

A powerful route minimization heuristic for the vehicle routing problem with time windows

Yuichi Nagata^a, Olli Bräysy^b

^a Interdisciplinary Graduate School of Science and Engineering, Tokyo Institute of Technology, 4259 Nagatsuta Midori-ku Yokohama, Kanagawa 226-8502, Japan

^b Agora Innoroad Laboratory, Agora Center, P.O. Box 35, FI-40014 University of Jyväskylä, Finland

article info

Article history:

Received 11 February 2008

Accepted 26 April 2009

Available online 24 May 2009

Keywords:

Vehicle routing

Heuristics

Time windows

Guided local search

abstract

We suggest an efficient route minimization heuristic for the vehicle routing problem with time windows. The heuristic is based on the ejection pool, powerful insertion and guided local search strategies. Experimental results on the Gehring and Homberger's benchmarks demonstrate that our algorithm outperforms previous approaches and found 18 new best-known solutions.

© 2009 Elsevier B.V. All rights reserved.

1. Introduction

The vehicle routing problem with time windows (VRPTW) is one of the most important and widely studied problems in the operations research. For recent literature, see e.g. [1–3]. The VRPTW can be described as the problem of designing least cost routes from a single depot to a set of geographically scattered customers. The routes must be designed in such a way that each customer is visited only once by exactly one vehicle, all routes start and end at the depot, the total demands of all customers on any route must not exceed the capacity of the vehicle (capacity constraint), and each customer must be serviced with a given time interval (time window constraint). The primary objective of the VRPTW is to find the minimum number of routes. Often a secondary objective is imposed to minimize the total distance traveled or to minimize the total schedule time.

Given the hierarchical objective, most of the recent and best heuristics for the VRPTW use a two-stage approach where the number of routes is minimized in the first stage and the total distance is then minimized in the second stage [4,5,2,3,6–8]. Minimizing the number of routes is also important by itself as it provides a basis for a more long-term evaluation of costs, service areas and fleet size. It has also been shown that minimizing the number of routes is often the most time consuming and challenging part of solving VRPTWs [1,2]. Therefore, there is a clear need

for developing more efficient and generally applicable procedures for minimizing the number of routes in VRPTWs.

In this paper we suggest an efficient heuristic method for reducing the number of routes in VRPTWs. The suggested method is based on the idea of the ejection pool [6] that is combined here with a concept reminiscent of the guided local search [9,10] to guide the ejections. Moreover, we incorporate a powerful insertion procedure that accepts temporal infeasible insertions, followed by an attempt to restore the feasibility. The suggested method was tested on the 300 well-known large-scale benchmark problems of Gehring and Homberger [11]. The computational results demonstrate that the proposed method outperforms the best heuristics that have been applied to these benchmarks in terms of the number of routes. It found all best-known and 18 new best-known solutions.

The remainder of this paper is arranged as follows. First, the notations are shortly described in Section 2. Section 3 describes the main framework and details of the powerful insertion and ejection procedures. The experimental setting and results are given in Section 4.

2. Notations

Let node f_0 be the depot and nodes f_1, \dots, f_N be the set of customers. Each node v has a time window $[e_v, l_v]$, a demand q_v ($q_0 = 0$) and a service duration s_v ($s_0 = 0$). Let $c_{v,v'}$ be the travel time from v to v' . The vehicle capacity is denoted by Q .

For each route consisting of n customers, let $h = v_0, v_1, \dots, v_n, v_{n+1}$ be a sequence of the customers in the route where v_0 and v_{n+1} represent the depot. The route satisfies the capacity constraint

Corresponding author.

E-mail address: nagata@fe.dis.titech.ac.jp (Y. Nagata).

