World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:1, No:7, 2007

Genetic Algorithms with Oracle for the Traveling Salesman Problem

Robin Gremlich, Andreas Hamfelt, Héctor de Pereda, and Vladislav Valkovsky

International Science Index, Computer and Information Engineering Vol:1, No:7, 2007 waset.org/Publication/4895

***Abstract*—**By introducing the concept of Oracle we propose an approach for improving the performance of genetic algorithms for large-scale asymmetric Traveling Salesman Problems. The results have shown that the proposed approach allows overcoming some traditional problems for creating efficient genetic algorithms.

***Keywords*—**Genetic algorithms, Traveling Salesman Problem, optimal decision distribution, Oracle.

I. Introduction

THE Traveling Salesman Problem (TSP) was formulated in the 30th year of the 20th century and till now it is one of the most popular and important combinatorial problems.

The TSP is stated as follows:

Given a finite number of "cities" along with the cost of travel between each pair of them, find the cheapest way of visiting all cities and returning to the starting point [7].

Although it seems to be quite simple, solving it is very computationally expensive. As no polynomial time algorithm has been discovered, the only way to obtain the best solution is by calculating all possible routes. If we have a map with N cities, the number of possible routes will be N!. For example solving the problem for 30 cities, using a computer with 10^9 adds per second, would take over 8×10^{15} years.

The reason for TSP's popularity is its practical and theoretical importance. Classical TSP applications include areas such as transportation logistics, PCB drilling, and X-rays analysis. Modern TSP applications include DNA/genome sequencing, bio-informatics, statistical physics, astronomy, etc. TSP belongs to the class of NP-complete problems. It is known [2] that all NP-complete problems could be transformed one to another in polynomial time, so the decision of TSP is also of great theoretical interest.

There are two types of TSP: symmetric and asymmetric [5]. For the symmetric TSP the cost of traveling from city A to city B is the same as the cost of traveling from B to A. For asymmetric TSP these costs can be different.

Manuscript received July 15, 2005.

R. Gremlich received his M. Sc. in Computer Science in 2004 from the University of Zurich, Zurich, Switzerland (email: gremlich@gmx.ch).

A. Hamfelt is a professor of the Department of Information Sciences in Uppsala University, Uppsala, Sweden (phone: +46-18-4711037, e-mail: andreas.hamfelt@dis.uu.se).

H. de Pereda was a Computer Science student in Uppsala University, Uppsala, Sweden. He is now with AtosOrigin, Madrid, Spain (email: hector.depereda@atosorigin.com).

V. Valkovsky is an associate professor of the Department of Information Sciences in Uppsala University, Uppsala, Sweden (phone: +46-18-4711041, e-mail: vladislav.valkovsky@dis.uu.se).

Research dedicated to solving the symmetric TSP abounds. For practical applications, currently the large-scale asymmetric TSP is starting to be extremely important (e.g. DNA/genome sequencing). However, concerning the asymmetric TSP there has been far less research. It appears to be a more difficult problem, both with respect to optimization and approximation [4]. Whereas the TSPLIB library of the best-known TSP decisions [3] contains symmetric TSP instances with as many as 85900 cities, its largest asymmetric TSP instance has only 443 cities and over half of its asymmetric TSP instances have fewer than 100 cities [4]. Computational complexity of the TSP was the main reason for creating the great number of heuristic algorithms to get approximate TSP decisions in polynomial time [5]. One of the most efficient heuristic approaches is the class of genetic algorithms. Genetic algorithms are general-purpose searching techniques based on the Darwinian Principle of Natural Selection [6]. Genetic algorithms combine selection, crossover and mutation operations with the goal of finding the best solution of a problem. Genetic algorithms search for this solution until a specified termination criterion is met. One of the main problems of genetic algorithms is local minimums [8]. It is very common that genetic algorithm converge to a solution that might be the best among the ones close/similar to it, but it could be far from the best solution we can get. Another duty is generating the initial population [6]. Starting from a sufficiently good initial population can save a lot of evolution time, as well as facilitating the genetic algorithm to deliver better quality solutions. To overcome the above-mentioned problems we propose a concept that we call "Oracle." Thanks to information provided by Oracle it is possible to avoid local minimums and generate good quality initial populations.

II. Oracle

The approach for finding the optimal decision distributions for large-scale asymmetric TSP was proposed in [1].

