ACO-PSO Optimization for Solving TSP Problem with GPU Acceleration

Olfa Bali[1(✉)], Walid Elloumi[1], Ajith Abraham[2], and Adel M. Alimi[1]

[1] REGIM-Laboratory: Research Groups on Intelligent Machines, National Engineering School of Sfax (ENIS), University of Sfax, BP 1173, 3038 Sfax, Tunisia
bali.olfa@gmail.com, {walid.elloumi, adel.alimi}@ieee.org
[2] Intelligence Research Labs (MIR Labs), P.O. Box 2259, Auburn, WA 98071-2259, USA
ajith.abraham@ieee.org

Abstract. In this paper, we present a novel approach named "ACO-PSO-TSP-GPU" to run PSO and ACO on Graphical Processing Units (GPUs) and applied to TSP (Parallel-PSO&ACO-A-TSP). Both algorithms are implemented on GPUs. Well-known benchmark problems for many heuristic and meta heuristic algorithms presented by Travelling Salesman Problem (TSP) are known as NP hard complex problems.TSP was investigated using classical approaches as well as intelligent techniques employing Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO). Parallel computing is well suited to the execution of nature and bio-inspired algorithms due to the rapidity of parallel implementation. Results show better performance optimization when using parallelism compared to results using sequential CPU implementation.

Keywords: PSO · ACO · TSP · GPU · Optimization · Parallelism

1 Introduction

In the field of Engineering, the optimum solution of a problem is defined by using optimality criteria.

Swarm intelligence techniques are bio-inspired methods, where group behavior is used to solve a problem based on the individualities of its members.

Particle swarm optimization (PSO), is a branch of Swarm intelligence used for solving many engineering optimization problems. Among the stochastic global optimization techniques initially designed for non-linear continuous function optimization, Swarm Intelligence algorithms offer a number of attractive features, global search capability and easy implementation. Since 1995, Kennedy and Eberhart developed a meta-heuristic population based on global optimization called PSO [1], presented in Fig. 1.

PSO suffer from premature convergence for small population size but can be improved by increasing the population. Fortunately, the PSO is very easy to be

A.M. Madureira et al. (eds.), *Intelligent Systems Design and Applications*,
Advances in Intelligent Systems and Computing 557,
DOI 10.1007/978-3-319-53480-0_55

parallelized since the particles do not depend on each other while moving in the search space. Many approaches simulate multiple particles at a time or propose multiple swarm versions of the PSO [2]. The task of calculation can be heavy and the speed of the course PSO can be seen as slow as the operation of the PSO algorithm requires a number of iterations and a stop condition; which are formed in a sequential manner on the processor.

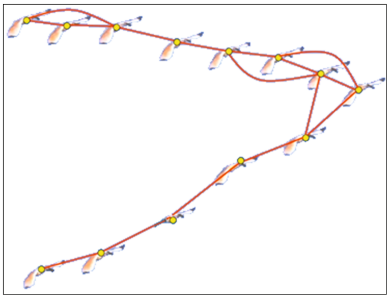

Fig. 1. Bird flocks

Ant systems [3] are inspired from the real behavior of real ants and are employed for combinatorial optimization problems. The basics of ant systems are founded on the theory of self-organizing systems [4] and the notion of stigmergy is presented in Fig. 2.

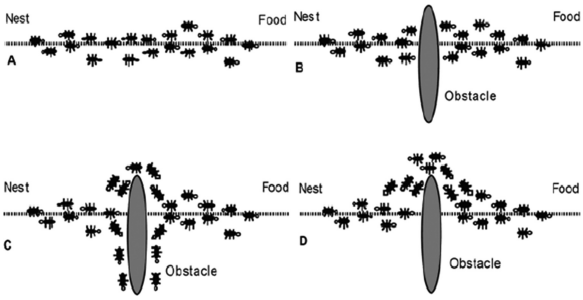

Fig. 2. The collective behavior changes

In [5], the authors have given an overview on the state of the art of the theoretical analysis of Ant Colony Optimization (ACO) algorithm. On a second stage, PSO is coupled with ACO for combinatory optimization.

Elloumi et al. [6] have presented an improved ACO algorithm supervised by PSO to solve continuous optimization problems. PSO algorithms are used to resolve continuous optimization problems while ACO algorithms are used for the discrete ones.

In [7], the authors have studied the multi-objective Particle Swarm Optimization (MOPSO) and found that more the number of the swarm increases more the accuracy of object achievement is increased.

