Computers & Operations Research 39 (2012) 728–735

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

An adaptive large neighborhood search heuristic for the cumulative capacitated vehicle routing problem

Glaydston Mattos Ribeiro [a,b], Gilbert Laporte [a,*]

[a] CIRRELT and HEC Montréal, Canada H3T 2A7
[b] Federal University of Espírito Santo, São Mateus - ES, 29932-540, Brazil

ARTICLE INFO

Available online 14 May 2011

Keywords:
Cumulative objective
Metaheuristics
Adaptive large neighborhood search
Disaster relief

ABSTRACT

The *cumulative capacitated vehicle routing problem* (CCVRP) is a variation of the classical *capacitated vehicle routing problem* in which the objective is the minimization of the sum of arrival times at customers, instead of the total routing cost. This paper presents an adaptive large neighborhood search heuristic for the CCVRP. This algorithm is applied to a set of benchmark instances and compared with two recently published memetic algorithms.

1. Introduction

The purpose of this paper is to describe an adaptive large neighborhood search heuristic (ALNS) for the *cumulative capacitated vehicle routing problem* (CCVRP). This problem is a variation of the classical *capacitated vehicle routing problem* (CVRP) in which the objective is the minimization of the sum of arrival times at the customers instead of the total routing cost. The CCVRP is an extension of the *delivery man problem* (DMP) [1,2] to the case of several vehicles. The DMP has applications in job scheduling problems on a single machine [3,4]. All applications of the DMP extend naturally to the CCVRP. The CCVRP with time windows can be used to model some school bus routing problems [5,6].

An important application of the CCVRP arises in the procurement of humanitarian aid in the context of natural disasters, such as tsunamis or earthquakes, where it is crucial to reduce suffering and life losses [7–10]. Campbell et al. [8] show that routes created by optimizing with respect to the CVRP objective can serve communities significantly later than those generated with a CCVRP objective and are not completely fair for a humanitarian supply chain. Besides, optimal solutions for the traditional CVRP can be significantly different from those observed in the context of the delivery of humanitarian aid [8,9].

Ngueveu et al. [9] have recently studied the CCVRP with a homogeneous fleet and a single depot. These authors have proposed lower bounds and two memetic algorithms (MA) which were applied to some instances with up to 199 vertices. Nolz et al. [10] have presented a bi-objective metaheuristic for a problem in which vehicle routes must be planned to deliver drinking water in a post-disaster situation. More precisely, the authors consider the joint minimization of the sum of distances between population centers and their nearest facility, and of the latest arrival time of the supply vehicle at a population center. They have applied their algorithm to instances with up to 41 vertices, based on data from Manabí, Ecuador. Campbell et al. [8] have considered the *traveling salesman problem* and the CVRP with two alternative objective functions: one that minimizes the maximal arrival time, and one that minimizes the average (or total) arrival time. They have derived lower bounds, as well as some insertion and local search heuristics. Tests were performed on 50 instances with 30 to 79 customers proposed by Augerat et al. [11], where customers are distributed uniformly or grouped into clusters, and on 11 instances proposed by Golden et al. [12], with at most 100 customers, which are based on special structures such as concentric circles, diamonds, squares and stars.

The CCVRP is NP-hard [9] and difficult to solve to optimality for even moderate instance sizes. There is, therefore, a need to develop efficient heuristics for this problem. To our knowledge, the only available heuristics for the CCVRP are the MAs of Ngueveu et al. [9]. Our aim is to develop an alternative adaptive large neighborhood search heuristic (ALNS) for the CCVRP and to compare it to the two heuristics of Ngueveu et al.

The remainder of this paper is organized as follows. Section 2 recalls the mathematical formulation, the memetic heuristics and the lower bounds proposed by Ngueveu et al. [9]. Our ALNS heuristic is described in Section 3, followed by computational results in Section 4. Conclusions are presented in Section 5.

2. Mathematical model, lower bounds and memetic algorithm

This section presents a brief overview of the paper presented by Ngueveu et al. [9]. We first recall their mathematical programming

* Corresponding author. Tel.: +1 514 343 6143; fax: +1 514 343 7121.
E-mail addresses: glaydston.ribeiro@cirrelt.ca (G. Mattos Ribeiro), gilbert.laporte@cirrelt.ca (G. Laporte).

0305-0548/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2011.05.005

formulation in order to state the CCVRP in a precise fashion. We then describe the lower bounds and the MA proposed by these authors.

2.1. Mathematical model and lower bounds

The CCVRP is defined on an undirected graph $G=(V,E)$ where $V=\{0,1,\ldots,n,n+1\}$ is the vertex set, vertices 0 and $n+1$ are two copies of the depot, and $V'=V\backslash\{0,n+1\}$ is the set of customers. The set $E\{(i,j):i,j\in V, i<j\}$ is the edge set. A travel time c_{ij} is associated with each edge $(i,j)\in E$. It is assumed that travel times are symmetric and satisfy the triangle inequality. Each customer $i\in V'$ has a demand q_i. Let R be a set of identical vehicles of capacity Q. The aim of the CCVRP is to define a set of vehicle routes starting and ending at the depot, visiting each customer once in exactly one route, having a demand not exceeding the vehicle capacity, and minimizing the sum of arrival times at the customers. Let t_i^k be the arrival time of vehicle k at customer i and let x_{ij}^k be a binary variable equal to 1 if and only if vehicle k traverses edge (i,j) from i to j. The CCVRP formulation is then

$$(CCVRP) \quad \text{minimize} \sum_{k\in R}\sum_{i\in V'} t_i^k \tag{1}$$

subject to

$$\sum_{i\in V} x_{ij}^k = \sum_{i\in V} x_{ji}^k \quad j\in V', \quad k\in R \tag{2}$$

$$\sum_{k\in R}\sum_{j\in V} x_{ij}^k = 1 \quad i\in V' \tag{3}$$

$$\sum_{i\in V'}\sum_{j\in V} q_i x_{ij}^k \le Q \quad k\in R \tag{4}$$

$$\sum_{j\in V} x_{0j}^k = 1 \quad k\in R \tag{5}$$

$$\sum_{i\in V} x_{i,n+1}^k = 1 \quad k\in R \tag{6}$$

$$(t_i^k + c_{ij})x_{ij}^k - t_j^k \le (1-x_{ij}^k)M \quad i\in V\ \{n+1\}, \quad j\in V, \ k\in R \tag{7}$$

$$t_i^k \ge 0 \quad i\in V, \quad k\in R \tag{8}$$

$$x_{ij}^k \in \{0,1\} \quad i\in V, \quad j\in V, \ k\in R. \tag{9}$$

The objective function (1) minimizes the sum of arrival times at the customers. Constraints (2) are flow conservation equations. Constraints (3) state that each customer i is served once by only one vehicle. Constraints (4) ensure that the total demand carried by vehicle k does not exceed its capacity. Constraints (5) and (6) ensure that each route starts and ends at the depot. Constraints (7) ensure that if customer j is served after customer i by vehicle k, then t_j^k must be greater than or equal to t_i^k, plus the travel time c_{ij}, and M is a sufficiently large positive constant. Finally, constraints (8) and (9) impose restrictions on the decision variables.

