Acceleration of Neighborhood Evaluation for a Multi-objective Vehicle Routing

Szymon Jagiełło, Jarosław Rudy(✉), and Dominik Żelazny

Institute of Computer Engineering, Control and Robotics,
Wrocław University of Technology,
Janiszewskiego 11-17, 50-372 Wrocław, Poland
{szymon.jagiello,jaroslaw.rudy,dominik.zelazny}@pwr.edu.pl

Abstract. In this paper a multi-objective vehicle routing problem (MOVRP) with the criteria being the total distance and the utilization of the vehicle space is considered. Two methods were developed to decrease the execution time of the main part of the algorithm – the neighborhood search procedure. First method, an accelerator, is defined in order to reduce the computational complexity of the algorithm from $O(n^3)$ to $O(n^2)$. The second method utilizes multiple threads of execution to speedup the neighborhood search. Both methods were applied to tabu search metaheuristic and tested against the basic version of the algorithm. In result, we concluded that the enhanced version allows for significant reduction of execution time (2500 times for 5000 clients) that scales well with the number of clients. Moreover, this allows the enhanced algorithm to find significantly better approximations of the Pareto front in the same time as the original algorithm.

Keywords: Vehicle routing problem · Acceleration scheme · Multi-objective

1 Introduction

The Vehicle routing problem (VRP) has significant applications in both in transport and logistics as efficient methods of optimization allow to improve performance for a wide range of services and production systems. Reducing the time of deliveries, increasing the efficiency of the production or reducing costs are prime examples and can affect the positions of companies on the market.

Over the last years, researchers have studied routing problems with more than one criterion considered, called multi-objective vehicle routing problem (MOVRPs). In the following subsection, some approaches concerning MOVRPs and speeding up of VRP are briefly reviewed.

The swap operation is one of the most basic operations used in local optimization. Computational effort required to check a single new solution obtained by this operation is dependent on the size of the problem. In this paper we describe a method which can be used to check single new solutions in fixed time.

L. Rutkowski et al. (Eds.): ICAISC 2015, Part II, LNAI 9120, pp. 202–213, 2015.
DOI: 10.1007/978-3-319-19369-4_19

1.1 Literature Overview

In paper [11], Moura proposed an implementation of Genetic Algorithm (GA) for MOVRP with time windows and loading. Three optimization criteria were considered, namely the number of vehicles, the total travel distance and volume utilization. Results were compared with other heuristic approaches developed by Moura.

Wang *et al.* [12] studied a MOVRP that simultaneously considers the depot desires and clients expectations, which can better expound the real logistics operations than a single objective VRP. Objective function minimizes the total delivering path distance, while maximizing client satisfaction by fulfilling time-window requirements. The study proposed a hybrid algorithm based in GA and some greedy techniques. Moreover, military application was employed in order to confirm practical values of the proposed model and algorithm.

In paper [9], Guerriero *et al.* proposed an approach to unmanned aerial vehicle routing. Authors presented a distributed system of autonomous Unmanned Aerial Vehicles (UAVs) that are able to self-coordinate and cooperate. Considered criteria included the total distances, customer satisfaction and the number of UAVs. Case study was introduced as an application scenario.

Garcia-Najera *et al.* [6] considered solving MOVRP with time windows using evolutionary algorithm. Proposed heuristic incorporated methods for measuring the similarity of solutions and was tested using standard benchmark problems. Analysis showed that the proposed algorithm provided more diverse solutions of higher quality than standard evolutionary algorithm.

In paper [8], Grandinetti *et al.* considered Multi-objective Undirected Capacitated Arc Routing Problem. Objectives included minimizing the total transportation cost, the longest route cost (makespan) and the number of vehicles. An approximation of the optimal Pareto front is determined through an optimization-based heuristic procedure. Performance was tested and analyzed on benchmark instances known from literature.

Baños *et al.* [1] considered Capacited VRP with Time Windows with objectives of minimum distance and workload imbalance. Proposed algorithm combined evolutionary computations and simulated annealing. Authors used benchmark instances known from literature and tests confirmed good performance of proposed hybrid approach.