In [8], the authors have proposed an approach that consists in combining fuzzy logic with ant colony Optimization (FACO) and fuzzy particle swarm optimization (FPSO) for solving TSP optimization problems.

In [9], the authors have proposed an optimization technique using multi-objective PSO (MOPSO) and FACO. This technique consists in combining these two methods. The objective of this combination is to reduce computation time and getting the shortest path.

In [10], the authors have presented an improved hybrid method (PSO–ACO) using the TSP benchmarks to validate our results.

During the last years, the trend was to use and improve graphics processing units (GPUs) as aco-processor. Designed mainly for graphics and game industry, GPUs have attracted many researchers due to its arithmetic computation power [11].

The paper is organized as follows. In Sect. 2, we define TSP problem. In Sect. 3, we present the basics of NVIDIA GPU based computing. Our system is illustrated in Sect. 4. Implementations of two algorithms and a number of experiments are done on four benchmarks with details of experiments and the reported results are exhibited in Sect. 5. Finally, we conclude our paper in Sect. 6.

2 The TSP Problem

TSP, the traveling salesman problem, which may be defined as follows: at first we initialize (n) cities that must be visited. Initially we start from a city chosen randomly and then returns to the starting city. The objective is to determine the overall distance and visit every city just once with respect to fixed start/end locations [12].

An illustration of this problem with 5 cities is given in Fig. 3; it shows two possible solutions, one in red and the other in green color. The two routes don't have the same length. A travelling salesman will choose the shortest path to reduce the cost of the travel. However, the TSP is said NP-complete. In fact, for n cities the number of possible route is equal to $(n-1)!/2$.

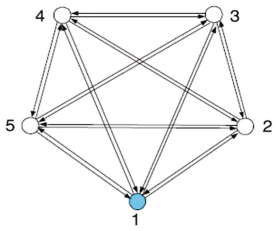

Fig. 3. TSP possible solutions for a simplified cities representation, here the number of cities is limited to 5

3 NVIDIA GPU Architecture

GPU processors are supported by the image processing and 3D data, as well as display. We distinct two main types of memory: local memory and global memory. This distinction allows memory spaces clearly separate memory areas read-only (constant, texture), local to a thread (local), read/write (global), short latency and deterministic (shared). The latency of each type of memory that can be estimated precisely statically. This parallel architecture, composed of a large number of calculations units, heavily exploit modes of SIMD (Single Instruction Multiple Data)/MIMD (Multiple Instruction Multiple Data) [13], allowing the simultaneous execution of many parts code. Parallelism can be achieved by using the concept of threads, i.e. lightweight processes that can run in parallel. When running a program, several thread groups will be spears in parallel to perform operations on a large data set.

The CUDA language for Compute Unified Device Architecture is available to the public since 2007. CUDA is a programming language similar to C/C++ to exploit the capabilities of GPU and material resources, particularly in regard to memory management and organization of treatment.

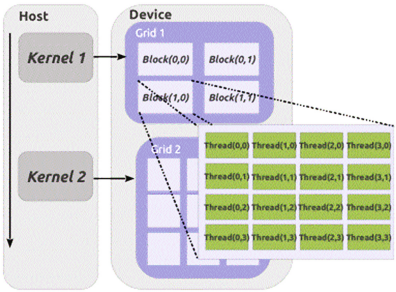

Fig. 4. Decomposition of blocks of grid data, where each block contains a number of elements executed in parallel. The term defines the Host CPU and Device term corresponds to the GPU. The size used for the definition of the grid is two-dimensional block-level and threads.

The principle of treatment of a problem on a highly parallel architecture is to break the problem into smaller problems that can be solved in parallel. Thus the partition of a wide array of data is performed by its decomposition into multiple blocks. Each block is run independently in parallel and the elements of each block are executed cooperatively in parallel. Figure 4 shows the decomposition of a set of data in a grid of 2×2 blocks are decomposed themselves in 4×4 elements.

All threads contained in a grid are sent to execution by a kernel, which can be defined as a simple function or program. To manage a large number of concurrent threads that can cooperate with each other, the architecture of graphics cards introduced threads of cooperation sets, also known as blocks of threads or thread blocks in CUDA terminology.

4 GPU-PSO and ACO-A-TSP

In this section, we will study our approach having two essential parts. The first part explain GPU PSO-A to TSP while the second part explain GPU ACO-A to TSP.