Ngueveu et al. [9] have shown that an optimal solution for the CCVRP uses exactly $\min\{n,|R|\}$ vehicles. When $|R|\ge n$, the optimal solution is a set of back and forth routes between the depot and each customer, which yields the lowest objective function value attainable for any value of $|R|$.

Thus a trivial lower bound is given by

$$LB_1 = \sum_{j\in V'} c_{0j}. \tag{10}$$

A second lower bound applicable when $|R|<n$ is computed by using the $|R|$ shortest edges incident to the depot and the remaining $n-|R|$ shortest edges. Let c'_e be the cost of the eth shortest edge incident to the depot, and let c''_e be the cost of the eth edge between two customers. This bound can be calculated as

$$LB_2 = \sum_{e=1}^{|R|}\left(\left\lceil\frac{|R|+n-e-(n \bmod |R|)}{|R|}\right\rceil\right)c'_e + \sum_{e=1}^{n-|R|}\left(\left\lceil\frac{n-e-(n \bmod |R|)}{|R|}\right\rceil\right)c''_e. \tag{11}$$

2.2. Memetic algorithm

The MA proposed by Ngueveu et al. [9] is based on the hybrid genetic algorithm of Prins [13]. It defines a chromosome as a permutation of the n customers, without trip delimiters. The fitness of a chromosome is the total cost corresponding to the best extractable solution from it, which is obtained by applying an optimal splitting procedure based on the solution of a shortest path problem.

The population of chromosomes is initialized as follows. The first individual is a solution obtained by means of a nearest neighbor heuristic, while the second is obtained after applying local search on it. These solutions are transformed into chromosomes by concatenating their routes, and they are inserted into the population. The remaining chromosomes are random permutations of the customers.

For the crossover operation, two chromosomes are chosen randomly and the order crossover OX is applied to them to generate two children, as in Prins [13]. Ngueveu et al. [9] also consider children obtained by applying OX with one or both parents reversed. Thus these authors work with two MA implementations: MA_1 where OX is used with zero, one or two parents reversed, and MA_2 which only applies the traditional OX. Unlike what happens in the traditional CVRP, a route for the CCVRP has a different cost when it is reversed. It is, therefore, important to check whether a better solution can be obtained by reversing the parent solutions.

Local search is applied over the solutions obtained by the splitting procedure. It uses three moves for one or two distinct routes: 2-opt, exchange of two customers between two routes, and relocation of one customer to a different route. In addition to these three classical moves, a check is made to see whether the result of a move can be improved by reversing the candidate routes.

As a rule, MA_1 generates better solutions than MA_2, but it is more time consuming.

3. Adaptive large neighborhood search heuristic

We now present our ALNS heuristic for the CCVRP. The ALNS heuristic introduced by Ropke and Pisinger [14] extends the large neighborhood search heuristic of Shaw [15] by allowing the use of multiple destroy and repair methods within the same search process. It belongs to the class of very large scale neighborhood search algorithms [16].

At each iteration, the heuristic *destroys* part of the current solution s and *repairs* it in a different way to generate a new solution s''. This new solution is accepted according to a criterion defined by a search paradigm applied at the master level, such as an acceptance criterion from simulated annealing (SA) by which if s' is better than s, the search continues from s', and otherwise the search continues from s' with some probability.

The destroy and repair procedures are selected according to an adaptive probabilistic mechanism. At each iteration, the probability of selecting a given procedure depends on how well it has performed in the past. Recently, ALNS has provided good solutions for a wide variety of vehicle routing problems, see for instance [14,16–20].

We now describe the main elements of our ALNS implementation for the CCVRP. This heuristic contains some user-controlled parameters denoted by lower case Greek letters, which will be calibrated in Section 4.

