STUDIA INFORMATICA 2011
Volume 32 Number 3B (99)

Mirosław BŁOCHO, Zbigniew J. CZECH
Silesian University of Technology, Institute of Informatics

AN IMPROVED ROUTE MINIMIZATION ALGORITHM FOR THE VEHICLE ROUTING PROBLEM WITH TIME WINDOWS

Summary. A route minimization algorithm for the vehicle routing problem with time windows is presented. It was elaborated as an improvement of the algorithm proposed by Nagata and Bräysy (A powerful route minimization heuristic for the vehicle routing problem with time windows, Operations Research Letters 27, 2009, 333-338). By making use of the improved algorithm the two new-best solutions for Gehring and Homberger's (GH) benchmarks were found. The experiments showed that the algorithm constructs the world-best solutions with the minimum route numbers for the GH tests in a short time.

Keywords: vehicle routing problem with time windows, guided local search, heuristics, approximation algorithms

ULEPSZONY ALGORYTM MINIMALIZACJI LICZBY TRAS DLA PROBLEMU TRASOWANIA POJAZDÓW Z OKNAMI CZASOWYMI

Streszczenie. W pracy zaprezentowano algorytm minimalizacji liczby tras dla problemu trasowania pojazdów z oknami czasowymi. Został on opracowany przez ulepszenie algorytmu zaproponowanego przez Nagatę and Bräysy'ego (A powerful route minimization heuristic for the vehicle routing problem with time windows, Operations Research Letters 27, 2009, 333-338). Przy użyciu ulepszonego algorytmu znaleziono dwa nowe najlepsze rozwiązania dla testów wzorcowych Gehringa i Hombergera (GH). W eksperymentach wykazano, że za pomocą ulepszonego algorytmu są konstruowane w krótkim czasie najlepsze światowe rozwiązania testów GH o minimalnej liczbie tras.

Słowa kluczowe: trasowanie pojazdów z przedziałami czasowymi, lokalne poszukiwanie z przewodnikiem, heurystyki, algorytmy aproksymacyjne

1. Introduction

Reducing transportation costs has always been important in all transit companies. Nowadays, not only overestimating the number of required vehicles causes a lot of problems, but also underestimating them. Revaluing routes number brings about ineffective allocation of funds, which is especially essential due to expensive maintenance costs. In turn underestimating them might entail inadequate covering all scheduled services and the prospective loss of the reputation and the customers. Therefore, the proper prediction of the minimum number of vehicles needed to carry out the transportation tasks becomes increasingly indispensable. This problem occurs not only in distribution products from depot to the customers but also in the school bus routing, newspapers and mail delivery, armoured car routing, rail distribution, airline fleet routing, repairmen scheduling and many others.

The vehicle routing problem with time windows (VRPTW) is an extension to the capacitated vehicle routing problem (CVRP) which is formulated in the following manner [1]. There is a central depot of goods and n customers geographically scattered around the depot. The locations of the customers ($i = 1, 2, \ldots, n$) and the depot ($i = 0$), as well as the shortest distances d_{ij} and the corresponding travel times t_{ij} between any two customers i and j (including the depot) are known. Each customer asks for a quantity q_i of goods which has to be delivered by a vehicle of capacity Q. Each vehicle after serving a subset of customers must return to the depot for reloading. Each route starts and terminates at the depot. A solution to the CVRP is a set of routes of minimum travel distance (or travel time) which visits each customer i exactly once. The total demand for each route cannot exceed Q.

The CVRP is extended into the VRPTW by introducing for each customer and the depot a service time window $[e_i, f_i]$ and a service time s_i ($s_0 = 0$). The values e_i and f_i determine the earliest and the latest start time of service, respectively. Each customer i has to be served within the time window $[e_i, f_i]$ and the service of all customers must be accomplished within the time window of the depot $[e_0, f_0]$. The vehicle can arrive to the customer i before the earliest start time of service e_i, but then it has to wait until time, when the service can begin. If the vehicle arrives to the customer i after the latest start time of service f_i, then the solution is not feasible. The routes are travelled simultaneously by a fleet of K homogenous vehicles (i.e. of equal capacity) with each vehicle assigned to a single route. A feasible solution to the VRPTW is the set of routes which guarantees the delivery of goods to all customers and satisfies the time window and the vehicle capacity constraints. The primary objective is to minimize the number of needed vehicles and the secondary objective is to minimize the total travel distance.

Formally, there are three types of decision variables in this two-objective optimization problem. The first decision variable $x_{i,\,j,\,k}$, $i, j \in \{0,1,\ldots,n\}$, $k \in \{1,2,\ldots,K\}$, $i \neq j$, is 1 if

vehicle k travels from customer i to j, and 0 otherwise. The second decision variable, t_i, indicates the time when a vehicle arrives at the customer i, and the third decision variable, b_i, denotes the waiting time at this customer. The objective is to:

$$\text{minimize } K, \text{ and then} \tag{1}$$

$$\text{minimize } \sum_{i=0}^{n}\sum_{j=0}^{n}\sum_{k=1}^{K} d_{i,j}x_{i,j,k} \tag{2}$$

subject to the following constraints:

