International Conference on Automation, Control Engineering and Computer Science (ACECS)
Proceedings of Engineering and Technology – PET
Vol.21pp.58 - 65

Parallel Implementation of Travelling Salesman Problem using Genetic Algorithm on GPU

Yassine Moumen, Otman Abdoun, Ali Daanoun

Department of Computer Science Polydisciplinary Faculty Abdelmalek Essaadi University
B.P 745, Poste Principale 92004 Larache, Morocco
Moumen.yassine@gmail.com
otman.fpl@gmail.com
a_daanoun@yahoo.fr

***Abstract*— Genetic algorithms are metaheuristic algorithms, which mean that it generates useful solutions to solve NP-hard optimization problems in moderate execution times. However, Genetic algorithms usually require more computation power than other heuristic approaches do. Due to the complexity of problem such as TSP, finding a good solution with traditional ways needs a huge computational power (in term of processing power and memory usage) as well as time to solve. In this mater parallelism is an approach that not only reduce the resolution time but also improve the quality of the provided solutions. The latter holds since parallel algorithms usually run a different search model with respect to sequential ones. In this paper we have proposed a parallel implementation of Genetic algorithm on NVidia GPUs. We have done comparison on different parameters of the GA, which directly or indirectly affect the result, parallel comparison of speedup between CPU and GPU and best-known solution for both.**

***Keywords*— GPU, Parallel Genetic Algorithms, Sequential Genetic algorithms, Travelling Salesman Problem, CUDA.**

I. INTRODUCTION

In the last two decades, the researcher from all over the world has been searching for new ways to optimize and improve the traditional techniques to solve complex NP-hard problems like TSP (Traveling SalesMan Problem), who require an enormous computational time if we considered a real life size example, which make it impossible to solve it in an acceptable time.

In this context, some metaheuristic methods has been developed based on natural phenomenon observation, for example Genetic Algorithm (AG) that can be suitable to solve NP-hard optimization problems in moderate execution times. However, yet none of these algorithms have been able to reach the optimal solution for large-scale problem instances, and since there is no exact algorithm to solve an optimization problem in polynomial time, the minimal expected time to obtain optimal solution is exponential. Therefore, it is only possible to use metaheuristic algorithms to find approximate solutions for a given problem (a "good" solution).

Even with the use of one of the metaheuristic methods, a complex problem such as TSP needs huge computational power as well as time to solve. It takes lots of time for a single processor to solve such large problems single handedly. To solve these type problems in real time some additional mechanism must be taken into consideration to speed up the calculation time. Metaheuristic algorithms can be implemented parallel with high efficiency by using multi processors, multi-cores, or Graphic Processing Units (GPUs).

Graphical processing units (GPUs) are specialized processors with dedicated memory that conventionally perform floating point operations required for rendering graphics. In response to commercial demand for real-time graphics rendering, the current generation of GPUs have evolved into many-core processors that are specifically designed to perform data-parallel computation. Due to the inherently parallel nature of Genetic Algorithm, it is relatively easy way to implement on GPUs. However, it also brings some significant challenges due to its synchronization points and memory access patterns.

The aim of this paper is to propose efficient parallelization of Genetic Algorithms on CUDA architecture with Graphics Processing Units. The experimental results showed that the GPU overpass the CPU for the large population size.

II. PROBLEMS ADDRESSED

To solve the traveling salesman problem, researchers have come up with several algorithms

ISSN 2356-5608

to achieve the best-known solution; still, those methods always fail to find an acceptable solution in a reasonable time. Which leaves us with two options; either minimize the size of the problem or give up and accept the poor results. Luckily, some of the algorithm such as genetic algorithm can be implemented in a parallel environment to improve the performance. The popular method to parallelize an algorithm is to have a cluster of computers each with its own processor and memory, split the problem into sub-problems, and then assign each one to a computer. But then again, by adding computational power into the cluster, we can face the problem of overloading the network connecting the computers.

III. Traveling SalesMan Problem

The Viennese mathematician Karl Menger made the first statement of the Traveling Salesman Problem (TSP) as we know it today in 1930. It arose in connection with "A new definition of curve length" that Menger proposed. As he defined the length of a curve as the least upper bound of the set of all numbers that could be obtained by taking each finite set of points of the curve and determining the length of the shortest polygonal graph joining all the points. "We call this the messenger problem, because in practice the problem has to be solved by every postman, and also by many travellers: finding the shortest path joining all of a finite set of points whose distances from each other are given. Of course, the problem can be solved by a finite number of trials. However, there is no such a rule that would reduce the number of trials to less than the number of permutations of the given points. The rule of proceeding from the origin to the nearest point, then to the nearest point to that, and so on, does not generally give the shortest path" [1].