1. *Large neighborhood*: Given a solution s, at each iteration, γ customers are removed from it and are then reinserted. This is accomplished by using one of several removal and insertion heuristics.
   Because the number of vehicles is limited to $|R|$, it may not be possible to reinsert some customers without violating the capacity constraint. In this case, they are placed in a *request bank U* and the solution s' is penalized accordingly. If a solution s' with customers in the request bank is accepted by the SA criterion, at the next insertion operation the customers in the request bank are pooled with the newly removed customers. The request bank is necessary to allow the heuristic to consider intermediate infeasible solutions, thus improving the overall search [14].
2. *Adaptive search engine*: The choice of the removal and insertion heuristics is governed by a roulette-wheel mechanism in which each heuristic is assigned a weight that depends on its past behavior. More precisely, let w_i be a measure of how well heuristic i has performed in past iterations. Then, given h heuristics with weights w_i, heuristic j is selected with probability $w_j/\sum_{i=1}^{h} w_i$. The insertion heuristic is chosen independently of the removal heuristic.
3. *Adaptive weight adjustment*: The search is divided into a number of *segments* of φ consecutive iterations. In the first segment, all heuristics have the same weight i.e., $w_i=1$ for $i=1,\ldots,h$. After φ iterations, the weights used to select removal and insertion heuristics are updated according to the *score* obtained during the segment. The scores show how well the removal and insertion heuristics have performed in the last segment. The score of a heuristic is increased by a parameter equal to σ_1,σ_2 or σ_3 when it identifies new solutions. For the CCVRP, if a pair of removal–insertion heuristic finds a new best solution, their scores are increased by $\sigma_1=50$, if it finds a solution better than the current one, their scores are increased by $\sigma_2=20$, and if it finds a non-improving solution which is accepted, their scores are increased by $\sigma_3=5$. When a segment ends, new weights are calculated using the scores obtained, and all scores are reset to zero for the next segment. Let π_i and o_{ij} be, respectively, the score of heuristic i, and the number of times that heuristic i has been chosen in the last segment j. Then,
$$w_{i,j+1}=\begin{cases} w_{ij} & \text{if } o_{ij}=0 \\ (1-\eta)w_{ij}+\eta\pi_i/o_{ij} & \text{if } o_{ij}\neq 0, \end{cases} \tag{12}$$
   where $\eta\in[0,1]$ is parameter called the *reaction factor*. This parameter controls how quickly the weight adjustment algorithm reacts to changes in the effectiveness of the heuristics.
4. *Penalized objective function*: Solutions s with customers in the request bank are considered during the search. This is done through the use of the penalized objective function
$$v(s)=\sum_{k\in R}\sum_{i\in V'} t_i^k+\lambda|U|, \tag{13}$$
   where λ is a user-defined penalty.
5. *Acceptance and stopping criteria*: The acceptance criterion from SA is used, i.e., given a current solution s, a neighbor solution s' is accepted if $v(s')<v(s)$, and it is accepted with probability $e^{-(v(s')-v(s))/T}$ otherwise, where $T>0$ is the current *temperature* and $v(s)$ is the penalized solution cost defined by (13). The temperature starts at T_{start} and is multiplied by c at every iteration, where $0<c<1$ is the *cooling rate*. Ropke and Pisinger [14,17] suggest that the best choice of T_{start} should be instance-dependent. In our experiments, given an instance and an initial solution s, T_{start} is set equal to $v(s)$ with $\lambda=0$, while the cooling rate is fixed to 0.99975 independently of the instance.
6. *Noise to the objective function*: Ropke and Pisinger [14] indicate that to avoid the myopic behavior of some insertion heuristics, it is necessary to add a noise term to the objective function. For the CCVRP this strategy did not improve the results, and, therefore, it was not used. Instead of it, we used a basic greedy insertion heuristic that inserts the customers respecting their order in the removed list. This heuristic is explained in Section 3.1.8.

The number γ of customers removed from the current solution has a significant impact on the ALNS performance. If only a small part of the solution is destroyed, then the heuristic may not be able to efficiently explore the search space because the effect of a large neighborhood is lost. If a very large part of the solution is destroyed, then the heuristic almost degrades into independent reoptimization threads. This can be time consuming or yield poor quality solutions, depending on how the partial solution is repaired [16].

3.1. Removal (H^-) and insertion (H^+) heuristics

This section describes seven removal and three insertion heuristics. Where applicable, let $v(s)$ be the penalized solution cost defined by (13) for a solution s, and let D be a set of removed customers, consisting of customers in the request bank U and of $\gamma-|U|$ newly removed customers from s.

3.1.1. Shaw removal heuristic based on arrival times

Following Ropke and Pisinger [14] and Shaw [15], the general idea of the Shaw removal heuristic is to remove customers that are somewhat similar, as it is expected to be reasonably easy to reshuffle similar requests and thereby create new, perhaps better solutions. The degree of similarity between two customers i and j is computed through a *relatedness measure* $R(i,j)$, where a lower value corresponds to more similar customers. The first Shaw removal heuristic (SRH1) is based on the absolute difference between the contributions of customers i and j to the objective function. More precisely,
$$R_{SHR1}(i,j)=|t_i^k-t_j^k|. \tag{14}$$

Algorithm 1. SRH1—Shaw removal heuristic

```
1  Input: SOLUTION $s$, REQUEST BANK $U$, $\gamma$ AND $\delta\geq 1$;
2  Let $D$ BE A SET OF REMOVED CUSTOMERS
3  Let $L$ BE AN ORDERED SET OF CUSTOMERS AND $L_i$ THE $i$th CUSTOMER OF $L$;
4  $D\leftarrow U$;
5  if $U=\{\}$ then
6      $r\leftarrow$ A CUSTOMER SELECTED AT RANDOM FROM $s$;
7      $D\leftarrow\{r\}$;
8      REMOVE $r$ FROM $s$;
9  end
10 while $|D|<\gamma$ do
11     $r\leftarrow$A CUSTOMER SELECTED AT RANDOM FROM $D$;
12     $L\leftarrow$ CUSTOMERS OF $s$;
13     SORT $L$ SUCH THAT $i<j\Rightarrow R_{SHR1}(r,L_i)<R_{SHR1}(r,L_j)$;
14     CHOOSE A RANDOM NUMBER $y$ FROM [0,1];
15     $i\leftarrow\lfloor y^{\delta}|L|\rfloor$;
16     $D\leftarrow D\cup\{L_i\}$;
17     REMOVE $L_i$ FROM $s$;
18 end
19 RETURN $s$ AND $D$;
```

The relatedness measure is used to remove γ customers as in Shaw [15]. Given a solution s and a set D of removed customers, the algorithm randomly selects a customer r from D, calculates the relatedness measure between it and all customers not yet removed, and then chooses a new customer to be inserted in D. The process is repeated while $|D| < \gamma$. See Algorithm 1 for details and note that the parameter $\delta \geq 1$ introduces randomness in the selection of a customer.

3.1.2. *Shaw removal heuristic based on distances*

For this Shaw removal heuristic (SRH2), the similarity between two customers i and j is measured by the relatedness measure $R_{SHR2}(i,j) = d_{ij}$, where d_{ij} is the distance between i and j. Apart from the relatedness measure, this procedure is identical to Algorithm 1. Note that some precomputations can be performed to streamline SRH2. If a nearest customer matrix is kept at hand, i.e., a matrix where each line (customer) i has in the columns the index (customer) j in non-decreasing order of distance d_{ij}, lines 13, 14 and 16 of Algorithm 1 can be merged to avoid unnecessary computations.

3.1.3. *Random removal heuristic*

This simple removal heuristic removes $\gamma - |U|$ customers selected at random from the current solution s, given that the customers of U are already considered removed. This heuristic tends to generate a poor set of removed customers, but it helps diversify the search.

