Parallel Cost Function Determination on GPU for the Vehicle Routing Problem

Mieczysław Wodecki[1], Wojciech Bożejko[2(✉)], Szymon Jagiełło[2], and Jarosław Pempera[2]

[1] Institute of Computer Science, University of Wrocław, Joliot-Curie 15, 50-383 Wrocław, Poland
mieczyslaw.wodecki@ii.uni.wroc.pl

[2] Department of Automatics, Mechatronics and Control Systems, Faculty of Electronics, Wrocław University of Technology, Wyb. Wyspiańskiego 27, 50-370 Wrocław, Poland
{wojciech.bozejko,szymon.jagiello,jarolaw.pempera}@pwr.edu.pl

Abstract. The paper deals with parallel variants of optimization algorithms dedicated to solve transportation optimization issues. The problem derives from practice of logistics and vehicle routes planning. We propose parallelization method of the cost function determination dedicated to be executed on GPU architecture. The method can be used in metaheuristic algorithms as well as in exact approaches. [1]

1 Introduction

Vehicle Routing Problem (VRP) is an NP-hard problem, which solution constitutes a key element for effectiveness improvement in transportation. The problem, introduced by Dantzig and Ramser [8] in 1959, appears in many variants with different level of complexity. From this time literature relating to it has been expanded by thousands of items. The software used to solve VRP problems generates considerable savings of 5% – 30% for Companies which are using it [10]. Up to now, the exact methods are not able to solve in a reasonable time solutions for instances with more than 50 - 100 clients [10]. This is a very serious limit if we take into account that customer lists of large companies may contain thousands of items. Consequently, for such cases, it is necessary to use approximate methods.

In Capacitated Vehicle Routing Problem (CVRP) we consider the following scenario: we own a delivery business that sends goods to clients via vehicles. Transport begins at the base station. The time needed to travel from base station to every client (and from every client to every other client) is known. We can look at this set-up as a full weighted graph with one highlighted vertex. The goal is to deliver every package to clients in the smallest possible time according to their demands. The capacity of each vehicles is fixed. The vehicles needs to

[1] The work was partially supported by the OPUS grant DEC-2012/05/B/ST7/00102 of Polish National Centre of Science.

go back to the base station to reload when empty. A general CVRP assume that demand of every client, number of vehicles and their capacity are not bound by any assertion. The vehicle routing problems have been attracting the interest of combinatorial optimization experts for over 50 years. The motivation to study this class of problems lies in its relevance to the real world as well as in its difficulty. One of books that are worth mentioning is [17]. It is an overview of main VRP variations (including CVRP). The authors show both exact and heuristic methods of finding the solutions. Large portion of the book covers the main variations, like: VRP with time windows, backhauls, pickup and delivery. Parallel Tabu Search for multi-criteria VRP problem was proposed in [11]. Other efficient methods, for multi-criteria discrete problems, were proposed and tested. Among them simulated annealing [13], genetic algorithm with local search method [15] or GACO hybrid algorithm [16]. Our further research on the multi-criteria VRP will certainly draw from the above mentioned work.

In our variation of the CVRP we assume that every client demands exactly one package and we have only one delivery vehicle with fixed capacity. It is easy to see that with these constraints our problem transforms into a permutation problem. Furthermore, it is very similar to the classical Travelling Salesman Problem (TSP) [7] with only difference being exclusion of the base station from permutation. Therefore, only vertices that represent clients are being permutated. Next, we can evenly partition the resulting permutation into sets of size equal to the capacity of the vehicle. These sets represent paths the vehicle will make with each round of deliveries.

The main goal of the research presented in this paper is to propose a new methodology of parallelization of the cost function determination function dedicated to be executed on the GPGPU (*General-Purpose computing on Graphics Processing Units*). The paper is organized as follows: firstly, we give a specification of the CVRP variation we will be solving. Next, we introduce multi-GPU algorithm. After that we show the results of the performed experiments. The main goal is to show the scalability of the algorithm.