$$\sum_{k=1}^{K}\sum_{j=1}^{n} x_{i,j,k} = K \text{ , for } i = 0, \tag{3}$$

$$\sum_{j=1}^{n} x_{i,j,k} = \sum_{j=1}^{n} x_{j,i,k} = 1, \quad \text{for } i = 0 \text{ and } k \in \{1,2,...,K\}, \tag{4}$$

$$\sum_{k=1}^{K}\sum_{j=0,j\neq i}^{n} x_{i,j,k} = \sum_{k=1}^{K}\sum_{i=0,i\neq j}^{n} x_{i,j,k} = 1, \quad \text{for } i, j \in \{1,2,...,n\}, \tag{5}$$

$$\sum_{i=1}^{n} q_i \sum_{j=0,j\neq i}^{n} x_{i,j,k} \leq Q, \quad \text{for } k \in \{1,2,...,K\}, \tag{6}$$

$$\sum_{k=1}^{K} q_i \sum_{i=0,\, i\neq j}^{n} x_{i,j,k}(t_i + b_i + h_i + t_{i,j}) \leq t_j, \quad \text{for } j \in \{1,2,...,n\} \text{ and } t_0 = b_0 = h_0 = 0, \tag{7}$$

$$e_i \leq (t_i + b_i) \leq f_i, \quad \text{for } i \in \{1,2,...,n\}. \tag{8}$$

Formulas (1) and (2) identify the minimized functions. Eq. (3) specifies that there are K routes starting at the depot. Eq. (4) denotes that every route starts and terminates at the depot. Eq. (5) assures that every customer is visited only once by a single vehicle. Eq. (6) denotes the capacity constraints. Eqs. (7)-(8) identify the time windows constraints. Eqs. (3)-(8) define the feasible solutions to the VRPTW.

In this paper the improvements of the route minimization algorithm for the VRPTW by Nagata and Bräysy [7] are presented. Similarly to this heuristic the improved algorithm is based on the idea of the ejection pool, originated from the heuristic by Lim and Zhang [5], combined with the guided local searches and the diversification strategy [12]. The powerful insertion procedure which temporarily accepts an infeasible solution supplemented with further attempts to restore the feasibility have been also included in the improved algorithm. Moreover, the additional algorithm modifications which allowed for speeding up the local search strategies have been introduced. The experimental tests provided helpful hints on the algorithm steps needed strong modifications in order to achieve better results. The remainder of this paper is arranged as follows. In Section 2 the algorithm modifications are described. Section 3 contains the discussion of the experimental test results. Section 4 concludes the paper.

2. Route minimization algorithm

2.1. Algorithm description

The first version of the algorithm was implemented based upon the route minimization heuristic for the vehicle routing problem with time windows [7]. The main framework of the heuristic consists of the consecutive route elimination steps performed until the total computation time reaches a specified time. These route elimination steps are included in the *RemoveRoute* function [7], which is repeatedly called during the algorithm execution.

```
function RemoveRoute(φ)
begin
1:    choose a random route and remove it from the solution φ
2:    initialize Ejection Pool (EP) with the customers from the removed
      route
3:    initiate the penalty counters for all customers p[j]:= 1
      (j = 1,…,n)
4:    while EP ≠ ∅ and currTime < maxTime do
5:       select and eject the customer v_ins from EP using LIFO stack
6:       if N_insert(v_ins, φ) ≠ ∅ then
7:          φ:= the new solution φ′ selected randomly from N_insert(v_ins, φ)
8:       else
9:          φ:= Squeeze(v_ins, φ)
10:      end if
11:      if v_ins is not inserted into φ then
12:         p[v_ins] := p[v_ins] + 1 (increase a penalty counter for the
                                      customer v_ins)
13:         select φ′ from N_ej(v_ins, φ) with minimized
            P_sum = p[v_out^(1)]+p[v_out^(2)]+…+p[v_out^(k)]
14:         update φ:= φ′
15:         add the ejected customers: v_out^(1), v_out^(2), … , v_out^(k) to EP
16:         φ:= Perturb(φ)
17:      end if
18:   end while
19:   if EP ≠ ∅ then
20:      restore φ to the beginning solution
21:   end if
22:   return φ
End
```

The *Ejection Pool* (*EP*) is used to hold the unserved customers coming from the removed route (line 2) or from the ejected customers list (line 15) during finding solution with the minimum penalty sum. According to Nagata proposition [7] the ejection pool should be constructed using the *LIFO* (*Last In First Out*) queue in order to prevent the customers which are hard to reinsert from those remaining in the ejection pool. The penalty counters for all customers are initialized each time the function *RemoveRoute* is called. These counters indicate how many times the attempts to insert given customers failed. The bigger value of the penalty counter for a specified customer, the more difficult is to reinsert it into the

solution. After the random route is removed from the current solution, continuous attempts to include the ejected customers in the rest of the routes are performed. These attempts are carried out until all customers from the *Ejection Pool* are inserted or the execution time of the algorithm reaches a specified time limit *maxTime*.

