2013 Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing

A parallel memetic algorithm for the vehicle routing problem with time windows

Mirosław Błocho*†
*Silesian University of Technology
Gliwice, Poland
† ABB ISDC
Krakow, Poland
email: Miroslaw.Blocho@pl.abb.com

Zbigniew J. Czech*†
*Silesian University of Technology
Gliwice, Poland
†Silesia University
Sosnowiec, Poland
email: Zbigniew.Czech@polsl.pl

***Abstract*—A parallel memetic algorithm for the NP-hard vehicle routing problem with time windows (VRPTW) is proposed. The algorithm consists of components which are executed as parallel processes. A process runs either a heuristic algorithm or a hybrid of a genetic algorithm and some local refinement procedures. In order to improve the results, processes co-operate periodically using a novel randomized scheme. During each phase of co-operation processes exploit their best solutions found so far. The purpose of the work is to devise the parallel memetic algorithm which determines the VRPTW solutions of the highest possible quality. The experiments on Gehring and Homberger's (GH) benchmarking tests show that the algorithm achieves very good results. By making use of it the best-known solutions to 171 out of 300 GH tests were improved.**

***Keywords*-parallel memetic algorithm, parallel processes co-operation schemes, genetic and local search algorithms, vehicle routing problem with time windows**

I. Introduction

The vehicle routing problem with time windows (VRPTW) is an important NP-hard optimization problem. It consists in determining the minimum cost routing plan to deliver goods from a single depot to a set of customers. The primary objective is to minimize the number of vehicles used, and the secondary one is to minimize the total distance traveled by the vehicles.

A number of approximate algorithms were proposed for the VRPTW. The most effective approximate algorithms to solve this problem rely on the construction heuristics, improvement heuristics and meta-heuristics. The construction heuristics create a feasible solution by inserting customers iteratively into the partial routes according to some criteria. The examples of using them can be found in [18], [21]. The improvement heuristics modify a current solution by executing local search moves to find better neighbor solutions. The most successful applications of these heuristics are described in [5], [17], [20]. The meta-heuristics usually embed construction and improvement heuristics and their examples can be found in [8], [9], [11], [19].

The memetic algorithms are built upon a population-based search approach. They combine an evolutionary algorithm for the global exploration of a solution space with a local search algorithm for the local exploitation of solutions already found [11]. The most efficient applications of the memetic algorithms to the VRPTW have been proposed so far in [1], [10], [14].

In this work a parallel memetic algorithm for the VRPTW is proposed. The algorithm consists of components which are executed in parallel as processes. A process runs either a heuristic algorithm or a hybrid of a genetic algorithm (GA) and some local refinement procedures. The edge assembly crossover (EAX) operator for reproduction of solutions in the GA is applied. In order to improve the final results parallel processes co-operate periodically using a novel randomized scheme. During each phase of co-operation processes exploit their best solutions found so far. The exploitation may involve the use of the EAX operator.

The purpose of the work is to devise the parallel memetic algorithm which determines the VRPTW solutions of the highest possible quality. In the experimental part of the work the speedup of the parallel algorithm and quality of achieved solutions on the MPI implementation of the algorithm are investigated.

The remainder of this paper is arranged as follows. Section II formulates the VRPTW problem. In section III the parallel memetic algorithm is presented. Section IV contains the discussion of the computational experiments. Section V concludes the paper.

II. Problem formulation

The VRPTW is defined on a complete graph $G = (V, E)$ with a set of vertices $V = \{v_0, v_1, ..., v_M\}$ and a set of edges $E = \{(v_i, v_j): v_i, v_j \in V, i \neq j\}$. The node v_0 represents a single depot and the set of nodes $\{v_1, v_2, ..., v_M\}$ represents the customers to be serviced. With each node $v_i \in V$ there are associated a load q_i ($q_0 = 0$), a service time s_i ($s_0 = 0$) and a time window $[e_i, l_i]$. Every edge (v_i, v_j) has a travel distance d_{ij} and a non-negative travel time c_{ij}. A feasible solution to the VRPTW is the set of m routes in graph G such that (a) each route starts and ends at the single depot, (b) every customer v_i belongs to exactly one route, (c) the total load of each route does not exceed the maximum vehicle capacity Q, (d)