The TSP is stated as, given a complete graph, G, with a set of vertices, V, a set of edges, E, and a cost, Cij, associated with each edge in E. The value Cij is the cost incurred when traversing from vertex $i \in V$ to vertex $j \in V$. Given this information, a solution to the TSP must return the shortest Hamiltonian cycle of G (A Hamiltonian cycle is a cycle that visits each node in a graph exactly once. This is referred to as a tour in TSP terms).

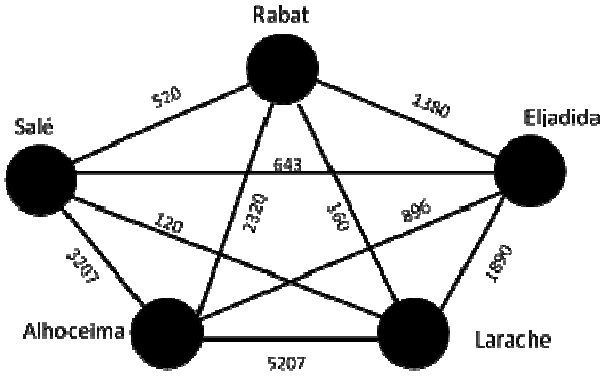

Fig. 1 Example of distances between cities

Traveling Salesman Problem is one of the most studied combinatorial problems because it is simple to comprehend but hard to solve [2]. The problem is to find the shortest tour of a given number of cities which visits each city exactly once and returns to the starting city [3]

At the first sight, TSP seems to be limited for a few application areas; however, it can be used to solve tremendous number of problems. Some of the application areas are; printed circuit manufacturing, industrial robotics, time and job scheduling of the machines, logistic or holiday routing, specifying package transfer route in computer networks, and airport flight scheduling.

The Travelling Salesman Problem is one of the best-known NP-hard problems, which means that there is no exact algorithm to solve it in polynomial time. The minimal expected time to obtain optimal solution is exponential. Therefore, we usually use heuristics to help us to obtain a “good” solution. Many algorithms were applied to solve TSP with more or less success. There are various ways to classify algorithms, each with its own merits. The basic characteristic is the ability to reach optimal solution: exact algorithms or heuristics.

There are various approaches to solve TSP the classical approaches are dynamic programming, branch and bound which uses heuristic and exact method and results into exact solution. Still, TSP is an NP-hard problem so the time complexity of these algorithms are exponential. Therefore, they can solve the small problem in optimal time but as compared to the large problem time taken by these

ISSN 2356-5608

algorithms are quite high. Unfortunately no classical approach can solve this type of problem in reasonable time as the size of the problem increases complexity increases exponentially.

Many alternate approaches are used to solve TSP, which may not give the exact solution but an optimal solution in a reasonable time. Methods like nearest neighbour, spanning tree based on the greedy approach are efficiently used to solve such type of problems with small size. To overcome this different other approaches based on natural and population techniques such as genetic algorithm, stimulated annealing, bee colony optimization, particle swarm optimization etc. are inspired from these techniques.

IV. Genetic Algorithms

Genetic Algorithms (GAs) are powerful search methods based on the biological concept of natural selection and genetics. They are being applied successfully to find acceptable solutions to problems in business, engineering, and science. GAs obeys the postulate of Charles Darwin's theory of Survival Of The Fittest as occurs in nature. This algorithm starts with an initial population from a random assemblage of individuals that evolves from one generation to another, through the creation of new individuals with better fitness values and elimination of individuals with low fitness values. In GAs the populations evolves by applying genetic operators such as selection, crossover and mutation, whose functionality and implementation depends on the problem to solve. One of the main features of the genetic algorithm is its ease of parallelization, since they are based on populations of independent individuals, thereby calculating the fitness function and the results for an individual is not depend on the calculation of other individuals [4].

A. Individual Representation

By finding the minimum distance between fixed cities, we can solve the stander Traveling Salesman Problem; therefore, the distance function depend on the order the cities in one tour. For this reason, each individual in the population will be an instance of vectors, where a vector consists of a list of randomly arranged cities. Note that the same cities must be used for each individual instance, ensuring that the salesman travels through the same set of cities, each time.

