Available online at www.sciencedirect.com
SCIENCE DIRECT®
European Journal of Operational Research 147 (2003) 629–643
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/dsw

Discrete Optimization

A guided local search heuristic for the capacitated arc routing problem

Patrick Beullens *, Luc Muyldermans, Dirk Cattrysse, Dirk Van Oudheusden

Centre for Industrial Management, Katholieke Universiteit Leuven, Celestijnenlaan 300A, B-3001 Heverlee, Belgium

Accepted 10 April 2002

Abstract

This paper presents a new local search algorithm for the capacitated arc routing problem (CARP). The procedure uses single vehicle moves and moves that operate on two routes, both derived from a node routing context but properly adapted to work well for arc routing problems. We combine the algorithm with the meta-heuristic guided local search and further use the mechanisms of neighbor lists and edge marking to improve the solution quality and to save computation time. Experiments on standard benchmark problems from the literature show that our algorithm outperforms the existing heuristics for the CARP. On a set of new test problems, the local search approach consistently produces high quality solutions and often detects an optimal solution within limited computation time.

Keywords: Routing; Capacitated arc routing problem; Local search

1. Introduction

In the capacitated arc routing problem (CARP) (Golden and Wong, 1981), we are given an undirected, connected graph $G = (V, E)$ with vertex set V and edge set E. Every edge $e \in E$ has a non-negative cost or length c_e and a non-negative demand for service q_e. The edges with positive demand make up the subset of the required edges E_R. Given a vehicle capacity Q, the CARP consists of finding a set of vehicle routes of minimum cost, such that every required edge is serviced by exactly one vehicle, each route starts and ends at a pre-specified vertex v_0 (the depot) and the total demand serviced by a route does not exceed the vehicle capacity Q. Golden and Wong (1981) show that the CARP is NP-hard. In fact, they prove that even finding a solution with a cost (counting only the deadheading) less than 5/3 times the optimal cost, is NP-hard.

Several, mostly heuristic, solution procedures for the CARP are reported in the literature. Surveys on these algorithms and the various applications of the CARP, as well as on other related

* Corresponding author. Tel.: +32-16-32-24-99; fax: +32-16-32-29-86.

E-mail addresses: patrick.beullens@cib.kuleuven.ac.be (P. Beullens), luc.muyldermans@cib.kuleuven.ac.be (L. Muyldermans), dirk.cattrysse@cib.kuleuven.ac.be (D. Cattrysse), dirk.vanoudheusden@cib.kuleuven.ac.be (D. Van Oudheusden).

0377-2217/03/$ - see front matter
doi:10.1016/S0377-2217(02)00334-X

arc routing problems, can be found in Assad and Golden (1995), Eiselt et al. (1995) and the recent book of Dror (2000). In this paper, we present a new local search algorithm for the CARP. In contrast with the extensive literature on local search approaches for node routing problems, only few local search heuristics are available for the CARP. In the context of winter gritting (salt spreading on roads in wintertime), Eglese (1994) presents a solution procedure for a multi-depot CARP with additional side constraints. In a first stage, an Eulerian graph is partitioned into small cycles and an initial solution is constructed by aggregating cycles into routes using a greedy saving heuristic. Next, within a simulated annealing framework, cycles are exchanged between the routes to improve the solution. The use of cycles to build routes in the CARP, is criticized in Eglese and Li (1994). Obviously, when the total demand serviced in the cycles is rather large compared with the truck capacity, it will be more difficult to construct good routings by aggregating cycles instead of individual edges. Another approach using cycles to build routings is given in Amberg et al. (2000). More efficient algorithms for the CARP include the tabu search procedures of Greistorfer (1994, 2000) and of Hertz et al. (2000); the latter, called CARPET, is one of the best procedures developed so far. The success of CARPET relies on a number of routines to perform edge removal, re-insertion and re-optimization operations. Some of the routines were originally developed to solve the undirected rural postman problem (URPP), Hertz et al. (1999). CARPET uses a neighborhood structure similar to that of TABUROUTE, a tabu search heuristic developed for the vehicle routing problem (VRP) by Gendreau et al. (1994). The subroutines in CARPET are however rather intricate. Recently, Muyldermans et al. (2001a) developed k-opt local search algorithms for (uncapacitated) arc and general routing problems. These procedures are very similar to the well-known k-opt procedures for the traveling salesman problem (TSP) (Croes, 1958; Lin, 1965), and are much simpler than the approaches developed by Hertz et al. (1999). In this paper, we extend the procedures of Muyldermans et al. (2001a) to deal with CARP. New moves (*relocate*, *exchange* and *cross*) operating between two routes, are introduced, as was done for the VRP in Savelsbergh (1988). We present a local search algorithm for the CARP embedded within a guided local search procedure (GLS), a meta-heuristic successfully applied to several combinatorial optimization problems among which: the TSP (Voudouris and Tsang, 1999), the VRP (Kilby et al., 1999) and the URPP (Muyldermans et al., 2001a). Experiments on standard benchmark problems from the literature and newly developed instances indicate that the new algorithm is capable of finding optimal or near-optimal solutions within a limited computation time.

The remainder of the paper is organized as follows. Section 2 describes the solution representation, the neighborhood moves and some implementation issues, and finally gives a high-level description of the local search algorithm for the CARP. Computational experiments are presented in Section 3 and the conclusions follow in Section 4.

2. A local search algorithm for the CARP

2.1. Solution representation

As for the URPP in Muyldermans et al. (2001a), we construct a CARP solution in a directed graph $D(V,A)$ instead of in the original undirected graph $G(V,E)$. The reason to do so is that the direction of travel along an arc is uniquely defined while this is not the case for edges. The graph D has the same vertex set V as G, the arc set A contains for every edge $e = (v_i, v_j) \in E$, the arcs $a = (v_i, v_j)$ and $a^{\mathrm{Rev}} = (v_j, v_i)$, both with cost c_e, and demand q_e. The arcs a and a^{Rev} are called representatives of edge e, and only one representative of a required edge has to be serviced in a CARP solution. In $D = (V,A)$, we define the bijective function $reversal : A \to A$ such that for every arc $a = (v_i, v_j) \in A \mapsto a^{\mathrm{Rev}} = reversal(a)$ and,

(i) $a^{\mathrm{Rev}} = (v_j,\ v_i)$,
(ii) $reversal(a^{\mathrm{Rev}}) = a$.

Property (i) ensures that *reversal* deserves its name while property (ii) ensures that *reversal* be-

haves properly when E contains parallel edges. A vehicle route S_i servicing a subset of the required edges $E_{R'} \subseteq E_R$ with $|E_{R'}| = m$ is represented by a $V^2 \times A^m$ vector, $S_i = (v_0, \sigma_1, \ldots, \sigma_j, \ldots, \sigma_m, v_0)$, where v_0 is the depot vertex; σ_j, for $j = 1, \ldots, m$, denotes a representative arc of an edge of $E_{R'}$ on the $j+1^{\text{th}}$ position of S_i, and S_i contains exactly one representative of every edge in $E_{R'}$. Since the cost of servicing the edges in E_R is a constant (sunk cost), we only take the deadheading (the distance without servicing) into account. For a tour S_i, the deadheading $z(S_i)$, is given by

$$z(S_i) = \sum_{j=1}^{m-1} d(h(\sigma_j), t(\sigma_{j+1})) + d(v_0, t(\sigma_1)) + d(h(\sigma_m), v_0), \quad (1)$$

where $h(\sigma_j)$, $t(\sigma_j)$ denote the head and tail of arc σ_j and $d(h(\sigma_j), t(\sigma_{j+1}))$ is the length of a shortest path between $h(\sigma_j)$ and $t(\sigma_{j+1})$. The total load serviced by S_i is denoted by $q(S_i)$. Route S_i is feasible if $q(S_i) \leqslant Q$, otherwise it is infeasible and the excess demand with respect to the vehicle capacity is denoted by $ex(S_i)$. A solution S for the CARP is represented by a collection of routes S_i, $i = 1, \ldots, K$, such that every required edge $e \in E_R$ has exactly one representative in one of the K routes. The solution is feasible, if $ex(S) = \sum_{i=1}^{K} ex(S_i) = 0$. As in Hertz et al. (2000), we allow intermediate infeasible solutions during the search process and evaluate a solution S according to

$$z(S) = \sum_{i=1}^{K} z(S_i) + \alpha ex(S), \quad (2)$$

where α is a self-adjusting parameter (see Section 2.3.3). The number of routes K is chosen suitably, to allow feasible solutions. Our procedure can terminate with a solution containing empty routes $S_i = (v_0, v_0)$, which, obviously, can be disregarded in the final solution.

