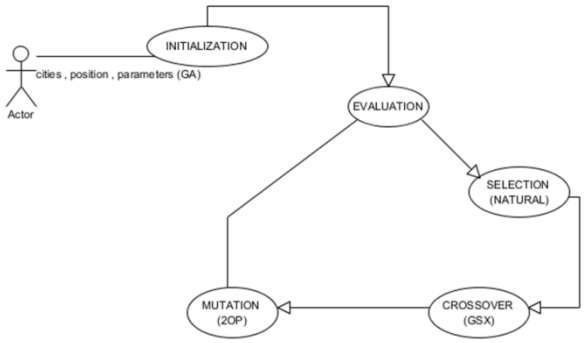

Diagram 1: Use case diagram

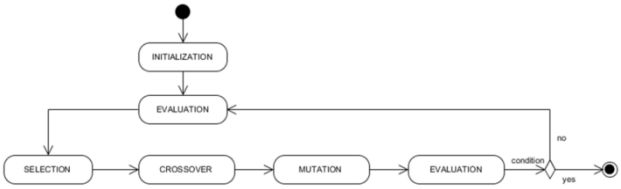

Diagram 2: Activity diagram

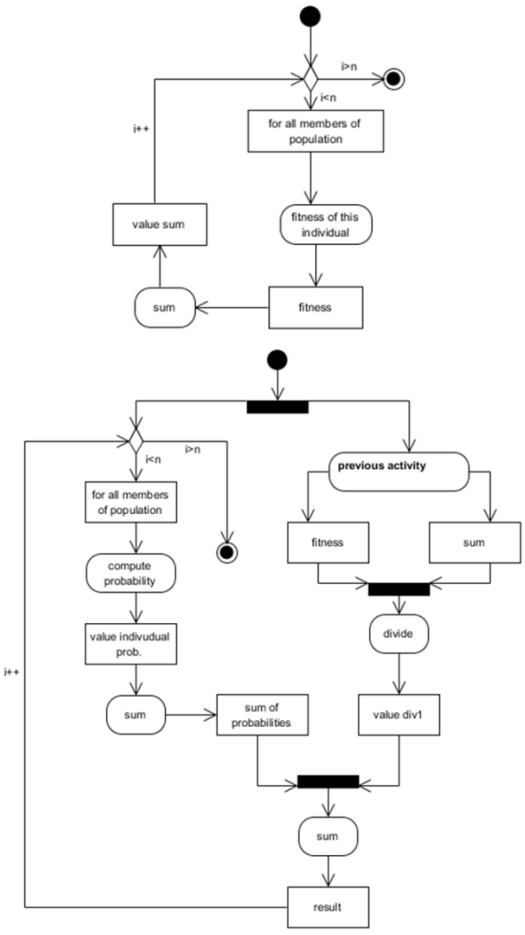

Diagram 3: Activity diagram - Generic selection procedure (fitness, probability)