978-0-7695-5094-7/13 $31.00 © 2013 IEEE
DOI 10.1109/3PGCIC.2013.28

```
1: // First stage - generating initial population of solutions
2: m ← R(RM(0));                                          ▷ m - minimum number of routes
3: for j ← 1 to N_pop do
4:     σ_j ← RM(m);  σ_j ← LocalSearch(σ_j);
5: end for
6: // Second stage - minimizing total travel distance
7: for P_i ← P_1 to P_N do in parallel
8:     while not finished do                              ▷ compute a generation of solutions
9:         generate a random permutation r(j) ∈ {1, 2, ..., N_pop};
10:        for j ← 1 to N_pop do
11:            σ_A ← σ_r(j);  σ_B ← σ_r((j+1)%N_pop);
12:            if σ_A ≠ σ_B then
13:                σ_best ← σ_A;
14:                for k ← 1 to N_ch do
15:                    σ_ch ← EAX(σ_A, σ_B);
16:                    σ_ch ← Repair(σ_ch);  σ_ch ← LocalSearch(σ_ch);
17:                    if σ_ch is feasible and F_td(σ_ch) < F_td(σ_best) then
18:                        σ_best ← σ_ch;
19:                    end if
20:                end for
21:                σ_r(j) ← σ_best;
22:            else
23:                σ_A ← Perturb(σ_A);
24:            end if
25:        end for
26:        finished ← Cooperate();
27:    end while
28: end for
29: return the best individual solution σ_best in the population;   ▷ P_1 returns σ_best
```

Figure 1. Parallel memetic algorithm (PMA)

the service at each customer v_i begins between e_i and l_i. The VRPTW is a bicriterion optimization problem with the hierarchical objectives. A desired solution to the VRPTW is a feasible solution with the minimum number of routes (primary objective) and the minimum total travel distance (secondary objective). As mentioned before, the VRPTW is NP-hard. Therefore the purpose of this work is to construct the parallel memetic algorithm which finds the VRPTW solution of the highest possible quality. Let (m, D) represent a solution α, where m is a number of routes and D is a total travel distance of this solution. Then we say that a solution β represented by (m_1, D_1) is of better quality, if either $m_1 < m$ or ($m_1 = m$ and $D_1 < D$).

III. Parallel memetic algorithm

The parallel memetic algorithm (PMA) consists of two stages in which the number of routes and the total travel distance are minimized independently (Fig. 1). In the first stage (lines 1–5) the minimum number m of routes in the VRPTW solutions is established. Furthermore, an initial population of solutions consisting of m routes is generated in each parallel process. In the second stage (lines 6–29) employing these populations a solution with the minimum total travel distance is found. In order to improve the final results and accelerate convergence of computation, the parallel processes executing the algorithm co-operate with each other in both stages.

A. Route number minimization

At the beginning of the first stage of the PMA (line 2, Fig. 1) the minimum number m of routes is established by calling the parallel heuristic algorithm RM shown in Fig. 2 [4]. The RM algorithm uses the EAX operator which is performed in co-operation phases (line 18, Fig. 2). The EAX operator creates offspring solutions by removing and replacing edges in two parent solutions without altering the orientation of edges[1]. The execution of the operator takes $\mathcal{T}_{EAX}(M) = O(M^2)$ time, where M is a number of customers in each parent solution [2]. The RM heuristic

[1]The EAX operator was first suggested for the traveling salesman problem by Nagata and Kobayashi [15], and was afterwards adapted for the capacitated vehicle routing problem by Nagata [12] and for the VRPTW by Nagata and Bräysy [13], [14].

```
1: function RM(m)
2:     for P_i ← P_1 to P_N do in parallel
3:         create an initial solution σ;
4:         while not finished do                                   ▷ remove a randomly chosen route
5:             initialize EP and penalty counters p[v_j] := 1, j = 1, 2, ..., M;
6:             while EP ≠ ∅ and not finished do                    ▷ insert ejected customers
7:                 if S_i^fe(v_r, σ) ≠ ∅ then
8:                     σ ← a new solution σ' chosen randomly from S_i^fe(v_r, σ);
9:                 else
10:                    σ ← Squeeze(v_r, σ);
11:                end if
12:                if v_r ∉ σ then
13:                    p[v_r] := p[v_r] + 1;
14:                    σ ← σ' from S_ej(v_r, σ) with minimum P_sum = Σ_{j=1}^{k_m} p[v_out^(j)];
15:                    add the ejected customers: v_out^(1), v_out^(2), ..., v_out^(k_m) to the EP;
16:                    σ ← Perturb(σ);
17:                end if
18:                finished ← Cooperate();
19:            end while
20:        end while
21:    end for
22:    return the best solution σ_best from all processes;          ▷ P_1 returns σ_best
23: end function
```

Figure 2. Parallel algorithm for minimizing the number of routes (RM)

algorithm includes N components which are executed in parallel as processes P_1, P_2, ..., P_N (line 2, Fig. 2). The algorithm starts with an initial solution σ in which each customer is served individually by a single vehicle (line 3). The attempts to decrease the number of routes in σ are then made, until the total execution time reaches a specified time limit T_{RM} (lines 4-20).