2 Problem Definition

We consider Capacity constrained VRP (CVRP) here, in which the main constraint is connected with capacity of a single vehicle. Assumptions of the *CVRP* are:

- a fixed number of identical vehicles with specific capacity,
- a fixed number of customers with a specific orders of sizes and locations
- vehicles are able to perform client orders, but the total size handled by vehicle orders cannot be larger than its carrying capacity,
- vehicles are located in a central base, in which they starts and finish its work,
- travel costs between the points are given,
- the goal of optimization is to minimize the cost of customers service.

An instance of the *CVRP* can be described with using following notions:

$$C = \{1, ..., n\} \text{ – set of clients,} \tag{1}$$

$$G = (N, E) \text{ – graph, where} \tag{2}$$

$$N = \{0\} \cup C \text{ – base with a set of clients (set of vertexes),} \tag{3}$$

$$E \subseteq N \times N \text{ – set of edges,} \tag{4}$$

$$D = \{d_1, ..., d_n\} \text{ – size of order of particular clients,} \tag{5}$$

$$q \geq d_i, i \in \{1, ..., n\} \text{ – capacity of a single vehicle,} \tag{6}$$

$$V = \{1, ..., K\} \text{ – set of vehicles,} \tag{7}$$

$$[A_{ij}] \text{ – matrix of transport costs for elements of the set } N,\ i, j = 0, 1, 2, \ldots, n. \tag{8}$$

A solution of the *CVRP* consists of above elements and fulfills constraints presented below:

- K routes(single route for each vehicle),
- beginning and end of each routes in the same base,
- total size of all order on the route $\leq q$,
- each client is served exactly one time,
- sum of costs of driving on the particular route the cost of this route,
- sum of routes costs is the cost of the solution.

The aim of the optimization can be formulated as:

Find a solution with the smallest cost.

An adequate formal description can be presented as:

$$\text{minimalize} \sum_{k \in V} \sum_{\substack{i=\{0,1,\ldots,n\} \\ j=\{0,1,\ldots,n\}}} A_{ij} x_{ij}^k, \tag{9}$$

$$\text{where } x_{ij}^k = \begin{cases} 1, \text{ if the vehicle } k \text{ drives from } i \text{ to } j \\ 0, \text{ otherwise.} \end{cases}$$

3 Parallel Algorithm

The role of parallel algorithms which solves NP-hard problems significantly increased in the last decade, in particular for vehicle routing problems.

Rego [14] proposed a parallel tabu search algorithm (see also [4,5,6]) for the VRP which uses *ejection chain* type neighborhood; four *slave* processors were used in the implementation. The SSMS (*Single Starting point Multiple Strategies*) model was used, according to the classification Vo ss and cite Voss, i.e. slave processors receive the same starting solution, but have used different search strategies. Evaluation of the objective function also takes place concurrently.

Alba and Dorronsoro [1] proposed parallel genetic algorithm for VRP (*cellular* version), wherein crossover and mutations operators used local search procedures. The algorithm allowed oppearing if unfeasible solutions using the appropriately modified cost (penalty) function. The population is organized in mesh structure with crossover limitation to four neighbors on the mesh (see [2]).

Multi-colonial ant colony optimization (ACO) algorithm for VRP was proposed in the work of Doerner et al. [9]. Authors propose three different strategies of *master-slave* type with using standard local search operators and focusing exclusively on sequential algorithm speedup without improvement of the quality of obtained solutions. Tests were performed on 32 processors. Computational experiments show good scalability of the algorithm using up to 8 processors, for the greater number of them, as a result of increase the role of communication in parallel algorithm, the efficiency deteriorating.

On the other hand, for the problem of routing with time windows (*VRP with Time Windows*, VRPTW) Bouthillier and Crainic [12] proposed an effective procedure of parallel optimization based on four metaheuristics (two based on the tabu search method and two genetic-based approaches) and post-optimization procedure which uses *ejection chains* technique as well as local search procedures. The authors analyzed the profit resulting from co-operating nature of their algorithm by comparing the quality of obtained solutions with the approach without communication between processors.