2.2. Neighborhood moves

Single vehicle moves (*flip*, *reverse* and *dir-opt*) in an arc routing context are described in detail in Muyldermans et al. (2001a). We therefore summarize them and refer to the original reference for details. The other moves (*relocate*, *exchange* and *cross*) operate on two routes and are well-known from a node routing context (Savelsbergh, 1988). We illustrate how they can be applied for arc routing.

2.2.1. Single vehicle moves

Consider a route $S_i = (v_0, \sigma_1, \ldots, \sigma_j, \sigma_{j+1}, \ldots, \sigma_{j+k}, \sigma_{j+k+1}, \ldots, \sigma_m, v_0)$ and call $s_{j+1,k} = (\sigma_{j+1}, \ldots, \sigma_{j+k})$ a subsequence of route S_i starting at σ_{j+1} of size k ($1 \leqslant j+1 \leqslant m$ and $1 \leqslant k \leqslant m-j$). The deadheading associated with $s_{j+1,k}$, is denoted by $z(s_{j+1,k}) = \sum_{h=j+1}^{j+k-1} d(h(\sigma_h), t(\sigma_{h+1}))$.

flip$(s_{j+1,k})$: The move *flip*$(s_{j+1,k})$ replaces a subsequence with size $\geqslant 2$, $s_{j+1,k} = (\sigma_{j+1}, \ldots, \sigma_{j+k})$, by the subsequence $s^{\text{flip}}_{j+1,k} = (\sigma^*_{j+1}, \ldots, \sigma^*_{j+k})$ where $\sigma^*_{j+h} = \sigma_{j+k+1-h}$ for $h = 1, \ldots, k$. This move is illustrated in Fig. 1. Clearly, the new route contains the same representatives as the original route, but the sequence of the representatives in $s_{j+1,k}$ is 'reversed'. Hence, other shortest paths will link consecutive representatives in $s^{\text{flip}}_{j+1,k}$, such that, in general, $z(s_{j+1,k}) \neq z(s^{\text{flip}}_{j+1,k})$ and the length should be calculated.

reverse$(s_{j+1,k})$: The move *reverse*$(s_{j+1,k})$ replaces a subsequence with size $\geqslant 1$, $s_{j+1,k} = (\sigma_{j+1}, \ldots, \sigma_{j+k})$, by the subsequence $s^{\text{reverse}}_{j+1,k} = (\sigma^*_{j+1}, \ldots, \sigma^*_{j+k})$ where $\sigma^*_{j+h} = \mathit{reversal}(\sigma_{j+k+1-h})$ for $h = 1, \ldots, k$ (Fig. 2). The only difference with *flip*$(s_{j+1,k})$ is that each representative on the subsequence is replaced by its reversal and that this move makes sense for subsequences of size one. Due to the symmetry of shortest path distances when the direction of travel is changed, it is clear that $z(s_{j+1,k}) = z(s^{\text{reverse}}_{j+1,k})$.

dir-opt$(s_{j+1,k})$: The move *dir-opt*$(s_{j+1,k})$ replaces a subsequence with size $\geqslant 1$, $s_{j+1,k} = (\sigma_{j+1}, \ldots, \sigma_{j+k})$, by the subsequence $s^{\text{dir-opt}}_{j+1,k} = (\sigma^*_{j+1}, \ldots, \sigma^*_{j+k})$ where $\sigma^*_{j+h} = \sigma_{j+h} \vee \mathit{reversal}(\sigma_{j+h})$ for $h = 1, \ldots, k$, such that $z(s^{\text{dir-opt}}_{j+1,k}) + d(v, t(\sigma^*_{j+1})) + d(h(\sigma^*_{j+k}), v')$ is minimized (minimal deadheading), with $v = v_0$ if $j+1 = 1$, $v = h(\sigma_j)$ otherwise; and $v' = v_0$ if $j+k = m$, $v' = t(\sigma_{j+k+1})$ otherwise. Thus *dir-opt*$(s_{j+1,k})$ selects the 'right' representatives (directions) in $s_{j+1,k}$, without changing the sequence in which each edge is serviced in the route. As shown in Muyldermans et al. (2001a), this can

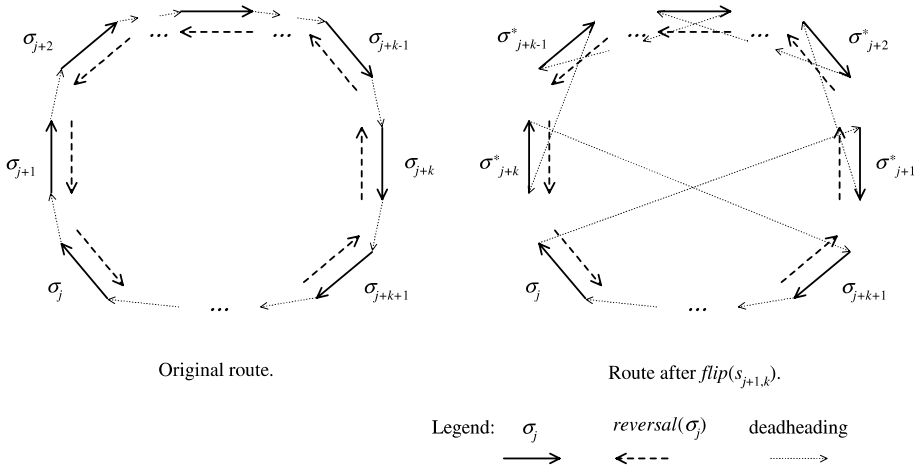

Fig. 1. The $flip(s_{j+1,k})$ move.

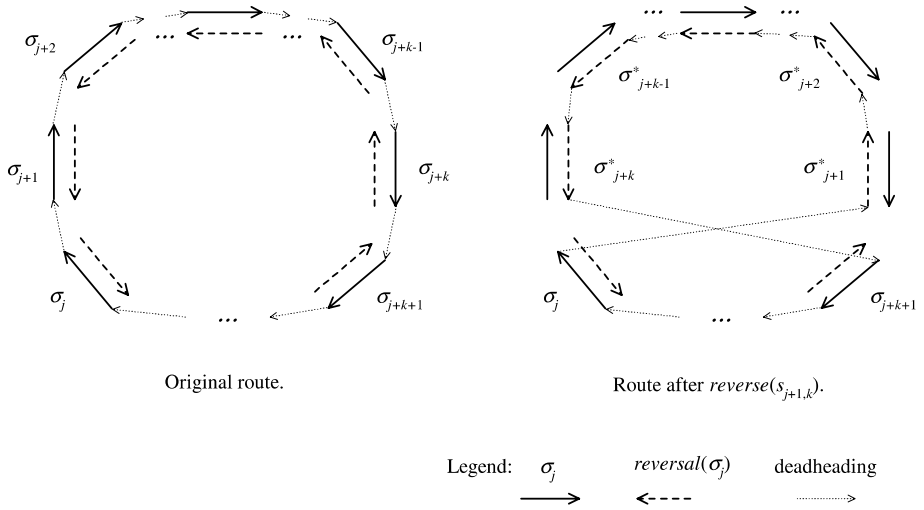

Fig. 2. The $reverse(s_{j+1,k})$ move.

be done by solving a shortest path problem in a staged network in $O(k)$ time.