The RM algorithm proceeds as follows. A randomly chosen route is removed from the current solution σ and the set of unserved customers from this route is used to initialize the ejection pool *EP* (line 5). The pool holds all customers currently missing from σ. The penalty counters $p[v_j]$ initialized to 1 (line 5) indicate how many times the attempts to re-insert a given customer failed. Then, the continuous attempts to insert the unserved customers into the remaining routes are undertaken (lines 6–19) without violating the capacity and time windows constraints. These attempts are made until all customers from *EP* are inserted into σ, or the execution time exceeds its limit. At first, a customer v_r is selected from *EP* and a set $S_i^{fe}(v_r, \sigma)$ containing all possible insertions of v_r into σ is formed. If there exists at least one such solution then it is selected randomly from this set (line 8), otherwise the squeezing of σ is carried out (line 10). The Squeeze function temporarily accepts an infeasible insertion of v_r with the minimal value of some penalty function $F_p(\sigma)$, and a solution with the smallest penalty is selected. Then, a number of local search moves is carried out on this solution to restore its feasibility. If the squeezing is not successful then the penalty counter $p[v_r]$ is increased by 1 (line 13) and the ejections of previously inserted customers are performed (up to a limit k_m of ejected customers). The set $S_{ej}(v_r, \sigma)$ is formed, which includes a number of solutions with various combinations of ejected customers and v_r inserted at different route positions. A solution σ' with the minimum sum of the penalty counters is selected from $S_{ej}(v_r, \sigma)$ (line 14) and it replaces the current solution, while the ejected customers are inserted into the *EP* (line 15). In order to obtain better diversification of populations, the solution σ is perturbed by a number of local search moves (line 16).

As mentioned earlier, at the first stage of the PMA the initial population of N_{pop} feasible solutions consisting of m routes in every process is created (lines 3–5, Fig. 1). A single execution of each of the refinement functions LocalSearch, Perturb, Squeeze and Repair takes $O(M^2)$ time [2]. Let $\mathcal{T}_{RM}(N, M)$ denote the execution time of the RM algorithm, where N and M are numbers of processes and customers, respectively. Then the time taken by the first stage of the PMA (lines 1–5) is $\mathcal{T}_1(N, M) = \mathcal{T}_{RM}(N, M) + N_{pop}(\mathcal{T}_{RM}(N, M) + cM^2) = O(\mathcal{T}_{RM}(N, M) + M^2)$, for some constant c. It can be shown [2] that the average execution time of the RM algorithm is $\mathcal{T}_{RM}(N, M) = O(M^{2.7} + N \cdot M^2)$ (the second term corresponds to co-operation cost; see subsection III-B), thus

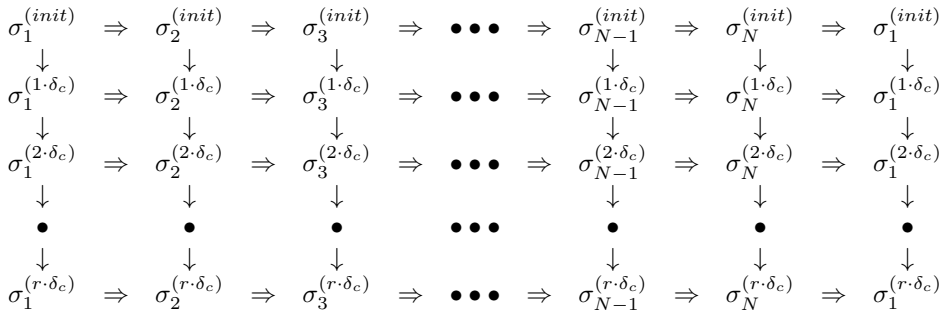

Figure 3. Co-operation of parallel processes ($\sigma_i^{(init)}$ - initial solution of P_i; $\sigma_i^{(j\cdot\delta_c)}$ - j co-operation phase of P_i; δ_c - number of steps between co-operations; r - number of co-operation phases; $\Rightarrow$ - data transfer; $\downarrow$ - progress of process execution)

$\mathcal{T}_1(N, M) = O(\mathcal{T}_{RM}(N, M) + M^2) = O(M^{2.7} + N \cdot M^2)$.