PSO and ACO optimization require high CPU computation resources. That's explain the necessity to use GPU accelerated system.

4.1 GPU PSO

The Fig. 5 shows roughly the Graphical Process Unit - Particle Swarm Optimization Applied to Travelling Salesman Problem.

Our goal is to cover all cities (designated nodes) once (if the particle passes through the city i to j it does not cross the town in the other direction, from j to i). Finally, the particle returns to the starting city, so we get a cycle.

The "**gpuArray**()" function allows copying data from the memory of the CPU to the GPU memory brings us to manipulate the table on the GPU memory.

First, the global best (P_{ig}) and local best (P_{il}) are elected, then we update the positions and velocities of the particles. These particles are assigned for N data and N threads.

We had to repeat these steps until reaching the maximum number of iterations; it is assigned to each node. This allows us to obtain an archive, according to the latter; we can make a comparison between the different obtained paths. We choose the best way in terms of its execution time. Finally, we return the GPU data to the CPU through the control "**gather**()".

4.2 GPU ACO

The Fig. 6 illustrates the diagram of the Graphical Process Unit - Ant Colony Optimization Applied to Travelling Salesman Problem.

The operation of the classic ACO is based on parameters that are often defined by the user of the algorithm. Thus, found settings that are appropriate for a problem are not suitable for other problems, forcing the user to perform numerous tests to define the parameters.

After coping data to GPU memory, the algorithm starts by assigning a city for each ant until all cites are affected to ants. When ants return to the starting city the amount of pheromone is updated in a cycle. Then, we Assign each node a thread and repeat these steps until reaching the maximum number of iteration. If we arrive at stopping criterion the concept of pheromone placing procedure guides the building procedure to each thread. The solutions to the intermediate partial problems are seen that we display the best lap, otherwise we return in step 2.

The dynamic memory structure that is inspired by the movement of the ant k for each iteration of the algorithm, see Fig. 6.