We use the former moves in the local search algorithm in a 2-opt way; meaning, that we apply the moves on one single subsequence at a time. Let $s_{j+1,k}$ denote the subsequence under investigation. Checking if $reverse(s_{j+1,k})$ leads to an improvement can be done, using local information only, in constant time. Performing $reverse(s_{j+1,k})$ requires the updating of the representatives in $s_{j+1,k}$, spending time proportional to the size of the subsequence, $O(k)$, that is reversed. Using the move $flip(s_{j+1,k})$ instead, is not a good idea. Testing for an improvement is slower than with *reverse*, since it cannot be based on local information only ($z(s_{j+1,k}) \neq z(s^{\text{flip}}_{j+1,k})$), and, more importantly, a large part of the solution space would not be attainable since always the same representatives are used. Taking *dir-opt*°*flip* (*dir-opt* after *flip*) as a single move however, allows the exploration of new combinations of representatives. Checking for an improvement with this move as well as carrying

out the move takes $O(k)$ time. Finally, a *hybrid* strategy using *reverse* and *dir-opt°flip* can be considered as well: first test *reverse*, if it leads to an improvement, perform the move; otherwise if the size of $s_{j+1,k}$ is smaller than a pre-specified k^*, test *dir-opt°flip* and perform the move if it improves the route.

2.2.2. Moves between two routes

By exchanging edge representatives between different routes the deadheading distance and the excess demand may change. Thus, both have to be checked for an improvement. Let σ_p denote an edge representative in route S_i and σ_q an edge representative in route S_j.

relocate$(S_i, \sigma_p, S_j, \sigma_q)$: *relocate* removes σ_p from route S_i and inserts σ_p or *reversal*(σ_p), depending on which of the two gives the best improvement, after σ_q in route S_j. Testing for an improvement and performing the move can be done in constant time, since only local information is needed.

exchange$(S_i, \sigma_p, S_j, \sigma_q)$: *exchange* removes σ_p and σ_q from the routes S_i and S_j, and re-inserts σ_p or *reversal*(σ_p) on position $q+1$ in S_j and σ_q or *reversal*(σ_q) on position $p+1$ in route S_i. Again, the move yielding the best improvement is executed and checking for an improvement and carrying out the move can be done in constant time.

cross$(S_i, \sigma_p, S_j, \sigma_q)$: *cross* swaps the end portions of two vehicle routes. This means that the parts $(\sigma_{p+1}, \ldots, v_0)$ and $(\sigma_{q+1}, \ldots, v_0)$ of the original routes S_i and S_j, are pasted after respectively σ_q in S_j and σ_p in S_i to form two new routes. It is easy to see that *cross* can combine two routes into one, if the capacity constraint allows it. Testing if *cross* improves the deadheading and the excess demand (capacity violation) can be done in constant time. To check the capacity constraint in constant time however, we need to store, along each route, for every arc in the route, the cumulative demand serviced so far. Performing a cross move requires that the cumulative demands for the arcs in the swapped portions are updated. Moreover, if we want to evaluate *cross* in constant time, each time a route changes by applying another move (*reverse*, *relocate* and *exchange*), the cumulative demands in this route have to be adapted, which increases the time requirement to perform these moves. Therefore, a better strategy is to try *cross* only occasionally with a linear time implementation.

Obviously, *relocate* and *exchange* can operate with subsequences instead of individual edge representatives. If so, one should consider applying *dir-opt* on the relocated or exchanged subsequences. Checking for improvement and performing such a move requires time proportional to the size of the subsequence. Finally, note that the start and end depot vertex of every route may be treated as special edge representatives (with the same head and tail), so that *relocate* can be used to move a required edge to an empty vehicle for example. However, we do not allow (except with a cross) the depot vertex (start or end) to be removed from any route.

2.3. Neighbor lists, edge marking and GLS

Instead of investigating complete neighborhoods, we only check some particular moves in the local search procedure, using the mechanisms of neighbor lists and edge marking.

2.3.1. Neighbor lists

Let $|E_R| = m$, then for every required edge e the neighbor list contains the mN_L $(0 < N_L \leqslant 1)$ closest required edges, sorted in non-decreasing value of the distance to e. The distance between e and another required edge e' is the minimal shortest path distance between two vertices on the respective edges, i.e. $\min\{d(v_i, v_j) : v_i \in e, v_j \in e'\}$. The parameter N_L is specified at the beginning of the procedure. Besides the usual saving in computation time (for the TSP see Johnson and McGeoch, 1997), neighbor lists allow the investigation of only the moves that look somewhat promising (edges that are close to each other should (possibly) be serviced after each other in the same route). Applied with *reverse* for example, let σ and σ' denote two arcs in the same route, representing the required edges e and e'. The edge e is chosen from an active list AL (see Section 2.3.2) and e' is chosen from the neighbor list of e. We investigate the move *reverse*$(s_{j+1,k})$, where, in case arc σ is serviced before σ' in the tour, it is assumed that $\sigma = \sigma_j$ and

$\sigma' = \sigma_{j+k}$ and, when arc σ is serviced after σ', $\sigma = \sigma_{j+k+1}$ and $\sigma' = \sigma_{j+1}$.

We also create a neighbor list for every start depot vertex. Furthermore, next to the required edges, each neighbor list contains the end depot vertex of every route. Thus, the starts and ends of each route are treated as different objects in the procedure, while for the CARP, they correspond to the same vertex.

2.3.2. Edge marking

Next to neighbor lists, computation times for the TSP can be reduced by using vertex marking (do not look bits), introduced by Bentley (1992). We apply this idea as follows. Every required edge can be marked or unmarked. Initially, they are all marked. Edges that are marked are stored in an active list AL. We begin searches from marked edges, in the order in which they appear in AL (FIFO). We unmark an edge after an unsuccessful search and remove it from AL. When AL is empty, the local search terminates. The (re-)marking of edges depends upon the move involved. For example, suppose e is the edge of AL currently under investigation, σ_j its representative in a route and, $reverse(s_{j+1,k})$ leads to improvement, then we mark the edges corresponding with σ_{j+1}, σ_{j+k} and σ_{j+k+1} and append them to AL, unless they are already in the list. The edge marking policy for the different moves considered in the local search procedure, is shown in Table 1.

We allow the start depot vertex of a route to be marked as well and it thus may appear on the AL. Obviously, the moves and marking in Table 1 are only executed if they are meaningful and possible. For example, suppose we selected the start vertex v_0 of route S_i from AL and consider an edge from the neighbor list of v_0 represented by σ_q in route S_j; then only the moves $relocate(S_j, \sigma_q, S_i, v_0)$, $cross(S_i, v_0, S_j, \sigma_{q-1})$ and $exchange(S_i, \sigma_1, S_j, \sigma_q)$ with σ_1 the first required arc in S_i, are tested for possible improvement. To not overload the paper with specific details, in Sections 2.3.3 and 2.4, we will not further differentiate between cases where start and end vertices or 'real' edges are involved in the moves and marking process. It should be clear, however, that the algorithm includes the necessary tests to select and execute the moves and marking properly when start or end vertices are involved.