B. Co-operation of parallel processes

The processes co-operate periodically in both stages of the PMA. In the first stage they co-operate every δ_{RM} step of inserting the ejected customers (smaller δ_{RM} means higher frequency of co-operation), and in the second stage every δ_{DM} computed generation. At the beginning of each co-operation phase (line 26, Fig. 1, and line 18, Fig. 2) the master process[2] P_1 generates a random permutation of process numbers and broadcasts it asynchronously to the remaining processes. This permutation determines the order in which processes co-operate in a given phase. Each co-operation phase is started by process P_1 which sends its best solution found so far to process P_2 (Fig. 3). Such a chain of transfers continues in a pipeline fashion for the pairs of processes (P_2, P_3), (P_3, P_4), ..., until the last process P_N in a current permutation obtains the best solution from all the processes. The co-operation scheme constitutes a ring, thus this solution is sent back from process P_N to P_1.

If during a co-operation phase any process P_i receives from its neighbor a solution of better quality (see section II) then this solution replaces the current one in P_i. Otherwise, the EAX operator is performed on a pair of solutions (received and the current one in P_i) what results in a child solution σ_{ch}. If σ_{ch} is not feasible, then the local search moves are performed to restore its feasibility. If the repair is successful then a child solution σ_{ch} replaces the current solution. The co-operation among processes is carried out in order to improve the quality of a final solution through the deeper exploration of the search space and better convergence of computation. The randomized scheme of co-operation allows for a more diverse reproduction of solutions, since the EAX operator can be performed on a pair of solutions of neighbor processes. One of these solutions is always the best achieved so far by a process. The way the processes co-operate guarantees that after each co-operation phase the master process P_1 holds the best solution found by all the processes.

Last but not least, non-blocking operations for data transferring are used to make the co-operation as fast as possible. This is important because in the presence of limits imposed on the execution time of the PMA (see subsection IV-A), a fast scheme allows for more co-operation phases to be carried out. At the end of each co-operation phase a process has to wait for already started operations to be finished (all communication operations are asynchoronous). During this waiting time the process continues its work, e.g. creation of a generation of solutions. As a result, computation is overlapped with communication. A two-step procedure of passing data between processes is also introduced. In the first step, only a number of routes m or a total travel distance D (depending on the PMA stage) is sent to a neighbor process, along with the information whether a complete solution will be sent in the second step. The complete solution is sent only if m or D of solutions have decreased, comparing to their values in the last co-operation phase. Clearly, a single cooperation phase takes time $\mathcal{T}_C(N, M) = O(N \cdot M^2)$.

C. Distance minimization

The total travel distance is minimized in the second stage of the PMA (lines 6–29, Fig. 1). To this aim the set of processes P_1, P_2, ..., P_N, each executing a memetic algorithm is run. The processes co-operate with each other using the randomized co-operation scheme described in subsection III-B. The main **while** loop of the memetic algorithm contains the operations required to create a single generation of solutions (lines 9–25), and to carry out the co-operation phase (line 26). In each iteration a random permutation of solutions is generated, so as to diversify the search process (line 9). Then, parent solutions σ_A and σ_B are selected (line 11). In order to avoid having the same individuals in a population, some perturbing steps on σ_A are performed in case when $\sigma_A = \sigma_B$ (line 23). If these

[2]We distinguish process P_1 calling it master. It decides on the order in which processes co-operate. It also controls the execution time of both stages of the algorithm by sending to processes the signal to finish computation when this time elapses.

solutions are different, then solution σ_A is copied into σ_{best}, which holds the best solution in a current iteration (line 13). The EAX operator is carried out on solutions σ_A and σ_B, and a child solution σ_{ch} is obtained. If it is not feasible, then the repair is undertaken to fix the violated constraints (line 16). In addition it is enhanced by a number of local search moves which reduce the total travel distance (line 16). If a child solution σ_{ch} has a smaller total travel distance than the best solution σ_{best}, then the latter is replaced with σ_{ch} (line 18). Note that processes P_1, P_2, ..., P_N co-operate periodically every δ_{DM} step (line 26). In a step a single generation of solutions is computed (lines 9–25). The analysis shows [2] that the average execution time of the second stage of the PMA is $\mathcal{T}_2(N, M) = O(M^2 + N \cdot M^2)$. The time complexities of both stages of the PMA grow linearly with the number of processes N. Therfore one may expect that in the presence of execution time limits imposed on the PMA (see subsection IV-A), when N increases, a number of attempts to decrease a route number in the RM algorithm, and a number of computed generations in the second stage of the PMA will decrease. Most likely, however, this will be compensated by co-operation of a larger number of processes.