3.1.4. *Worst removal heuristic*

This heuristic removes customers with a high cost in the current solution s and attempts to insert them in better positions. Let $Cost^{-}(i,s) = v(s) - v_{-i}(s)$ be the cost associated with customer i in current solution s, where $v_{-i}(s)$ represents the solution cost without customer i into s. The heuristic first sorts the customers according to $Cost^{-}(i,s)$, chooses one to be removed, recalculates $Cost^{-}(i,s)$ for the remaining customers, and the process is repeated. See Algorithm 2 for details and note that the removal is also randomized, but controlled by a parameter $\rho \geq 1$.

Algorithm 2. WRH—Worst removal heuristic

```
1  Input: SOLUTION $s$, REQUEST BANK $U$, $\gamma$ AND $\rho \geq 1$;
2  LET $D$ BE SET A OF REMOVED CUSTOMERS
3  LET $L$ BE AN ORDERED SET OF CUSTOMERS AND $L_i$ THE $i$th
   CUSTOMER OF $L$;
4  $D \leftarrow U$;
5  while $|D| < \gamma$ do
6      $L \leftarrow$ CUSTOMERS OF $s$;
7      SORT $L$ SUCH THAT $i < j \Rightarrow Cost^{-}(L_i,s) < Cost^{-}(L_j,s)$;
8      CHOOSE A RANDOM NUMBER $y$ FROM [0,1];
9      $i \leftarrow \lfloor y^{\rho}|L| \rfloor$;
10     $D \leftarrow D \cup \{L_i\}$;
11     REMOVE $L_i$ FROM $s$;
12 end
13 RETURN $s$ AND $D$;
```

3.1.5. *Cluster removal*

Some CCVRP instances contain clusters of customers. Suppose that a vehicle serves customers from two geographically distant clusters. Then, it is important to divide these customers into two sets, and remove one of them entirely to be reinserted in a correct way. Given a route r, the cluster removal heuristic (CRH) splits it into two groups of related customers based on strongly connected components criteria. To define this groups, we use a modified version of Kruskal's algorithm [21] for the *minimum spanning tree problem*, as in Ropke and Pisinger [17]. Instead of running Kruskal's algorithm to the end, we stop it when two connected components are found. These components approximate the clusters. One of these clusters is chosen at random and its customers are removed. If it is necessary to remove more customers, an already removed customer i is chosen at random and another customer j from a different route more related to i is found. The route of the customer j is then partitioned into two clusters and the process is repeated until the desired number of removed customers has been reached. See Algorithm 3 for details.

Algorithm 3. CRH—Cluster removal heuristic

```
1  Input: SOLUTION $s$, REQUEST BANK $U$ AND $\gamma$;
2  LET $D$ BE A SET OF REMOVED CUSTOMERS;
3  $D \leftarrow U$;
4  $r_{prior} \leftarrow -1$;
5  while $|D| < \gamma$ do
6      if $|D| = \{\}$ then
7          $r \leftarrow A$ NON – EMPTY ROUTE SELECTED AT RANDOM FROM $s$;
8      else
9          $i \leftarrow$ A CUSTOMER SELECTED AT RANDOM FROM $D$;
10         $r \leftarrow$ A NON – EMPTY ROUTE DIFFERENT OF $r_{prior}$ THAT HAS THE CLOSEST CUSTOMER TO $i$;
11     end
12     APPLY THE KRUSKAL's ALGORITHM ON $r$ TO FIND $Cluster_1$ AND $Cluster_2$;
13     $L \leftarrow$ CUSTOMERS FROM ONE CLUSTER CHOSEN AT RANDOM
14     $D \leftarrow D \cup L$;
15     $r_{prior} \leftarrow r$;
16     REMOVE CUSTOMERS IN $L$ FROM $s$;
17 end
18 RETURN $s$ AND $D$;
```

3.1.6. Neighbor graph removal

This removal heuristic uses historical information to remove customers. The historical information is stored in a complete directed and weighted graph called the neighbor graph [17,18]. A node in this graph represents a visit in the original problem. All edge weights are initially set to infinity. The cost of an edge (i,j) in this graph is the best objective function value found until the current ALNS iteration for which customer i is visited just before customer j. When a new solution is found, the edge weights in the neighbor graph are updated if necessary. Given a current solution, this removal heuristic calculates a score for each customer by summing up the edge weights in the neighbor graph corresponding to the neighbor configuration in s. The customers with high scores are likely to be in wrong positions and are selected for removal. When a customer is removed, the scores of the surrounding customers are recalculated. See Algorithm 4 for details and note that the removal is also randomized, but controlled by a parameter $\phi \geq 1$.

Algorithm 4. NGR—Neighbor graph removal

```
1  Input: SOLUTION $s$, REQUEST BANK $U$, $\gamma$ AND $\phi \geq 1$;
2  LET $D$ BE A SET OF REMOVED CUSTOMERS;
3  LET $L$ BE AN ORDERED SET OF CUSTOMERS AND $L_i$ THE $i$th CUSTOMER OF $L$;
4  $D \leftarrow U$;
5  $L \leftarrow$ CUSTOMERS OF $s$;
6  CALCULATE THE SCORE FOR EACH CUSTOMER $L_i$ USING THE NEIGHBOR GRAPH;
7  while $|D| < \gamma$ do
8      SORT $L$ SUCH THAT $i < j \Rightarrow SCORE(L_i) > SCORE(L_j)$;
9      CHOOSE A RANDOM NUMBER $y$ FROM [0,1];
10     $i \leftarrow \lfloor y^{\phi}|L| \rfloor$;
11     $D \leftarrow D \cup \{L_i\}$;
12     UPDATE THE SCORES FOR THE CUSTOMERS CLOSE TO $L_i$;
13     REMOVE $L_i$ FROM $L$;
14 end
15 RETURN $s$ AND $D$;
```

3.1.7. Request graph removal

Our heuristic also uses historical information to remove customers. This information is stored in a complete and undirected graph called the request graph [17,18], in which a node represents a customer. All edge weights are initially set to zero. The cost of an edge (i,j) in this graph is the number of times customers i and j have been served by the same vehicle in the B best unique solutions (top-B) observed so far in the search. When a new unique top-B solution is found, the weights are incremented or decremented according to the solutions entering and leaving the top-B solutions. In our computational experiments, we use $B = 100$. Given a customer i, customer j is considered more related to i if it presents the largest edge weight in the current request graph. This relatedness measure is used as in the Shaw removal heuristic described in Section 3.1.1.