2.3.3. Guided Local Search

We embedded the local search for the CARP in GLS, a meta-heuristic originally proposed by

Table 1
The edge marking policy

Representative of edge e[a]	Representative of edge e'[b]	Move	Mark edges represented by[c]
σ_j in route S_i	σ_{j+k} in route S_i	$reverse(s_{j+1,k})$	$\sigma_{j+1}, \sigma_{j+k}, \sigma_{j+k+1}$
		$dir\text{-}opt(flip(s_{j+1,k}))$	idem
σ_{j+k+1} in route S_i	σ_{j+1} in route S_i	$reverse(s_{j+1,k})$	σ_j, σ_{j+1}, σ_{j+k}
		$dir\text{-}opt(flip(s_{j+1,k}))$	idem
σ_p in route S_i	σ_q in route S_j	$relocate(S_i, \sigma_p, S_j, \sigma_q)$	$\sigma_{p-1}, \sigma_{p+1}, \sigma_q, \sigma_{q+1}$
		$relocate(S_i, \sigma_p, S_j, \sigma_{q-1})$	$\sigma_{p-1}, \sigma_{p+1}, \sigma_{q-1}, \sigma_q$
		$relocate(S_j, \sigma_q, S_i, \sigma_p)$	$\sigma_{p+1}, \sigma_{q-1}, \sigma_q, \sigma_{q+1}$
		$relocate(S_j, \sigma_q, S_i, \sigma_{p-1})$	$\sigma_{p-1}, \sigma_{q-1}, \sigma_q, \sigma_{q+1}$
		$exchange(S_i, \sigma_p, S_j, \sigma_q)$	$\sigma_{p-1}, \sigma_{p+1}, \sigma_{q-1}, \sigma_q, \sigma_{q+1}$
		$exchange(S_i, \sigma_p, S_j, \sigma_{q+1})$	$\sigma_{p-1}, \sigma_{p+1}, \sigma_q, \sigma_{q+1}, \sigma_{q+2}$
		$exchange(S_i, \sigma_{p+1}, S_j, \sigma_q)$	$\sigma_{p+1}, \sigma_{p+2}, \sigma_{q-1}, \sigma_q, \sigma_{q+1}$
		$exchange(S_i, \sigma_p, S_j, \sigma_{q-1})$	$\sigma_{p-1}, \sigma_{p+1}, \sigma_{q-2}, \sigma_{q-1}, \sigma_q$
		$exchange(S_i, \sigma_{p-1}, S_j, \sigma_q)$	$\sigma_{p-2}, \sigma_{p-1}, \sigma_{q-1}, \sigma_q, \sigma_{q+1}$
		$cross(S_i, \sigma_p, S_j, \sigma_{q-1})$	$\sigma_{p+1}, \sigma_{q-1}, \sigma_q$
		$cross(S_i, \sigma_{p-1}, S_j, \sigma_q)$	$\sigma_{p-1}, \sigma_q, \sigma_{q+1}$

[a] Chosen from the AL.
[b] Chosen from the neighbor list of edge e.
[c] In the original routes (i.e. before the move is performed), and provided they are not yet marked.

Voudouris and Tsang (1996). GLS penalizes the deadheading $d(h(\sigma_j), t(\sigma_{j+1}))$ between two edge representatives in the same route: $d'(h(\sigma_j), t(\sigma_{j+1})) = d(h(\sigma_j), t(\sigma_{j+1})) + \lambda p(\sigma_j, \sigma_{j+1})$. Here, $p(\sigma_j, \sigma_{j+1})$ is the penalty parameter between σ_j and σ_{j+1} and $\lambda = 0.3z(S^0)/(m + K^0)$, with $|E_R| = m$, S^0 the local minimum obtained after the first call to the local search procedure and K^0 the number of non-empty vehicles used in S^0. Initially all penalty parameters are zero.

At the $k + 1^{\text{th}}$ GLS iteration, given solution S^k from the local search, look in the routes of S^k for the subsequence $s_{j,2}$ with the largest value for $d(h(\sigma_j), t(\sigma_{j+1}))/(1 + p(\sigma_j, \sigma_{j+1}))$. Then, increment the penalty parameters $p(\sigma_j, \sigma_{j+1})$ and $p(reversal(\sigma_{j+1}), reversal(\sigma_j))$ (due to symmetry) with one, mark the edges corresponding with σ_j and σ_{j+1} and put them in AL. The local search procedure is called, using the modified costs d'. If the new solution S^{k+1} is feasible and, evaluated according to the original deadhead distances, improves the current best solution, we replace the current best by S^{k+1}; otherwise, we start a new GLS iteration. A combination of two stop criteria is used: the first is a limit on the number of GLS iterations, the second checks if the gap with a known lower bound is smaller than a prescribed value. We also adapt the parameter α in Eq. (2) in the GLS loop: α is halved (doubled) when the previous μ different solutions were (in)feasible. Initially, we set α to 10 times the maximal shortest path distance to reach the depot from a vertex on a required edge and, $\mu = 5$.

2.4. A CARP local search algorithm

We now give a high-level algorithmic description of the local search procedure. As mentioned earlier, not all features are described in detail. At iteration $k + 1$, the local search procedure starts with a solution S^k, a modified cost matrix d', the parameter α and the AL as an input. It generates the solution S^{k+1} which is possibly different from S^k, indicated by the Boolean variable *changed*.

LocalSearch($S^k, d', \alpha, AL; S^{k+1}, changed$)

Set *changed* := false; $S^{k+1} := S^k$.

START: If AL is empty, Stop; else select the first edge e from AL. All the edges in the neighbor lists are unexamined.

NEXT: Select the first unexamined edge e' in the neighbor list of e. If all edges are examined, go to UNMARK. When the representatives of e and e' are in the same route, go to SAME. Test the improvement with *relocate* and *exchange* (Table 1) and if there is an improvement, implement the best move and mark the edges involved (Table 1), set *changed* := true, call the new solution S^{k+1}, and go to START. If there is no improvement, test *cross*, and if successful, implement the best move, mark the edges involved, set *changed* := true, call the new solution S^{k+1}, and go to START. If no improvement is found, e' is examined. Go to NEXT.

SAME: Test improvement for 2-opt *hybrid*, and if successful, implement the move, mark the edges involved, set *changed* := true, call the new solution S^{k+1}, and go to START. If no improvement is found, e' is examined. Go to NEXT.

UNMARK: Delete e from AL, and go to START.

In combination with GLS, the procedure is as follows:

GuidedLocalSearch

INITIALIZATION: Compute the distance matrix d (deadheading between edge representatives) and initialize the penalty matrix p (all penalties are zero). For a given value of N_L, construct the neighbor lists, and add all required edges to AL. Set the modified distance matrix $d' := d$. Construct an initial feasible solution S; set $z^* := z(S)$, $S^* := S$. Calculate a lower bound z_{lb} on z^* and specify values for the GLS iteration limit k_{max}, α and μ. Call LocalSearch (S, d', α, $AL; S^0, changed$) and calculate λ. If $ex(S^0) = 0$, set $S^* := S^0$ and $z^* := z(S^0)$. Set the GLS iteration counter $k := 0$. To check if all the μ previous and different solutions found with GLS were feasible or infeasible, use the parameters μ_f and μ_i. Initially, $\mu_f := 0$, $\mu_i := 0$.