For light left tail TSP weight matrixes distributions it will be Rayleigh distribution:

$$p(x) = \frac{x}{\sigma^2} \cdot e^{-\frac{x^2}{2\cdot\sigma^2}} \tag{1}$$

For heavy left tail TSP weight matrixes distributions the optimal decision distribution will be γ-distribution:

$$p(x) = \frac{1}{\Gamma(\alpha+1)\cdot\beta^{\alpha+1}}\cdot x^{\alpha}\cdot e^{-\frac{x}{\beta}} \qquad \alpha = 0.5 \qquad (2)$$

Methods for finding parameters of these distributions by a given asymmetric TSP weight matrix, calculating a prediction of the optimal tour length and its variance are also presented in [1]. The result of this research provides an opportunity to reduce drastically the search space for solving the asymmetric TSP. For example, for the dimension of the problem n = 1000 we can delete nearly 99% of the elements of the TSP weight matrix as irrelevant for the optimal decision. For the dimension n = 10000 nearly 99.9 % of the elements can be deleted, etc. After such pruning according to the predicted optimal decision distribution, we can determine the probability for the remaining elements of the weight matrix to be included in the optimal decision to support finding an optimal TSP solution.

We will call information provided by this approach “Oracle” and will use it to improve the performance of genetic algorithms for the asymmetric TSP.

III. Classical Genetic Algorithms for TSP

If we want to solve TSP using genetic algorithms, we have to consider some facts. An appropriate way of encoding the tours that the salesman can choose is by assigning every city a different number and representing tours as a list of these numbers in the order they must be visited. Every city/number is intended to be a gene of a full chromosome. Tours/lists represent chromosomes. Lists must have the same length as the number of cities in the TSP and no city can appear twice in the same list. The initial set of solutions can be created either by randomizing the list of cities or creating tours using heuristic algorithms. A simple evaluation function could just calculate the total distance of the tour represented by each list. In this way, lower values will mean routes are closer to the optimum solution. Reproduction can be done by crossover and mutation or by combining both techniques. There is a wide range of mutations to be applied to the members of a population of TSP solutions. Choosing the most appropriate one will determine in some way the success of the algorithm. We analyzed the performance of the six different mutations: swap adjacent nodes mutation, swap nodes mutation, move node mutation, relocate mutation, invert mutation and permute mutation.

All the mutation functions have at least one randomized factor, which makes them more efficient when trying to emulate a specie evolution. The more randomized these functions are, the more possibilities of evolution we will have. Swap adjacent nodes mutation turned out to be quite inappropriate for our goal, due to its limited range of variations. On the other hand, invert mutation provided us with best results. This mutation also came out as the less prone to get stuck in local minimums. Fig. 1 shows an example of the rd100 TSP instance from TSPLIB being solved by means of the six different mutations during a certain number of generations. We can appreciate how swap adjacent gets quickly stuck in a local minimum, whereas invert mutation is the one that faster gets close to the optimum solution.

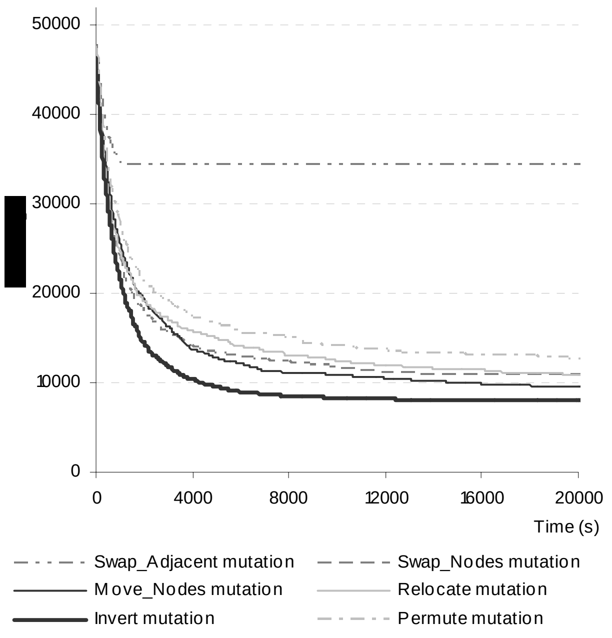

Fig. 1 Performance of mutations for 'rd100.tsp' from TSP-LIB

To continue the analysis, we need large-scale asymmetric TSP instances, but as it was mentioned above, TSPLIB has not appropriate ones. Because of this fact we will continue our investigation by using generated large-scale instances. Further we will use asymmetric TSP instances up to 500-nodes with elements of weight matrixes generated randomly on the interval [0, 1] according to distribution density f(x) = 2x (triangle distribution).