B. Fitness Function

The fitness function determines the probability of the solution that individuals have inside the population [4]. In this work, the cost function which represents the fitness function is given by the following equation, and the objective of the genetic algorithm is to minimize this cost function.

$$f(\pi) = \sum_{i=1}^{n-1} d_{\pi(i)\pi(i+1)} + d_{\pi(n)\pi(1)}$$

Where $d\pi(i)\pi(i+1)$ is the distance between city i and city i+1 and $d\pi(n)\pi(1)$ is the distance between city n and the first city.

C. Selection

Based on the value of the fitness function, the algorithm determines which individual will leave offspring for the next generation. There are various selection methods used in GAs, such as roulette selection, tournament selection, and fitness proportional, all having the basic goal of selecting the individuals with the top fitness functions, which consists of three steps. In the first step, calculating the sum of the fitness function across all individuals. The second step, calculate the individual probabilities, which is simply the individual's fitness divided by the sum of the fitness of all individuals in the same population. In the third step, we select two parents.

D. Crossover

After the two parents are selected, they are combined and the resulting individual has the ability to replace one of the parents or the individual with the worst fitness in the population. This step is known as crossover, as traits from each parent are used to create one or more children. There are several ways to apply the crossover operator such as crossing a single point, multi-point crossover, uniform crossover, among others. In this work, each individual in the population makes cross with another individual chosen by the selection

ISSN 2356-5608

operator in each iteration and cross-used by a single point. Only one child is created per set of parents, copying all cities up through crossing point, from the first parent, into a new child, and then adding the rest of the elements from the second parent(when cities are added from the second parent, only cities that are not currently in the child will be added). This process is illustrated below.

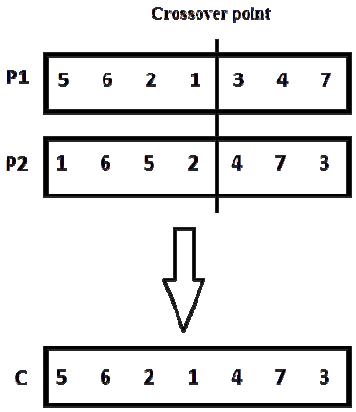

Fig. 2 Example of single point crossover

E. Mutation

The mutation creates an individual performing a change in chromosomes, usually small, on an individual of the population chosen randomly. The main objective of mutation is to give a chance to individuals with new genetic material. In this paper, the process of mutation used is by inversion where two positions of chromosomes randomly selected and two positions are inverted.

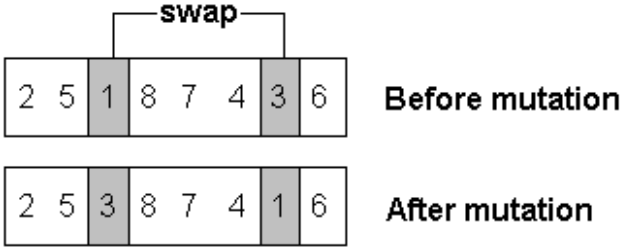

Fig. 3 Example of swap mutation

V. Sequential Genetic algorithms

Sequential programming involves a consecutive and ordered execution of processes one after another. In other words with sequential programming, processes are run one after another in a succession fashion, computation is modelled after problems with a chronological sequence of events.

The program in such cases will execute a process that will in turn wait for user input, then another process is executed that processes a return according to user input creating a series of cascading events.

To use the genetic algorithm in an iterative manner, first we have to guess some initial solutions randomly and then combine the choosing ones (the fittest) to create a new generation of solutions which theoretically should be better than the previous generation. We also include a random mutation in the genes with some probability like in the real word. The genetic algorithm consists of the following steps:

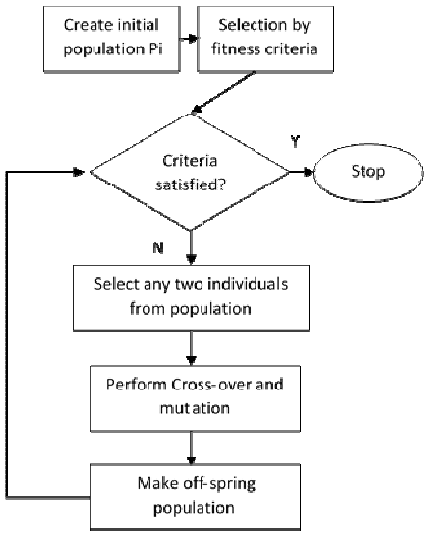

Fig. 4 Sequential Genetic algorithms

Usually, the first population of GA is created by generating a group of individuals randomly. The size of the initial population is important because it influences the way the algorithm work and if it can find good solutions and the time need to do so. If the population is too small, it may reduce search space, and it will be difficult to identify good solutions [3]. If the population is too large, it is even harder to find a solution among those individuals since the algorithm have to use a lot of computational resources and processing time. In each iteration of the GA a new population of individuals is created based on the previous generation, having more chance to reproduce those with better fitness function. GAs are usually able to

ISSN 2356-5608

find good solutions of combinatorial optimization problems in a reasonable time, but as applied to the biggest problems, the time needed to accomplish the task increase significantly. Therefore, many researcher have put so much efforts to implement faster GAs, and one of the most promising alternatives is to use parallel implementations, which can help in reducing the processing time. In recent years the development of powerful graphics processors has had a major boost, as a result we may have computing platforms with high performance and low cost.

VI. Parallel Genetic Algorithms

There are two main possible methods to parallelism. The first of which is data parallelism, where the same instruction will be executed on numerous data simultaneously. The second one is control parallelism, which involves execution of various instructions concurrently [5].

Data parallelism is sequential by its nature as only the data manipulation is paralysed while the algorithm will be executed as the sequential one instruction in certain time. Thus, the majority of parallel genetic algorithms were data based parallelism.

Parallel genetic algorithms arise from the need of computation required for extremely complex problems whose running time using sequential genetic algorithms is a limitation [6]. The use of parallel genetic algorithms aims to break a problem into several sub problems, solve them simultaneously on multiple processors, which improves the performance of search, and increase the probability of finding the best solution. In general, parallel algorithms behave in the same way as in the sequential algorithms, but with small different. In theory, parallel genetic algorithms can be divided into subtasks to maintain some balance in the distribution of activities, then; each processor can handle one of the sub-populations derived from the initial population of the genetic algorithm. There are several methods to parallelize a genetic algorithm. The first and most intuitive is the global, which is basically to parallelize the evaluation of the fitness function of individuals holding a single stock. A better option for parallelization of genetic algorithm is to divide the population into subpopulations that evolve separately and exchange individuals every few generations [6]. In this research, the TSP case will be solved by using the Genetic Algorithm on Graphical Processing Unit (GPU).

VII. Related Work

In its early emergence, the GPU is known as a graphics processor. But the numbers of processor cores in it attract the researchers to implement it data computing. A number of studies have been carried are related to GPU utilization in numeric data computing. Lefonn et al. developed a library to access the GPU data structures in generic and efficient way [7]. Mendez-Lojo et al. use GPU to run irregular algorithms that operate on pointer-based data structures as graphs[8], resulting an average speed up of 7x compared to a sequential CPU implementation and outperforms a parallel implementation of the same algorithm running on 16 CPU cores. The TSP case is frequently studied by the researchers of Yang and Nygard to observe the impact of genetic algorithm initial population in order to solve the TSP case by using the approach of time windows by using regular CPU [9]. Some researchers also investigate the TSP case in mathematical pint of view. Bartal et al. shown the algorithmic tractability of metric TSP depends on the dimensionality of the space and not on its specific geometry [10]. In another TSP study, Fekete et al. can solve the Fermat-Weber-Problem (FWP) with high accuracy in order to find a good heuristic solution for the MWMP [11].

VIII. Experiment and Result

To test our algorithm we used some particular problems from TSPLIB[12] to mimic real word scenarios and have an idea on how the proposed algorithm may work, This parallel algorithm is implemented by using both standard C++ for the serial version and CUDA C++ for the parallel version of the algorithm. The parallel version is compiled via CUDA compiler. The test libraries include 47, 50 and 279 cities. Each problem is solved with 10, 100, 1000, 10000 and 100000 populations by applying 10, 100 and 1000

ISSN 2356-5608

generations respectively. The GPU in this study is the NVIDIA GeForce GTX 1060. The serial version is run on an Intel Intel Xeon CPU E3-1220 V2 3.10 GHz CPU. The operating system for this experiment is windows 8. The compiler for the parallel version is CUDA SDK 8.0. Table I shows the details of the underlying system.