$N_{insert}(v_{ins}, \varphi)$ is constructed as a set of the feasible partial solutions, that are obtained by inserting the customer v_{ins} into all insertion positions in the solution φ. In this set only the insertions between two consecutive nodes are considered. If the constructed $N_{insert}(v_{ins}, \varphi)$ set is empty (line 6) then the function *Squeeze* is called in order to help the insertion of the selected customer into the solution φ.

```
function Squeeze(v_ins ,φ)
begin
1:    φ:= the selected solution φ' from N_insert(v_ins , φ) with min F_p(φ')
2:    while F_p(φ') ≠ 0 do
3:       randomly choose an infeasible route r
4:       select a solution φ' from N_r(φ), such that F_p(φ') is minimum
5:       if F_p(φ') < F_p(φ) then
6:          φ:= φ'
7:       else
8:          break
9:       end if
10:   end while
11:   if F_p(φ) ≠ 0 then
12:      restore φ to the beginning solution
13:   end if
14:   return φ
End
```

The idea of this method is to choose the temporally infeasible insertion with the minimal penalty function value $F_p(\varphi)$ defined as follows:

$$F_p(\varphi) = P_c(\varphi) + \alpha \bullet P_{tw}(\varphi), \tag{9}$$

where

$F_p(\varphi)$ – the solution penalty function,

$P_c(\varphi)$ – the capacity penalty (the sum of total excess of the demands in all routes [7]),

α – the penalty coefficient, which value is adapted iteratively according to the $P_c(\varphi)$ and $P_{tw}(\varphi)$ comparisons,

$P_{tw}(\varphi)$ – the sum of the total time window penalties $P_{tw}(i,\varphi)$, (i = 0,1,..,n) of all customers and the depot in the solution φ [6]*;* $P_{tw}(i,\varphi)$ is defined by:

where

a_{v_i} – the earliest possible start time of service at customer vi and is defined recursively by the following equations:

$$a_{v_0} = e_0 \tag{11}$$

$$a_{v_i} = \max \{ a_{v_{i-1}} + s_{v_{i-1}} + c_{v_{i-1}v_i}, e_{v_i} \}, \; i = 1, 2, \ldots, n+1. \tag{12}$$

After that the local search moves are performed in order to restore the feasibility of the solution. In the *Squeeze* function no customer ejections are allowed.

If the *Squeeze* function fails then it informs that the selected customer v_{ins} was not inserted into the solution φ. The value of the penalty counter for a chosen customer must be then increased and after that the ejections of the customers are tested. In these ejections the limit k_m for the number of removed customers was introduced [7]. Before searching there is need to construct the $N_{ej}(v_{ins}, \varphi)$ set which contains the possible solutions with the combinations of inserted customers at different route positions and various ejected customers. Only the solutions with the minimum penalty sum value (line 13) are taken into account during the local searches in the *Perturb* function. Choosing the solution with the minimized penalty counters sum value gives the largest probability [7] of finding the feasible solution after the local search moves.

2.2. Algorithm improvements

In the *RemoveRoute* function the *Ejection Pool* is initialized with the customers from the randomly removed route (line 2). It was observed that if this function fails to insert the customers from the removed route and fails to insert other customers from the removed routes in the next *RemoveRoute* calls, then after some steps it generates again the same solution which was previously considered. In such a case the customers are inserted into the ejection pool exactly in the same order which may lead to a similar execution of the algorithm as in the previous phase. For that reason it was proposed to insert the customers from the removed route into the ejection pool always in a random order to obtain the diversification of the search. This situation may happen very frequently while checking the ejections for the solutions with the route number close to the minimum number of routes for a given test case.

In the original algorithm [7] the attempts to insert the ejected customers into the solution are carried out until all customers are inserted or the execution time of the algorithm reaches a specified time limit *maxTime*. While analyzing the algorithm we observed that in some cases always the same customers were ejected and the algorithm got stuck inside the while loop (lines 4 to 18 in *RemoveRoute*) no matter how large were the values of the penalty counters for those customers. Obviously in the original algorithm it is forbidden to eject just inserted customer v_{ins} , what is especially important in the case of ejecting more than two customers. Based on these observations we suggest the following improvements in the algorithm:

- the ejecting customers which have been inserted into the solutions during the last l_c loop executions should be forbidden (recommended, experimentally established value of $l_c = 4, 5$)

- the attempts of inserting the customers from the ejection pool should be given up (together with the original break conditions), if the loop count exceeds the specified limit $p_{maxTrials}$ (recommended $p_{maxTrials} = 1000$).

These algorithm improvements are vital while removing the routes from the solutions with the routes count very close to the number of routes of the best known solutions.

During the process of continuous insertions and ejections of the customers in the *RemoveRoute* function, the ejection pool size is usually alternatingly increased and decreased. During experimental tests we observed that in many cases, if the ejection pool size exceeds the certain limit, then it is quite difficult to insert the customers from this pool again into the solution. Therefore we propose the additional limit for the maximum ejection pool size EPs_{max}. This size should be calculated each time the random route is removed from the solution. The recommended formula for the EPs_{max} is defined as follows:

$$EPs_{max} = rr_s + rr_x \tag{13}$$

where

rr_s – the removed route size,

rr_x – the coefficient which denotes the number of customers which may reside in the ejection pool together with the customers from the removed route.

If the *Squeeze* function fails to insert the selected customer into the solution, then the customers' ejections in order to find the feasible solution are tested. It was observed that ejecting more than one customer is useful only if the routes count of the current solution is very close to the minimum number of routes for a given solution. If k (*RemoveRoute* function, line 13) is set to 3, 4 or more, then testing the ejections of routes takes too much time. In many cases setting $k = 1$ is sufficient and allows for finding the feasible solutions in much shorter time. Therefore we suggest an improvement to test first the ejections with $k = 1$, if they fail then test the ejections with $k = 2$ and so on, up to a specified maximum limit k_{max}.