Fig. 2 demonstrates the length of the solutions found for a 500-node generated asymmetric TSP instance in relation with the time elapsed until these solutions were reached. The average length of the solutions found so far is also showed.

The expected length for the optimal solution delivered by Oracle is around 30.46 (+/- 0.45). It is possible to see that, in average, the later the algorithm gets stuck in a local minimum, the closer the solution found is to the optimal path length. However, the length of the best solution found is 57.69. This means that the solutions found are still quite far from the optimal one.

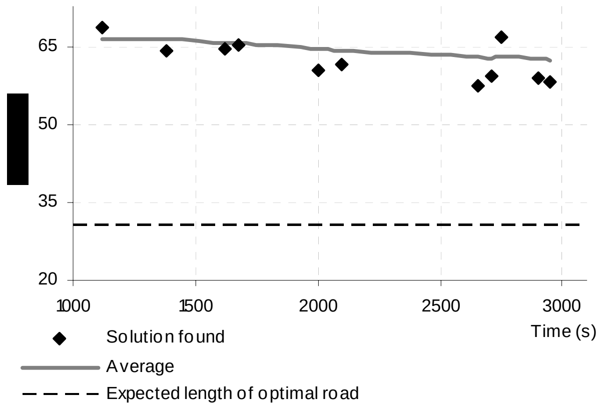

Fig. 2 Solutions found for a 500-node TSP

Fig. 3 demonstrates expected distribution density of the optimal decision, delivered by Oracle, and distribution density of the decision delivered by the classical genetic algorithm for a 500-nodes generated asymmetric TSP instance, which shows poor preliminary precision. In the successive results we should try to get a distribution density much closer to the expected one, which will mean better solutions for the TSP.

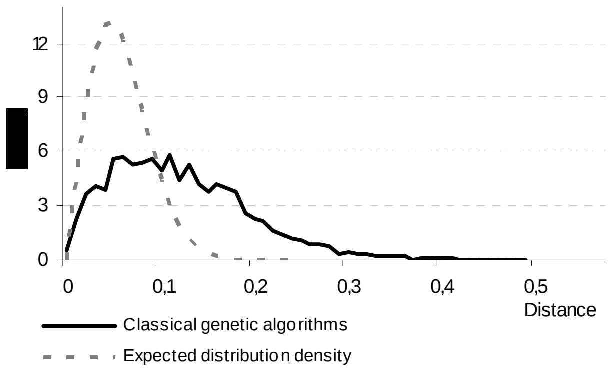

Fig. 3 Distribution density for classical genetic algorithms

IV. Local Minimums Problem

As it was mentioned above, local minimums are one of the main problems of genetic algorithms. So, detecting and avoiding local minimums is the first goal of our investigation. By Oracle it is possible to know the expected length of the optimal route and its variance. This information can be used for evaluating whether we have found an appropriate solution or have to keep on searching. Local minimums are detected by counting the number of generations that have been created without improving a best solution, which is sufficiently far from the one predicted by Oracle. When this number reaches a certain quantity, it means that we are in a local minimum and we have to act accordingly.

The way of skipping a local minimum is going back in the evolution and trying to evolve in a different direction. This is accomplished by mutating a whole population enough so it does not get back to the same solution we had in the local minimum, but not so much that we loose all the evolution we had already performed. A lot of combinations have been tested in the quest for the most suitable way of skipping a local minimum. The best way we have found to manage this problem is by applying several single-node oriented mutations. We have used a combination of swap node and move node mutations, each of them used twice, alternatively. This technique means an improvement in the algorithm performance. Fig. 4 shows the performance improvement regarding the quality of the solution for a 500-nodes TSP instance.

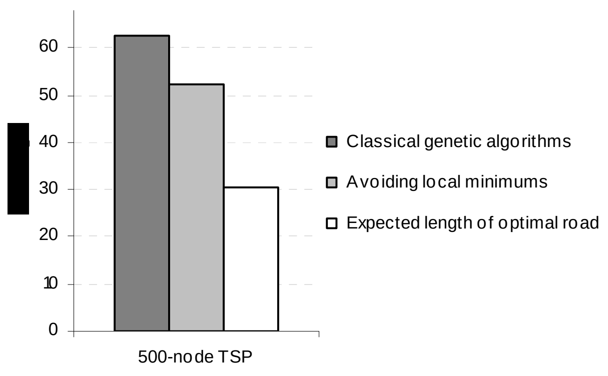

Fig. 4 Average solutions length for a 500-node TSP

However, this increase in the precision of the algorithm causes a significant increase of the time needed too, as we can see in Fig. 5.