In paper [7], Ghoseiri *et al.* considered another VRP with Time Windows problem. Authors used goal programming and genetic algorithm. Additionally, aspiration levels and their deviations are specified by decision maker. Considered objectives include number of vehicles and total distance. Algorithm was tested using Solomon's benchmark instances and results show that the approach is effective and provided solutions are competitive with those known from literature.

Concerning speeding up for vehicle routing problems Bożejko *et al.* [2] studied a memetic algorithm for CVRP in parallel computing environment. The study included theoretical analysis using PRAM model and practical research on multi-GPU with the use of CUDA platform.

Żelazny *et al.* proposed another concept of parallel solving DVRP in [10]. Proposed MOTS algorithm was implemented in CUDA architecture and tested using new benchmark instances and outperformed classic MOTS in terms of speed and quality of solutions.

Ant Colony Optimization (ACO) is an example of metaheuristic based on many seperate agents and is thus well-suited for parallel computation. Such parallel ACO approach with the use of GPU for the VRP was proposed by Diego *et al.* [5].

As a final note, readers interested in more information about several methods and approaches to parallel VRP should refer to the review by Crainic [4].

2 Problem Description

The problem can be formulated by using the following symbols: c_{nr} (number of clients), v_{nr} (number of vehicles), 0 (the base), $C = \{0, c_1, c_2, ..., c_{c_{nr}}\}$ (sequence that includes all clients and the base), $V = \{v_1, v_2, ..., v_{v_{nr}}\}$ (vehicles sequence), $\Xi = \{\xi_1, \xi_2, ..., \xi_{c_{nr}}\}$ (clients demands sequence), max_Ξ such that $\bigvee_{\xi \in \Xi} max_\Xi \geq d$ (the capacity of a single vehicle), max_Θ (maximum track cost), $\Theta M \subseteq (c_{nr} + 1) \times (c_{nr} + 1)$ (travel cost matrix between elements form the C sequence).

Let V_{v_i}, where $1 \leq v_i \leq v_{nr}$ denote the vehicle with the number v_i and S_{v_i} denote the track of the vehicle V_{v_i}. Moreover, let $\Theta[S_{v_i}]$ denote a function returning the cost of traveling the track S_{v_i}, $\Xi[S_{v_i}]$ denote a function returning the total clients demands of the track S_{v_i}. Let C_{c_i}, where $1 \leq c_i \leq c_{nr}$ denote the client with the number c_i and $\Xi[C_{c_i}]$ denote the demand of the client C_{c_i}.

The optimal solution of our vehicle routing problem is a sequence of v_{nr} tracks $S^* = \{S_1, ..., S_{v_{nr}}\}$ that minimizes the given cost function f:

$$f(S^*) = \min_{s \in S_{feas}} f(s), \tag{1}$$

where S_{feas} is the set of all feasible solutions. A solution is feasible if: each client $C_{c_i} \in C$ is visited exactly once, each track $S_{v_i} \in S$ begins and ends in 0, $\bigvee_{1 \leq v_i \leq v_{nr}} \Theta[S_{v_i}] \leq max_\Theta$ and $\bigvee_{1 \leq v_i \leq v_{nr}} \Xi[S_{v_i}] \leq max_\Xi$. The exact cost function used in our case is mentioned in subsection 2.3.

2.1 Representation

The Giant Track Representation (GTR) is a method which provides an easy way for storing solutions inside computer memory and manipulating them. Let Sg be a sequence and $Sg_{v_i} \in Sg$ be the track of vehicle v_i where: $\bigvee_{1 \leq v_i \leq v_{nr}} Sg_{v_i}$ does not contain the 0 signifying the return to the base (it is default) and additionally track Sg_1 does not contain the 0 signifying the departure from the base (it is default). Then the sequence $Sg = \{Sg_1, ..., Sg_{v_{nr}}\}$ is called the giant track. For 9 clients (1 to 9) and 3 vehicles the giant track can look as follows:

$$4, 1, 6, 0, 3, 7, 0, 2, 9, 8, 5$$

and represents tracks: $0 \to 4 \to 1 \to 6 \to 0$, $0 \to 3 \to 7 \to 0$ and $0 \to 2 \to 9 \to 8 \to 5 \to 0$. The number of elements of a GTR equals $|Sg| = c_{nr} + v_{nr} - 1$.