The experiments indicated that constructing $N_{insert}(v_{ins}, \varphi)$ in the *RemoveRoute* function takes much more time than expected. The set $N_{insert}(v_{ins}, \varphi)$ has to contain only the feasible partial solutions. Therefore there is no need to construct a new solution object for each test case while checking whether the partial solution is feasible or not. We suggest that it can be checked in advance whether the potential solution (after inserting a new customer) will remain feasible or not. For all tested customer insertions, the positions which give the feasible solutions are recorded on a list. Then a random item from this list is chosen and a new solution object is created together with all customers' updates of the earliest arrival times, the latest start times of service, etc. In order to check whether the solution will be feasible after the customer insertion the forward and backward time window penalty slacks described in [8] was implemented. Introducing this feature allowed for calculation the change in a time

window penalty (in case of the feasible solutions this change is equal to zero) in constant time what decreases the time for constructing the $N_{insert}(v_{ins}, \varphi)$ set.

In a similar manner the implementation of finding the solution with the minimum penalty value $F_p(\varphi')$ in the *Squeeze* function (line 1) was modified. Only one solution is required and since the change in the time window penalty may be calculated in a constant time, the best solution may be found very fast. This feature was implemented using the *2Opt**, *OutRelocate* and *Exchange* local search moves [8] for the *Squeeze* as well as in the *Perturb* function for the same operators but with accepting only the feasible solutions. If more than one solution with a minimum penalty is found, it is suggested to record all solutions with the current minimum penalty (during the consecutive customer insertions while constructing $N_{insert}(v_{ins}, \varphi)$ set) and then to choose randomly one of them.

The experimental tests of finding the solution from $N_{ej}(v_{ins}, \varphi)$ with the minimized penalty sum value of ejected customers indicated that there is need to implement the changes similar to the changes in construction of $N_{insert}(v_{ins}, \varphi)$ also in this case. The solutions constructed after inserting the customer v_{ins} and ejecting the particular customers must be partially feasible. It is proposed to test first the backward and forward time window penalty slacks as well as the changes in the route demands. This helps to check very quickly whether the changed solution is feasible or not.

3. Analysis of the experimental result

The improved algorithm was implemented in C++ and tested with the following settings:

$maxTime = 240$ – the maximum time (in seconds) for a single test,
$\alpha = 1$ – the initial penalty coefficient,
$l_c = 5$ – the number of customers inserted during the last k_{lc} loop executions in which the customers are not allowed to be ejected,
$p_{maxTrials} = 1000$ – the loops count limit in the *RemoveRoute* function,
$x = 7$ – the coefficient for the maximum ejection pool size,
$k_{max} = 4$ – the maximum ejected customers count,
$I_{rand} = 400$ – the *Perturb* function execution count,
$np_{Sq} = 60$ – the percent of close customers for the *Squeeze* function.

The algorithm was executed on an Intel Core 2 Duo 2.4 GHz (2 GB RAM) processor. The Figures 1-5 show the Gehring and Homberger's test results for 200, 400, 600, 800 and 1000 customers for all problem instances. The achieved test results are compared with the world results based on the cumulative number of the vehicles (CVN).