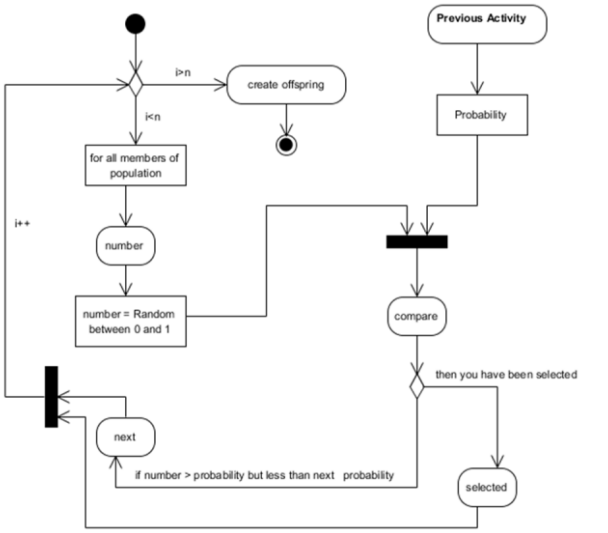

Diagram 4: Activity diagram - Generic selection procedure

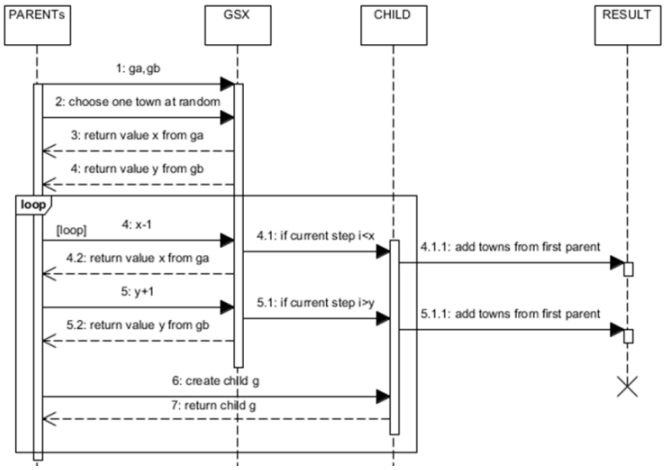

Diagram 5: Sequence diagram - Greedy Subtour Crossover (GSX)

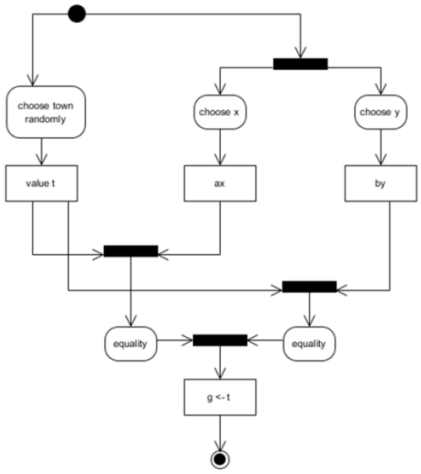

Diagram 6: Activity diagram - Greedy Subtour Crossover (GSX)

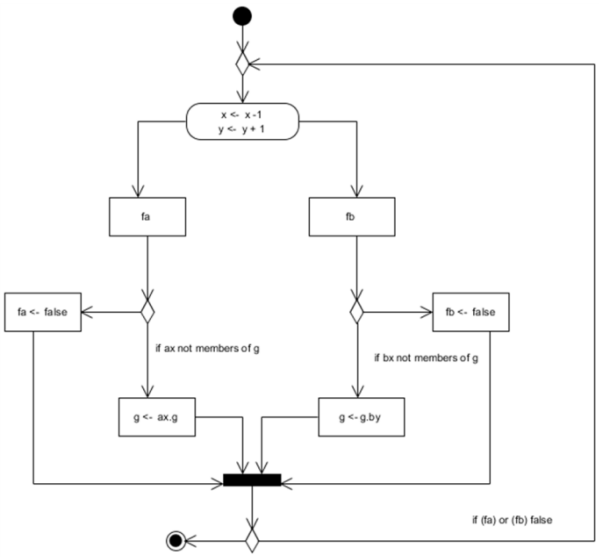

Diagram 7: Activity diagram – Greedy Subtour Crossover (GSX)

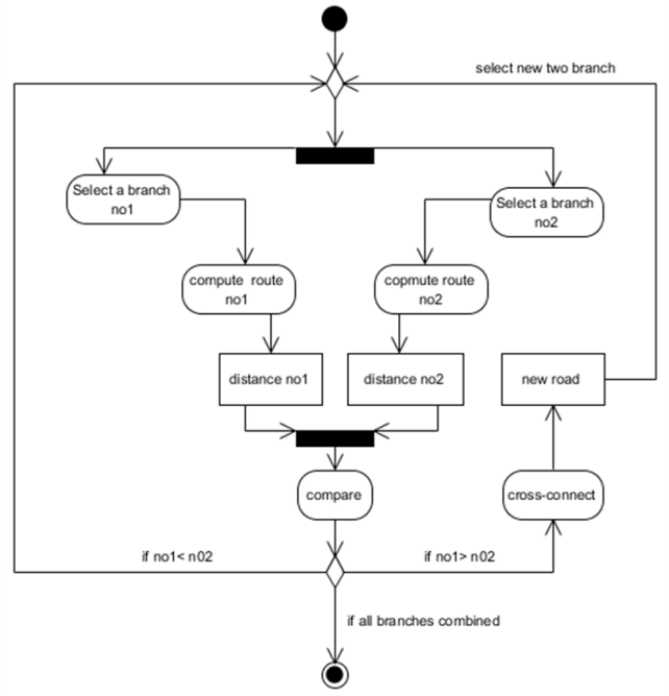

Diagram 8: Mutation by 2-opt