2.2 Swap Neighborhood

Let N be a swap neighborhood for GTR solution Sg. Checking such neighborhood involves generating all solutions Sg' that can be obtained by switching two elements (from positions p_1 and p_2) of the solution Sg and calculating the values of the $\Theta[Sg']$, $Max_{\Theta}[Sg']$, $Min_{\Xi}[Sg]$ and $Min_{\Xi}[Sg']$ functions. The size of the neighborhood (the number of possible different Sg' solutions) is:

$$|N| = \frac{|Sg|^2 - |Sg|}{2}. \tag{2}$$

Thus, the computational complexity of generating all Sg' solutions without calculating the function values is $O(n^2)$.

2.3 Cost Function

The bi-criteria cost function considers the cost (distance) of visiting all clients:

$$\Theta[Sg] = \sum_{v_i=1}^{v_{nr}} \Theta[Sg_{v_i}] \tag{3}$$

and the maximum wasted space of all vehicles/tracks. It is computed by subtracting the minimum demand of all vehicles $Min_{\Xi}[Sg]$ from the capacity of a single vehicle:

$$Max_{\Delta}[Sg] = max_{\Xi} - Min_{\Xi}[Sg], \tag{4}$$

$$Min_{\Xi}[Sg] = \min_{S_{v_i} \in Sg} \Xi[Sg_{v_i}] \tag{5}$$

Now our cost function f is constructed as a normalized sum of both criteria:

$$f[Sg] = \frac{\Theta[Sg]}{\Theta^W} + \frac{Max_{\Delta}[Sg]}{Max_{\Delta}^W}, \tag{6}$$

where Θ^W and Max_{Δ}^W are the worst (highest) total track distance and vehicle space waste (utilization) respectively. These values are updated during the course of the optimization algorithm. The algorithm also needs the values of $Max_{\Theta}[Sg]$ and $Max_{\Xi}[Sg]$ (the maximum distance and maximum demand of all tracks in Sg), in order to test the feasibility of a given solution.

As stated before, we aim to minimize the cost function $f[Sg]$. The computational complexity of a brute-force approach to this kind of DCVRP is $O(n!)$.

3 Acceleration

Acceleration is used to speedup the evaluation of solutions Sg' from the swap neighborhood by quickly computing values $\Theta[Sg']$, $Max_{\Theta}[Sg']$, $Min_{\Xi}[Sg']$ and $Max_{\Xi}[Sg']$. The method utilizes additional data and specific properties of the *DCVRP* problem. It has been designed for parallel computing with the primary focus on minimizing the maximal number of necessary memory operations as memory access takes tens to hundreds times longer than a single processor cycle.

3.1 Assumptions and Symbols

Data with size depending on the problem instance are kept as arrays in memory:

- standard DCVRP data: Sg, ΘM, Ξ,
- support data: v_A (array assigning the positions in the solution to indexes of vehicles), v_P (array assigning vehicle/track indexes to positions in a solution), p_{Θ} (array of cumulated costs for the consecutive elements of the sequence Sg), p_{Ξ} (array of cumulated demands for the consecutive members of the sequence Sg).

The rest of the data is kept in registers of the processor:

- standard DCVRP data: values of cost functions ($\Theta[Sg]$, $Max_{\Theta}[Sg]$, $Max_{\Xi}[Sg]$ and $Min_{\Xi}[Sg]$), other (c_{nr} and v_{nr}),
- standard swap move data: swap positions (p_1, p_2),
- support data (for accelerator): second, third, and forth biggest cost values (denoted by $Max_{\Theta 2}[Sg]$, $Max_{\Theta 3}[Sg]$, $Max_{\Theta 4}[Sg]$), biggest demand values ($Max_{\Xi 2}[Sg]$, $Max_{\Xi 3}[Sg]$, $Max_{\Xi 4}[Sg]$) and smallest demand values ($Min_{\Xi 2}[Sg]$, $Min_{\Xi 3}[Sg]$, $Min_{\Xi 4}[Sg]$) for a given solution Sg.

