Proceedings of the Federated Conference on
Computer Science and Information Systems pp. 131–136

ISBN 978-83-60810-51-4

Comparison of a memetic algorithm and a tabu search algorithm for the Traveling Salesman Problem

Eneko Osaba
Deusto Institute of Technology.
University of Deusto, Av.
Universidades, 24, Bilbao, Spain.
Email: e.osaba@deusto.es

Fernando Díaz
Deusto Institute of Technology.
University of Deusto, Av.
Universidades, 24, Bilbao, Spain.
Email: fernando.diaz@deusto.es

***Abstract*—The traveling salesman problem, or TSP, is one of the most famous and well studied problems in combinatorial optimization. There are many studies with the objective of finding an optimal solution for this problem. These studies have not been successful, since it is considered to be an NP-Hard problem. This means that is not possible to find a method that ensures an optimal solution for all instances of this problem. In this paper we present two techniques to solve this problem, a tabu search based algorithm and a memetic algorithm. The results of both techniques are shown and compared to decide which one of the two alternatives gets better results. Apart from this, several studies are performed to determine certain aspects of the algorithms, such as the crossover function for the memetic algorithm or the size of the tabu list.**

I. Introduction

The traveling salesman problem, or TSP, is a very well studied problem in combinatorial optimization [1]. Its formulation is simple: Given a finite number of cities or nodes N and the corresponding costs cij of travelling from the city i to the city j, the objective is to find a cyclic permutation that minimizes the cost of visiting every city only once. The TSP is very easy to describe, yet very difficult to solve. It is considered an NP-Hard [14] problem. This means that is not possible to find a method that ensures an optimal solution for all instances of this problem within reasonable execution time. For this reason, there are numerous strategies to find an acceptable solution, taking into account the basic criteria of the computational complexity: execution time and resources used. In this paper we present two algorithms, one algorithm is based on tabu search, the other is a memetic algorithm, which combines a genetic algorithm with a tabu. After presenting the algorithms, we show the results offered by each one applied to different instances.

II. Tabu search

The tabu search is an algorithm designed to find good solutions for combinatorial optimization problems. It is one of the most widely known metaheuristics, due to the quality of its solutions. In this section, the different parts of the implemented algorithm are presented.

A. Initial solution

This kind of algorithms starts with an initial solution, called initial state, which is modified during execution in order to find a near optimal solution. The initial solution can be generated at random or can be generated by a function. This function is used with the aim of starting the process with an acceptable solution. Logically, this initial solution is not the optimal one, but in most cases it is a good starting point to begin the search. In this paper we present the implementation and comparison of four initialization functions, starting with a partial path and building step by step the initial solution.

Nearest Neighbor: This function begins by selecting a random node as start node. Then iteratively, the nearest node to the last node added is added to the partial path.

Insertion: This function starts the process with a partial path consisting of two cities. These cities are the closest on the map. This means that the process starts with the shortest edge. From this point and iteratively, the nearest node to any node in the partial path is obtained. This node is inserted into the path at the position that involves lower cost increase.

Farthest Insertion: This is a modification of the above function. In this case, the node farthest from any of the nodes that are already in the partial path is taken as the node to insert. After that, the node is inserted into the position that involves lower cost increase.

Solomon Heuristic I1 [2]: In 1987, Solomon proposed three heuristics for the VRPTW problem. From the three, the best is called I1. In this article, a modification of this heuristic has proposed, to adapt it to the TSP problem. This function starts execution with the longest edge, in other words, begins with a partial path consisting of the two most distant nodes. Then, iteratively, it calculates for each node (which is not yet in the path) the cost increase of inserting it at each one of the possible positions in the partial tour. The node that involves smaller increase is inserted into the path in the optimal position.

These functions have been tested with different instances of the TSP problem, concluding that the two functions that offer better results are Farthest Insertion and I1. The instances are taken from TSPLIB [3] and [4].

TABLE I.
INITIAL SOLUTIONS OBTAINED BY EACH FUNCTION

TSP Instance	Oliver30	Eilon50	Eilon75	Eil101
Optimal result	420	421	535	629
NN*	509	566	620	817
Insertion	466	475	622	738
Insertion_Farthest	**452**	485	**588**	720
I1	471	**464**	597	**719**

NN*= these values are approximated, since the result depends on the randomly selected initial node.