Procedure DELETEROUTE(σ)**begin**

```

1 : Select and remove a route randomly from  $\sigma$ ;
2 : Initialize  $EP$  with the customers in the removed route;
3 : Initialize all penalty counters  $p[v] := 1$  ( $v = 1, \dots, N$ );
4 : while  $EP \neq \emptyset$  and  $time < maxTime$  do
5 :   Select and remove customer  $v_{in}$  from  $EP$  with the LIFO strategy;
6 :   if  $\mathcal{N}_{insert}^{fe}(v_{in}, \sigma) \neq \emptyset$  then
7 :     Select  $\sigma' \in \mathcal{N}_{insert}^{fe}(v_{in}, \sigma)$  randomly; Update  $\sigma := \sigma'$ ;
8 :   else
9 :      $\sigma := SQUEEZE(v_{in}, \sigma)$ ;
10 :   endif
11 :   if  $v_{in}$  is not included in  $\sigma$  then
12 :     Set  $p[v_{in}] := p[v_{in}] + 1$ ;
13 :     Select  $\sigma' \in \mathcal{N}_{Ej}^{fe}(v_{in}, \sigma)$  such that  $P_{sum} = p[v_{out}^{(1)}] + \dots + p[v_{out}^{(k)}]$ 
        is minimized; Update  $\sigma := \sigma'$ ;
14 :     Add ejected customers  $\{v_{out}^{(1)}, \dots, v_{out}^{(k)}\}$  to  $EP$ ;
15 :      $\sigma := PERTURB(\sigma)$ ;
16 :   endif
17 : endwhile
18 : if  $EP \neq \emptyset$  then Restore  $\sigma$  to the input state;
19 : return  $\sigma$ ;
end

```

Fig. 1. The algorithm for minimizing the number of routes.

if $\sum_{i \in D} q_{v_i} \leq Q$. The departure time of the depot, a_{v_0} , the earliest possible starting time of customer v_i , a_{v_i} , $i \in D \setminus \{1, \dots, n\}$, and the earliest arrival time to the depot, $a_{v_{n+1}}$, are defined recursively as shown in Eq. (1). The route satisfies the time window constraint if $a_{v_i} \leq l_{v_i}$, $i \in D \setminus \{1, \dots, n\}$.

$$\begin{aligned} a_{v_0} &= e_0 \\ a_{v_i} &= \max\{a_{v_{i-1}} + c_{v_{i-1}v_i}, e_{v_i}\} \quad i \in D \setminus \{1, \dots, n\} \end{aligned} \quad (1)$$

A route is called feasible if it satisfies both the capacity and time window constraints, and infeasible otherwise. A feasible solution consists of a set of feasible routes that pass through all customers exactly once. In addition, we define a *partial* solution as a set of routes that pass through a subset of all customers exactly once. A partial solution is called feasible if it consists of feasible routes, and infeasible otherwise.

3. The route elimination algorithm**3.1. Main framework**

The suggested route elimination algorithm starts with an initial solution where each customer is served individually by a separate route. Procedure DeleteRoute(.) is then repeatedly applied to an incumbent solution to reduce the number of routes one by one until the total computation time reaches a given limit. Fig. 1 depicts the algorithm of procedure DeleteRoute(.)

The procedure is started by selecting and removing a route randomly from the current feasible solution (line 1). Thus, σ is a feasible partial solution. The set of the resulting unserved customers is used to initialize the ejection pool (EP) (line 2). The main idea of the ejection pool [6] is to always hold the set of unserved (ejected) customers, currently missing from σ .

In each iteration (lines 5–15), attempts are made to insert customer v_{in} , selected from EP , into σ without violating the capacity and time window constraints. At first only straightforward insertions (inserting a customer between two consecutive nodes) are considered. Let $\mathcal{N}_{insert}^{fe}(v_{in}, \sigma)$ be the set of feasible partial solutions that are obtained by inserting v_{in} into all insertion positions in σ . If

there is a feasible insertion position, the next solution is randomly selected from $\mathcal{N}_{insert}^{fe}(v_{in}, \sigma)$ (lines 6–7). Otherwise, a more sophisticated insertion method, procedure Squeeze(v_{in}, σ), is applied (lines 8–9). Here the main idea is to accept a temporally infeasible insertion, followed by a series of local search moves to restore the feasibility of the partial solution without removing any customer from σ . For more details, see Section 3.2.