3.1.8. Basic greedy insertion heuristic (BGI)

Let Δv_{ik} be the change in the solution cost incurred by inserting customer i into route k in the position that increases the objective function the least in the current solution s. If a customer i cannot be inserted into route k, then $v_{ik} = \infty$. Let $f^+(i,s) = \min_{k \in R}\{\Delta v_{ik}\}$ be the minimum change found in the solution representing the minimum cost position. If D is a set of removed customers and $\bar{i}$ is the first customer in D, we calculate $f^+(\bar{i},s)$ and customer $\bar{i}$ is removed from D and inserted in the corresponding route in the best position. Note that this greedy insertion heuristic respects the order of the customers in D. After the first customer has been inserted, $f^+(\bar{i},s)$ is calculated again and the process is repeated. If a customer cannot reinserted, it is left in the request bank.

3.1.9. Deep greedy insertion heuristic (DGI)

This heuristic works differently from the previous one. Instead of inserting the first customer i from D into the current solution s, it inserts the customer i having the minimum global cost position. Formally, it inserts customer i yielding $\min_{i \in D}\{f^+(i,s)\}$ when i is inserted in its least cost position. This process is repeated until no more customers can be inserted. Consequently, this heuristic is slower than BGI.

3.1.10. Regret-κ insertion heuristic

The regret heuristic tries to improve the myopic behavior of greedy heuristics [14,19,22]. Given a set D of removed customers, for each customer $i \in D$, this heuristic calculates a regret value equal to the difference in cost between two solutions in which i is inserted in its best route or in its second best route. The customer i with the maximum regret value is chosen to be inserted in the current solution s. Formally, let $\omega_{ik} \in R$ be and index indicating the route for which customer i has the kth lowest insertion cost, i.e., $\Delta v_{i\omega_{ik}} \leq \Delta v_{i\omega_{ik'}}$ for all $k \leq k'$. Thus, the regret value is $RegretValue_i = \Delta v_{i\omega_{i2}} - \Delta v_{i\omega_{i1}}$ and at each iteration, the heuristic chooses customer i according to $\max_{i \in D}\{RegretValue_i\}$. Ties are broken by selecting the lowest cost insertion.

This concept can be extended by considering not only the cost difference just defined, but also the difference in cost of inserting customer i in its best, $2nd$ best, $\ldots, \kappa th$ best route, where κ is a user-defined parameter. With this idea, the regret-κ heuristic chooses a customer i according to $\max_{i \in D}\{\sum_{j=2}^{\kappa}(\Delta v_{i\omega_{ij}} - \Delta v_{i\omega_{i1}})\}$ s and inserts it in its least cost position. Ties are broken by selecting the lowest cost insertion.

3.2. Discussion

Ropke and Pisinger [14,17] and Pisinger and Ropke [18] add a noise term to the objective function during the insertion phase of heuristics DGI and regret-κ. As explained, this strategy did not improve our results and we then decided to work with heuristic BGI described in Section 3.1.8. Although this heuristic seems inappropriate because it inserts the customers according to their order in D, it serves two major functions: to ensure the insertion of customers who are in the request bank, and to introduce some noise in the insertion process.

All removal heuristics defined in Section 3.1.1– 3.1.7 first consider the customers of the request bank as removed. So, when the BGI is invoked, the customers not previously inserted are the first ones to be inserted, and since the current solution s' has $n - \gamma$ customers, it is more likely that they will be successfully inserted. This insertion heuristic is not as efficient as DGI and the regret-κ heuristics to find new best solutions. However, even at the end of the search, ALNS sometimes invokes BGI, which contributes to diversify the search.

3.3. Initial solution and stopping criteria

Our initial solution is constructed by means of a regret-3 heuristic. All customers are initially placed in the request bank, and the regret-3 heuristic is run in parallel for all vehicles.

In the ALNS, the master framework is based on the SA mechanism and the removal and insertion heuristics are repeated until one of the following two stopping criteria is satisfied: (1) the number of iterations reaches 50 000; and (2) the temperature reaches 10^{-2}.

4. Computational results

The ALNS heuristic was coded in C++ and run on a laptop with Pentium Core 2 Duo processor of 2.0 GHz with 3 GB of RAM Memory under the Windows operating system.

We have tested the ALNS heuristic on the seven 50–199 customer instances used by Ngueveu et al. [9], and first proposed by Christofides et al. [23], and on the 20 large scale instances proposed by Golden et al. [12], ranging from 240 to 483 customers.

Our results were compared to the ones obtained with MA1 and MA2 [9]. To ensure fair comparisons, all heuristics were run five times for each instance on the same computer. The computer codes for MA1 and MA2 were kindly provided by Ngueveu et al. [9]. The travel times are equal to the Euclidean distances calculated using double precision.

4.1. Tuning instances and parameters

To tune the parameters of ALNS, we have chosen four instances at random: CMT_2, CMT_4, $GWKC_2$ and $GWKC_9$. To set the main ALNS parameters, we have followed the methodology presented by [17]. The parameters not yet fixed are: γ, φ, η, λ, δ, ρ, ϕ and κ. The parameter γ has a significant impact on the results. As shown by Pisinger and Ropke [18], it is not necessary to remove a large number of customers in the removal phase. We have used the following strategy: if $n \leq 200$, γ is chosen at random in the interval [10,40], otherwise γ is chosen at random in the interval [10,60].

To set all parameters, each parameter was in turn allowed to take several values, while the others were kept fixed. We ran ALNS five times for each parameter setting, and the setting

Table 1
Values for the ALNS parameters after the tuning phase.

Parameter	Meaning	Best value
γ	Defines the number of customers removed at each ALNS iteration	If $n \leq 200$, $\gamma \in [10,40]$, otherwise, $\gamma \in [10,60]$
φ	Defines the number of consecutive iterations in a segment	50
η	Reaction factor	10^{-2}
λ	Penalizes a violated solution in (13)	10^4
δ	Avoids determinism in the Shaw and Request graph removal heuristics	2
ρ	Avoids determinism in the worst removal heuristic	2
ϕ	Avoids determinism in the neighbor graph removal heuristic	3
κ	Indicates which regret heuristic must be used	3

Table 2
Results for the seven medium-size instances proposed by Christofides et al. [23] and used by Ngueveu et al. [9].