B. Successors generation function

As explained above, the tabu search algorithm begins with an initial solution. Then iteratively, the current state will be modified by a successors generation function with the aim of finding better solutions. In this paper, we have implemented two functions, widely used for this type of algorithms [5] [6]:

Vertex Insertion: In this function, the generator examines a part of current solution's neighborhood by executing a finite number of random movements. A single movement consists of selecting randomly one node and removing it from the current route. Then, this node is inserted at another position between two nodes to generate a new solution. The best move of all is selected and applied. This means that in every movement, function moves to the best neighbor found.

Swapping: In this function, a movement consists of selecting two nodes and swapping their positions. Like the Vertex insertion, a finite number of movements in the neighborhood are examined and the best of them is selected and applied.

C. Tabu memory

The tabu search algorithm has the property of accepting worse solutions than the current one, with the aim of avoiding premature local optima. To prevent this, there is a structure called memory. This structure is used to prevent the search from re-visiting solutions visited in the immediate past. The forbidden movements are normally stored in a list called the tabu list. Whenever the successors generation function is going to make a movement, that is, a change in the current state, the tabu list is examined to check whether the movement is forbidden. If it is, the movement is discarded and another one is tried When an allowed movement is found, the new state is accepted and the movement is stored in the list to prevent the algorithm from visiting the same state in a near future.

There are many criteria to decide whether a movement is tabu or not, some of them more restrictive than others. There are many studies on this subject [7]. In addition, there are different approaches according to the successors generation function that is being applied. Here are some examples of these criteria, sorted from least to most restrictive and valid for the VertexInsertion function, which is the one applied in this algorithm:

- **Vector (I, POSITION (I), POSITION (J)):** This vector is used to prevent the node I move from POSITION (I) to POSITION (J).
- **Vector (I, POSITION (I)):** Similar vector, to prevent the node I move from POSITION (I) to any other position.
- **Node I, L:** The list includes the identifier of the node, and prevents it from being displaced to the left
- **Node I:** The node identifier is stored, in the same way as the above criteria, and prevents it from any move.

D. Tabu list and duration of tabu status

When the algorithm generates new successors using the generating function, the last movement is inserted into the tabu list. The information inserted in this list is maintained a certain time, which means that the tabu movements will be prohibited only for a period of time, usually translated into a number of iterations of the algorithm.

Another topic widely studied, and even undetermined, is the ideal size of the tabu list [8]. If the size is small, cycling phenomena will be evident, whereas, if it is large, the process might be driven away from the vicinity of global optimum.

In the implemented algorithm the restrictive criterion "Node I" has been used, which is the most restrictive one. This implies that the tabu list will have a small size and movements will be prohibited for a short period of time. In our case, this period is defined by the size of the list. For example, if the list stores the last 10 movements, each of them remains on the list for 10 iterations, since the new movements are added to the list as a queue.

A small survey that allows to decide the optimal size of the list for our algorithm will be shown later.

E. Aspiration criterion

The execution of the algorithm finishes after a finite number of iterations without improving the best solution found. The ideal is to set this number according to the neighborhood size, which depends on the successors generation function. This number, in our algorithm, is equal to the neighborhood size multiplied by 15 in instances with few cities and neighborhood size multiplied by 10 in problems with many nodes, such as Eil101.

F. Results of the initialization function

Here are the results obtained by the algorithm, using different initialization functions. For these tests we used the successors VertexInsertion generation function and no tabu mechanism. A hit is an execution in which the optimal solution has been found.

TABLE II.
TESTS FOR INSTANCE OLIVER30

Oliver30 (420)	Hits	Average	Avg. %	Ex. Time
NN	29/30	420.03	0.007%	< 1 sec.
I1	**30/30**	**420**	**0%**	< 1 sec.
Insertion	23/30	420.23	0.23%	< 1 sec.
Insertion_F	**30/30**	**420**	**0%**	< 1 sec.