If Squeeze(v_{in}, σ) also fails to insert v_{in} into σ , we allow removals (ejections) of customers from a route in σ up to a given maximum limit, denoted by k_{max} , in order to insert v_{in} . Let $\mathcal{N}_{Ej}^{fe}(v_{in}, \sigma)$ be the set of feasible partial solutions that are obtained by inserting v_{in} into all insertion positions in σ and, for each insertion, ejecting at most k_{max} customers, $v_{out}^{(1)}, \dots, v_{out}^{(k)}$ ($k \leq k_{max}$), from the resulting infeasible route in all possible ways. Note that here also v_{in} itself can be ejected. The next solution is selected from $\mathcal{N}_{Ej}^{fe}(v_{in}, \sigma)$ according to a new innovative criterion based on a concept reminiscent of the guided local search to diversify the ejections. In the beginning all penalty counters $p[v]$, $v \in D \setminus \{1, \dots, n\}$, are initialized (line 3). The penalty counter denotes how many times an attempt to insert (and squeeze) a customer v has failed (line 12). In other words, the value of $p[v]$ gives an estimate how difficult it is to re-insert a customer v into σ . The next partial solution (insertion ejection combination) is chosen so that the sum of the penalty counters of the ejected customers, $P_{sum} = p[v_{out}^{(1)}] + \dots + p[v_{out}^{(k)}]$, is minimized (line 13). This criterion is motivated by the expectation that ejecting the customers whose sum of the penalty counters is the smallest possible will benefit the subsequent insertions even if the number of the customers in the ejection pool is increased (i.e., $k \geq 2$). However, the number of all possible ejections for each insertion may be very large and it will therefore be impractical to test all ejections. In Section 3.3, we present an efficient search limitation strategy for finding the best insertion–ejection combination.

Another important feature of our algorithm is the selection sequence of v_{in} . Our algorithm removes and adds customers into the ejection pool with the last-in first-out (LIFO) strategy (line 5, 14). This strategy prevents the customers that are hard to insert from remaining in the ejection pool.

Procedure SQUEEZE(v_{in}, σ)**begin**

```

1 : Search  $\sigma' \in \mathcal{N}_{insert}(v_{in}, \sigma)$  such that  $F_p(\sigma')$  is minimum; Update  $\sigma := \sigma'$ ;
2 : while  $F_p(\sigma) \neq 0$  do
3 :   Randomly select an infeasible route  $r$ ;
4 :   Search  $\sigma' \in \mathcal{N}_r(\sigma)$  such that  $F_p(\sigma')$  is minimum;
5 :   if  $F_p(\sigma') < F_p(\sigma)$  then Update  $\sigma := \sigma'$ ;
6 :   else break;
7 : endwhile
8 : if  $F_p(\sigma) \neq 0$  then Restore  $\sigma$  to the input state;
9 : return  $\sigma$ ;
end

```

Fig. 2. Algorithm of the squeeze procedure.



Fig. 3. An example of the lexicographic ejections. Customer nodes indicated by gray circles are ejected. If $k_{max} = 3$, customer nodes are ejected in the following order: $f1g; f1; 2g; f1; 2; 3g; f1; 2; 4g; \dots; f1; 2; ng; f1; 3g; f1; 3; 4g; \dots; f1; n; 1; ng; f2g; \dots$. For example, when customers $f5; 6; 9g$ are ejected, we gain $k = 3$, $i.k/D = 9$, $i.k - 1/D = 6$ and $i.k - 2/D = 5$.

Each time after one or more customers have been ejected from σ , the resulting feasible partial solution σ is modified by procedure **Perturb** σ to diversify the search (line 15). Here random local search moves are executed inside the partial solution for a given number of times I_{rand} ($=1000$), without violating the feasibility. For more details, see Section 3.2.

3.2. Squeeze and perturb procedures

Procedure **Squeeze** $(v_{in}; \sigma)$ can be considered as an improved insertion heuristic. In the beginning customer v_{in} is inserted into feasible partial solution σ by allowing the violation of the capacity and/or time window constraints. Then an attempt is made to restore the feasibility by applying local search without ejecting any customers. The local search is based on the well-known move operators, 2-opt* [12], intra- and inter-route relocation, and intra- and inter-route exchange [13]. We define a composite neighborhood, denoted by \mathcal{N} , as the sets of the partial solutions that are obtained by applying these operators to σ where the moves are executed inside the partial solution. To reduce the size of \mathcal{N} , only moves involving geographically close customers are considered (e.g., a customer node must be relocated between two consecutive nodes, at least one of which must be among the 100 closest nodes from the relocated one).

The infeasible partial solution σ is evaluated with a penalty function $F_p(\sigma)$ defined in Eq. (2). $P_c(\sigma)$ and $P_{tw}(\sigma)$ are the penalty terms for the violations of the capacity and time window constraints, respectively, and α is a penalty coefficient whose value is adapted iteratively during the search process.

$$F_p(\sigma) = \alpha P_c(\sigma) + P_{tw}(\sigma) \quad (2)$$

$P_c(\sigma)$ is straightforwardly defined as the sum of the total excess of the demands in all routes. For the $P_{tw}(\sigma)$ penalty term, we use the definition proposed by Nagata [7].