Name	*n*	|*R*|	*LB*	MA1				MA2				ALNS			
				Best	Avge	Time (s)	Dev (%)	Best	Avge	Time (s)	Dev (%)	Best	Avge	Time (s)	Dev (%)
CMT_1	50	5	1873.91	**2230.35**	2230.35	10.63	0.00	**2230.35**	2245.74	3.70	0.00	**2230.35**	2235.27	30.29	0.00
CMT_2	75	10	1861.56	2421.90	2443.07	27.78	1.27	2429.18	2556.24	2.04	1.57	**2391.63**	2401.72	60.77	0.00
CMT_3	100	8	2947.74	4073.12	4073.12	97.91	0.68	4073.12	4079.69	40.46	0.68	**4045.42**	4063.98	172.45	0.00
CMT_4	150	12	3561.67	**4987.52**	5020.75	449.44	0.00	**4987.52**	4996.84	188.41	0.00	**4987.52**	4994.93	235.12	0.00
CMT_5	199	17	4804.20	**5810.12**	5842.00	1035.45	0.00	**5810.12**	5840.77	629.27	0.00	5838.32	5857.76	277.37	0.49
CMT_{11}	120	7	6119.66	7317.98	7395.83	160.64	0.03	7347.49	7395.46	68.83	0.43	**7315.87**	7341.28	202.07	0.00
CMT_{12}	100	10	2885.48	**3558.92**	3559.23	38.20	0.00	3559.43	3559.43	23.66	0.01	**3558.92**	3566.06	152.74	0.00
Avge	113.43	9.86	3436.32	4342.84	4366.34	260.01	0.28	4348.17	4382.02	136.62	0.38	**4338.29**	**4351.57**	**161.54**	**0.07**

Table 3
Results for the 20 large instances proposed by Golden et al. [12].

Name	*n*	|*R*|	*LB*	MA1				MA2				ALNS			
				Best	Avge	Time (s)	Dev (%)	Best	Avge	Time (s)	Dev (%)	Best	Avge	Time (s)	Dev (%)
$GWKC_1$	240	9	35423.72	54815.17	54878.25	1593.60	0.03	54826.53	54917.44	866.80	0.05	**54786.92**	54853.76	1038.27	0.00
$GWKC_2$	320	10	62561.85	100836.90	100918.54	4549.05	0.17	100800.33	100924.25	2054.68	0.14	**100662.53**	100934.34	1484.74	0.00
$GWKC_3$	400	10	105915.57	**171277.26**	171400.35	9295.64	0.00	171311.81	171523.42	3336.09	0.02	171613.59	172231.14	2061.75	0.20
$GWKC_4$	480	10	165358.69	**262584.23**	262830.96	12810.31	0.00	262646.37	263020.99	6557.67	0.02	263433.03	265207.46	2626.89	0.32
$GWKC_5$	200	5	91037.70	**114163.64**	114237.00	471.45	0.00	**114163.64**	114338.39	246.05	0.00	114494.66	114846.27	1200.67	0.29
$GWKC_6$	280	7	101234.28	**140430.09**	140456.96	2358.40	0.00	140463.67	140556.57	788.71	0.02	140804.64	140929.71	1547.43	0.27
$GWKC_7$	360	8	116694.54	183282.64	186702.15	1537.36	1.11	190371.53	192578.18	258.45	5.02	**180481.56**	181610.82	1926.19	0.00
$GWKC_8$	440	10	116945.52	194312.60	194510.99	9598.42	0.02	**194273.58**	194454.29	5165.83	0.00	194988.74	195174.85	2330.34	0.37
$GWKC_9$	255	14	4197.81	4730.70	4740.42	2453.61	0.10	4737.32	4744.14	1582.86	0.24	**4725.58**	4728.05	864.89	0.00
$GWKC_{10}$	323	16	6014.12	6732.36	6747.10	7652.00	0.24	6721.16	6742.76	4332.24	0.07	**6713.92**	6717.76	1092.34	0.00
$GWKC_{11}$	399	18	8290.60	9243.05	9259.66	8835.74	0.27	9240.37	9257.23	7429.60	0.24	**9214.07**	9216.60	1356.99	0.00
$GWKC_{12}$	483	19	11079.18	12629.37	12649.21	9881.24	0.73	12659.15	12708.54	3449.06	0.97	**12526.17**	12543.04	1540.32	0.00
$GWKC_{13}$	252	26	3331.28	3653.07	3660.93	2012.17	0.68	3686.62	3965.98	74.31	1.61	**3628.30**	3638.50	632.72	0.00
$GWKC_{14}$	320	29	4719.74	5770.02	6045.20	2263.84	10.07	5826.43	6420.08	9.59	11.14	**5216.80**	5257.95	682.19	0.00
$GWKC_{15}$	396	33	6336.77	7077.48	7140.11	5597.98	0.89	8576.61	8576.61	0.00	22.27	**7010.41**	7023.12	855.25	0.00
$GWKC_{16}$	480	37	8254.46	9300.74	9339.45	16204.52	0.54	9347.02	9420.36	5307.29	1.04	**9250.98**	9268.30	1104.53	0.00
$GWKC_{17}$	240	22	2583.17	3089.99	3103.99	1631.86	0.76	3095.32	3117.79	431.31	0.93	**3065.46**	3068.29	618.70	0.00
$GWKC_{18}$	300	27	3576.10	4528.16	4582.44	2410.31	7.22	5083.70	5138.01	0.10	20.37	**4221.14**	4244.60	630.47	0.00
$GWKC_{19}$	360	33	4905.86	5570.35	5589.12	5535.25	0.84	5600.05	5631.68	763.56	1.37	**5523.38**	5531.78	853.43	0.00
$GWKC_{20}$	420	38	6568.06	7413.58	7473.69	6264.59	2.64	8453.96	8453.96	0.00	17.04	**7223.08**	7240.86	881.92	0.00
Avge	347.40	19.05	43251.45	65072.07	65313.33	5647.86	1.32	65594.26	65824.53	2132.71	4.13	**64979.25**	**65213.36**	**1266.50**	**0.07**

yielding the best average results was chosen. Table 1 shows the best values found for all parameters.