4 Case Study

The huge computational complexity of the problem forces the use of more sophisticated techniques than a complete overview. No matter what technique will be used common fragments which are specific for a problem will occur. For VRP problems one of such parts is computing the cost of a solution. The following paper provides a study of scalability for computing the solution cost by using GPU units for three problems: VRP (without any limitations), Distance constrained Vehicle Routing PRoblem (DVRP) and Distance and Capacity Constrained Vehicle Routing Problem (DCVRP).

4.1 Generation of Test Data

Test data was generated in the initialization phase of the algorithm. In order to generate the cost (distance) matrix a pseudo-random number generator (srand function from gcc 4.4.3) was initialized with the constant value 7378342. Matrix elements were calculated as follow:

- if the element is located on the main diagonal of the matrix it takes the value 0,
- if the element is located above the main diagonal it is set to a random value in the range from 1 to 30 (computed by using rand function from gcc 4.4.3),

– if the element is located below the main diagonal then it will take the value of an element obtained by swapping its indexes. Next a pseudo-random test number is drawn. If it is even then the element value will increase by 0.20 and decrease by 0.20 otherwise.

In order to generate the capacity demands of each client a pseudo-random number generator (srand function from gcc 4.4.3) was initialized with the constant value 478 342. Each value was pseudo-randomly selected within the range from 0 to 99 (computed by using rand function from gcc 4.4.3).

The solutions were constructed in two stages. In the first stage the length of the route for a single vehicle was calculated by dividing the number of customers by the number of available vehicles (1 was added to the length of the route for the first R routes, where R is the rest of the division). Next the vehicle routes were populated with rising client numbers. In the second stage the order of solution elements was changed pseudo-randomly. For this purpose, a pseudo-random number generator was initialized with the constant value 5625 (srand function from gcc 4.4.3). Next two positions were picked pseudo-randomly and their values were swapped. The action was repeated as many times as the solution length.

4.2 Hardware

Studies were performed by using the following hardware:

– Intel R Core i7 CPU X 980 @ 3.33GHz (12 available cores),
– 24GB RAM memory,
– nVidia Tesla S2050 - 4 GPU units, each with 448 cores, 14 multiprocessors and 3GB RAM memory (global).

Used GPU units are equipped with the following types of memory (OpenCL names):

– global memory - the largest and slowest volume of available memory,
– constant memory - heavily buffered (an therefore much faster) read-only global memory. For Tesla S2050 GPU units its volume is limited to 64KB,
– local memory - very fast on-chip memory. For Tesla S2050 GPU units its volume is limited to 48KB per multiprocessor,
– private memory - this term usually refers to the registers.

Used GPU units provide a mechanism that can hide the global memory latency by minimising memory transactions. Unfortunately random accesses which are present in cost determination for the VRP problems make it impossible to take advantage of it.

4.3 Solution Methods

The maximum number of threads used was calculated by multiplying the total number of available cores with a multiplier. The resulting value is rounded up to the nearest multiple of the block size. Solutions are divided between the available devices (GPU) and threads (not necessarily the maximum number). While running the algorithm each threads calculates the costs for its set of solutions.

For the VRP problem the cost (distance) matrix and appropriate solutions were copied to the global memory of available devices. Additional global memory area is reserved for returning the calculated total cost value for each solution. The calculation is mostly associated with data reads of the first two global memory areas.

For the DVRP problem no additional data was copied to the devices but an additional global memory region was allocated for returning the maximum vehicle distance for each solution. The algorithm was slightly more complex due to the calculation of two costs but it does not involve additional reading of global memory. Computational overhead associated with parallelism remains unchanged, so the scalability of the problem DVRP should be slightly better than the VRP problem.