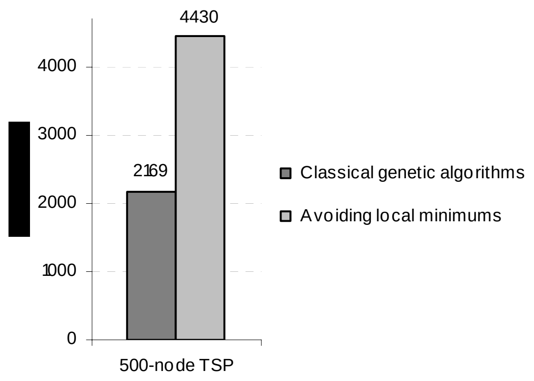

Fig. 5 Average time spent for a 500-node TSP

Consequently, the distribution density improves the one we got for the results of classical genetic algorithms and gets closer to the expected optimal one, as we can see in Fig. 6.

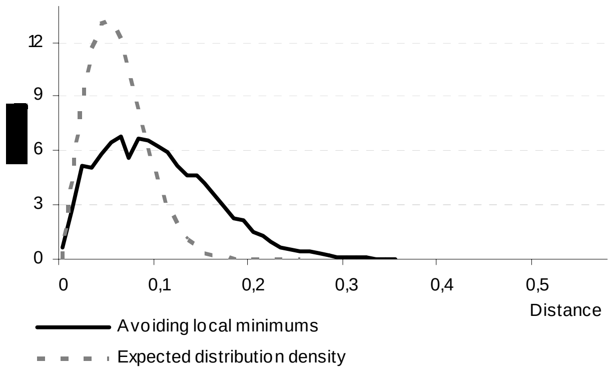

Fig. 6 Distribution density avoiding local minimums

V. Generation of Initial Population

Starting from a sufficiently good initial population can save a lot of evolution time, as well as facilitating the genetic algorithm to deliver better quality solutions. Our task now is to create an efficient and quick way to create initial population members. We will use "predicted by Oracle optimal decision distribution" as a guide that can lead us to solutions looking quite similar to the optimal path.

We came up with the idea of sorting each node's neighbors by "predicted Oracle distance density distribution." Then, when it comes to choose the next node in the path, we pick an unvisited neighbor with a probability that is directly proportional to its distance expected density. In this way, distances with an insignificant density will not be picked unless the rest of the neighbors have been visited already. This method reminds of the nearest-neighbor algorithm [9], but introduces a pseudo-random factor and considers the expected distance density in the final path instead of the distance itself. From now on, we will call this method most-probable-neighbor algorithm. Fig. 7 represents the outstanding performance of the new algorithm, which, in addition, was not as time consuming as preliminary attempts. For each TSP size from 10 to 500, we calculated the average total length of the initial population members, both generated randomly and using the most-probable-neighbor algorithm. As we can see, the members of initial populations generated by this new method are much closer to the optimal route than the solutions provided by genetic algorithms before implementing this improvement, even after performing genetic algorithms' evolution through a huge number of generations. This will allow us to start the evolution process from a point where it would have taken an enormous amount of time to get to applying mutations.

Another fact we can extract from the graph is that when possible solutions are generated randomly, their length differs from the optimal route length more and more as the number of nodes of TSP instance increases. However the length of solutions generated by the most–probable-neighbor algorithm remains somehow constant in relation to the expected length of the optimal route, as we can see in Fig. 8.