IV. Computational experiments

A. Settings

The PMA was implemented in C++ using the MPI (Message Passing Interface) library. The computational experiments were conducted on Galera supercomputer (www.task.gda.pl/kdm/sprzet/Galera). The total execution time of the PMA consists of three elements: $T = T_{RM} + T_{PopGen} + T_{DM}$, where T_{RM} is the time for establishing the minimum number of routes (line 2, Fig. 1), T_{PopGen} the time for creating the initial population (lines 3–5), and T_{DM} the time for the minimization of the total travel distance (lines 7–29). The following execution time limits were set: $T_{RM} = 10$ min., $T_{PopGen} = 1$ h and $T_{DM} = 3$ h. The experiments showed that the time $T_{RM} = 10$ min. was sufficient to find the best-known minima of route numbers for GH tests in nearly 100% cases. The RM algorithm run on 64 processors found those minima for 89% tests in time less than 10 sec. So in most cases this time is negligible as compared to the total execution time T of the algorithm. In terms of generation of the initial population, its size was fixed to $N_{pop} = 100$ and the number of children in each reproduction step was fixed to N_{ch} = 20. In most cases (96%) the time for finding a single member of the population was at most 10 sec., for a total of 1000 sec. $\approx$ 17 min., i.e. well below the limit of $T_{PopGen} = 1$ h. However for 4% of GH tests it was not possible to find the required number of $N_{pop} = 100$ solutions. In those cases the missing solutions were obtained by means of the Perturb function applied to solutions which were successfuly generated. The maximum number of ejected customers (see subsection III-A) was set to k_m = 4. The periods of process co-operations were set to δ_{RM} = 100 and δ_{DM} = 2. Furthermore, the execution of the algorithm was terminated when the best solution in the population was not improved for the consecutive 100 generations.

The algorithm was run on the benchmarking tests by Gehring and Homberger (GH) [7]. Six groups of tests were designed to highlight several factors. The C1 and C2 groups include the customers located in clusters, while in the R1 and R2 groups the locations are generated randomly. The RC1 and RC2 groups contain a mix of random and clustered customers. The groups C1, R1 and RC1 have smaller vehicle capacities and shorter time windows than the groups C2, R2 and RC2. The GH tests consist of five sets containing $M = 200$, 400, 600, 800 and 1000 customers, with 60 instances in each set, resulting in 300 instances.

B. Speedup analysis

The speedup of the MPI implementation of the PMA was examined on RC2_6_3 and C1_10_4 tests. The target solutions were set to 11/9600 for RC2_6_3 and 90/40000 for C1_10_4, which should be reached in each test case (in a/b, a stands for the number of routes and b for the total travel distance). For these targets the execution times of the PMA were recorded. Overall 60 experiments were performed on each number of processors: $N = 1$, 8, 16, ..., 88, 96. The speedup was calculated as a ratio of the average execution time of the PMA for $N = 1$ to the average execution time of the PMA for a given $N > 1$, (Fig. 4). The achieved speedup for the test RC2_6_3 was slightly worse than for the test C1_10_4. For example, for 96 processors the speedups were 44.71 and 50.21, respectively (the maximum possible speedup in this case equals 96). Note that the speedup comes from two factors. Firstly, a larger number of processors searching the solution space simultaneously reach a target solution faster than a single one. Secondly, due to co-operation of processors a faster convergence to a target solution is achieved. For both tests, the PMA obtained larger speedupa for a small number of processors (8, 16). The worse speedup for a larger number of processors can be explained by the higher cost associated with the co-operation and synchronization of processors. However, conducting computation with a large number of processors is beneficial in terms of higher quality of solutions.

C. Solutions quality analysis

In order to evaluate the quality of solutions of the PMA, the algorithm was executed seven times on each GH test instance using 64 processors. The best results obtained are presented in Tab. I. The naming convention of tests needs some clarification, e.g. test R2_4_3 denotes the 3rd test from group R2 having 400 customers. The best achieved result for this test is 8/5911.50. By making use of the MPI implementation of the PMA we improved, in a reasonable time, the

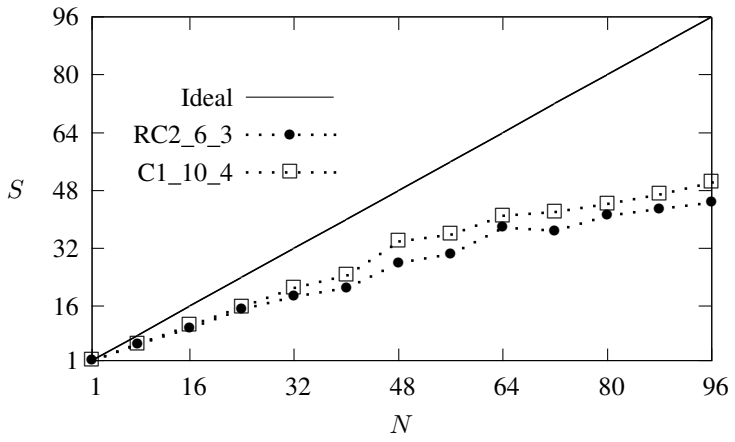

Figure 4. Speedup S vs. number of processors N for tests RC2_6_3 and C1_10_4 (Ideal stands for the maximum speedup, i.e. equal to N)