For the DCVRP problem additional data containing the clients capacity demands was copied to the devices. This data was read for calculating the maximum load of a single vehicle for each solution. Due to the necessity of additional cost calculation the algorithm gained complexity again but it is associated with additional global memory readings. In order to minimize the latency impact the algorithm was developed in two versions. The first version holds the additional DCVRP specific data in the global memory and the second in the constant memory. Computational overhead associated with parallelism is identical for both versions, so the scalability for the version using constant memory should be better. Fig. 1 contains the GPU code fragment which is responsible for calculating the costs for the DCVRP problem with constant memory.

4.4 Computational Experiments

The study was focused on the analysis of the scalability of calculating solutions costs for three problems (VRP, DVRP, DCVRP). The DCVRP problem was tested in two variants. In total four problems were tested.

The maximum number of threads was determined by multiplying the number of available cores by the multiplier. Each problem was tested with three different multiplier values (1, 2, 3).

The figures Fig.1.A., Fig.1.B, Fig.1.C, Fig.1.D. present the speed up (code executed on the GPU) for an increasing number of threads with respect to one thread for all four problems, the multiplier value set to 3, the length of a single solution set to 1099 and the total number of solutions set to 40000. The speedup is increasing even when the number of threads exceeds the number of available cores. This increase is relatively stable when the number of threads is less than about 2400.

Maximal speedups obtained for a particular problem was:

– VRP: 1098.17,

```
//This is one of the cost functions used to calculate the costs
//__global and __constant - address space used to refer to memory
//        objects allocated in the global or constant memory pool
//COST_TYPE - data type used to store track costs
//CAPA_TYPE - data type used to store vehicle capacities/client demands
//matrix, matrixRank - cost matrix and rank of the matrix
//clientCapas - demands of all clients
//solution, solLen - the solution and length of the solution
//cost, maxVehicle, maxVehicleCapa - variables used to return the calculated
//        values of cost of all tracks, maximum track cost, maximum track demand
void dcvrpCountCost(__global COST_TYPE* matrix, int matrixRank,
                __constant CAPA_TYPE * clientCapas,
                __global int * solution, int solLen,
                __global COST_TYPE * cost, __global COST_TYPE * maxVehicleCost,
                __global CAPA_TYPE * maxVehicleCapa) {

        //following 5 variables are stored in private memory (usually registers)
        //private memory is the fastest type of memory avialable
        COST_TYPE totalSolCost = 0;

        //dvrp
        COST_TYPE vehicleCost = matrix[0 + solution[0]];
        COST_TYPE tmp_maxVehicleCost = 0;

        //cvrp
        CAPA_TYPE vehicleCapa = clientCapas[solution[0]-1];
        CAPA_TYPE tmp_maxVehicleCapa = 0;

        //iterate over the elements of the solution
        for(int i=0; i<sol_len-1; i++) {
                //add travel cost between two clients or base to current track cost
                vehicleCost += matrix[solution[i]*matrixRank + solution[i+1]];
                //add client demand to current track demand
                if(solution[i] != 0) {
                        vehicleCapa += clientCapas[solution[i]-1];
                }
                //if this is true then all costs and demands of
                //the current track have been added
                if(solution[i+1]==0) {
                        //dvrp
                        //check if the current track has the biggest cost
                        if(i==0 || tmp_maxVehicleCost<vehicleCost) {
                                tmp_maxVehicleCost = vehicleCost;
                        }
                        //cvrp
                        //check if the current track has the biggest demand
                        if(i==0 || tmp_maxVehicleCapa<vehicleCapa) {
                                tmp_maxVehicleCapa = vehicleCapa;
                        }
                        //add current track cost to total cost of all tracks
                        totalSolCost += vehicleCost;
                        vehicleCost = 0;
                        vehicleCapa = 0;
                }
        }
        //following operations are used for the last track
        vehicleCost += matrix[solution[sol_len-1]*matrixRank + 0];
        vehicleCapa += clientCapas[solution[sol_len-1]-1];
        totalSolCost += vehicleCost;

        //dvrp
        if(tmp_maxVehicleCost<vehicleCost) {
                tmp_maxVehicleCost = vehicleCost;
        }
        //cvrp
        if(tmp_maxVehicleCapa<vehicleCapa) {
                tmp_maxVehicleCapa = vehicleCapa;
        }

        //copy the results to global memory
        *cost = totalSolCost;
        *maxVehicleCost = tmp_maxVehicleCost;
        *maxVehicleCapa = tmp_maxVehicleCapa;
}
```

Fig. 1. GPU code

– DVRP: 1419.27,
– DCVRP with global memory: 1395.84,
– DCVRP with constant memory: 1457.15.