All temporary variables are also kept in registers.

3.2 Basic Operations

The idea of the accelerator relies on several basic operations (some descriptions might seem obvious but they are essential for understanding the algorithm). Here we will describe a few of them in detail, while other cases will be described briefly for the sake of brevity. Before we start, let P_i be the value of element on position p_i in solution Sg. Moreover, let bP_i and aP_i be the values of elements on positions $p_i - 1$ and $p_1 + 1$ respectively. The basic operations are as follows:

1. Read the value of one of the following elements: P_1, P_2, bP_1, bP_2, aP_1 or aP_2. The operation reads the value from Sg from the position p_1, p_2, $p_1 - 1$, $p_2 - 1$, $p_1 + 1$ or $p_2 + 1$. It involves *one* read from memory for each operation but:
   (a) if $p_1 = p_2 - 1$ then $aP_1 = P_2$, $bP_2 = P_1$ so we can save up to *two* reads from memory,

   (b) if $p_1 = 0$ then bP_1 equals 0 by default so we can save up to *one* read from memory,
   (c) if $p_2 = |Sg| - 1$ then aP_2 equals 0 by default so we can save up to *one* read from memory.
2. Find the index v_i of the vehicle/track which is handling the Sg element from position e_i. The operation reads the value from v_A from the position e_i. It involves *one* read from memory.
3. Find the beginning of the v_i track. The operation reads the value from v_P from the position v_i. It involves *one* read from memory.
4. Find the beginning of the track which includes the element from the specified position e_i. The operation includes execution of operation 2 and 3, which involves *two* reads from memory but:
   (a) if $e_i = p_1$ and $P_1 = 0$ or $P_1! = 0$ and $bP_1 = 0$ then the first track element is placed on position p_1 or bP_1 respectively so we can save up to *two* reads from memory,
   (b) if $e_i = p_2$ and $P_2 = 0$ or $P_2! = 0$ and $bP_2 = 0$ then the first track element is placed on position p_2 or bP_2 respectively so we can save up to *two* reads from memory,
   (c) the end of the track Sg_{v_i-1} is also the beginning of the track Sg_{v_i} so if it is known we can save up to *one* read from memory.
5. Find the end of the track which includes the element from the specified position e_i. Similarly to operation 4, this operation includes execution of operation 2 and 3 which involves *two* reads from memory. Similar conditions as for operation 4 allows as to save up *one* or *two* reads from memory.
6. Read the travel cost between two clients or the base (c_1 and c_2). The operation reads the value from ΘM from the position $[c_1, c_2]$. It involves *one* read from memory. $\Theta M[c_1, c_2]$ denotes this operation.
7. Read the demand of the element E_i from the solution Sg. The operation reads the value from Ξ from the position E_i. It involves *one* read from memory.
8. Calculate the travel cost between two Sg elements placed on positions e_1 and e_2. The operation includes calculating the difference of two values from p_Θ placed on positions e_2 and e_1 and involves *two* reads from memory but:
   (a) if $e_1 < 0$ then the value from the position e_1 is set to 0 by default so we can save up to *one* read from memory,
   (b) if a value from p_Θ has been read already it is not read again so we can save up to *two* reads from memory,
   (c) if $e_1 >= e_2$ then the travel cost is set to 0 so we can save up to *two* reads from memory,
   (d) if $e_1 = e_2 - 1$ and $E_1 = 0$ and $E_2 = 0$ then then the travel cost is set to 0 so we can save up to *two* reads from memory.

   $ACC\_Dist[e_1, e_2]$ denotes this operation. Executing with a track as the argument means executing it for the beginning and end of the track .Using standard procedure would require a number of memory reads equal to the length of the track + 1.