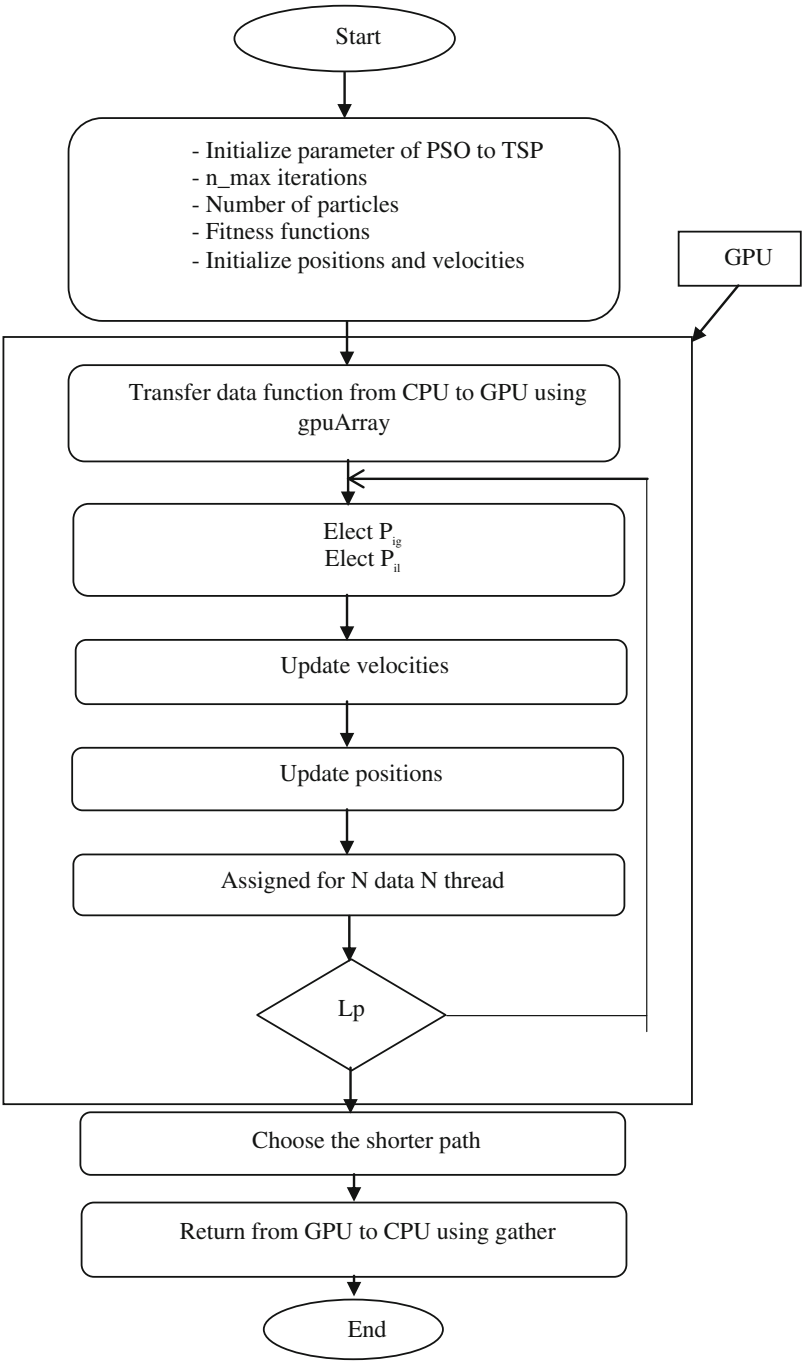

Fig. 5. The process of GPU-PSO-A-TSP

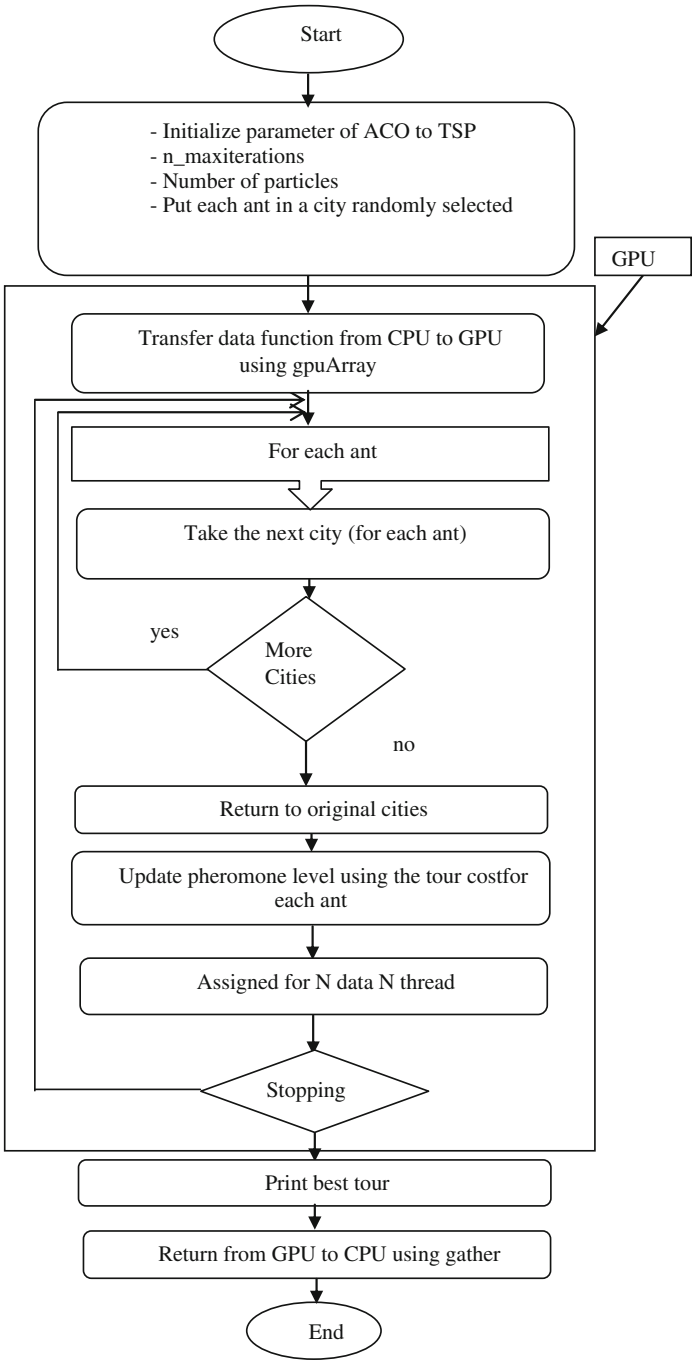

Fig. 6. The process of GPU-ACO-A-TSP

5 Experimental Results

We are based on well known benchmarks to validate the developed algorithms [10]. We have chosen in this paper from the benchmarks four TSP problems which consist in finding the optimal path to travel between graph cities of 22, 29, 30 and 48 nodes. In our approach, we begin by resetting the parameters feature of PSO applied to the TSP; to say the number of nodes contained in a graph, a weight of every particle and the coefficients of acceleration. The maximum number of iterations in this case was taken as 1000 iterations. The number of used particles depends on the graph.