The significant speedup improvement is very explicit between the DVRP and VRP problems which confirms previous assumptions. Increased complexity of the DCVRP algorithm was not enough to compensate the additional global memory readings. Therefore the maximal speedup obtained for DCVRP with global memory was lower than the maximal speedup for the DVRP problem. The use of constant memory has significantly increased this value so that it was even better than the speed up value obtained for the DVRP problem.

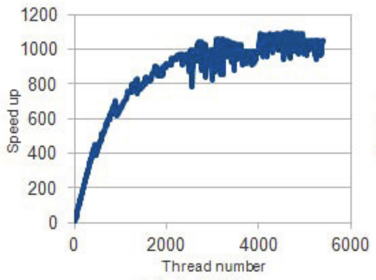

Fig.1.A. VRP

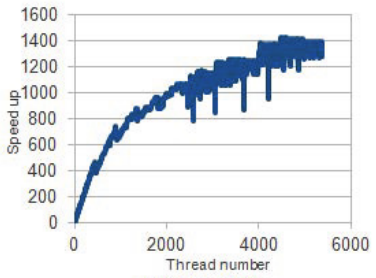

Fig.1.B. DVRP

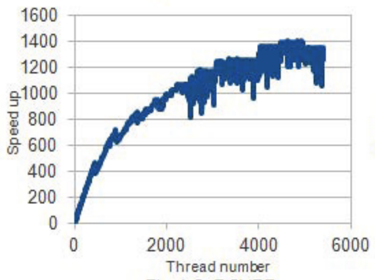

Fig.1.C. DCVRP

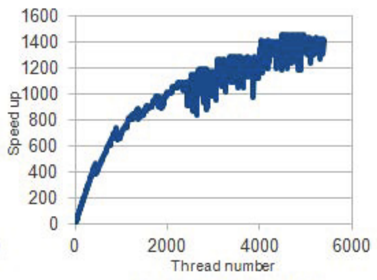

Fig.1.D. DCVRP_const

The figures Fig.2.A., Fig.2.B, Fig.2.C, Fig.2.D. present the change of the optimum block size with respect to the number of threads (block size was tested in the range from 1 to 32). A relatively stable growth can be observed when the number of threads is less than about 2400. Beyond this value the optimum size of the block still does not fall below certain base levels. For example for the DCVRP with constant memory the block size base level increases after the number of threads exceeds 448, 904, 1368, 1808, 2340. The number of available multiprocessors is 4x14=56 so the number of blocks is never greater than 8-9 per multiprocessor.

The figures Fig.3.A., Fig.3.B, Fig.3.C, Fig.3.D. present the obtained speedup for different number of devices (in respect to one device) for all four problems. For four devices the speed up was never less than 3.5 times.

The figure Fig.4. presents the execution time (GPU code) for the studied problems depending the number of solutions (the solution length is 1099). Explicit