Given a route r , the time window constraint is violated at a customer or the depot v in case $t_v < a_v$ (See Eq. (1)). In this case, we assume that the vehicle can travel back in time t_v by paying a penalty $\alpha_v \cdot t_v$ and can arrive at v without delay. Then, the vehicle continues the service (a_v is set to t_v). The total of time window violation in the route, denoted by TW_r , is defined by the sum of the penalties that the vehicle must pay to service the customers and

to arrive at the depot without delay. Note that the vehicle may pay the penalty more than once in the route. Finally, $P_{tw}(\sigma)$ is defined as $P_{tw}(\sigma) = \sum_{r \in \mathcal{R}} TW_r$, where m is the number of routes in σ . The change in the $P_{tw}(\sigma)$ can be calculated in $O(1)$ time for any move in \mathcal{N} , except for intra-route relocation. For more details, we refer to [7].

Fig. 2 describes the algorithm of procedure **Squeeze** $(v_{in}; \sigma)$. The procedure begins with inserting v_{in} into the best insertion position in σ that minimizes the value of $F_p(\sigma)$ (line 1). The algorithm continues by randomly selecting an infeasible route r (line 3). Then, the move that decreases the value of $F_p(\sigma)$ most is executed within a neighborhood $\mathcal{N}_r(\sigma)$ (lines 4–5). Here, $\mathcal{N}_r(\sigma)$ is the subset of $\mathcal{N}(\sigma)$ that affects route r (i.e., route r must be changed by the moves). If the value of $F_p(\sigma)$ is not decreased, the procedure is terminated (line 6) and it returns the input state of σ (lines 8–9).

In our algorithm, the parameter α is adapted with an adaption mechanism. In the beginning of the search, α is set to 1.0. Then, α is changed each time procedure **Squeeze** $(v_{in}; \sigma)$ fails to find a feasible insertion. If $P_c(\sigma) < P_{tw}(\sigma)$, α is divided by 0.99 to emphasize the penalty term of the time window constraint. On the other hand, if $P_c(\sigma) > P_{tw}(\sigma)$, α is multiplied by 0.99.

Procedure **Perturb** (σ) modifies a feasible partial solution σ by executing moves randomly selected from $\mathcal{N}(\sigma)$ for a given number of times (I_{rand}). Here only feasible partial solutions are accepted.

3.3. Finding the best insertion–ejection combination

The procedure for finding the best insertion–ejection combination that minimizes P_{sum} proceeds as follows. All insertion positions in σ are considered for v_{in} . For each insertion, all possible ejections (of at most k_{max} customers) are tested; check the feasibility of the new route after ejecting the customers and record the ejected customers (and insertion position) if the new route is feasible and P_{sum} is less than P_{best} (the minimum value of P_{sum} found so far).

For each insertion, let $h[0; 1; \dots; n; n+1]$ be a sequence of the customers in the resulting infeasible route (we call it *original route*) where the customers are labeled according to the order in which they appear in the original route to simplify the notations. We denote all possible ejections as follows: $f1; 1; \dots; i.k/g; 1; i.1/ < \dots < i.k/ < n; k \leq k_{max}$. In order to test the ejections, we eject customers from the original route in the lexicographic order (See Fig. 3 for illustration). Before starting the lexicographic ejections, we calculate the latest possible arrival times [13], $z_i, i \in [1; \dots; n+1]$, for the original route (i.e., if the vehicle arrives at customer i no later than time z_i , the time window constraints of i and the subsequent customers in the route are satisfied). Let Q^0 be the total demand of the original route. The lexicographic ejections proceed by incrementing k (e.g., $f1; 2g \rightarrow f1; 2; 3g$), incrementing $i.k/$ (e.g., $f1; 2; 3g \rightarrow f1; 2; 4g$), or backtracking (e.g., $f1; n-1; ng \rightarrow f2g$). Let a_i^{new} denote the earliest starting time

Table 1

The performances of our algorithms.