solutions to 171 out of 300 GH tests, as compared to the best solutions published on Sintef website [7] on January 30, 2013. The cumulative total distance (CTD) was calculated to evaluate quality of solutions for the particular GH test groups as a whole (Tab. II). Considering the percentage, $\%B$, of the CTD, it can be seen that the best results were achieved for the GH test groups RC2 and C2 (85.86% and 96.42%) (the smaller CTD the better). The values of CTD indicate that the PMA gives better results for the problems with large vehicle capacities and wide time windows, and for customers located in clusters.

V. Conclusions

The parallel memetic algorithm for the NP-hard vehicle routing problem with time windows is presented. The experiments on GH tests show that the algorithm achieves very good results. We believe that this is due to the novel scheme of process co-operation, which is the main contribution of this work. In the scheme, processes are arranged into a ring, however their order within the ring changes randomly in consective phases of co-operation. During each phase processes pass through the ring and exploit their best solutions found so far. On these solutions the EAX operator is performed, what is adventegous since the created child solutions have likely even better quality than their parents. The cost of co-operation is relatively small, because in a single phase processes communicate only in pairs, as opposed to a case when processes co-operate in larger groups using all-to-all communication. Due to randomization these pairs of co-operating processes changes from phase to phase, what results in better diversifiaction of offspring solutions. Clearly, the efficiency of the EAX operator and the memetic computation paradigm also contribute to high quality of the VRPTW solutions.

Acknowledgments

We thank the following computing centers where the computations of our project were carried out: Academic Computer Centre in Gdańsk TASK, Academic Computer Centre CYFRONET AGH, Kraków (computing grant 027/2004), Interdisciplinary Centre for Mathematical and Computational Modeling, Warsaw University (computing grant G27-9), Wrocław Centre for Networking and Supercomputing (computing grant 30).

References

[1] Berger J., Barkaoui M.: *A parallel hybrid genetic algorithm for the vehicle routing problem with time windows*. Computers & OR, 2004: 2037–2053.

[2] Błocho M.: *A parallel memetic algorithm for the vehicle routing problem with time windows*, PhD thesis, mansucript, 2013 (in Polish).

[3] Błocho M., Czech Z.J.: *A parallel algorithm for minimizing the number of routes in the vehicle routing problem with time windows*, Parallel Processing & Applied Mathematics (PPAM'2011), Part I, LNCS 7203, pp. 255–265. Springer, Heidelberg, 2012.

[4] Błocho M., Czech Z.J.: *A parallel EAX-based algorithm for minimizing the number of routes in the vehicle routing problem with time windows*, Proc. 2012 IEEE 14th Intern. Conf. on High Performance Computing and Communications, Liverpool, 1239–1246.

[5] Bräysy O., Hasle G., Dullaert W.: *A multi-start local search algorithm for the vehicle routing problem with time windows*, Eur. Jour. of OR, 2002.

[6] Crainic T.G., Nourredine H.: *Parallel meta-heuristics applications*, In: Alba, E. (ed.) Parallel Metaheuristics: A New Class of Algorithms, Wiley, Hoboken, NJ, 2005, 447-494.

Table I
THE BEST RESULTS FOR GH TESTS (NEW WORLD-BEST RESULTS ARE MARKED IN BOLDFACE)