TABLE III.
TESTS FOR INSTANCE EILON50

Eilon50 (425)	Hits	Average	Avg. %	Ex. Time
NN	6/30	427.16	0.50%	1.5 sec.
I1	**8/30**	**426.26**	**0.29%**	1.5 sec.
Insertion	7/30	426.96	0.46%	1.5 sec.
Insertion_F	**8/30**	426.43	0.33%	1.5 sec.

TABLE IV.
TESTS FOR INSTANCE EILON75

Eilon75 (535)	Hits	Average	Avg. %	Ex. Time
NN	6/30	537.63	0.45%	9 sec.
I1	**17/30**	**536.16**	**0.21%**	6 sec.
Insertion	10/30	536.33	0.24%	6 sec.
Insertion_F	5/30	539.56	0.85%	8 sec.

TABLE V.
TESTS FOR INSTANCE EIL101

Eil101 (629)	Hits	Average	Avg. %	Ex. Time
NN	0/30	635.56	1.04%	25 sec.
I1	**24/30**	**630.5**	**0.23%**	15 sec.
Insertion	0/30	637.23	1.30%	25 sec.
Insertion_F	0/30	632.96	0.62%	25 sec.

After analyzing the results, several conclusions can be obtained. In instances where there are few stations, such as Oliver30, the tabu list size is irrelevant, because with a good initial solution, it is easy to reach the global optimum. However, in instances with larger number of nodes, the size of the list is very important. These results confirm the theory explained above:

- **Conclusion 1:** If the size of the list is small, it is easy to fall into local optima. In contrast, if the size is large, the search space can be very constrained, and can deviate from the vicinity of global optimum.

Based on this conclusion, the best option is to perform tests with different sizes to see which is best suited. There is no universal rule to decide the optimal size of the list since it is very dependent on the size of the instance, the tabu criterion and the successors generation function.

In this case, we can distinguish a best choice, achieving remarkable results in Eil101 and Eilon75 instances, and with exceptional results in the other alternatives. This option is N/4. In Eilon50, however, the best alternative is N/2, but the alternative N/4 also gives good results.

Analyzing the result set, and more specifically the instance Eil101, we can see clearly how a large size of the list causes the deterioration of the results. In this instance the size N/8 is a really good option and N/10 becomes the best alternative.

Anyway, in case of selecting one of the 5 options studied, we arrive at the following deduction.

- **Conclusion 2:** Using VertexInsertion successors generator and a strict criterion of tabu movements, for instances of up to 101 cities, the ideal size of the tabu list is an approximate size of N/4, one quarter of the total number of nodes.

Anyway, we have seen in these results how the optimal size can vary, depending on the instance. It is for this reason that the right thing is to reach the following conclusion.

- **Conclusion 3:** There is no an optimum size for the tabu list, because this depends on the size of the instance, the successors generation function and the movement prohibition criteria.

G. Final results of the algorithm

The final results of the implemented algorithm can be summarized in the following table. The tests were performed on an Intel Core i5 – 2410 laptop, with 2.30 GHz and a RAM of 4 GB. This time, instead of 30 executions per instance, we have carried out 50.

TABLE VI.
FINAL RESULTS OF THE ALGORITHM

	Hits	Average	Avg%	Time
Oliver30 (420)	50/50	420	0%	0.28 sec.
Eilon50 (425)	22/50	425.7	0.16%	3.14 sec.
Eilon75 (535)	40/50	535.6	0.11%	20.25 sec.
Eil101 (629)	40/50	629.46	0.07%	34 sec.

III. PROPOSED MEMETIC ALGORITHM

Evolutionary algorithms are based on the laws of the evolution of species. These algorithms work on populations of organisms, in other words, on sets of intermediate solutions. The organisms interact with each other to generate new organisms and these are added to the population. After a finite number of generations, or iterations, the algorithm terminates execution and returns as a final solution the best organism in the population.

In this paper we have developed a new hybrid algorithm, which combines a genetic algorithm with a tabu search. There are different types of hybrid or memetic algorithms, which combine different techniques, two examples are [9] and [10]. The aim is to study the effectiveness of this new algorithm and see if it is worth using an algorithm of this nature rather than a tabu search algorithm.

In the next sections we explain in detail the algorithm and the results obtained.