N	Best CVN	Algorithm-B			Algorithm-S			Algorithm-D			Algorithm-DS			
		1 min	10 min	60 min	1 min	10 min	60 min	1 min	10 min	60 min	1 min	10 min	60 min	$\frac{N}{200}$ h
200	694	C1	C0	C0	0, C0	0, C0	0, C0	0, C0	0, C0	0, C0	0, C0	0, C0	0, C0	0, C0
400	1382	C15	C14	C13	0, C6	0, C0	0, C0	0, C3	1, C3	1, C3	0, C3	1, C1	2, C0	2, C0
600	2068	C28	C26	C20	0, C6	1, C1	1, C0	0, C7	1, C0	1, C0	0, C1	1, C0	3, C0	3, C0
800	2739	C32	C26	C21	0, C11	0, C5	1, C4	0, C15	2, C1	3, C0	2, C4	3, C1	4, C0	5, C0
1000	3425	C37	C28	C24	0, C9	2, C2	3, C1	0, C12	2, C2	3, C1	0, C2	5, C0	6, C0	8, C0
Total	10308	C113	C94	C78	0, C32	3, C8	6, C5	0, C37	6, C6	8, C4	2, C10	10, C2	15, C0	18, C0

Table 2

New best-known solutions.

Instances	N	Best-known	New best	Time (min)
C2_4_8	400	12	11	60
RC2_4_5	400	9	8	10
C1_6_6	600	60	59	60
C1_6_7	600	58	57	60
RC2_6_5	600	12	11	10
C1_8_2	800	73	72	1
C1_8_6	800	80	79	10
C1_8_8	800	74	73	240
C2_8_6	800	24	23	60
RC2_8_1	800	19	18	1
C110_6	1000	100	99	10
C110_7	1000	98	97	60
C110_8	1000	93	92	300
C210_3	1000	29	28	10
C210_6	1000	30	29	10
C210_7	1000	30	29	10
C210_8	1000	29	28	300
C21010	1000	29	28	10

of customer i in a new route obtained through the lexicographic ejections. Here, a_i^{new} is dynamically updated; each time $i.k/ \rightarrow j/$ is ejected or incremented, $a_{i.k/C1}^{\text{new}}$ or a_j^{new} can be updated in $O(1/\text{time})$ in the same way as in Eq. (1). When customer $i.k/$ is ejected, we can analyze the feasibility of the new route by checking whether both $a_{i.k/C1}^{\text{new}} \geq z_{i.k/C1}$ and $Q^0 \leq \sum_{s \in D1} q_{i.s/} \leq Q$ hold.

However, it would be very time consuming to test all possible ejections. To speed up the search, the lexicographic ejections can be pruned if one of the following conditions (a)–(c) is met. Here, we calculate in advance the earliest starting time $a_{i.k/} \cdot i \in D1; \dots; n \in C1/$ for the original route.

After $i.k/$ is ejected,

(a) $\sum_{s \in D1} p_{i.s/} \geq P_{\text{best}}$ (the minimum value of P_{sum} found so far)

After $i.k/ \rightarrow j/$ is incremented,

(b) $I_j < a_j^{\text{new}}$

(c) $a_j^{\text{new}} \geq a_j$ and $Q^0 \leq Q$

In the case (a), the lower-level lexicographic ejections ($i.1/; \dots; i.k/; \dots; j/$) can be pruned because these ejections never improve P_{best} where \cdot refers to any subsequent customer after $i.k/$. In both cases (b) and (c), the lower-level lexicographic ejections ($i.1/; \dots; i.k-1/; i.k/; \dots; j/$) can be pruned because (b) the time window constraint is violated already at customer j , or (c) the ejection of customers $i.1/; \dots; i.k-1/$ does not accelerate the earliest starting time of customer j (and the capacity constraint need not to be considered).

4. Experimental results

4.1. Benchmark instances

The proposed algorithm was tested on the well-known and widely used benchmark instances of Gehring and Homberger [11].

The benchmark set consists of 5 sets of 200, 400, 600, 800 and 1000 customers, with 60 instances in each set, resulting in 300 problems. For each set the problems are divided into 6 groups, R1, R2, RC1, RC2, C1, and C2. The C1 and C2 classes have customers located in clusters and in the R1 and R2 classes the customers are at random positions. The RC1 and RC2 classes contain a mix of both random and clustered customers. Each class contains 10 individual problem instances. The C1, R1 and RC1 problems have a short scheduling horizon and a small vehicle capacity (each route consists of about 10 customers on the average) whereas classes C2, R2 and RC2 have a long scheduling horizon and large vehicle capacity (each route consists of about 50 customers on the average).