For example, for a graph with 22 cities, the possible number of particles of the population to use in this first test is 22; the second test is 70 for the third test is 100.

We had 1000 iteration because we have tests by using 2000 and 3000 iterations.

Our method consists in using 1000 iterations because the number and the time of the cycle are proportional among iterations to avoid the wasting.

Our approaches GPU Optimization Swarm of particles and Ant Colony Optimization applied to the TSP (GPU-PSO ACO-A-TSP) is coded in Matlab 2014b and executed on a processor Intel ® Core T i7-4700MQ (6 MB cache memory, 3.40 GHz) PC with memory of 12 GB and NVIDIA GeForce 480 M and Windows 7.

There are many parameters used for our approach. The size of the population, which we are three times going to increase, is the number of knots of the social and cognitive probability, having c1 and c2, defined as c1 = c2 = 2. The mass of inertia w is taken as 0,9 and the maximum of the speed live taken as 100 and the dimension of the space as 10. Every cycle of TSP is executed during five replications and 1000 iterations. Both has and ß control the relative importance of pheromone trail and the distance between cities TSP where has $\alpha = 1.5$, ß = 2 Refers to the speed of pheromone evaporation $\rho = 0{,}7$. Each test TSP is performed for 5 replications iterations and 1000.

5.1 TSP Solved by PSO and GPU PSO

In Table 1, N refers to the number of nodes, CPU-T.PSO refers to the best time for PSO (per seconds) and CPU-L.PSO refers to CPU-PSO the length for PSO. GPU used the same indices.

In the same table, we have tried to represent the different numbers of nodes, after that, we have attempted to increase the population of PSO keeping the same number of nodes. Thus, It was found from number 22 to 48 nodes, that the route of the shortest path decreases when the number of PSO population increases. Therefore, when the execution time increases, the number of (CPU-GPU) PSO population increases too.

5.2 TSP Solved by ACO and GPU ACO

In Table 2, N refers to the number of nodes, T.ACO refers to the best time for ACO (per seconds), L.ACO is the best Length for ACO.

Table 1. PSO and GPU PSO for TSP

		CPU		GPU	
N	Size of population of PSO	T.PSO (s)	L.PSO (km)	T.PSO (s)	L.PSO (km)
22	22	0.1406	90.6884	**0.0848**	90.6884
	70	0.4556	90.4220	**0.2791**	90.4220
	100	0.5707	89.4898	**0.3950**	89.4898
29	29	0.4181	1.1761e+004	**0.2346**	1.1761e+004
	70	0.9632	1.0900e+004	**0.5831**	1.0900e+004
	100	1.2177	1.0472e+004	**0.7763**	1.0472e+004
30	30	0.3577	584.0341	**0.2736**	584.0341
	70	0.7334	562.0160	**0.5969**	562.0160
	100	1.1059	545.7844	**0.8535**	545.7844
48	48	3.0961	4.5973e+004	**0.6366**	4.5973e+004
	70	4.3249	4.4654e+004	**0.9269**	4.4654e+004
	100	4.8480	4.1158e+004	**1.4414**	4.1158e+004

Table 2. ACO and GPU ACO for TSP

		CPU		GPU	
N	Size of population of ACO	T.ACO (s)	L.ACO (km)	T.ACO (s)	L.ACO (km)
22	22	0.2438	77.8000	**0.0930**	77.8000
	70	0.4969	77.1834	**0.2964**	77.1834
	100	0.6866	76.1212	**0.4223**	76.1212
29	29	0.4190	1.1621e+004	**0.2602**	1.1621e+004
	70	1.0881	1.0530e+004	**0.6132**	1.0530e+004
	100	1.3713	1.0432e+004	**0.9036**	1.0432e+004
30	30	0.3724	537.9874	**0.2933**	537.9874
	70	0.7652	495.5985	**0.6699**	495.5985
	100	1.4225	491.7651	**1.0160**	491.7651
48	48	4.0005	4.2086e+004	**1,1433**	4.2086e+004
	70	4.5047	4.1420e+004	**1,6658**	4.1420e+004
	100	6.8139	4.0585e+004	**2.3992**	4.0585e+004

In the second table we have tried to represent the different numbers of nodes. After that we have tried to increase the number of people of ACO keeping the same number of nodes. In this Table it was found that the route of the shortest path decreases when the number of ACO population increases. When the execution time increases the number of population increases too. The execution time depends on the complexity of the TSP as well.