A. Generation of initial population

The starting point of the algorithm is a set of solutions called population of individuals. Every individual in the population consists of a single chromosome, a vector of integers that represents a route. In many cases, these individuals are randomly generated. In our case, we use a technique that combines initialization heuristics and random generation. This technique creates an individual for every initialization heuristic described in the tabu search section, while the others are generated randomly, until completing the entire population. Thus, the process begins with a population in which there are a number of "acceptable" solutions and a number of random solutions.

B. Selection of parents

The crossover is the process in which the chromosomes of a population interact to generate new individuals. Generated individuals are called children, and as in natural law, every child must have a father and a mother. Therefore, for this process, the algorithm has to choose the chromosomes that will form part of the process.

There are many ways for selecting parents. In this case we have used an elitist method, in which the individuals with the highest fitness value are selected. In other words, the best individuals of the population are selected.

C. Crossover

There are many crossover functions [15]. In this section, the functions we have been implemented will be described and some results will be shown to see which of them is the best.

Order Crossover (OX1): The Order crossover was proposed by Davis [11]. This operator builds the children by choosing a sub-route of one of the parents and maintaining the order of the cities of the remaining parent. For example, suppose these two individuals.

P = (1 2 3 4 5 6 7 8)
M = (2 4 6 8 7 5 3 1)

Now, two cut points are selected, identical for both parents. Assuming that these breakpoints are located between the position two and three and between five and six:

P = (1 2 | 3 4 5 | 6 7 8)
M = (2 4 | 6 8 7 | 5 3 1)

The children will be created as follows. First, the segments between the cut points are preserved, as follows:

H1 = (* * | 3 4 5 | * * *)
H2 = (* * | 6 8 7 | * * *)

Then, starting with the second breakpoint, the remaining nodes are inserted in the same order they appear in the other parent, considering that the cities that has already been inserted have to be omitted. When the end of the string is reached, it continues through the beginning of this. The children resulting from this example would be these:

H1 = (8 7 | 3 4 5 | 1 2 6)
H2 = (4 5 | 6 8 7 | 1 2 3)

Modified Order Crossover (MOX): This crossover proposed by Shubhra [12] selects a cut point that divides each parent into two sections. Assuming the following parents:

P = (1 2 3 4 | 6 9 8 5 7)
M = (2 1 9 8 | 5 6 3 7 4)

The cities on the left of the cut point impose their order on the other parent:

H1 = (1 2 * * * 9 8 * *)
H2 = (2 1 * * * * 3 * 4)

The remaining cities are inserted into the children in the same order they appear in the other parent. This means that the resulting children would be:

H1 = (1 2 5 6 3 9 8 7 4)
H2 = (2 1 6 9 8 5 3 7 4)

Very Greedy Crossover (VGX): This operator introduced by Bryant [13] is more "customized" than previous ones, since it takes into account the distances between cities to generate the children. First, the function randomly selects a node. Assuming that these two chromosomes are the parents:

P = (1 2 3 4 5 6 7 8)
M = (2 4 6 8 7 5 3 1)

Randomly selected as initial node number 2, momentarily, the child would be the next:

H = (2 * * * * * * *)

Now, looking at the parents, the nodes adjacent to the last node added (and still not part of the child) are selected as “possible nodes”. In this case these nodes are 1, 3 and 4. From all “possible nodes”, the closest to the last node added is added to the child. Suppose that in this case is number 4. At the moment, this would be the new child:

H = (2 4 * * * * * *)

This process is repeated until all cities have been inserted.

D. Results of the crossover

With these three functions, various tests were performed to check which of them was the most efficient. The test results will not be shown, as they have a very long length. Even so, we will explain what are the conclusions obtained.

First of all is that the operator Very Greedy is the most efficient function, because it creates the best individuals. This is because it is a specifically designed operator for this type of problem. This function takes into account the distances between cities, therefore, whenever two chromosomes are crossed, resulting child will always be equal or better than either parent.