4.2. Experimental settings

The suggested algorithm was tested with four different settings. Algorithm-B that does not include the diversification criterion, nor the squeeze procedure is considered to investigate the baseline performance. Algorithm-S includes only the squeeze procedure whereas Algorithm-D includes only the diversification criterion (selection mechanism based on the penalty counter). Algorithm-DS includes both the diversification criterion and squeeze procedure. When the diversification criterion is not considered, the insertion ejection combinations are determined randomly, giving priority to fewer k (the number of customers to be ejected). Here, ejecting v_{in} is forbidden because it always results in a feasible route and makes it impossible to eject more than two customers.

The parameters of the algorithms were set as follows: $k_{\text{max}} \in \{5, 10\}$, $I_{\text{rand}} \in \{1000, 10000\}$, and $\text{maxTime} \in \{1; 10, \text{or } 60 \text{ min}\}$. Moreover, algorithm-DS was executed with longer time limits for more exhaustive searches; $\text{maxTime} \in \{1; 2; 3; 4 \text{ and } 5 \text{ h}\}$ for 200, 400, 600, 800, 1000 customers, respectively. For each problem instance, one trial is executed. The proposed algorithm was implemented by C++ and was executed on an AMD Opteron 2.4 GHz (4 GB memory) computer.

4.3. Analysis of the results

The results of the four variants of our algorithm are listed in Table 1. The performance is evaluated with the cumulative number of vehicles (CNV). In the table, Best CNV corresponds to the cumulative number of vehicles in the best-known solutions, taken from <http://www.top.sintef.no> (three wrong best-known solutions are neglected) and [3,14,8,6].

Each result is represented by the number of instances in which a trial found a new best-known solution or could not reach the previous best-known solution. For example, "2, C1" means that the algorithm found new best-known solutions in two cases and could not reach the best-known solution in one case. We noticed that the difference in the number of routes between our solutions and the best-known solutions was always less than or equal to one (except for Algorithm-B). Therefore, the sum of these values $\cdot 2 \in C1/$ indicates the difference in the CNV wrt. the Best CNV.

Table 3

The results for all problem size.