BEGIN: If $k = k_{max}$ or $z^* = z_{lb}$, go to END, else do the following: Look in the routes of S^k for the

subsequence of size two with the largest weighed value for the deadheading. Penalize this deadhead path and mark the adjacent edges (add them to AL) and update d'. Call LocalSearch $(S^k, d', \alpha, AL;\ S^{k+1}, changed)$. If $changed = \text{false}$, set $k := k + 1$ and go to BEGIN. If $ex(S^{k+1}) > 0$, set $\mu_i := \mu_i + 1$ and $\mu_f := 0$; else set $\mu_f := \mu_f + 1$, $\mu_i := 0$, and if $z(S^{k+1}) < z^*$, set $z^* := z(S^{k+1})$, $S^* := S^{k+1}$. If $\mu_i = \mu$, set $\alpha := 2\alpha$, $\mu_i := 0$, $\mu_f := 0$; else, if $\mu_f = \mu$, set $\alpha := \alpha/2$, $\mu_i := 0$, $\mu_f := 0$. Set $k := k + 1$ and go to BEGIN.

END: If $z^* = z_{lb}$, return z^* and S^*, else set α very high to allow only feasible solutions, add all required edges to AL and call LocalSearch$(S^*, d, \alpha, AL;\ S^*, changed)$, using the original distances d, and return z^* and S^*.

3. Computational experiments

3.1. The test problems and calculation of the lower bounds

We tested our algorithm on CARP instances from the literature as well as on new instances. The test problems from the literature are the following:

- 23 instances of Golden et al. (1983), named gdb1–gdb23 in Table 2,
- 6 instances of Kiushi et al. (1995), named kshs1–kshs6 in Table 2,
- 34 instances of Benavent et al. (1992), denoted 1.A–10.D in Table 2.

The data of these problems can be found in Benavent (1997). The individual instance characteristics are shown in Table 2: $|V|$, $|E_R|$, the continuous lower bound K^* on the number of vehicles to use and the best upper UB and lower bound LB reported in the literature (Belenguer and Benavent, 2000; Hertz et al., 2000). Note that we adapted the bounds and the gaps to take into account only the deadheading distance.

The new instances (100 in total) are created by partitioning the inter-city road network in Flanders (Belgium) into districts, in the context of winter gritting. Each district contains one depot where the vehicles are stationed and from where the salt spreading activities, within the district borders, are organized. Table 3 lists the characteristics $(|V|, |E|, |E_R|)$ of the subnetworks or districts, named C1–C25 and E1–E25 in Table 3, obtained by applying two districting procedures on the road network in Flanders (Muyldermans et al., 2001b). In each district we solve two CARPs: once with $Q = 300$ and once with $Q = 600$. For each instance, the continuous lower bound K^* on the number of trucks to use and, a lower bound LB on the optimal solution are reported in Table 3.

To calculate a lower bound for the instances in Table 3, we implemented a cutting plane approach based on the *supersparse* integer linear programming formulation for the CARP, Belenguer and Benavent (1998) and Benavent et al. (2000); and used exact and heuristic separation routines to identify violated valid inequalities (capacity constraints and odd cut sets). At the end of the cutting plane phase (i.e. when no additional violated constraints could be detected with both separation routines), we invoke CPLEX' standard branch and bound method to end up with an integer lower bound. In the next iteration, if new violated constraints are found, we cut off the integer lower bound and invoke branch and bound, otherwise the procedure is halted. The quality of the bounds obtained with this approach is, in general, very good, but can be further improved by implementing additional separation routines for the so-called *disjoint path* inequalities (Belenguer and Benavent, 2000). These routines are not included in our lower bound procedure. Therefore, we were not able to improve the best known lower bounds for the instances from the literature with this procedure. Using a totally different approach, in particular, the (somewhat modified) exact branch and bound scheme for the CARP of Hirabayashi et al. (1992), we were able to calculate the optimal solutions for gdb12 and kshs4 in Table 2. The latter procedure, however, is only applicable to very small instances.

Table 2
The test problems from the literature

Problem	$(\|V\|, \|E_R\|, K^*)$	LB	UB	Gap (%)	Problem	$(\|V\|, \|E_R\|, K^*)$	LB	UB	Gap (%)
gdb1	(12, 22, 5)	64	64	0.00	1.A	(24, 39, 2)	27	27	0.00
gdb2	(12, 26, 6)	48	48	0.00	1.B	(24, 39, 3)	27	27	0.00
gdb3	(12, 22, 5)	42	42	0.00	1.C	(24, 39, 8)	89	99	11.24
gdb4	(11, 19, 4)	49	49	0.00	2.A	(24, 34, 2)	42	42	0.00
gdb5	(13, 26, 6)	61	61	0.00	2.B	(24, 34, 3)	74	74	0.00
gdb6	(12, 22, 5)	38	38	0.00	2.C	(24, 34, 8)	270	272	0.74
gdb7	(12, 22, 5)	63	63	0.00	3.A	(24, 35, 2)	16	16	0.00
gdb8	(27, 46, 10)	134	138	2.99	3.B	(24, 35, 3)	22	22	0.00
gdb9	(27, 51, 10)	84	89[a]	5.95	3.C	(24, 35, 7)	73[b]	73	0.00
gdb10	(12, 25, 4)	23	23	0.00	4.A	(41, 69, 3)	57	57	0.00
gdb11	(22, 45, 5)	39	39	0.00	4.B	(41, 69, 4)	69	69	0.00
gdb12	(13, 23, 7)	122[c]	122	0.00	4.C	(41, 69, 5)	85	87	2.35
gdb13	(10, 28, 6)	27	35	29.63	4.D	(41, 69, 9)	179[a]	203	13.41
gdb14	(7, 21, 5)	4	4	0.00	5.A	(34, 65, 3)	56	56	0.00
gdb15	(7, 21, 4)	2	2	0.00	5.B	(34, 65, 4)	79	79	0.00
gdb16	(8, 28, 5)	8	8	0.00	5.C	(34, 65, 5)	102	107	4.90
gdb17	(8, 28, 5)	7	7	0.00	5.D	(34, 65, 9)	204	226	10.78
gdb18	(9, 36, 5)	6	6	0.00	6.A	(31, 50, 3)	33	33	0.00
gdb19	(8, 11, 3)	10	10	0.00	6.B	(31, 50, 4)	41	43	4.88
gdb20	(11, 22, 4)	16	16	0.00	6.C	(31, 50, 10)	121	127	4.96
gdb21	(11, 33, 6)	7	7	0.00	7.A	(40, 66, 3)	30	30	0.00
gdb22	(11, 44, 8)	9	9	0.00	7.B	(40, 66, 4)	34	34	0.00
gdb23	(11, 55, 10)	10	10	0.00	7.C	(40, 66, 9)	84	85	1.19
					8.A	(30, 63, 3)	39	39	0.00
Average gap (%)				1.68	8.B	(30, 63, 4)	48	48	0.00
Maximum gap (%)				29.63	8.C	(30, 63, 9)	170	181	6.47
Number of optima				20	9.A	(50, 92, 3)	45	45	0.00
					9.B	(50, 92, 4)	48	48	0.00
kshs1	(8, 15, 4)	5956	5956	0.00	9.C	(50, 92, 5)	54	54	0.00
kshs2	(10, 15, 4)	3135	3135	0.00	9.D	(50, 92, 10)	104	121	16.35
kshs3	(6, 15, 4)	1438	1438	0.00	10.A	(50, 97, 3)	52	52	0.00
kshs4	(8, 15, 4)	3507[c]	3507	0.00	10.B	(50, 97, 4)	60	60	0.00
kshs5	(8, 15, 3)	1679	1679	0.00	10.C	(50, 97, 5)	70	70	0.00
kshs6	(9, 15, 3)	1672	1672	0.00	10.D	(50, 97, 10)	148[b]	160	8.11
Average gap (%)				0.00	Average gap (%)				2.51
Maximum gap (%)				0.00	Maximum gap (%)				16.35
Number of optima				6	Number of optima				22

[a] Reported only in Belenguer and Benavent (2000).
[b] Reported only in Hertz et al. (2000).
[c] Improved bound found by the authors.