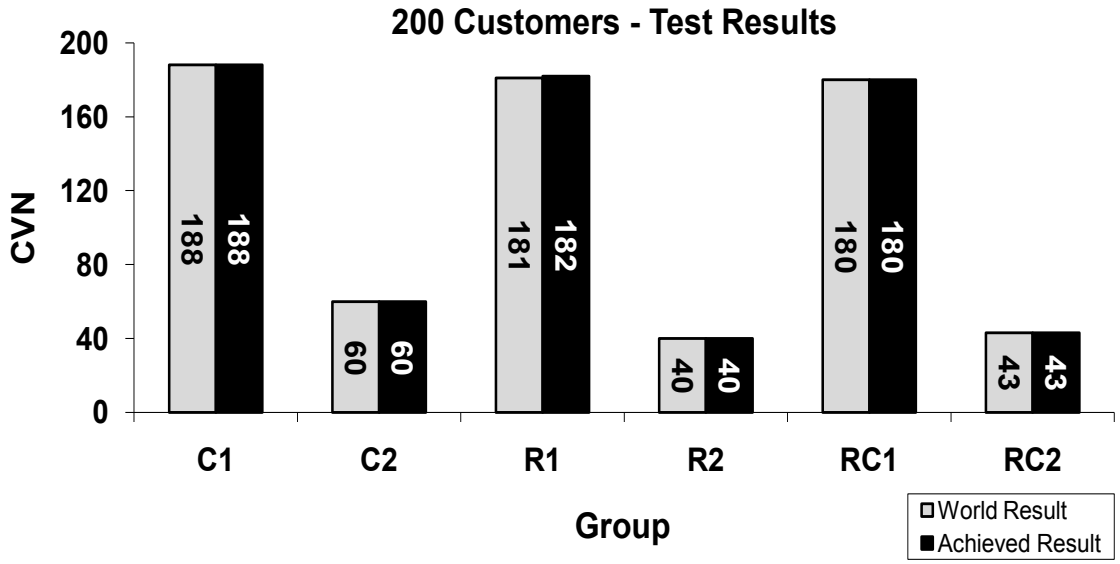

Fig. 1. Test Results – 200 Customers
Rys. 1. Wyniki – 200 Klientów

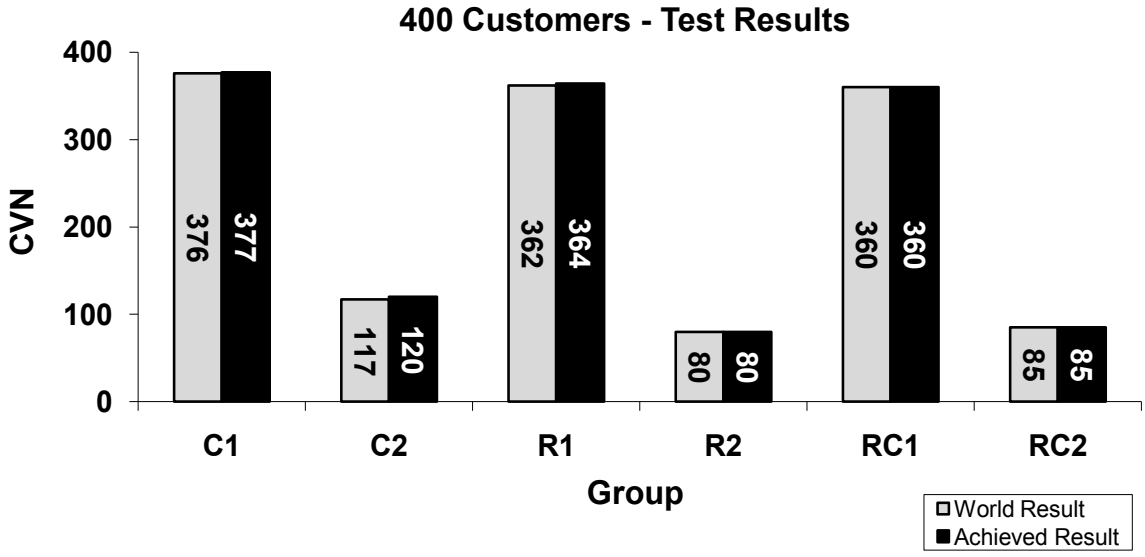

Fig. 2. Test Results – 400 Customers
Rys. 2. Wyniki – 400 Klientów

In Table 1 the results obtained with the improved algorithm (MBL) are compared with the results known from the literature:

- GH (Gehring and Homberger [2])
- IBA (Ibaraki et al. [3])
- LZ (Lim and Zhang [5])
- GD (Gagnon and Desaulniers [10])
- PR (Pisinger and Ropke [9])

The results are compared based on the number of routes only. The route minimization step is carried out independently in all those algorithms, so the comparison is well-founded.

The CVNs, computer specifications and average CPU time together with the number of runs are listed.