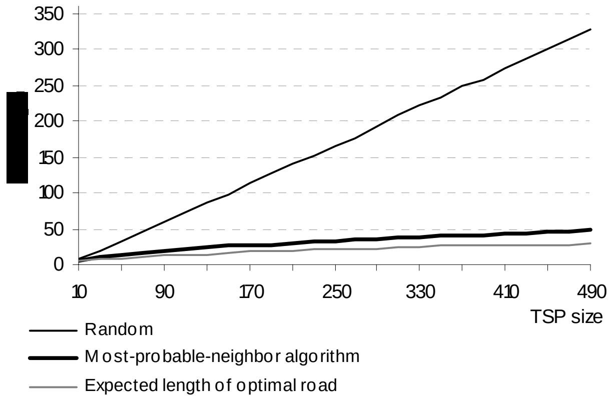

Fig. 7 Average length of initial population members

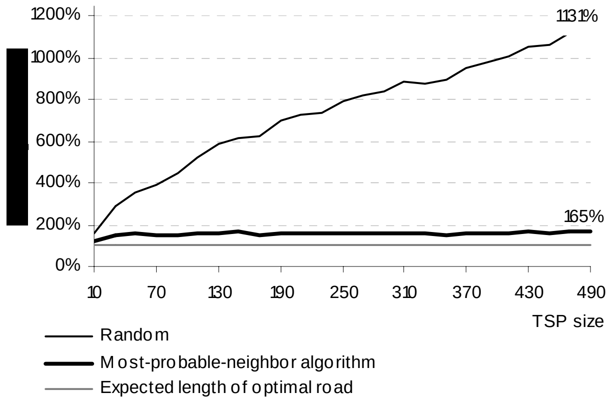

Fig. 8 Percentage length difference of initial population members with expected optimal road

The members of the initial population generated by the most-probable-neighbor algorithm are high quality solutions for the TSP to start the evolution process from. However, it would take too long to generate the entire set of initial population members using this technique. Instead, we generate a few members and mutate them to fill up the entire population.

If we take a look at the distribution density for the members of the initial population generated by the most-probable-neighbor algorithm (Fig. 9), we will appreciate how the curve is much closer to the expected optimal decision distribution.

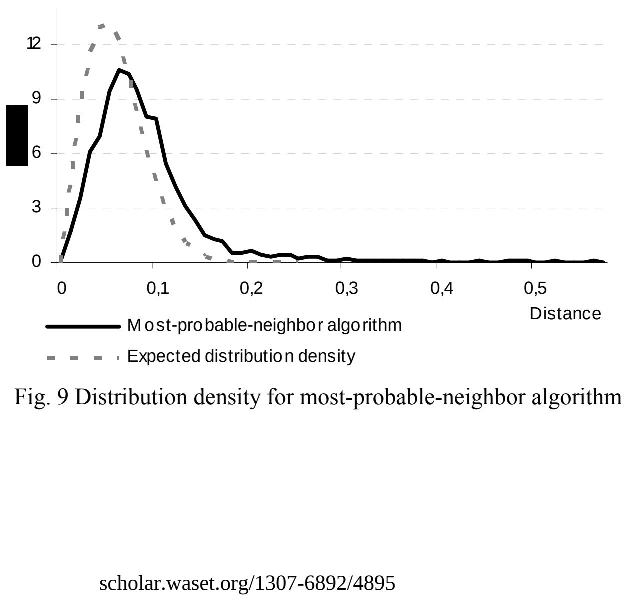

Fig. 9 Distribution density for most-probable-neighbor algorithm

As these solutions are not generated completely randomly, it was our task to find out an appropriate way to mutate them. By analyzing the paths created by the most-probable-neighbor algorithm we developed a mutation function, which tries to find the place in the path where the last node could be placed in order to decrease the total path length. If it does not find a better place for the last node, it continues with the preceding one, and so on.

VI. Final Results

Now it is time to merge all the improvements we have developed and check how they work all together.

The final algorithm works as follows:

1. The initial population is created using the most-probable-neighbor algorithm we have implemented (or by mutating members created by it, in case the process is too slow).
2. Then, the specific mutation that we have created places the last nodes of the path in more proper places in order to make the path more homogeneous.
3. Finally, the genetic algorithm starts working in the quest for the expected optimal route length. It will detect whether we have reached a good solution for the TSP, or are just in a local minimum. For the second case, the algorithm tries to skip the local minimum and lets the evolution go on.

The distribution density for the solution provided looks very similar to that predicted by Oracle, as we can see in Fig. 10.

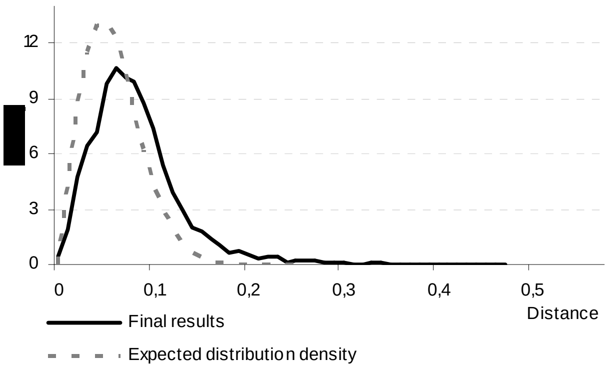

Fig. 10 Distribution density for final results

We now have to evaluate how accurate and efficient our new algorithm is in order to decide whether the improvements done to the classical genetic algorithms are worth applying. We can compare the solutions provided by the algorithm by placing all the solutions obtained in a chart. The Y axe of the chart will represent the length of the solution and the X axe will represent the amount of time spent to deliver that solution. The solutions with shorter paths will be at the bottom and those, which have been delivered in less time, will be shown at the left. In this way, solutions placed at the bottom-left corner will be considered better solutions than those in the upper-right corner (Fig. 11).