Now comparing the results of the two tables, we notice that the results of the shortest path of (CPU-GPU) ACO are better compared to the (CPU-GPU) PSO, but the best performance is that of (CPU-GPU) PSO compared to that of (CPU-GPU) ACO time.

Compared to previous works ACO-A-TCP and PSO-A-TCP [10], our algorithms GPU-ACO-A-TSP and GPU-PSO-A-TSP implemented on GPU have the advantage of reducing the computational time for solving TSP problems.

6 Conclusion

In this paper we have given two approaches the GPU-PSO–A-TSP and GPU-ACO–A-TSP. We have used PSO and ACO, meta-heuristics optimization algorithms, for resolving TSP problem. We have also used the parallel GPU programming model to reduce the PSO and ACO algorithms computational time.

According to the results of the two tables, we notice that the results of the shortest path of (CPU-GPU) ACO are better compared to the (CPU-GPU) PSO, but the best performance is that of (CPU-GPU) PSO compared to that of (CPU-GPU) ACO time.

For this reason, we have achieved hybridization between PSO and ACO using the GPU based on [14].

Acknowledgments. The authors would like to acknowledge the financial support of this work by grants from General Direction of Scientific Research (DGRST), Tunisia, under the ARUB program.

References

1. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: IEEE International Conference on Neural Networks, vol. 4, pp. 1942–1948 (1995)
2. Hendtlass, T.: WoSP: a multi-optima particle swarm algorithm. In: The IEEE Congress on Evolutionary Computation, vol. 1, pp. 727–734 (2005)
3. Dorigo, M., Maniezzo, V., Colorni, A.: The ant system: optimization by a colony of cooperating agents. IEEE Trans. Syst. Man Cybern. B Cybern. **26**(2), 29–41 (1996)
4. Garnier, S., Gautrais, J., Theraulaz, G.: The biological principles of swarm intelligence. Swarm Intell. (2007)
5. Elloumi, W., Alimi, A.M.: Combinatory optimization of ACO and PSO. In: International Conference on Metaheuristique and Nature Inspired Computing, pp. 1–8, October 2008
6. Elloumi, W., Rokbani, N., Alimi, A.M.: Ant supervised by PSO. In: International Symposium on Computational Intelligence and Intelligent Informatics, pp. 161–166, October 2009
7. Elloumi, W., Alimi, A.M.: A more efficient MOPSO for optimization. In: The Eight ACS/IEEE International Conference on Computer Systems and Applications, AICCSA, pp. 1–7, May 2010
8. Elloumi, W., Baklouti, N., Abraham, A., Alimi, A.M.: Hybridization of fuzzy PSO and fuzzy ACO applied to TSP. In: 13th International Conference on Hybrid Intelligent Systems (HIS), pp. 106–111, December 2013
9. Elloumi, W., Baklouti, N., Abraham, A., Alimi, A.M.: The multi-objective hybridization of particle swarm optimization and fuzzy ant colony optimization. J. Intell. Fuzzy Syst., 515–525 (2014). http://dx.doi.org/10.3233/IFS-131020

10. Elloumi, W., El Abed, H., Abraham, A., Alimi, A.M.: A comparative study of the improvement of performance using a PSO modified by ACO applied to TSP. J. Appl. Soft Comput. **25**, 234–241 (2014)
11. Kirk, D.B., Hwu, W.W.: Programming Massively Parallel Processors: A Hands-on Approach. Morgan Kaufmann, San Francisco (2010)
12. Gavish, B., Graves, S.C.: The travelling salesman problem and related problems (1978)
13. Flynn, M.J.: Some computer organizations and their effectiveness. IEEE Trans. Comput. **21**, 948–960 (1972)
14. Bali, O., Elloumi, W., Abraham, A., Alimi, A.M.: GPU PSO and ACO applied to TSP for vehicle security tracking. J. Inf. Assur. Secur. **11**, 369–384 (2016)