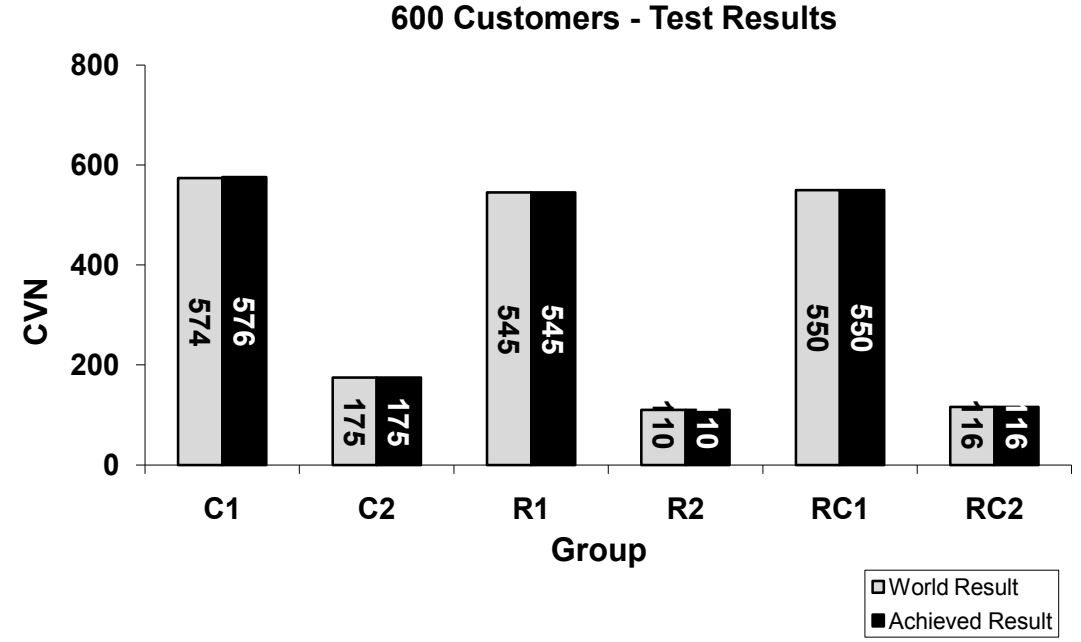

Fig. 3. Test Results – 600 Customers
Rys. 3. Wyniki – 600 Klientów

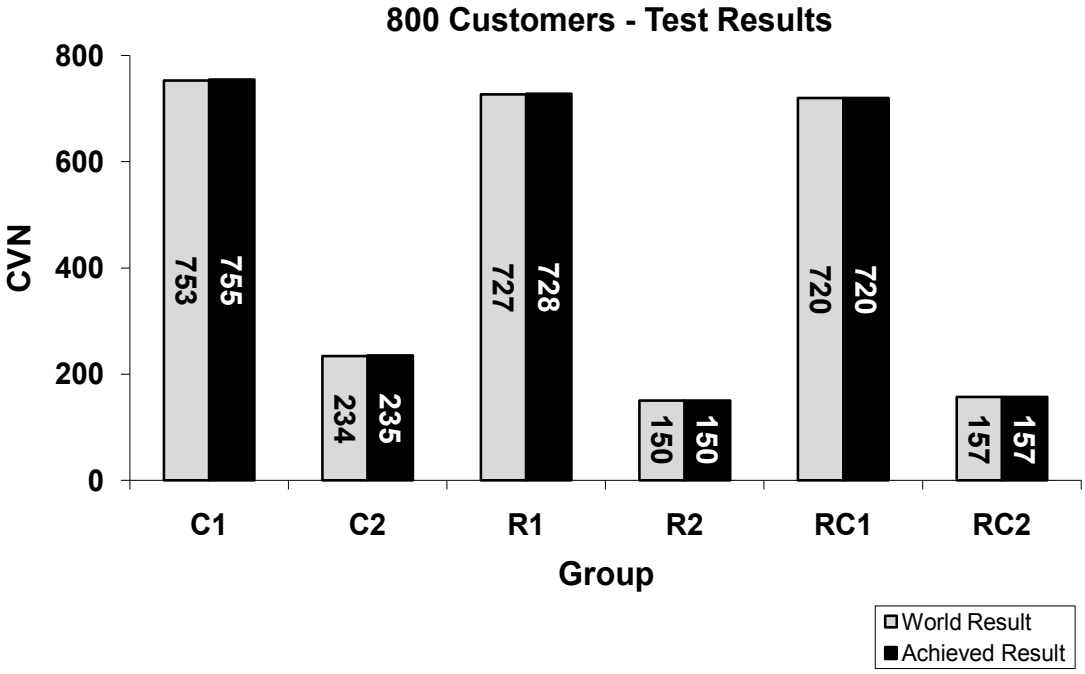

Fig. 4. Test Results – 800 Customers
Rys. 4. Wyniki – 800 Klientów

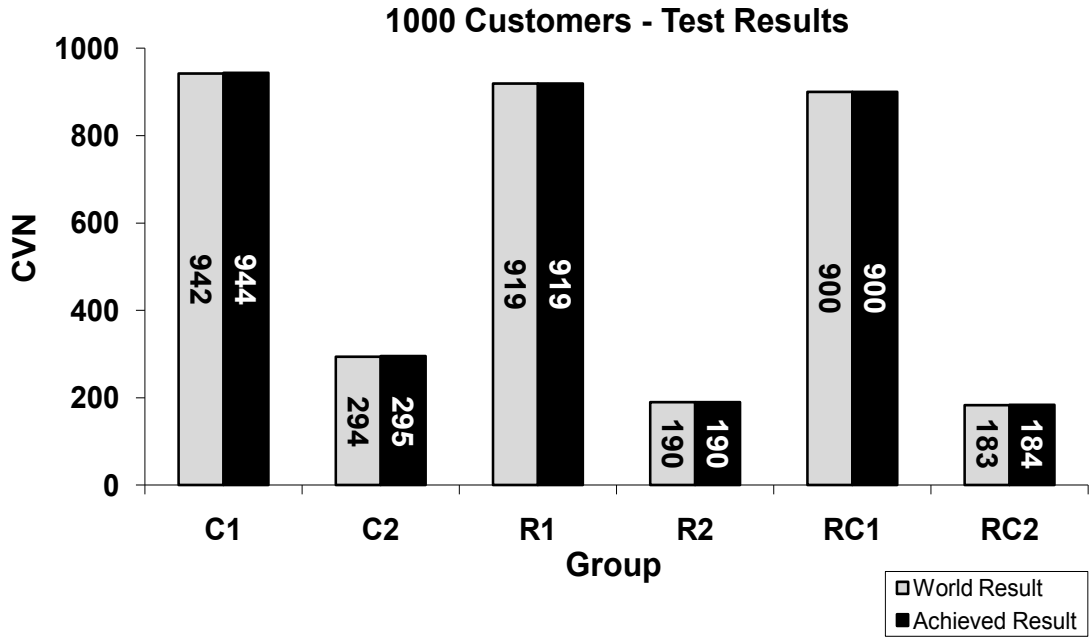

Fig. 5. Test Results – 1000 Customers
Rys. 5. Wyniki – 1000 Klientów

The obtained results prove the efficiency and powerfulness of the improved algorithm. The maximum execution time of the algorithm was set to 4 minutes and within this limit the algorithm found the world results of the minimum route number in 96% test cases (Table 3). The results in Table 1 compared to results of other well-known algorithms show the similar, competitive cumulative number of the vehicles with a much smaller amount of time needed to obtain those results, especially for 400, 600 and 1000 customers. Moreover the real average times needed to obtain those results are gathered in Table 2 and show that in most cases the 4 minutes limit was not necessary. The real average time for all 300 Gehring and Homberger's tests was only 25 seconds.