200 customer	GH	PR	IIN	GD	LZ	Algorithm-DS			
						1 min	10 min	60 min	1 h
R1	18.2	18.2	18.2	18.2	18.2	18.2	18.2	18.2	18.2
R2	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0
RC1	18.1	18.0	18.0	18.0	18.0	18.0	18.0	18.0	18.0
RC2	4.4	4.3	4.3	4.3	4.3	4.3	4.3	4.3	4.3
C1	18.9	18.9	18.9	18.9	18.9	18.9	18.9	18.9	18.9
C2	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0
CVN	696	694	694	694	694	694	694	694	694
Computer	P 400M	P 3.0G	P 2.8G	O 2.3G	P 2.8G	O 2.4G	O 2.4G	O 2.4G	O 2.4G
CPU (min)	(4 2:1)	(7.7)	n/a	(53)	10	1	10	60	60
Runs	3	10	1	5	2	1	1	1	1
400 customer	GH	PR	IIN	GD	LZ	Algorithm-DS			
						1 min	10 min	60 min	2 h
R1	36.4	36.4	36.4	36.4	36.4	36.4	36.4	36.4	36.4
R2	8.0	8.0	8.0	8.0	8.0	8.0	8.0	8.0	8.0
RC1	36.1	36.0	36.0	36.0	36.0	36.0	36.0	36.0	36.0
RC2	8.8	8.5	8.6	8.6	8.5	8.5	8.4	8.4	8.4
C1	38.0	37.6	37.7	37.6	37.6	37.6	37.6	37.6	37.6
C2	12.0	12.0	12.0	11.9	11.7	12.0	11.8	11.6	11.6
CVN	1392	1385	1387	1385	1382	1385	1382	1380	1380
Computer	P 400M	P 3.0G	P 2.8G	O 2.3G	P 2.8G	O 2.4G	O 2.4G	O 2.4G	O 2.4G
CPU (min)	(4 7:1)	(15.8)	n/a	(89)	20	1	10	60	120
Runs	3	5	1	5	4	1	1	1	1
600 customer	GH	PR	IIN	GD	LZ	Algorithm-DS			
						1 min	10 min	60 min	3 h
R1	54.5	54.5	54.5	54.5	54.5	54.5	54.5	54.5	54.5
R2	11.0	11.0	11.0	11.0	11.0	11.0	11.0	11.0	11.0
RC1	55.0	55.0	55.0	55.0	55.0	55.0	55.0	55.0	55.0
RC2	11.9	11.6	11.6	11.7	11.5	11.6	11.4	11.4	11.4
C1	57.7	57.5	57.5	57.4	57.4	57.4	57.4	57.2	57.2
C2	17.8	17.5	17.4	17.5	17.4	17.4	17.4	17.4	17.4
CVN	2079	2071	2070	2071	2068	2069	2067	2065	2065
Computer	P 400M	P 3.0G	P 2.8G	O 2.3G	P 2.8G	O 2.4G	O 2.4G	O 2.4G	O 2.4G
CPU (min)	(4 12:9)	(18.3)	n/a	(105)	30	1	10	60	180
Runs	3	5	1	5	6	1	1	1	1
800 customer	GH	PR	IIN	GD	LZ	Algorithm-DS			
						1 min	10 min	60 min	4 h
R1	72.8	72.8	72.8	72.8	72.8	72.8	72.8	72.8	72.8
R2	15.0	15.0	15.0	15.0	15.0	15.0	15.0	15.0	15.0
RC1	72.3	73.0	72.4	72.0	72.0	72.1	72.0	72.0	72.0
RC2	16.1	15.7	15.7	15.8	15.6	15.6	15.4	15.4	15.4
C1	76.1	75.6	75.7	75.4	75.4	75.2	75.1	75.0	74.9
C2	23.7	23.7	23.4	23.5	23.4	23.4	23.4	23.3	23.3
CVN	2760	2758	2750	2745	2742	2741	2737	2735	2734
Computer	P 400M	P 3.0G	P 2.8G	O 2.3G	P 2.8G	O 2.4G	O 2.4G	O 2.4G	O 2.4G
CPU (min)	(4 23:2)	(22.7)	n/a	(129)	40	1	10	60	240
Runs	3	5	1	5	8	1	1	1	1
1000 customer	GH	PR	IIN	GD	LZ	Algorithm-DS			
						1 min	10 min	60 min	5 h
R1	91.9	92.2	91.9	91.9	91.9	91.9	91.9	91.9	91.9
R2	19.0	19.0	19.0	19.0	19.0	19.0	19.0	19.0	19.0
RC1	90.1	90.0	90.0	90.0	90.0	90.0	90.0	90.0	90.0
RC2	18.5	18.3	18.3	18.5	18.3	18.4	18.2	18.2	18.2
C1	95.4	94.6	94.5	94.3	94.4	94.1	94.0	93.9	93.8
C2	29.7	29.7	29.4	29.5	29.3	29.3	28.9	28.9	28.8
CVN	3446	3438	3431	3432	3429	3427	3420	3419	3417
Computer	P 400M	P 3.0G	P 2.8G	O 2.3G	P 2.8G	O 2.4G	O 2.4G	O 2.4G	O 2.4G
CPU (min)	(4 30:1)	(26.2)	n/a	(162)	50	1	10	60	300
Runs	3	5	1	5	10	1	1	1	1

For Algorithm-B, only the excess of the CNV (wrt. the Best CNV) is given.

Compared to the baseline performance, the results of both Algorithm-S and Algorithm-D clearly indicate the importance of

the squeeze procedure and the diversification criterion. Moreover, the excess of the total CNV of Algorithm-S and Algorithm-D with the 10 min time limit are only five . 3 C 8/ and zero . 6 C 6/, respectively, indicating effectiveness of each strategy. In

fact, both algorithms outperform all previous heuristics presented in Table 3 already with 10 min time limit wrt. the total CNV. However, Algorithm-S (Algorithm-D) could not reach the best-known solutions in five (four) instances even if the time limit was increased to 60 min, although six (eight) new-best solutions were found, respectively.

The results of the full version of our algorithm (Algorithm-DS) are better than those of Algorithm-S and Algorithm-D with each time limit condition. In particular, all the best-known and 15 new best-known solutions were obtained when the time limit was set to 60 min. This fact indicates the robustness of Algorithm-DS given the difficulty of reducing the number of routes and the intensive research on these benchmarks. By increasing the time limit to $\frac{N}{200}$ h, Algorithm-DS finally could find in total 18 new best-known solutions in terms of the number of routes. The new best-known solutions are detailed in Table 2 with the time limit needed to find these solutions.