TABLE I
DETAILS OF THE UNDERLYING SYSTEM

	CPU	GPU
Manufacturer	Intel	NVIDIA
Model	Xeon E3-1220 V2	GTX 1060
Architecture	x86-64	Pascal
Clock Frequency	3.10 GHz	1544 MHz
Cores	4	1280
DRAM Memory	8 GB	6 GB

The solution quality as well as the execution time of the GA mainly depends on the parameter decisions. The parameters of the algorithm chosen in this study are as shown in Table II.

TABLE IIIII
PARAMETERS OF THE ALGORITHM

Parameter	Value
Number of cities	48, 51 and 280
Population size	10, 100, 1000, 10000 and 100000
Crossover	2 Points
Mutation rate	15%
Generations	10, 100 and 1000

A. Results

The table below shows the average results of 3 tries on every line for the CPU and GPU on problem a280_xy with 280 cities:

TABLE IVVVI
RESULTS FOR THE A280_XY

Population	Generations	Time CPU (ms)	Time GPU (ms)
10	10	89.5	1234.666667
	100	742.6666667	9559
	1000	7094.666667	92032.33333
100	10	203.1666667	1675.666667
	100	1859.5	13400.66667
	1000	18527.5	121213.6667
1000	10	1310.166667	2787.833333
	100	12454.33333	22796.33333
	1000	123513	207465
10000	10	14605	7266
	100	140356	36558
	1000	1412560	343638
100000	10	***	***
	100	***	***
	1000	***	***

Form the two tables we can make a comparison on both CPU and GPU:

For every population below 1800 individuals we can clearly see that the CPU deliver better performance over the GPU, which take a little longer to return the results. However, by increasing the population size, we notice that the CPU is struggling to run the algorithm and sometimes it cannot start the program due to huge work needed to initialize the massive data.

In the other side, the GPU has no problem running the algorithm. In addition, the time needed for the GPU to complete the process did not growth badly as the CPU.

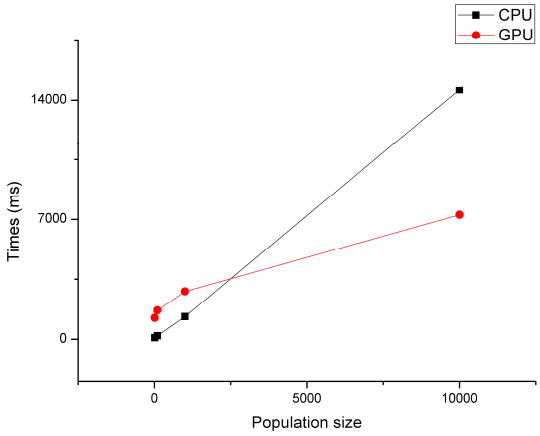

Fig. 5 Graph showing the time needed to complete GA for 10 generations

The graph shows that for larger population the GPU can produce the same results for less amount of time.

ISSN 2356-5608

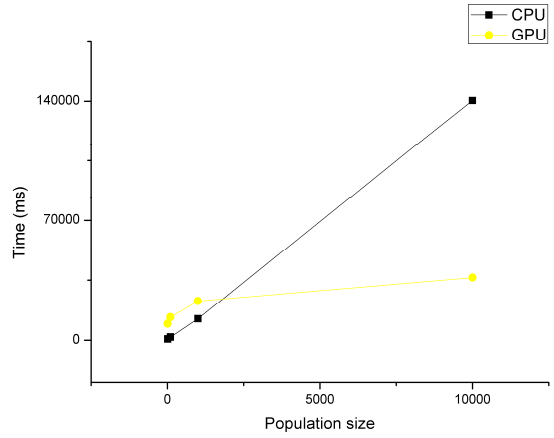

Fig. 6 Graph showing the time needed to complete GA for 100 generations

Increasing the number of time the GA is executed (number of generations) the CPU take longer period to return any results (almost linear graph).