Our computational experiments are summarized in Tables 2 and 3. The column headings are: the instance name, the number of customers, the number of vehicles used, the larger of the two lower bounds LB1 and LB2 (column LB), the best solution found in five runs of the heuristic (column Best), the average solution found (column Avge), the average computational time in seconds over five runs of the heuristic (Time (s)), and the deviation (column Dev %) calculated as Dev(%) = 100(Best − Best *)/Best *, where Best * is the best known solution value obtained by any of the three heuristics for a given instance. Best values are shown in boldface.

On the CMT instances, all heuristics exhibit a good behavior, in particular ALNS which identifies all best known solutions, except for instance CMT_5, and yields the best average solution values. It is important to mention that MA1 and MA2 provided slightly

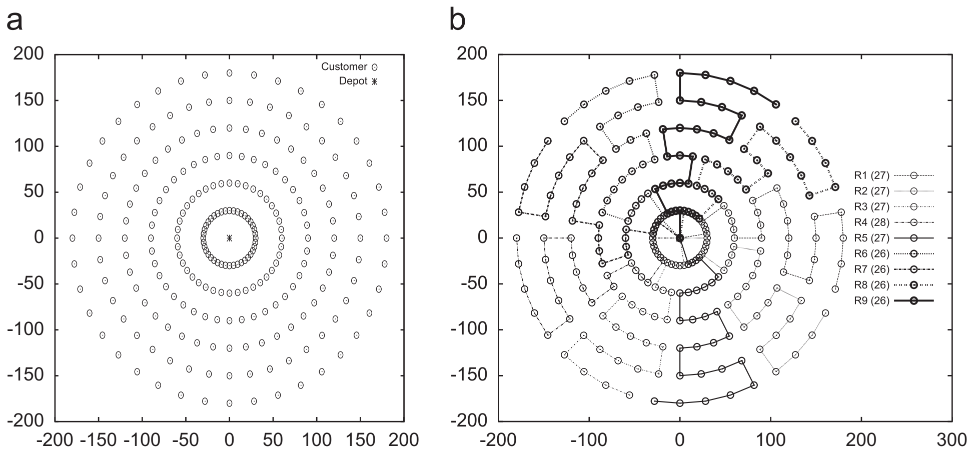

Fig. 1. Best known solution found by ALNS for instances $GWKC_1$. The number of customers served by each route is in parentheses. (a) $GWKC_1$—Vertices, (b) $GWKC_1$—Routes.

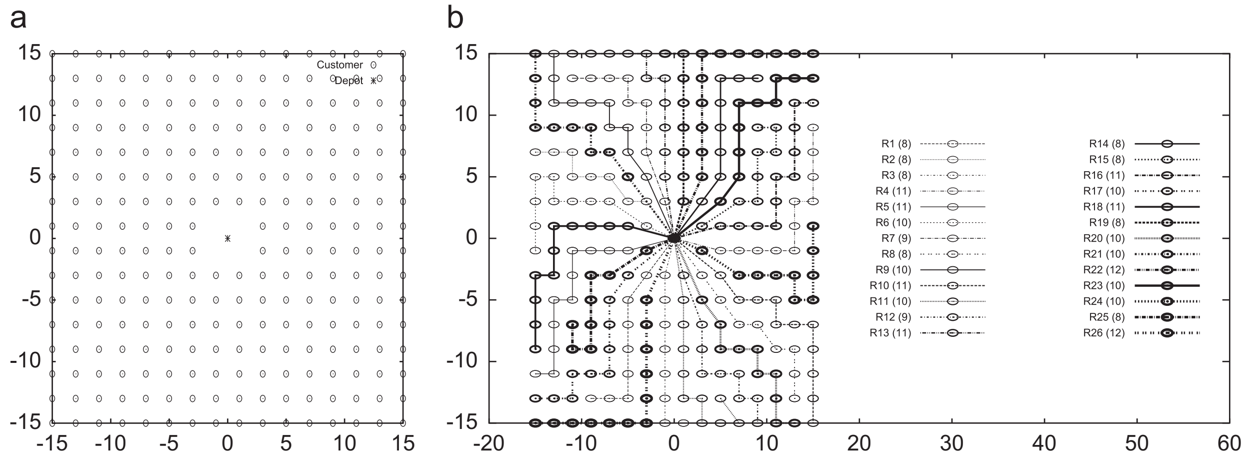

Fig. 2. Best known solution found by ALNS for instances $GWKC_{13}$. The number of customers served by each route is in parentheses. (a) $GWKC_{13}$—Vertices, (b)$GWKC_{13}$—Routes.

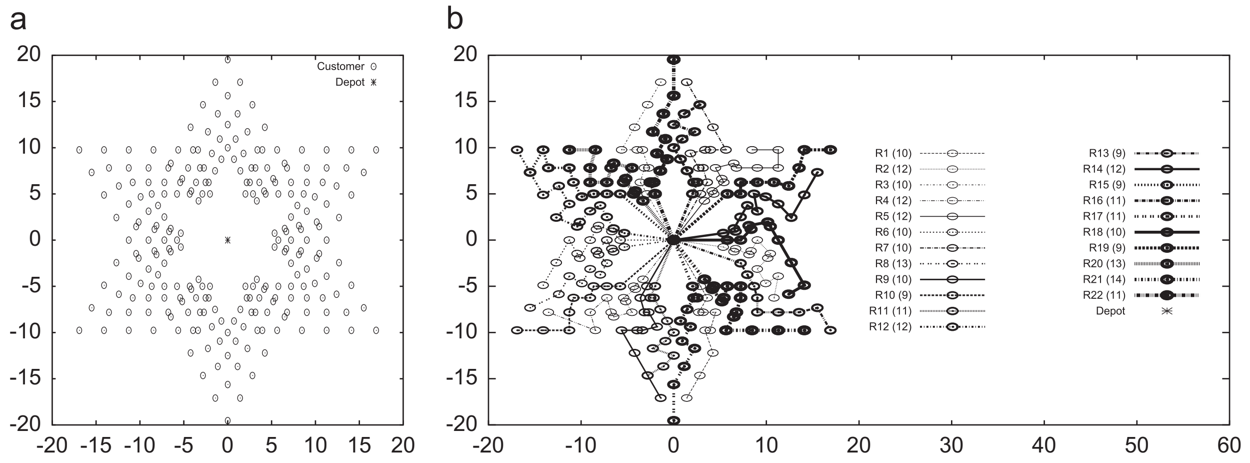

Fig. 3. Best known solution found by ALNS for instances $GWKC_{17}$. The number of customers served by each route is in parentheses. (a) $GWKC_{17}$—Vertices, (b) $GWKC_{17}$—Routes.

different results from those reported in [9]. Ngueveu et al. [9] report only the best known solution for each instance found by MA1 and MA2. If we compare our solutions to those in [9], we obtain better results for instances CMT_2 and CMT_5 and a worse result for instance CMT_3. The average best results for MA1 goes up from 4340.58 in [9] to 4342.84 in our experiments.