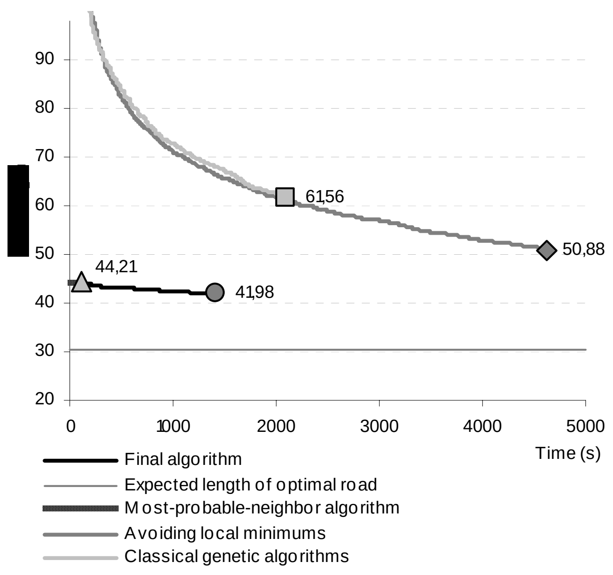

Fig. 11 Examples of evolutions for a 500-node TSP

It is obvious that the resulting algorithm delivers better solutions in less time than the classical genetic algorithms, but on the other hand, it spends a considerable amount of time after generating the initial population which does not result in a great improvement of the solutions delivered by the most-probable-neighbor algorithm.

If we take a look at the different distribution densities we have obtained, we will be able to notice how each time they get closer to the "predicted by Oracle optimal distribution density," as showed in Fig. 12.

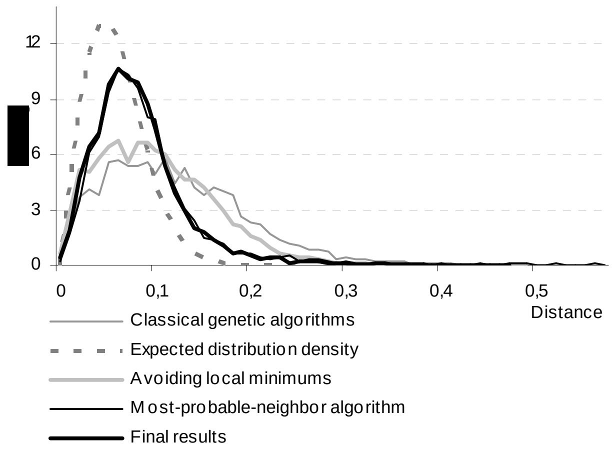

Fig. 12 Distribution density

VII. Conclusion

The presented results confirm that the concept of Oracle is an efficient means for improving the performance of genetic algorithms.

REFERENCES

[1] Robin Gremlich, Andreas Hamfelt, and Vladislav Valkovsky, “Prediction of the Optimal Decision Distribution for the Traveling Salesman Problem”, Proceedings of IPSI International Conf., Sveti Stefan, Montenegro, 2004.
[2] Papadimitriou C.H., Steiglitz K. Combinatorial Optimization: Algorithms and Complexity. Englewood Cliffs, NJ: Prentice Hall, 1982.
[3] http://www.iwr.uni-heidelberg.de/iwr/comopt/software/TSPLIB95/
[4] Gutin, Punnen (eds.), The Travelling Salesman Problem and its Variations, Kluwer Academic Publishers, 2002.
[5] http://www.tsp.gatech.edu/
[6] http://en.wikipedia.org/wiki/Genetic_algorithm
[7] http://tracer.ull.es/academic/Travelling_Salesman_Problem.html
[8] http://en.wikipedia.org/wiki/Local_optimum
[9] http://en.wikipedia.org/wiki/Nearest_neighbour_algorithm