3.2. Computational results

The local search algorithm for the CARP is coded in C/C++ and run on a Pentium II, 500 MHz, 64 Mb RAM. The results presented are based on single runs for each instance and the gaps reported only take the deadheading distance into account. In all the experiments we used $N_L = 0.5$ if $|E_R| \geqslant 40$, $N_L = 1$ otherwise and, when we used *hybrid*, *dir-opt°flip* was only evaluated for subsequences with size $k < 0.7|E_{R'}|$, with $|E_{R'}|$ the number of required edges currently serviced in the route under investigation.

Table 4 shows the results for several variants of the local search procedure on the 25 C-instances, $Q = 300$. In some preliminary tests, these problems

Table 3
The new CARP instances

Problem	$(\|V\|, \|E\|, \|E_R\|)$	(K^*, LB) $Q = 300$	(K^*, LB) $Q = 600$	Problem	$(\|V\|, \|E\|, \|E_R\|)$	(K^*, LB) $Q = 300$	(K^*, LB) $Q = 600$
C1	(69, 98, 79)	(9, 1590)	(5, 725)	E1	(73, 105, 85)	(10, 1855)	(5, 1065)
C2	(48, 66, 53)	(7, 1095)	(4, 480)	E2	(58, 81, 58)	(8, 1580)	(4, 920)
C3	(46, 64, 51)	(6, 875)	(3, 415)	E3	(46, 61, 47)	(5, 750)	(3, 400)
C4	(60, 84, 72)	(8, 1285)	(4, 615)	E4	(70, 99, 77)	(9, 1580)	(5, 930)
C5	(56, 79, 65)	(10, 2410)	(5, 1040)	E5	(68, 94, 61)	(9, 2130)	(5, 1180)
C6	(38, 55, 51)	(6, 855)	(3, 485)	E6	(49, 66, 43)	(5, 670)	(3, 490)
C7	(54, 70, 52)	(8, 1735)	(4, 735)	E7	(73, 94, 50)	(8, 1780)	(4, 1080)
C8	(66, 88, 63)	(8, 1640)	(4, 615)	E8	(74, 98, 59)	(9, 2080)	(5, 1135)
C9	(76, 117, 97)	(12, 1775)	(6, 680)	E9	(93, 141, 103)	(12, 2160)	(6, 1145)
C10	(60, 82, 55)	(9, 2190)	(5, 900)	E10	(56, 76, 49)	(7, 1690)	(4, 1010)
C11	(83, 118, 94)	(10, 1725)	(5, 920)	E11	(80, 113, 94)	(10, 1810)	(5, 1015)
C12	(62, 88, 72)	(9, 1510)	(5, 680)	E12	(74, 103, 67)	(9, 1580)	(5, 900)
C13	(40, 60, 52)	(7, 1050)	(4, 690)	E13	(49, 73, 52)	(7, 1300)	(4, 835)
C14	(58, 79, 57)	(8, 1620)	(4, 920)	E14	(53, 72, 55)	(8, 1780)	(4, 1025)
C15	(97, 140, 107)	(11, 1765)	(6, 910)	E15	(85, 126, 107)	(9, 1555)	(5, 945)
C16	(32, 42, 32)	(3, 580)	(2, 170)	E16	(60, 80, 54)	(7, 1785)	(4, 775)
C17	(43, 56, 42)	(7, 1590)	(4, 675)	E17	(38, 50, 36)	(5, 1290)	(3, 605)
C18	(93, 133, 121)	(11, 2315)	(6, 930)	E18	(78, 110, 88)	(8, 1600)	(4, 835)
C19	(62, 84, 61)	(6, 1345)	(3, 650)	E19	(77, 103, 66)	(6, 1400)	(3, 685)
C20	(45, 64, 53)	(5, 665)	(3, 415)	E20	(56, 80, 63)	(7, 950)	(4, 610)
C21	(60, 84, 76)	(8, 1705)	(4, 695)	E21	(57, 82, 72)	(7, 1700)	(4, 905)
C22	(56, 76, 43)	(4, 1070)	(2, 690)	E22	(54, 73, 44)	(5, 1155)	(3, 790)
C23	(78, 109, 92)	(8, 1620)	(4, 715)	E23	(93, 130, 89)	(8, 1395)	(4, 705)
C24	(77, 115, 84)	(7, 1330)	(4, 620)	E24	(97, 142, 86)	(8, 1695)	(4, 975)
C25	(37, 50, 38)	(5, 905)	(3, 410)	E25	(26, 35, 28)	(4, 655)	(2, 430)

Table 4
Evaluation of the local search variants on the C-instances, $Q = 300$

Algorithm	#gls $= 10^5$			#gls $= 2.5 \times 10^5$			#gls $= 5 \times 10^5$		
	Gap (%)	Time (s)	N_{best}	Gap (%)	Time (s)	N_{best}	Gap (%)	Time (s)	N_{best}
LS *reverse* $\mu = 5$	3.64	44.93	14	3.41	105.10	20	3.37	199.88	20
LS *reverse* $\mu = 20$	3.62	40.95	14	3.42	97.64	18	3.40	190.09	19
LS *hybrid* $\mu = 5$	3.93	54.11	14	3.55	126.53	17	3.40	239.41	20
LS *hybrid* $\mu = 20$	3.67	50.21	15	3.47	119.52	17	3.38	229.47	19
LSX *reverse* $\mu = 5$	3.44	52.92	17	3.38	125.85	20	3.30	235.99	22
LSX *reverse* $\mu = 20$	3.55	50.04	15	3.39	118.41	19	3.31	224.22	21
LSX *hybrid* $\mu = 5$	3.47	63.07	18	3.38	149.14	20	3.31	283.02	20
LSX *hybrid* $\mu = 20$	3.47	57.94	18	3.36	138.56	20	3.35	270.05	21

appeared to be the most 'difficult' ones to solve (largest average gap). The variants tested are indicated in Table 4: LS and LSX denote respectively the local search variants without and with the *cross* move; *reverse* and *hybrid* further specify which single vehicle moves are considered and, the parameter μ is related to the frequency to adjust α in the objective function. In total, eight variants are tested for the three values of the GLS iteration limit: 10^5, 2.5×10^5 and 5×10^5. For each run, we report the average gap, the average solution time and the number of best solutions N_{best} found in the run.

Considering the computation time, the results indicate that an algorithm using *reverse* is on average 20–25% faster than when using *hybrid* (ceteris paribus). Similarly, the LS variants are 20–25% faster than their LSX counterparts and, setting $\mu = 20$ instead of $\mu = 5$, decreases the average solution time by 5–10%. The average computation time seems to grow almost linearly in the range of the GLS iteration limits considered. With respect to the solution quality, LSX variants perform systematically better than their LS relatives. Also, using *hybrid* instead of *reverse* does not seem to payoff. It is more difficult to draw firm conclusions about the effect of μ on the average solution quality,