Furthermore, during the experiments the two new world best results were found – for test ***RC2_10_1*** (1000 customers) the solution with **20 routes** and for test ***C1_8_2*** (800 customers) the solution with **73 routes**. These results have been published on the *Sintef* website: http://www.sintef.no/Projectweb/TOP/Problems/VRPTW/Homberger-benchmark/.

Table 1

The results for all problem sizes – compared to other algorithms GH, IBA, LZ, GD, PR

200 customers	GH	IBA	LZ	GD	PR	MBL
C1	189	189	189	189	189	**188**
C2	60	60	60	60	60	**60**
R1	182	182	182	182	182	**182**
R2	40	40	40	40	40	**40**
RC1	181	180	180	180	180	**180**
RC2	44	43	43	43	43	**43**
Total CVN	696	694	694	694	694	**693**
CPU	P 400M	P 2.8G	P 2.8G	O 2.3G	P 3.0G	P 2.4G
(min.) x runs	8.4 x 3	N/A	10 x 2	53 x 5	7.7 x 10	4 x 1
400 customers	GH	IBA	LZ	GD	PR	MBL
C1	380	377	376	376	376	377
C2	120	120	117	119	120	120
R1	364	364	364	364	364	**364**
R2	80	80	80	80	80	**80**
RC1	361	360	360	360	360	**360**
RC2	88	86	85	86	85	**85**
Total CVN	1392	1387	1382	1385	1385	1386
CPU	P 400M	P 2.8G	P 2.8G	O 2.3G	P 3.0G	P 2.4G
(min.) x runs	28.4 x 3	N/A	20 x4	89 x 5	15.8 x 5	4 x 1
600 customers	GH	IBA	LZ	GD	PR	MBL
C1	577	575	574	574	575	576
C2	178	174	174	175	175	175
R1	545	545	545	545	545	**545**
R2	110	110	110	110	110	**110**
RC1	550	550	550	550	550	**550**
RC2	119	116	115	117	116	116
Total CVN	2079	2070	2068	2071	2071	2072
CPU	P 400M	P 2.8G	P 2.8G	O 2.3G	P 3.0G	P 2.4G
(min.) x runs	51.6 x 3	N/A	30 x 6	105 x 5	18.3 x 5	4 x 1
800 customers	GH	IBA	LZ	GD	PR	MBL
C1	761	757	754	754	756	755
C2	237	234	234	235	237	235
R1	728	728	728	728	728	**728**
R2	150	150	150	150	150	**150**
RC1	723	724	720	720	730	**720**
RC2	161	157	156	158	157	157
Total CVN	2760	2750	2742	2745	2758	2745
CPU	P 400M	P 2.8G	P 2.8G	O 2.3G	P 3.0G	P 2.4G
(min.) x runs	92.8 x 3	N/A	40 x 8	129 x 5	22.7 x 5	4 x 1
1000 customers	GH	IBA	LZ	GD	PR	MBL
C1	954	945	944	943	946	944
C2	297	294	293	295	297	295
R1	919	919	919	919	922	**919**
R2	190	190	190	190	190	**190**
RC1	901	900	900	900	900	**900**
RC2	185	183	183	185	183	184
Total CVN	3446	3431	3429	3432	3438	3432
CPU	P 400M	P 2.8G	P 2.8G	O 2.3G	P 3.0G	P 2.4G
(min.) x runs	120.4 x 3	N/A	50 x 10	162 x 5	26.2 x 5	4 x 1

Table 2

Comparison of average times needed to find test solutions

	200	400	600	800	1000	Avg
C1	< 1 s	11 s	16 s	56 s	95 s	36 s
C2	< 1 s	2 s	37 s	115 s	130 s	57 s
R1	< 1 s	< 1 s	2 s	< 1 s	1 s	1 s
R2	1 s	1 s	1 s	2 s	2 s	1 s
RC1	< 1 s	1 s	1 s	29 s	1 s	6 s
RC2	1 s	38 s	66 s	73 s	73 s	50 s
Avg	1 s	9 s	21 s	46 s	50 s	**25 s**

Table 3

Compared percentage of found solutions with minimal world-best number of routes

	200	400	600	800	1000	Avg
C1	100 %	90 % (9/10)	80 % (8/10)	80 % (8/10)	80 % (8/10)	86 % (43/50)
C2	100 %	70 % (7/10)	100 %	90 % (9/10)	90 % (9/10)	90 % (45/50)
R1	90 % (9/10)	90 % (9/10)	100 %	90 % (9/10)	100 %	94 % (47/50)
R2	100 %	100 %	100 %	100 %	100 %	100 %
RC1	100 %	100 %	100 %	100 %	100 %	100 %
RC2	100 %	100 %	100 %	100 %	90 % (9/10)	98 % (49/50)
Avg	98 % (59/60)	92 % (55/60)	97 % (58/60)	93 % (56/60)	93 % (56/60)	**96 % (288/300)**

4. Conclusions

The improved algorithm proved very competitive with respect to other well-known heuristics solving the VRPTW. The main advantage of the proposed algorithm is a short time of obtaining the solutions which contain the number of routes which are equal to or are slightly worse than the best known solutions. Therefore the improved algorithm can be used as the first stage in other heuristic algorithms for minimization of the routes count.