	C1	C2	R1	R2	RC1	RC2
200 customers						
1	20/2704.57	6/1931.44	**20/4784.11**	4/4483.16	18/3602.80	6/3099.53
2	18/2917.89	6/1863.16	18/4042.42	4/3621.20	**18/3249.05**	5/2825.24
3	18/2707.35	**6/1775.08**	18/3381.96	4/2880.62	18/3008.76	**4/2601.87**
4	18/2643.31	6/1703.43	**18/3058.91**	4/1981.30	18/2865.09	**4/2038.56**
5	20/2702.05	6/1878.85	18/4107.86	4/3366.79	**18/3371.00**	4/2911.46
6	20/2701.04	6/1857.35	18/3583.14	4/2913.03	18/3324.80	**4/2873.12**
7	20/2701.04	6/1849.46	18/3150.11	4/2451.14	18/3189.32	4/2525.83
8	**19/2775.48**	6/1820.53	18/2958.19	4/1849.87	18/3094.49	4/2295.97
9	18/2687.83	6/1830.05	18/3760.58	**4/3092.04**	**18/3081.38**	4/2175.04
10	18/2643.55	**6/1806.58**	18/3306.21	4/2654.97	18/3006.93	4/2015.61
400 customers						
1	40/7152.06	12/4116.14	**40/10372.31**	**8/9210.15**	36/8626.23	**11/6682.37**
2	36/7695.55	12/3930.05	36/9009.64	8/7606.75	36/7984.53	**9/6180.62**
3	36/7075.98	**11/4018.22**	36/7898.63	**8/5911.50**	36/7579.90	**8/4930.84**
4	36/6816.71	**11/3720.51**	36/7322.16	**8/4241.47**	36/7348.34	**8/3632.43**
5	40/7152.06	12/3938.69	36/9304.56	**8/7136.90**	36/8265.36	**8/6710.12**
6	40/7153.45	12/3875.94	36/8450.80	**8/6122.60**	**36/8190.19**	**8/5766.61**
7	39/7417.92	12/3894.16	36/7673.40	**8/5018.53**	**36/7965.04**	**8/5360.34**
8	**37/7347.23**	12/3789.89	36/7291.25	**8/4021.86**	36/7775.24	**8/4793.06**
9	**36/7045.55**	12/3865.65	36/8782.08	**8/6400.10**	**36/7767.43**	**8/4551.80**
10	36/6869.82	**11/3833.90**	36/8156.40	**8/5791.79**	**36/7655.50**	**8/4280.79**
600 customers						
1	60/14095.64	18/7774.16	59/21412.22	**11/18214.90**	**55/17231.05**	**14/13324.93**
2	56/14163.31	**17/8264.91**	**54/18972.64**	**11/14817.98**	55/16135.53	**12/11555.51**
3	**56/13777.81**	**17/7528.11**	54/17229.91	**11/11224.81**	55/15425.99	**11/9461.25**
4	**56/13563.30**	**17/6924.42**	54/15997.90	**11/8042.38**	55/14883.65	**11/7141.71**
5	60/14085.72	18/7575.20	54/20133.02	**11/15096.20**	55/16912.68	**11/13066.52**
6	**59/16345.44**	18/7471.17	54/18358.93	**11/12577.36**	**55/16787.63**	**11/11933.88**
7	**58/14816.55**	**18/7512.07**	54/16941.41	**11/10114.29**	**55/16369.30**	**11/10773.25**
8	**56/14487.58**	**17/7596.28**	54/15852.44	**11/7655.51**	55/16198.94	**11/10047.86**
9	**56/13715.48**	**17/7958.26**	54/19047.80	**11/13377.56**	**55/16122.07**	**11/9599.28**
10	**56/13661.85**	**17/7255.85**	**54/18179.64**	**11/12253.47**	**55/15897.65**	**11/9078.79**
800 customers						
1	80/25184.38	24/11662.08	**80/36889.96**	**15/28114.25**	**72/33412.03**	**18/20981.14**
2	**72/27189.16**	**23/12286.65**	72/33023.10	**15/22797.63**	**72/30272.08**	**16/18183.64**
3	72/24516.10	**23/11411.53**	**72/29910.18**	**15/17839.00**	**72/28404.90**	**15/14512.94**
4	**72/23914.01**	**23/10850.36**	72/28160.50	**15/13307.72**	**72/27395.14**	**15/11102.47**
5	80/25166.28	**24/11425.23**	**72/34051.67**	**15/24285.89**	**72/31420.09**	**15/19136.03**
6	**79/28665.57**	**23/13150.63**	**72/31476.28**	**15/20563.41**	**72/31240.49**	**15/18151.67**
7	**77/26492.64**	**24/11370.84**	72/29356.48	**15/16837.74**	**72/31007.16**	**15/16883.26**
8	74/26595.85	**23/11303.24**	72/28077.76	**15/12855.02**	**72/30289.94**	**15/15818.98**
9	**72/24653.26**	**23/11726.45**	**72/32814.47**	**15/22402.79**	**72/30479.51**	**15/15359.99**
10	**72/24218.37**	**23/10989.13**	72/31645.81	**15/20494.35**	**72/29969.74**	**15/14468.66**
1000 customers						
1	100/42478.95	30/16879.24	**100/53762.92**	**19/42219.21**	**90/46743.83**	**20/30278.50**
2	**90/42509.62**	**29/17126.39**	91/49819.61	**19/33586.49**	90/44720.62	**18/26677.87**
3	**90/40356.18**	**28/16884.08**	91/46136.08	**19/25309.46**	90/42886.30	**18/20177.96**
4	90/39641.46	**28/15743.88**	91/43402.31	**19/18182.09**	90/41841.87	**18/15820.37**
5	100/42469.18	**30/16561.57**	91/53971.76	**19/36335.72**	**90/45949.78**	**18/27269.46**
6	**99/44108.34**	**29/16920.33**	91/48615.92	**19/30247.98**	90/45799.43	**18/26965.51**
7	**97/44806.73**	**29/17882.42**	91/45383.56	**19/23381.36**	90/45361.67	**18/25317.29**
8	94/41853.36	**28/18343.01**	91/43072.25	**19/17598.63**	90/44791.07	**18/24010.00**
9	**90/41006.16**	**29/16442.47**	91/51093.89	**19/33131.99**	**90/44882.70**	**18/23341.21**
10	90/40229.20	**28/15988.21**	91/49423.01	**19/30656.00**	**90/44391.00**	**18/22372.41**