9. Calculate the total demand between two Sg elements placed on positions e_1 and e_2 ($e_1 \leq e_2$). The operation includes calculating the difference of two values from p_Ξ placed on positions e_2 and $e_1 - 1$ and involves *two* reads. Similarly to operation 8, four specific cases can be listed that nace save us up to *one* or *two* reads from memory.
   $ACC\_Dem[e_1, e_2]$ denotes this operation. Executing with a track as the argument means executing it for the beginning and end of the track. Using standard procedure would require a number of memory reads equal to the length of the track + 1.

Above operations are based on detailed analysis of the specific properties of the *DCVRP* problem. All operations might be used for neighborhood checking for the *CVRP* problem. For the *DVRP* problem operations 1–6 and 8 will suffice. Thus, all benefits of using the accelerator will be preserved in those subproblems as well.

3.3 Initialization Step

The initialization step has to be executed before the neighborhood check of the solution Sg can proceed. It is used to set the values of v_A, v_P, p_Θ, p_Ξ, $\Theta[Sg]$, $Max_\Theta[Sg]$, $Max_{\Theta_2}[Sg]$, $Max_{\Theta_3}[Sg]$, $Max_{\Theta_4}[Sg]$, $Max_\Xi[Sg]$, $Max_{\Xi_2}[Sg]$, $Max_{\Xi_3}[Sg]$, $Max_{\Xi_4}[Sg]$, $Min_\Xi[Sg]$, $Min_{\Xi_2}[Sg]$, $Min_{\Xi_3}[Sg]$ and $Min_{\Xi_4}[Sg]$.

The computational complexity of this step is $O(n)$ for the sequential version as it requires a single passage through all members of the solution. Minimum computational complexity for the parallel version is $O(\log_2 n)$. It is the only part of the accelerator that is dependent on the problem instance size.

3.4 New Maximum and Minimum

During a swap move on the Sg soluion up to three tracks can change. This is one of the most important observations used for calculating the values of $Max_\Theta[Sg']$, $Max_\Xi[Sg']$ and $Min_\Xi[Sg']$ for the Sg' solutions. Let Sg_{v_i}, Sg_{v_j}, Sg_{v_k} be those 3 tracks. The values of $\Theta[Sg'_{v_i}]$, $\Theta[Sg'_{v_j}]$ and $\Theta[Sg'_{v_k}]$ are compared only with four values ($Max_\Theta[Sg]$, $Max_{\Theta_2}[Sg]$, $Max_{\Theta_3}[Sg]$, $Max_{\Theta_4}[Sg]$) which were set during the initialization step. It is required to compare them with v_{nr} values when using the standard method. Similar methods are used to quickly calculate $Max_\Xi[Sg']$ and $Min_\Xi[Sg']$.

3.5 Algorithm

In order to test the proposed accelerator we decided to implement a TS algorithm. Our algorithm was based on the implementation proposed in [3]. Some changes to the original algorithm were made.

Tabu Search. The algorithm stopped at a local minimum very fast when using the max_Θ and max_Ξ values from the beginning. This problem was resolved by utilizing the following method:

- let cMax_Θ denote the current maximum distance and cMax_Ξ the current maximum total demand of a single track. Also let IS denote the initial solution for the tabu search agorithm,
- before the optimization $\mathrm{cMax}_\Theta = \mathrm{Max}_\Theta[\mathrm{IS}]$, $\mathrm{cMax}_\Xi = \mathrm{Max}_\Xi[\mathrm{IS}]$,
- let $\mathrm{TABU}[Sg]$ denote a function that returns the best solution obtained after a single tabu iteration executed on the solution Sg,
- after each iteration:
  - if $\mathrm{Max}_\Theta[TABU[Sg]] > \mathrm{cMax}_\Theta$ then $\mathrm{cMax}_\Theta = \mathrm{Max}_\Theta[TABU[Sg]]$ else if $\mathrm{cMax}_\Theta > \mathrm{max}_\Theta$ then $\mathrm{cMax}_\Theta = \mathrm{cMax}_\Theta - (\mathrm{cMax}_\Theta - \mathrm{Max}_\Theta[TABU[Sg]])/2$,
  - if $\mathrm{Max}_\Xi[TABU[Sg]] > \mathrm{cMax}_\Xi$ then $\mathrm{cMax}_\Xi = \mathrm{Max}_\Xi[TABU[Sg]]$ else if $\mathrm{cMax}_\Xi > \mathrm{max}_\Xi$ then $\mathrm{cMax}_\Xi = \mathrm{Max}_\Xi - (\mathrm{Max}_\Xi - \mathrm{Max}_\Xi[TABU[Sg]])/2$,
  - if $\mathrm{cMax}_\Theta - \mathrm{max}_\Theta \leq 1$ then $\mathrm{cMax}_\Theta = \mathrm{max}_\Theta$,
  - if $\mathrm{cMax}_\Xi - \mathrm{max}_\Xi \leq 1$ then $\mathrm{cMax}_\Xi = \mathrm{max}_\Xi$.