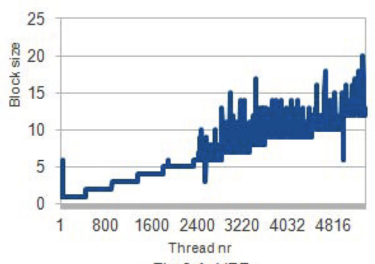

Fig.2.A. VRP

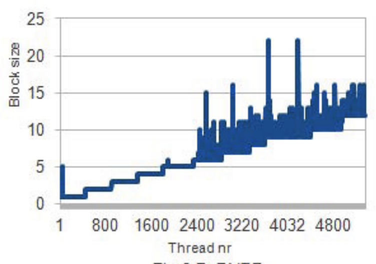

Fig.2.B. DVRP

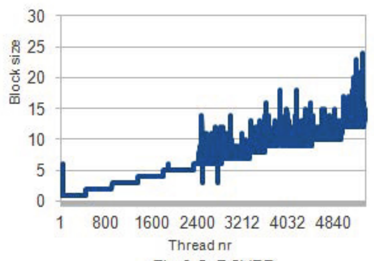

Fig.2.C. DCVRP

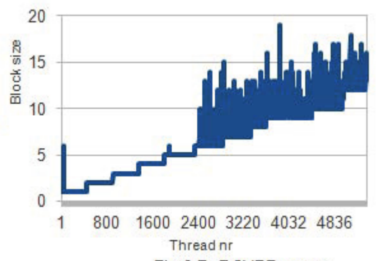

Fig.2.D. DCVRP_const

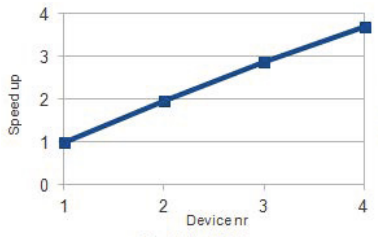

Fig.3.A. VRP

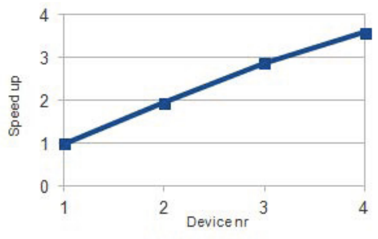

Fig.3.B. DVRP

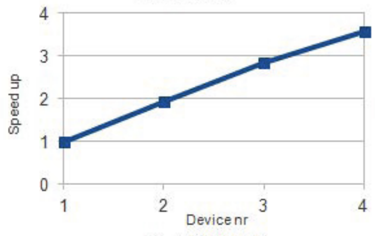

Fig.3.C. DCVRP

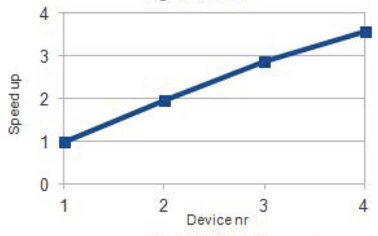

Fig.3.C. DCVRP_const

increase can be observed between the VRP, DVRP and DCVRP with global memory problems. For the DCVRP with constant memory problem the time has dropped so much that its chart practically overlaps with the time chart for the DVRP problem.

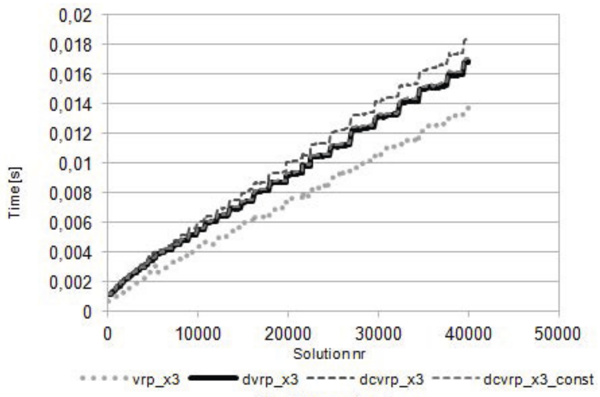

Fig.4. Execution time

5 Conclusions

Vehicles routes determination constitutes a major challenge for modern optimization algorithms. Most existing approaches are based on metaheuristic approach, due to the possibility of (approximate) solving of large size examples. Recently, the the role of parallel and distributed algorithms have been increasing, which further expands the range of potential applications of parallel discrete optimization methods in the field of logistics and management. Calculating the solution cost is one on of the most computationally intensive part of a local optimization algorithm of a VRP problem. In this paper we have checked the possibility to move cost calculation to the GPU in order to take advantage of the larger number of available cores. The method has been parallelized and tested against several aspects of the GPU architecture like the usage of different memory pools, different block sizes and number of threads in relation to the number of physical cores and multiprocessors.