Table II
PERCENTAGE OF OBTAINED CUMULATIVE TOTAL DISTANCE (CTD) VS. WORLD-BEST CTD FOR GH BENCHMARK GROUPS (NEW WORLD-BEST RESULTS ARE MARKED IN BOLDFACE)

	C1	C2	R1	R2	RC1	RC2	Avg
200	100.19%	100.00%	**99.37%**	100.00%	100.03%	**99.80%**	**99.90%**
400	100.07%	**98.70%**	**99.64%**	**99.32%**	100.27%	**97.28%**	**99.21%**
600	**98.86%**	**95.75%**	100.49%	**98.83%**	**99.92%**	**78.09%**	**95.32%**
800	**97.51%**	**97.17%**	**99.20%**	**98.77%**	**92.60%**	**66.87%**	**92.02%**
1000	**94.71%**	**90.48%**	**99.72%**	**99.54%**	**99.47%**	**87.27%**	**95.20%**
Avg	**98.27%**	**96.42%**	**99.68%**	**99.29%**	**98.46%**	**85.86%**	**96.33%**

[7] Gehring and Homberger's benchmarking tests: http://\-www.\-sintef.\-no/\-Projectweb/\-TOP/\-Problems/\-VRPTW/

[8] Glover F.: *Future paths for integer programming and links to artificial intelligence*, Comp. & OR(1986);13: 533–549.

[9] Holland JH.: *Adaptation in natural and artificial systems*, Ann Arbor: University of Michigan Press, 1975.

[10] Labadi N., Prins C., Reghioui M.: *A memetic algorithm for the vehicle routing problem with time windows.* RAIRO - OR 42(3): 415–431 (2008).

[11] Moscato P.: *On evolution, search, optimization, genetica lgorithms and martial arts towards memetic algorithms.* C3P Report 826, California Institute of Technology; 1989.

[12] Nagata Y.: *Edge assembly crossover for the capacitated vehicle routing problem.* In: Proceedings of the 7th European Conference on evolutinary computation in combinatorial optimization, 2007. p. 124–153.

[13] Nagata Y., Bräysy O.: *A powerful route minimization heuristic for the vehicle routing problem with time windows.* Oper. Res. Lett. 37(5): 333–338 (2009).

[14] Nagata Y., Bräysy O., Dullaert W.: *A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows.* Comp. & OR 37(4): 724–737 (2010).

[15] Nagata Y. Kobayashi S.: *Edge assembly crossover: a high-power genetic algorithm for the travelling salesman problem.* In: Proceedings of the 7th international conference on genetic algorithms, 1997. p. 450–457.

[16] Nalepa J., Czech Z.J.: *A parallel heuristic algorithm to solve the vehicle routing problem with time windows.* Studia Informatica, vol. 33, no. 1(104), 2012, 91–106.

[17] Potvin J-Y., Rousseau J-M.: *An exchange heuristic for routing problems with time windows*, Journal of the OR Society 1995;46: 1433–1446.

[18] Potvin J-Y., Rousseau J-M.: *A parallel route building algorithm for the vehicle routing and scheduling problem with time windows*, EJOR 1993;66: 331–340.

[19] Rechenberg I.: *Evolutionsstrategie*, Stuttgart: Fromman-Holzboog, 1973.

[20] Russell RA.: *Hybrid heuristics for the vehicle routing problem with time windows*, Transportation Science 1995;29: 156–166.

[21] Solomon MM.: *Algorithms for the vehicle routing and scheduling problems with time window constraints* Operations Research 1987;35(2):254–265.