Table 5
Computational results for the instances from the literature

Problem	#gls = 10^5			Problem	#gls = 10^5			#gls = 5×10^5		
	UB	Time (s)	Gap (%)		UB	Time (s)	Gap (%)	UB	Time (s)	Gap (%)
gdb1	64	0.00	0.00							
gdb2	48	0.06	0.00	1.A	27	0.06	0.00	27	0.06	0.00
gdb3	42	0.05	0.00	1.B	27	0.28	0.00	27	0.28	0.00
gdb4	49	0.00	0.00	1.C	99	45.26	11.24	99	198.23	11.24
gdb5	61	0.11	0.00	2.A	42	0.00	0.00	42	0.00	0.00
gdb6	38	0.05	0.00	2.B	74	0.27	0.00	74	0.27	0.00
gdb7	63	0.00	0.00	2.C	272	44.93	0.74	272	180.04	0.74
gdb8	138	36.36	2.99	3.A	16	0.06	0.00	16	0.06	0.00
gdb9	**84**	2.03	0.00	3.B	22	0.06	0.00	22	0.06	0.00
gdb10	23	0.00	0.00	3.C	73	0.60	0.00	73	0.60	0.00
gdb11	39	0.00	0.00	4.A	57	0.05	0.00	57	0.05	0.00
gdb12	122	0.00	0.00	4.B	69	0.11	0.00	69	0.11	0.00
gdb13	**27**	0.83	0.00	4.C	**85**	0.33	0.00	**85**	0.33	0.00
gdb14	4	0.00	0.00	4.D	**187**	63.00	4.47	**187**	275.89	4.47
gdb15	2	0.00	0.00	5.A	56	0.27	0.00	56	0.27	0.00
gdb16	8	0.11	0.00	5.B	79	0.38	0.00	79	0.38	0.00
gdb17	7	0.00	0.00	5.C	107	58.99	4.90	107	266.71	4.90
gdb18	6	0.06	0.00	5.D	**216**	56.74	5.88	**212**	241.15	3.92
gdb19	10	0.00	0.00	6.A	33	0.05	0.00	33	0.05	0.00
gdb20	16	0.16	0.00	6.B	43	31.64	4.88	43	148.85	4.88
gdb21	7	0.06	0.00	6.C	127	39.11	4.96	127	168.73	4.96
gdb22	9	0.06	0.00	7.A	30	0.06	0.00	30	0.06	0.00
gdb23	10	0.27	0.00	7.B	34	0.22	0.00	34	0.22	0.00
				7.C	85	48.01	1.19	85	225.75	1.19
Average time (s)			1.75	8.A	39	0.06	0.00	39	0.06	0.00
Average gap (%)			0.13	8.B	48	0.22	0.00	48	0.22	0.00
Maximum gap (%)			2.99	8.C	**176**	53.34	3.53	**174**	235.08	2.35
Number of optima			22	9.A	45	2.25	0.00	45	2.25	0.00
				9.B	48	0.60	0.00	48	0.60	0.00
kshs1	5956	0.00	0.00	9.C	54	1.16	0.00	54	1.16	0.00
kshs2	3135	0.00	0.00	9.D	**113**	81.13	8.65	**113**	372.01	8.65
kshs3	1438	0.00	0.00	10.A	52	0.28	0.00	52	0.28	0.00
kshs4	3507	0.00	0.00	10.B	60	1.54	0.00	60	1.54	0.00
kshs5	1679	0.00	0.00	10.C	70	1.59	0.00	70	1.59	0.00
kshs6	1672	0.00	0.00	10.D	**153**	96.12	3.38	**150**	441.43	1.35
Average time (s)			0.00	Average time (s)			18.49			81.31
Average gap (%)			0.00	Average gap (%)			1.58			1.43
Maximum gap (%)			0.00	Maximum gap (%)			11.24			11.24
Number of optima			6	Number of optima			23			23

but for the LSX algorithms, $\mu = 5$ performs somewhat better than $\mu = 20$. Finally, taking the best solutions for each instance over the 24 runs, yields an average gap of 3.23%, which can be compared with the average gaps for the individual runs. Based on the results in Table 4, we selected LSX *reverse* $\mu = 5$ for further experimentation. Table 5 presents the results obtained with this algorithm on the instances from the literature. For each problem the upper bound UB on the optimal solution, the computation time and the gap are reported.

For the gdb and kshs instances only the results with a GLS iteration limit $= 10^5$ are shown. For the Benavent et al. (1992) instances, we also give the results when 5×10^5 iterations are allowed. Overall, the results are very good. With a limit of 10^5 iterations, the algorithm reaches all the best known upper bounds and improves the bound for eight instances (indicated in bold in Table 5). For three problems (gdb9, gdb13 and 4.C) the improved solution is optimal. The computation times are good, in fact often negligible when an optimal solution was found. When 5×10^5 iterations are allowed, three of the Benavent instances are further improved (underlined in Table 5). Naturally, the computation times increase, approximately linear in the number of GLS iterations as observed in Table 4. The results for the new CARP instances, for an iteration limit of 10^5 and 5×10^5, are reported in Table 6 (C-instances) and Table 7

Table 6
Computational results for the C-instances

Problem	$Q = 300$, #gls $= 10^5$			$Q = 300$, #gls $= 5 \times 10^5$			$Q = 600$, #gls $= 10^5$			$Q = 600$, #gls $= 5 \times 10^5$		
	UB	Time (s)	Gap (%)	UB	Time (s)	Gap (%)	UB	Time (s)	Gap (%)	UB	Time (s)	Gap (%)
C1	1660	72.88	4.40	1660	325.38	4.40	725	11.76	0.00	725	11.76	0.00
C2	1095	7.91	0.00	1095	7.91	0.00	480	1.16	0.00	480	1.16	0.00
C3	925	40.54	5.71	925	172.63	5.71	415	0.44	0.00	415	0.44	0.00
C4	1340	64.26	4.28	1340	283.69	4.28	615	0.50	0.00	615	0.50	0.00
C5	2475	52.78	2.70	2475	234.31	2.70	1040	1.37	0.00	1040	1.37	0.00
C6	895	37.40	4.68	895	160.39	4.68	485	0.22	0.00	485	0.22	0.00
C7	1795	38.23	3.46	1795	166.58	3.46	835	30.65	13.61	835	138.91	13.61
C8	1730	52.07	5.49	1730	229.75	5.49	695	43.34	13.01	**685**	195.09	11.38
C9	1830	99.47	3.10	**1825**	445.12	2.82	680	1.09	0.00	680	1.09	0.00
C10	2290	41.75	4.57	2290	181.09	4.57	910	31.09	1.11	910	149.12	1.11
C11	1815	92.77	5.22	1815	423.58	5.22	930	75.69	1.09	930	368.34	1.09
C12	1610	64.53	6.62	1610	285.17	6.62	680	0.44	0.00	680	0.44	0.00
C13	1110	40.54	5.71	1110	173.62	5.71	690	0.76	0.00	690	0.76	0.00
C14	1680	44.44	3.70	1680	198.89	3.70	930	38.84	1.09	930	180.16	1.09
C15	1890	116.00	7.08	**1860**	552.27	5.38	920	92.11	1.10	**910**	368.82	0.00
C16	585	29.11	0.86	585	121.88	0.86	170	0.00	0.00	170	0.00	0.00
C17	1620	32.02	1.89	**1610**	137.48	1.26	675	0.17	0.00	675	0.17	0.00
C18	2425	120.72	4.75	**2410**	565.57	4.10	930	2.47	0.00	930	2.47	0.00
C19	1395	46.35	3.72	1395	210.15	3.72	680	32.79	4.62	680	154.83	4.62
C20	665	1.21	0.00	665	1.21	0.00	415	0.11	0.00	415	0.11	0.00
C21	1725	72.67	1.17	1725	326.92	1.17	810	53.55	16.55	**805**	251.83	15.83
C22	1070	2.75	0.00	1070	2.75	0.00	690	0.27	0.00	690	0.27	0.00
C23	1690	83.65	4.32	1690	381.63	4.32	735	71.62	2.80	735	335.32	2.80
C24	1365	68.60	2.63	**1360**	311.37	2.26	670	50.20	8.06	670	247.99	8.06
C25	905	0.33	0.00	905	0.33	0.00	410	0.00	0.00	410	0.00	0.00
Average time (s)		52.92			235.99			21.63			96.45	
Average gap (%)		3.44			3.30			2.52			2.38	
Maximum gap (%)		7.08			6.62			16.55			15.83	
Number of optima		4			4			15			16	