This feature is not present in the other two crossover functions, because they make "semi-blind" crosses. It is true that they attempt to maintain the structure and quality of the parents in the resulting individuals. Even so, these functions make "blind" unions, since in most of them, the routes are divided into sub-routes or small portions of routes, and then they are joined together regardless to the cost that this operation involves. This has the consequence that the resulting children may be worse than their parents, which is clearly seen in the presented results. Three conclusions can be drawn:

- **Conclusion 1:** Considering the three functions, Very Greedy Crossover (VGX) is which produces better individuals
- **Conclusion 2:** It is recommendable to use “customized” crossover operators for each problem. For example, for the TSP, it is possible to make operators taking into account the distances among cities. Thus, the resulting individuals will always be better than their parents.
- **Conclusion 3:** Crossover functions without "customization" do not ensure that the resulting children

improve the fitness of their parents, because they are based on "blind" unions.

Another thing to mention is that the VGX operator generates fewer individuals than the other operators. This shows that, despite generating a lot of children is a good quality, the fact of producing “customized” individuals is more advisable.

E. Mutation process

This process is done after the crossover, and it is performed on the resulting individuals. It is possible that during the execution of the algorithm, the population is moving towards a local optimum. For this reason exists this process, which selects with a certain percentage of probability the individuals produced by crossover and makes a small change in them.

There are different ways to mutate. In this case we have implemented a process in which each chromosome is mutated with a probability of 20%. In case of making the process, the mutation generates various node exchanges in the individual. The number of exchanges is given by an attribute of the algorithm, which is introduced beforehand and called mutation factor.

For example, suppose this chromosome:

C = (1 2 3 4 5 6 7 8 9 0)

The mutation factor is 0.2. This means that 20% of the chromosome will be changed. In this case the path is composed of 10 cities, therefore the number of exchanges will be two. Supposing that these two changes are between nodes 3 and 7, and node 2 and 9:

C’ = (1 2 7 4 5 6 3 8 9 0)

C’’ = (1 9 7 4 5 6 3 8 2 0)

Thus, the resulting chromosome is as follows:

CM = (1 9 7 4 5 6 3 8 2 0)

In order to preserve the quality of the population, the mutated chromosome and original chromosome are stored.

F. Application of Tabu Algorithm

This step is the one that distinguishes this algorithm from a traditional genetic algorithm. After crossover and mutation process, with the aim of optimize the new chromosomes, a reduced version of the tabu search algorithm described above is applied. As in the mutation process, this process is applied only to a small part of the population, in order not to increase excessively the execution time. This process is applied to each individual with 20% probability.

The characteristics of tabu algorithm are the same as the previously explained, with the difference that the initial solution is provided by the memetic algorithm. From this partial solution, the process runs autonomously.

In the simple version, the finish criterion of the tabu search was determined by a number of iterations without improvements in the quality of the solutions, close to the neighborhood multiplied by 10 or 15. In this case, this value is lower, so, the execution time is reduced drastically.

G. Selection of survivors

The population in a memetic algorithm has a finite size. Therefore, it is necessary to reduce the number of individuals after crossover and mutation and discard those that are less interesting. This is accomplished by the survivors selection function. In this case a function called ElitistRandomSurvivals is used. This function selects the surviving population considering two parts, one part will consist of the best individuals, while the other part will be selected at random. For example, if we have a population of 50 individuals, and we have to reduce it to 30 individuals, this function will select the best 15 chromosomes according to their fitness, and the other 15 will be selected at random from the remaining 35.

The reason to select half of the population at random is to maintain some diversity, in order to avoid local optima.

H. Results of the memetic algorithm

These are the results obtained with the implemented memetic algorithm. The computer used is the same as in section 2.8.We have executed 50 iterations for each instance:

TABLE VII.
FINAL RESULTS OF THE ALGORITHM

	Hits	**Average**	**Avg. %**	**Ex. Time**
Oliver30 (420)	50/50	420	0%	0.3 sec.
Eilon50 (425)	48/50	425.04	0.009%	11 sec.
Eilon75 (535)	40/50	535.3	0.05%	55 sec.
Eil101 (629)	50/50	629	0%	12 min.

IV. TABU VS. MEMETIC ALGORITHM

After analyzing the results, the first conclusion and clearer, is obtained by observing that the results obtained by the memetic algorithm are visibly better than those obtained by the tabu algorithm. This improvement in the results comes with an increase in execution time, which in the case of the first three instances, it is not very large. By contrast, in the instance Eil101 although the algorithm finds an optimum in all executions, the time increases in excess.