In Table 3 we compare the results obtained with Algorithm-DS with other state-of-the-art results from the literature: GH (Gehring and Homberger [4]), RP (Pisinger and Ropke [3]), IIN (Ibaraki et al. [14]), GD (Gagnon and Desaulniers [8]) and LZ (Lim and Zhang [6]). All algorithms considered in the table are based on a two-stage approach, where the number of routes is optimized first, followed by the distance optimization. We compare the results based on the number of routes only. This comparison can be justified by the fact that the route-minimization procedure is executed independently in these algorithms.

In the table, results are compared in each benchmark set of different sizes. The first row lists the authors and the subsequent rows list the average number of routes with respect to the instances in each problem type (R1, R2, RC1, RC2, C1 and C2), CNV, computer specifications, average CPU time and the number of runs. If multiple runs were executed, the best results are listed in the table, meaning that the average computation time to obtain these results is CPU Runs minutes. The computer specifications are represented by 'P' (Pentium), 'O' (Opteron). Here one must note that the CPU times of LZ and Algorithm-DS refer to the time limit for each run in the route-minimization procedure. For GH, PR and GD, the CPU times refer to average CPU times (runs are restricted by the maximum number of iterations) including both route- and distance-minimization. For IIN, the computation time for the route-minimization is not reported by the authors.

As can be seen from the table, Algorithm-DS with the 1-min runs outperforms all previous methods except for LZ with respect to CNV in each problem size. With the 10 min runs, Algorithm-DS outperforms all previous methods in the same sense. The

difference between Algorithm-DS and the previous methods also appears to increase with the problem size. Moreover, Algorithm-DS appears to be competitive also wrt. the CPU time, especially for larger problems.

Acknowledgements

This work was partially supported by Grant-in-Aid for Young Scientists (B) and by TEKES and the EDGE project funded by the Research Council of Norway. This support is gratefully acknowledged. We would like to thank also Dr. Wout Dullaert for his valuable feedback and help.

References

- [1] O. Bräysy, M. Gendreau, Vehicle routing problem with time windows, Part I: Route construction and local search algorithms, *Transportation Science* 39 (2005) 104–118.
- [2] O. Bräysy, M. Gendreau, Vehicle routing problem with time windows, Part II: Metaheuristics, *Transportation Science* 39 (2005) 119–139.
- [3] D. Pisinger, S. Ropke, A general heuristic for vehicle routing problems, *Computers & Operations Research* 34 (2007) 2403–2435.
- [4] H. Gehring, J. Homberger, Parallelization of a two-phase metaheuristic for routing problems with time windows, *Asia-Pacific Journal of Operational Research* 18 (2001) 35–47.
- [5] R. Bent, P. Van Hentenryck, A two-stage hybrid local search for the vehicle routing problem with time windows, *Transportation Science* 38 (2004) 515–530.
- [6] A. Lim, X. Zhang, A two-stage heuristic with ejection pools and generalized ejection chains for the vehicle routing problem with time windows, *Informatics Journal on Computing* 19 (2007) 443–457.
- [7] Y. Nagata, Efficient evolutionary algorithm for the vehicle routing problem with time windows: Edge assembly crossover for the VRPTW, in: *Proc. of the 2007 Congress on Evolutionary Computation*, CD-ROM, 2007, pp. 1175–1182.
- [8] E. Prescott-Gagnon, G. Desaulniers, L.-M. Rousseau, A branch-and-price-based large neighborhood search algorithm for the vehicle routing problem with time windows, Working Paper, University of Montreal, Canada, 2007.
- [9] C. Voudouris, E. Tsang, Guided local search, Technical Report CSM-247, Department of Computer Science, University of Essex, UK, August, 1995.
- [10] C. Voudouris, E. Tsang, Guided local search, in: F. Glover (Ed.), *Handbook of Metaheuristics*, Kluwer, 2003, pp. 185–218.
- [11] H. Gehring, J. Homberger, A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows, in: *Proc. of EUROGEN*, vol. 99, 1999, pp. 57–64.
- [12] J.-Y. Potvin, J.-M. Rousseau, An exchange heuristic for routing problems with time windows, *Journal of the Operational Research Society* 46 (1995) 1433–1446.
- [13] G.A.P. Kindervater, M.W.P. Savelsbergh, Vehicle routing: Handling edge exchanges, in: *Local Search in Combinatorial Optimization*, Wiley, 1997, pp. 337–360.
- [14] T. Ibaraki, S. Imahori, K. Nonobe, K. Sobue, T. Uno, M. Yagiura, An iterated local search algorithm for the vehicle routing problem with convex time penalty functions, *Discrete Applied Mathematics* 156 (2008) 2050–2069.