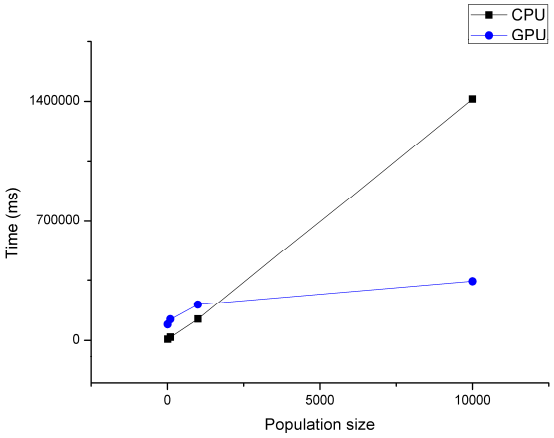

Fig. 7 Graph showing the time needed to complete GA for 1000 generations

IX. Conclusions

Apparently, the GPU made a better work in solving the TSP in all variations and with all parameters than the CPU if we considered a big population size as initial data, which take a long time to calculate. For the small size population, we can say that the CPU overpass the GPU in term of time. This phenomenon caused by the delay of write/read from the Graphic memory, every time we want to write or read from the GPU memory we have to have an equivalent amount of space in RAM.

X. Annex

A. *eil51_xy (51 cities):*

TABLE VIIV
RESULTS FOR THE EIL51_XY

Population	Generations	AVG Time CPU (ms)	AVG Time GPU (ms)
10	10	51.6666667	218.333333
	100	396.333333	1084.66667
	1000	3812	9480
100	10	66.6666667	276.333333
	100	498.5	1347.16667
	1000	4914.33333	11787.6667
1000	10	200.666667	666.833333
	100	1721.16667	2188.33333
	1000	16937.5	17231.5
10000	10	3633	4451
	100	35056	8777
	1000	349076	53940
100000	10	266834	56513
	100	1639040	229967
	1000	***	***

B. *att48_xy (48 cities):*

TABLE V
RESULTS FOR THE EIL51_XY

Population	Generations	AVG Time CPU (ms)	AVG Time GPU (ms)
10	10	48	250
	100	408	1134.66667
	1000	4021	9886.33333
100	10	62	272
	100	500.333333	1291.16667
	1000	4591.5	10889.8333
1000	10	191.666667	654.666667
	100	1671.5	2199
	1000	16232.1667	15841.1667
10000	10	3524	4468
	100	34169	8733
	1000	340358	53625
100000	10	267858	59638
	100	1656540	223404
	1000	***	***

***: hardware cannot run the algorithm.

References

[1] [1] Karl Menger, Ergebnisse eines Kolloquiums 3, 11-12 (1930).

ISSN 2356-5608

[2] [2] Fan Yang, Solving Traveling Salesman Problem Using Parallel Genetic Algorithm and Simulated Annealing, 2010.
[3] [3] Adewole Philip, Akinwale Adio Taofiki, Otunbanowo Kehinde, A Genetic Algorithm for Solving Travelling Salesman Problem, in International Journal of Advanced Computer Science and Applications, 2011.
[4] [4] S. N. Sivanandam, and S. N. Deepa. Introduction to Genetic Algorithms. Springer, 2008.
[5] [5] Zden˘ek Konfr˘st, “Parallel Genetic Algorithms: Advances, Computing Trends, Applications and Perspectives”, 18th International Parallel and Distributed Processing, 2004.
[6] [6] E. Cant-Paz. Efficient and accurante parallel genetic algorithms. Kluwer Academic Publishers, 2001.
[7] [7] A. E. Lefohn, S. Sengupta, J. O. E. Kniss, R. Strzodka, J. D. Owens, Glift: Generic, Efficient , Random-Access GPU Data Structures, ACM Trans. Graph., 25 (2006), no. 1, 60-99.
[8] [8] M. Mendez-Lojo, M. Burtscher, K. Pingali, A GPU Implementation of Inclusion-based Points-to Analysis, ACM SIGPLAN Notices, 47 (2012), 107-116.
[9] [9] C.-H. Yang and K. E. Nygard, The Effects of Initial for Time Constrained Population Traveling in Genetic Search Salesman Problems, ACM, (1993), 378-383.
[10] [10] Y. Bartal, L.-A. Gottlieb, R. Krauthgamer, The Traveling Salesman Problem: Low-dimensionality implies a polynomial time approximation scheme, Proceedings of the 44th symposium on Theory of Computing - STOC '12, (2012), 663-672.
[11] [11] P. Fekete, H. Meijer, Andre Rohe, W. Tietze, Solving a “Hard” Problem to Approximate an “Easy” One: Heuristics for Maximum Matchings and Maximum Traveling Salesman Problems, Chapter in Algorithm Engineering and Experimentation, Springer Berlin Heidelberg, 2001.
[12] [12] http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/index.html

ISSN 2356-5608