Each iteration the tabu search algorithm chooses the best solution from all possible Sg' solutions from the swap neighborhood of the current solution. If there exits a Sg' solution for which $\mathrm{Max}_\Theta[Sg'] \leq \mathrm{cMax}_\Theta$ and $\mathrm{Max}_\Xi[Sg'] \leq \mathrm{cMax}_\Xi$ then the iterations best solution will be the solution with the lowest bi-criteria function value. Otherwise the solution with the lowest $Err[Sg']$ function value will be chosen:

- set the return value to 0 ($ret = 0$),
- if $(\mathrm{Max}_\Theta[Sg'] - \mathrm{cMax}_\Theta)/\mathrm{cMax}_\Theta > 0$ then $ret = \mathrm{Max}_\Theta[Sg'] - \mathrm{cMax}_\Theta)/\mathrm{cMax}_\Theta$,
- if $(\mathrm{cMax}_\Xi[Sg'] - \mathrm{cMax}_\Xi)/\mathrm{cMax}_\Xi > 0$ then $ret = ret + \mathrm{Max}_\Xi[Sg'] - \mathrm{cMax}_\Xi)/\mathrm{cMax}_\Xi$,
- return the ret value.

Choosing the best solution from all iterations is being done the same way.

4 Computer Experiment

In order to test the theoretical properties of the proposed speedup methods – both the accelerator and the parallel approach – in practice, we run a series of tests on Intel i7 X980 (6 cores, 12 concurrent threads) machine running Ubuntu (kernel 3.2.0-70-generic). All C/C++ programs were compiled with gcc 4.6.3. The tests were performed by running the tabu search metaheuristic with various neighborhood search methods (standard/accelerated, sequential/parallel) in order to compare them.

In our tests we focused on two aspects: a) the speedup obtained using the developed methods and b) the quality of the obtained solutions. We start with the former.

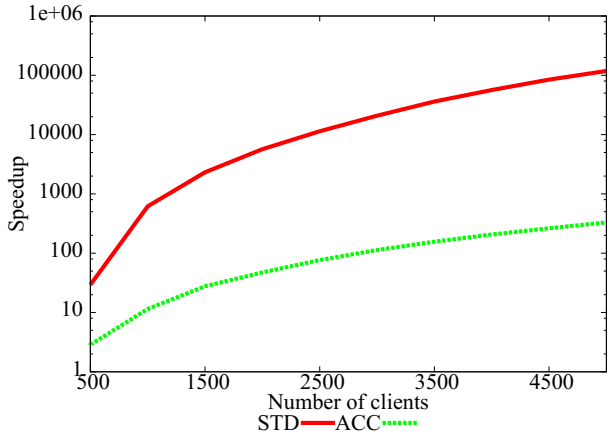

Fig. 1. Comparison of standard (STD) and accelarated (ACC) method for different number of clients (12 threads, log scale)

The relation between accelerated and standard version of the algorithm can be observed in Fig. 1. We have used logarithmic scale for clarity. We observe that accelerated version performs from approximately 10 to 350 times faster, depending on the number of clients.