It would be interesting to examine if the parallelization of the improved route minimization algorithm could give the better results while solving the VRPTW. Our further research concentrates on this topic and the results are very promising as the parallel heuristic has been able to discover new best solutions to Gehring and Homberger's benchmarks.

BIBLIOGRAPHY

1. Czech Z. J.: A Parallel Simulated Annealing Algorithm as a Tool for Fitness landscapes Exploration. Parallel and Distributed Computing, InTech, 2010, chapter 13, p. 247÷271.
2. Gehring H., Homberger J.: Parallelization of a two-phase metaheuristic for routing problems with time windows. Asia-Pacific Journal of Operational Research 18, 2001, p. 35÷47.
3. Ibaraki T., Imahori S., Nonobe K., Sobue K., Uno T., Yagiura M.: An iterated local search algorithm for the vehicle routing problem with convex time penalty functions. Discrete Applied Mathematics 156, 2008, p. 2050÷2069.
4. Lenstra, J., and Rinnooy Kan, A.: Complexity of vehicle routing and scheduling problems. Networks 11, 1981, p. 221÷227.
5. Lim A., Zhang X.: A two-stage heuristic with ejection pools and generalized ejection chains for the vehicle routing problem with time windows. Informs Journal on Computing 19, 2007, p. 443÷457.
6. Nagata Y.: Efficient evolutionary algorithm for the vehicle routing problem with time windows: Edge assembly crossover for the VRPTW. Proc. of the 2007 Congress on Evolutionary Computation, 2007, p. 1175÷1182.
7. Nagata Y, Bräysy O.: A Powerful Route Minimization Heuristic for the Vehicle Routing Problem with Time Windows. Operations Research Letters 37, 2009, p. 333÷338.
8. Nagata Y, Bräysy O., Dullaert W. (2010) A Penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows. Computers & Operations Research 37, 2010, p. 724÷737.
9. Pisinger D., Ropke S.: A general heuristic for vehicle routing problems. Computers & Operations Research 34, 2007, p. 2403÷2435.
10. Prescott-Gagnon E., Desaulniers G., Rousseau L.-M.: A branch-and-price-based large neighborhood search algorithm for the vehicle routing problem with time windows. Working Paper, University of Montreal, Canada, 2007.
11. Toth, P., and Vigo, D., (Eds.): The vehicle routing problem. SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, PA, 2002.
12. Voudouris C., Tsang E.: Guided local search. Handbook of Metaheuristics, Kluwer, 2003, p. 185÷218.

Recenzent: Dr hab. Urszula Boryczka

Wpłynęło do Redakcji 16 grudnia 2010 r.

Omówienie

W pracy został przedstawiony ulepszony algorytm minimalizacji liczby tras dla problemu trasowania pojazdów z oknami czasowymi. Algorytm ten jest oparty na zmodyfikowanej heurystyce zaproponowanej przez Nagatę i Bräysy'ego [7]. Idea algorytmu polega w pierwszej fazie na usuwaniu klientów z losowo wybranej trasy w celu zminimalizowania liczby tras w rozwiązaniu oraz na wieloetapowych próbach wstawiania usuniętych klientów do tworzonego rozwiązania w drugiej fazie. Próby wstawiania usuniętych klientów są przeprowadzane z zastosowaniem puli usuniętych klientów (ang. *ejection pool*) wraz z kierowanymi lokalnymi poszukiwaniami (ang. *guided local searches*) oraz dywersyfikacją uzyskanych rozwiązań (ang. *solution diversification*). W algorytmie jest istotna tymczasowa akceptacja rozwiązań z naruszonym ograniczeniem maksymalnej ładowności pojazdu lub z naruszonym ograniczeniem okien czasowych poszczególnych klientów.

Ulepszony algorytm powstał na podstawie testów eksperymentalnych zaimplementowanej heurystyki podanej w pracy [7]. Do najważniejszych modyfikacji algorytmu należą:

- wstawianie klientów w kolejności losowej do puli klientów usuniętych,
- zabronione usuwanie klientów wstawionych do aktualnego rozwiązania podczas określonej liczby ostatnich wykonań głównej iteracji algorytmu,
- dodatkowy limit na maksymalną pojemność puli usuniętych klientów,
- ulepszona funkcja sprawdzania z wyprzedzeniem, czy potencjalne zmiany w rozwiązaniu pozwolą na uzyskanie żądanego rezultatu.

Przeprowadzone testy ulepszonego algorytmu dla testów wzorcowych Gehringa i Hombergera dowiodły jego wysokiej konkurencyjności w stosunku do innych dobrze znanych algorytmów rozwiązywania problemu VRPTW [2, 3, 5, 9, 10]. Zdecydowaną przewagą ulepszonego algorytmu jest krótki czas znajdowania rozwiązań z liczbą tras równą lub niewiele większą od światowych wyników dla testów wzorcowych. W 96% przypadków testowych udało się uzyskać liczbę tras równą znanym światowym wynikom przy rzeczywistym średnim czasie działania 25 sekund na jeden test. Dla dwóch przypadków testowych (RC2_10_1 oraz C1_8_2) udało się uzyskać rozwiązania z liczbą tras mniejszą od światowych wyników.

Adresses

Mirosław BŁOCHO: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-100 Gliwice, Polska, blochom@gmail.com
Zbigniew J. CZECH: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-100 Gliwice, Polska, zbigniew.czech@polsl.pl