- **Conclusion 1:** The memetic algorithm obtains better results than the tabu algorithm, although in instances with more than 75 nodes the runtime is too high.

Generally, both algorithms offer good results, but each one has a characteristic that makes it different from the other. The tabu search algorithm gives better execution times, while the memetic algorithm provides a higher success rate. For this reason, depending on the needs an algorithm or the other should be used.

- **Conclusion 2:** If the runtime is more important than obtaining better results, the correct choice is to use the tabu search algorithm. Otherwise, the right choice is the memetic algorithm.

I. Conclusions and Future Work

In this paper we have presented two different algorithms for solving the TSP. On the one hand, a tabu algorithm, on the other hand, a memetic algorithm. We have described their characteristics, including various studies about some important aspects, such as the initialization function of the tabu search algorithm or the crossover functions of the memetic algorithm. Different instances have been used for testing, as Oliver30, Eilon50, Eilon75 and Eil101, and the results have shown. Finally, a comparison between the two algorithms has been made, concluding that the choice of algorithm is influenced by the priorities of the user. In case of giving priority to the execution time, the right choice is the tabu search algorithm. In case of giving priority to the quality of the solution, the correct choice is the memetic algorithm.

We are currently working on the adaptation of these algorithms to the VRP problem. We have planned several studies to determine the influence of the process of mutation in evolutionary algorithms and the influence of survivors selection functions.

Finally, this research is part of the PRODIS project (Grant PI2011-58, funded by the Basque Government in Spain).

References

[1] Lawler E. L., Lenstra J. K., Rinnooy K. and Shmoys D. B. The Traveling Salesman Problem: A guided tour of combinatorial optimization. Wiley-Interscience Publication, 1985.

[2] Solomon M. M. Algorithms for the vehicle routing and scheduling problems with time windows. IFORMS Operations Research, no. 35, pp. 254-265, 1987.

[3] TSPLIB, comopt.ifi.uni- heidelberg.de/software/TSPLIB95/

[4] Whitley D., Starkweather T. and Fuquay D. Scheduling Problems and Traveling Salesmen The Genetic Edge Recombination Operator International Conference on Genetic Algorithms, 3:133-140, 1989.

[5] Scheuerer S. A tabu search heuristic for the truck and trailer routing problem. Computers & Operations Research, 33:894-909, 2006.

[6] Montane F.A.T. and Galvao R.D. A tabu search algorithm for the vehicle routing problem with simultaneous pick-up and delivery service. Computers & Operation Research, 33:595-619, 2006.

[7] Malek M., Guruswamy M., Pandya M., and Owens H. Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem. Annals of Operations Research, 21:59-84, 1989.

[8] Tsubakitani S. and Evans J. R. Optimizing tabu list size for the traveling salesman problem. Computers & Operations Research, 25:91-97, 1998.

[9] Pop P.C., Iordache S. A hybrid heuristic approach for solving the generalized traveling salesman problem. Genetic and evolutionary computation conference, GECCO 2011:481-488, 2011.

[10] Gutin G. and Karapetyan D. A memetic algorithm for the generalized traveling salesman problem. International Journal of Natural Computing Research, 9: 47-60. 2010.

[11] David L. Applying Adaptive Algorithms to Epistatic Domains. Proceedings of the International Joint Conference on Artificial Intelligence, 162-164, 1985.

[12] Ray, S. S.; Bandyopadhyay, S. and Pal, S. K. New Operators of Genetic Algorithm for Traveling Salesman Problem. Proceedings of the 17th International Conference on Pattern Recognition, 497-500, Vol. 2, 2004.

[13] Julstrom B. A. Very Greedy Crossover in a Genetic Algorithm for the TSP. Proceedings of the 1995 ACM symposium on Applied computing, 324-328, 1995.

[14] M. R. Garey, and D.S. Johnson. Computers and Intractability; a Guide to the Theory of Np-Completeness. W. H. Freeman & Co. 1990.

[15] P. Larranaga, C. M. H. Kuijpers, R. H. Murga, I. Innza and S. Dizdarevic. Genetic Algorithms for the Traveling Salesman Problem: A Review of Representations and Operators. Artificial Intelligence Review,13:129-170.1999.