References

1. Alba, E., Dorronsoro, B.: Computing Nine New Best-So-Far Solutions for Capacitated VRP with a Cellular Genetic Algorithm. Information Processing Letters 98(6), 225–230 (2006)
2. Bożejko, W., Wodecki, M.: Parallel genetic algorithm for minimizing total weighted completion time. In: Rutkowski, L., Siekmann, J.H., Tadeusiewicz, R., Zadeh, L.A. (eds.) ICAISC 2004. LNCS (LNAI), vol. 3070, pp. 400–405. Springer, Heidelberg (2004)
3. Bożejko, W., Wodecki, M.: Parallel Evolutionary Algorithm for the Traveling Salesman Problem. Journal of Numerical Analysis, Industrial and Applied Mathematics 2(3-4), 129–137 (2007)
4. Bożejko, W., Wodecki, M.: Solving Permutational Routing Problems by Population-Based Metaheuristics. Computers & Industrial Engineering 57, 269–276 (2009)

5. Bożejko, W., Uchroński, M., Wodecki, M.: The new golf neighborhood for the flexible job shop problem. In: Proceedings of the ICCS 2010, Procedia Computer Science 1, pp. 289–296. Elsevier (2010)
6. Bożejko, W., Pempera, J., Smutnicki, C.: Parallel Tabu Search Algorithm for the Hybrid Flow Shop Problem. Computers & Industrial Engineering 65, 466–474 (2013)
7. Bożejko, W., Wodecki, M.: On the theoretical properties of swap multimuwes. Res. Lett. 35(2), 227–231 (2007)
8. Dantzig, G.B., Ramser, J.H.: The Truck Dispatching Problem, INFORMS. Management Science 6(1), 80–91 (1959)
9. Doerner, K.F., Hartl, R.F., Kiechle, G., Lucká, M., Reimann, M.: Parallel ant systems for the capacitated vehicle routing problem. In: Gottlieb, J., Raidl, G.R. (eds.) EvoCOP 2004. LNCS, vol. 3004, pp. 72–83. Springer, Heidelberg (2004)
10. Hasle, G., Kloster, O.: Industrial Vehicle Routing, SINTEF ICT, Department of Applied Mathematics, P.O. Box 124 Blindern, NO-0314 Oslo, Norway
11. Jagiełło, S., Źelazny, D.: Solving Multi-criteria Vehicle Routing Problem by Parallel Tabu Search on GPU. Procedia Computer Science (18), 2529–2532 (2013)
12. Le Bouthillier, A., Crainic, T.: A cooperative parallel meta-heuristic for the vehicle routing problem with time windows. Computers & Operations Research 32, 1685–1708 (2005)
13. Pempera, J., Smutnicki, C., Żelazny, D.: Optimizing bicriteria flow shop scheduling problem by simulated annealing algorithm. Procedia Computer Science (18), 936–945 (2013)
14. Rego, C.: Node-ejection chains for the vehicle routing problem: Sequential and parallel algorithms. Parallel Computing 27, 201–222 (2001)
15. Rudy, J., Żelazny, D.: Memetic algorithm approach for multi-criteria network scheduling. In: Proceeding of the International Conference on ICT Management for Global Competitiveness and Economic Growth in Emerging Economies (ICTM 2012), pp. 247–261 (2012)
16. Rudy, J., Żelazny, D.: GACO: a parallel evolutionary approach to multi-objective scheduling. In: Gaspar-Cunha, A., Henggeler Antunes, C., Coello, C.C. (eds.) EMO 2015. LNCS, vol. 9018, pp. 307–320. Springer, Heidelberg (2015)
17. Toth, P., Vigo, D.: Vehicle Routing Problem, Society for Industrial and Applied Mathematics, Philadelphia (2001)