Moreover, Tab. 1 can be consulted for the speedups of both accelerated and standard version when parallelization with multiple threads is used. All numbers were computed by dividing the execution time by the execution time of the sequential version (STD or ACC respectively). We observe that up to 6 threads

Table 1. Speedup for standard (STD) and accelerated (ACC) methods with different number of concurrent threads for 5000 clients

Method	1 thread	6 threads	12 threads
STD	1	5.66	6.31
ACC	1	5.41	7.02

(number of available physical cores) both algorithms can be sped up rather well, providing speedups of approximately 5.5. Further increase of the number of threads yielded much less increase in speedup (12 virtual cores available), however we can conclude that it is possible to speedup both versions of the method to comparable extent.

As a final test concerning the speedup aspect of our research, we decided to compare the slowest method (standard, only 1 concurrent thread) with the fastest (accelerated, 12 concurrent threads). The results, depending on the number of clients, are shown in Fig. 2. We observe that for as little as 500 clients we get a speedup of nearly 100. For 5 000 clients the speedup increases over 2 500 times compared to the original method. Moreover, the speedup increase is approximately linear in the function of the number of clients. We conclude that the

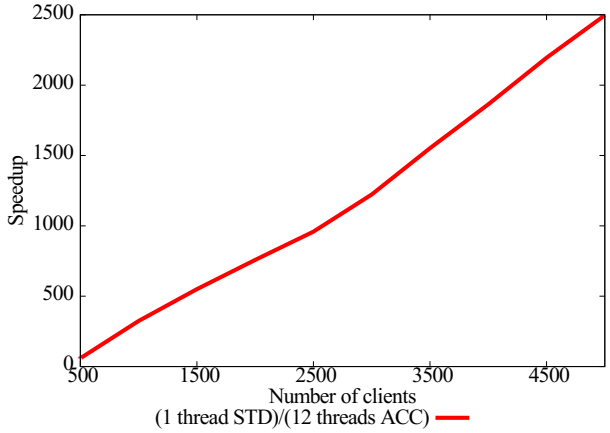

Fig. 2. Speedup of accelerated method (running 12 threads) over the standard method (running 1 thread)

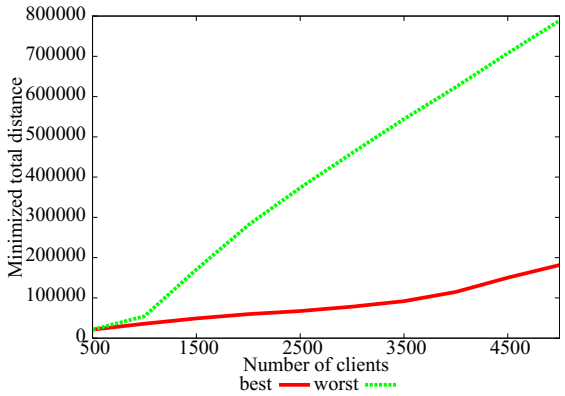

Fig. 3. Best values of total distance found by both methods for different number of clients (lesser is better)

improved method offers considerable speedup that scales well with the number of clients.

For the second aspect, the quality of obtained solutions, we simply decided to compare the best values of each criterion for different versions of the algorithm. For the sake of brevity we focused on two versions: "worst" (standard method, 1 thread) and "best" (accelerated method, 12 threads). In both cases the number of iterations of the tabu algorithm was adjusted so both algorithms would run for 1200 seconds. The results are shown in Fig. 3 (total distance, smaller is better) and Fig. 4 (vehicle utilization, greater is better).

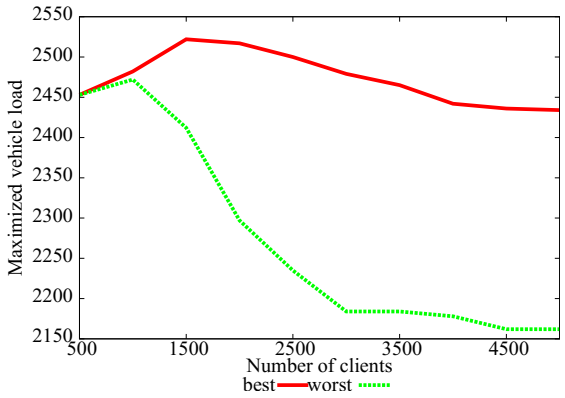

Fig. 4. Best values of vehicle utilization percentage found by both methods for different number of clients (greater is better)

It is clear that the improved version yielded much better results on both criteria. This is especially visible in the case of the total distance: the improved version yields even 4 times shorter distance (400% improvement over the original method). In the case of the vehicle utilization the improvement is much smaller, in the range of 5% to 15%. Those results are caused by the fact that the improved method is capable of running hundreds or thousands as many iterations in the same time as the original method, thus being able to check much more solutions. The greater speedup for greater number of clients only serves to amplify this effect.