On the GWKC instances (Table 3), ALNS found 15 best known solutions, whereas MA1 and MA2 found four and two best known solutions, respectively. On the average results, ALNS again provides good solutions. Our heuristic took on average 1266.50 s to converge, which is faster than the times obtained with MA_1 and MA_2.

Figs. 1–3 show the best known solutions found by our ALNS heuristic for instances $GWKC_1$, $GWKC_{13}$ and $GWKC_{17}$, respectively. Note that for each route, the return trip to depot is not shown.

5. Conclusions

In situations where the arrival times at the customers are important, such as after natural disasters, good routing methodologies must be developed to attend those in need in a fast and equitable manner. The cumulative capacitated vehicle routing problem objective function constitutes a good way to model such situations. We have provided a highly efficient local search algorithm for this problem. On test instances it outperforms the only two available published heuristics.

Acknowledgments

Glaydston Mattos Ribeiro acknowledges the National Council for Scientific and Technological Development (Process 201509/2009-1) and Espírito Santo Research Foundation (Process 45391998/09) for their financial support. Gilbert Laporte acknowledges the Canadian Natural Sciences and Engineering Research Council (Grant 39682-10). The authors thank Sandra U. Ngueveu, Christian Prins and Roberto Wolfler-Calvo for kindly providing their memetic algorithm code and for answering several questions.

References

[1] Lucena A. Time-dependent traveling salesman problem: the deliveryman case. Networks 1990;20:753–63.
[2] Fischetti M, Laporte G, Martello S. The delivery man problem and cumulative matroids. Operations Research 1993;41:1055–64.
[3] Picard J-C, Queyranne M. The time-dependent traveling salesman problem and its application to the tardiness problem in one-machine scheduling. Operations Research 1978;26:86–110.
[4] Stecco G, Cordeau J-F, Moretti E. A tabu search heuristic for a sequence-dependent and time-dependent scheduling problem on a single machine. Journal of Scheduling 2009;12:3–16.
[5] Heilporn G, Cordeau J-F, Laporte G. The delivery man problem with time windows. Discrete Optimization 2010;7:269–82.
[6] Park J, Kim B-I. The school bus routing problem: a review. European Journal of Operational Research 2010;202:311–9.
[7] Ogryczak W. Inequality measures and equitable approaches to location problems. European Journal of Operational Research 2000;122:374–91.
[8] Campbell AM, Vandenbussche D, Hermann W. Routing for relief efforts. Transportation Science 2008;42:127–45.
[9] Ngueveu SU, Prins C, Wolfler-Calvo R. An effective memetic algorithm for the cumulative capacitated vehicle routing problem. Computers & Operations Research 2010;37:1877–85.
[10] Nolz PC, Doerner KF, Gutjahr WJ, Hartl RF. A bi-objective metaheuristic for disaster relief operation planning. In: Coello CAC, Dhaenens C, Jourdan L, editors. Advances in multi-objective nature inspired computing. Berlin: Springer-Verlag; 2010. p. 167–87.
[11] Augerat P, Belenguer J, Benavent E, Corberán A, Naddef D, Rinaldi G. Computational results with a branch and cut code for the capacitated vehicle routing problem. Technical Report 949-M, Université Joseph Fourier, Grenoble; 1995.
[12] Golden BL, Wasil EA, Kelly JP, Chao I-M. Metaheuristics in vehicle routing. In: Crainic TG, Laporte G, editors. Fleet management and logistics. Boston: Kluwer; 1998. p. 33–56.
[13] Prins C. A simple and effective evolutionary algorithm for the vehicle routing problem. Computers & Operations Research 2004;31:1985–2002.
[14] Ropke S, Pisinger D. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation Science 2006;40:455–72.
[15] Shaw P. A new local search algorithm providing high quality solutions to vehicle routing problems. Technical Report, University of Strathclyde, Glasgow; 1997.
[16] Pisinger D, Ropke S. Large neighborhood search. In: Gendreau M, Potvin J-Y, editors. Handbook of Metaheuristics of International Series in Operations Research & Management Science, vol. 146. Boston: Springer; 2010. p. 399–419.
[17] Ropke S, Pisinger D. A unified heuristic for a large class of vehicle routing problems with backhauls. European Journal of Operational Research 2006;171:750–75.
[18] Pisinger D, Ropke S. A general heuristic for the vehicle routing problem. Computers & Operations Research 2007;34:2403–35.
[19] Azi N, Gendreau M, Potvin J.-Y. An adaptive large neighborhood search for a vehicle routing problem with multiple trips. Technical Report 2010-08, CIRRELT, Montréal, Available at 〈https://www.cirrelt.ca/DocumentsTravail/CIRRELT-2010-08.pdf〉; 2010.
[20] Pepin A-S, Desaulniers G, Hertz A, Huisman D. A comparison of five heuristics for the multiple depot vehicle scheduling problem. Journal of Scheduling 2009;12:17–30.
[21] Kruskal JB. On the shortest spanning subtree of a graph and the traveling salesman problem. In: Proceedings of the American Mathematical Society, vol. 7. American Mathematical Society; 1956. p. 48–50.
[22] Potvin J-Y, Rousseau J-M. A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. European Journal of Operational Research 1993;66:331–40.
[23] Christofides N, Mingozzi A, Toth P. The vehicle routing problem. In: Christofides N, Mingozzi A, Toth P, Sandi C, editors. Combinatorial optimization. Chichester: Wiley; 1979. p. 315–38.