5 Conclusions

Proposed acceleration scheme allowed us to speedup the algorithm up to 400 times over the classic one, independently from the the number of threads. Also, the accelerated version running on 12 threads was nearly 2500 times faster than classic algorithm running on single thread. Moreover, when comparing the quality of solutions, we observed that accelerated algorithm provided solutions with better values of both criteria. Hence, the proposed method allows to find better solutions to multi-criteria at the same time as the classic algorithm. This has a significant impact on solving multi- criteria problems of transport and logistics.

In the future research, an implementation of GPU version will be considered and tested using real-world data and order sets. Furthermore, some properties of multi-criteria problems and solutions evaluation will be considered in algorithm development, in order to further speed-up solving of MOVRPs.

Acknowledgements. The work was partially supported by the OPUS grant DEC-2012/05/B/ST7/00102 of Polish National Centre of Science and by the *Młoda Kadra*, grant no. B40129.

References

1. Baños, R., Ortega, J., Gil, C., Márquez, A.L., de Toro, F.: A hybrid meta-heuristic for multi-objective vehicle routing problems with time windows. Computers & Industrial Engineering (65), 286–296 (2013)
2. Wodecki, M., Bożejko, W., Karpiński, M., Pacut, M.: Multi-GPU parallel memetic algorithm for capacitated Vehicle Routing Problem. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Waśniewski, J. (eds.) PPAM 2013, Part II. LNCS, vol. 8385, pp. 207–214. Springer, Heidelberg (2014)
3. Bożejko, W., Pempera, J., Smutnicki, C.: Parallel tabu search algorithm for the hybrid flow shop problem. Computers & Industrial Engineering (65), 466–474 (2013)
4. Crainic, T.G.: Parallel Solution Methods for Vehicle Routing Problems, The Vehicle Routing Problem: Latest Advances and New Challenges. Operations Research/Computer Science Interfaces (43), 171–198 (2008)
5. Diego, F.J., Gómez, E.M., Ortega-Mier, M., García-Sánchez, A.: Parallel CUDA Architecture for Solving de VRP with ACO. Industrial Engineering: Innovative Networks, 385–393 (2012)
6. Garcia-Najera, A., Bullinaria, J.A.: An improved multi-objective evolutionary algorithm for the vehicle routing problem with time windows. Computers & Operations Research (38), 287–300 (2011)
7. Ghoseiri, K., Ghannadpour, S.F.: Multi-objective vehicle routing problem with time windows using goal programming and genetic algorithm. Applied Soft Computing (10), 1096–1107 (2010)
8. Grandinetti, L., Guerriero, F., Laganá, D., Pisacane, O.: An optimization-based heuristic for the multi-objective undirected capacitated arc routing problem. Computers & Operations Research (39), 2300–2309 (2012)
9. Guerriero, F., Surace, R., Loscrí, V., Natalizio, E.: A multi-objective approach for unmanned aerial vehicle routing problem with soft time windows constraints. Applied Mathematical Modelling (38), 839–852 (2014)
10. Jagiełło, S., Żelazny, D.: Solving Multi-criteria Vehicle Routing Problem by Parallel Tabu Search on GPU. Procedia Computer Science (18), 2529–2532 (2013)
11. Moura, A.: A Multi-Objective Genetic Algorithm for the Vehicle Routing with Time Windows and Loading Problem. Intelligent Decision Support, 187–201 (2008)
12. Wang, C.-H., Li, C.-H.: Optimization of an established multi-objective delivering problem by an improved hybrid algorithm. Expert Systems with Applications (38), 4361–4367 (2011)