Table 7
Computational results for the E-instances

Problem	$Q = 300$, #gls $= 10^5$			$Q = 300$, #gls $= 5 \times 10^5$			$Q = 600$, #gls $= 10^5$			$Q = 600$, #gls $= 5 \times 10^5$		
	UB	Time (s)	Gap (%)	UB	Time (s)	Gap (%)	UB	Time (s)	Gap (%)	UB	Time (s)	Gap (%)
E1	1945	74.92	4.85	**1940**	342.02	4.58	1065	3.41	0.00	1065	3.41	0.00
E2	1610	40.75	1.90	1610	188.61	1.90	920	2.25	0.00	920	2.25	0.00
E3	750	0.83	0.00	750	0.83	0.00	400	0.00	0.00	400	0.00	0.00
E4	1610	69.32	1.90	1610	328.29	1.90	940	53.61	1.08	940	247.77	1.08
E5	2245	48.00	5.40	**2170**	214.10	1.88	1180	0.22	0.00	1180	0.22	0.00
E6	670	0.05	0.00	670	0.05	0.00	490	0.00	0.00	490	0.00	0.00
E7	1900	37.68	6.74	1900	161.65	6.74	1080	0.11	0.00	1080	0.11	0.00
E8	2150	49.76	3.37	2150	221.63	3.37	1145	40.10	0.88	1145	185.54	0.88
E9	2280	95.68	5.56	**2250**	440.23	4.17	1145	3.74	0.00	1145	3.74	0.00
E10	1690	0.16	0.00	1690	0.16	0.00	1010	0.06	0.00	1010	0.06	0.00
E11	1850	91.72	2.21	1850	420.40	2.21	1015	4.45	0.00	1015	4.45	0.00
E12	1715	61.84	8.54	**1710**	264.68	8.23	910	50.26	1.11	910	228.66	1.11
E13	1325	42.02	1.92	1325	178.46	1.92	835	0.22	0.00	835	0.22	0.00
E14	1810	43.01	1.69	1810	190.21	1.69	1025	18.78	0.00	1025	18.78	0.00
E15	1615	109.47	3.86	**1610**	503.39	3.54	945	1.32	0.00	945	1.32	0.00
E16	1825	50.69	2.24	1825	199.32	2.24	775	0.05	0.00	775	0.05	0.00
E17	1290	6.26	0.00	1290	6.26	0.00	605	0.11	0.00	605	0.11	0.00
E18	1610	78.82	0.63	1610	363.06	0.63	850	56.84	1.80	850	274.52	1.80
E19	1435	46.36	2.50	1435	211.52	2.50	725	33.12	5.84	725	157.96	5.84
E20	990	51.19	4.21	990	232.22	4.21	610	1.32	0.00	610	1.32	0.00
E21	1715	64.70	0.88	**1705**	293.36	0.29	905	3.95	0.00	905	3.95	0.00
E22	1185	29.77	2.60	1185	129.02	2.60	790	0.38	0.00	790	0.38	0.00
E23	1450	86.89	3.94	**1430**	394.81	2.51	725	68.43	2.84	725	319.61	2.84
E24	1790	79.70	5.60	**1785**	362.12	5.31	975	26.59	0.00	975	26.59	0.00
E25	655	0.00	0.00	655	0.00	0.00	430	0.06	0.00	430	0.06	0.00
Average time (s)			50.38			225.86			14.78			59.24
Average gap (%)			2.82			2.50			0.54			0.54
Maximum gap (%)			8.54			8.23			5.84			5.84
Number of optima			5			5			19			19

(E-instances). Both tables clearly show the effect of the vehicle capacity Q on the average solution quality and the computation time. The $Q = 300$ problems are more difficult to solve than those with $Q = 600$. This is easily observed by comparing the average gaps and the number of optima found. Hence, it is recommended to allow more GLS iterations for the problems that involve more vehicles. For example, by allowing 5×10^5 instead of 10^5 iterations, the solution of eight E ($Q = 300$) and five C ($Q = 300$) instances is improved, indicated in bold in Tables 6 and 7. In contrast, increasing the iteration limit for the $Q = 600$ problems, hardly affects the solution quality: only for three C-instances is a better solution found. In any case, the algorithm consistently produces high quality solutions on relatively large problem instances, within a limited computation time. For the instances involving only few trucks, it often finds an optimal solution.

4. Conclusions

In this paper we present a new local search algorithm for the CARP. The procedure uses, apart from the single vehicle moves introduced in Muyldermans et al. (2001a), new moves (*relocate*, *exchange* and *cross*), that operate between two routes and are commonly used in a node routing context (Savelsbergh, 1988). We combine the algorithm with the meta-heuristic GLS and further

use the mechanisms of neighbor lists and edge marking to save computation time. Our approach is no more complex than the procedure of Hertz et al. (2000) and relatively easy to implement. Experiments on standard benchmark problems from the literature show that the algorithm finds all the best known upper bounds and improves the bound for eight instances, within a limited computation time. On a set of new test problems, the local search approach consistently produces high quality solutions and often detects an optimal solution. Finally, our procedure can be easily extended to deal with multi-depot CARP problems by allowing different starts and stops for the vehicles and adapting the cross move. Such an approach is currently being investigated.

References

Amberg, A., Domschke, W., Voss, S., 2000. Multiple center capacitated arc routing problems: A Tabu search algorithm using capacitated trees. European Journal of Operational Research 124, 360–376.

Assad, A.A., Golden, B.L., 1995. Arc routing methods and applications. In: Ball, M.O., Magnanti, T.L., Monma, C.L., Nemhauser, G.L. (Eds.), Network Routing, Handbooks in Operations Research and Management Science. North-Holland, Amsterdam, The Netherlands, pp. 375–483.

Belenguer, J.M., Benavent, E., 1998. The capacitated arc routing problem: Valid inequalities and facets. Computational Optimization and Applications 10, 165–187.

Belenguer, J.M., Benavent, E., 2000. A Cutting Plane Algorithm for the Capacitated Arc Routing Problem. Working paper, Departemento de Estadística e Investigación Operativa, Universidad de Valencia.

Benavent, E., 1997. ftp://indurain.estadi.uv.es/pub/CARP.

Benavent, E., Campos, V., Corberán, A., Mota, E., 1992. The capacitated arc routing problem. Lower bounds. Networks 22, 669–690.

Benavent, E., Corberán, A., Sanchis, J.M., 2000. Linear programming based methods for solving arc routing problems. In: Dror, M. (Ed.), Arc Routing: Theory, Solutions and Applications. Kluwer Academic Publishers, Dordrecht, The Netherlands, pp. 231–275.

Bentley, J.L., 1992. Fast algorithms for geometric traveling salesman problems. ORSA Journal on Computing 4, 387–411.

Croes, G.A., 1958. A method for solving traveling salesman problems. Operations Research 6, 791–812.

Dror, M. (Ed.), 2000. Arc Routing: Theory, Solutions and Applications. Kluwer Academic Publishers, Dordrecht, The Netherlands.

Eglese, R.W., 1994. Routing winter gritting vehicles. Discrete Applied Mathematics 48, 231–244.

Eglese, R.W., Li, L.Y.O., 1994. Modeling Issues in Arc Routing. In: Paper presented at EURO XIII/OR36, Glasgow, Scotland.

Eiselt, H.A., Gendreau, M., Laporte, G., 1995. Arc routing problems, Part II: The rural postman problem. Operations Research 43, 399–414.

Gendreau, M., Hertz, A., Laporte, G., 1994. A Tabu search heuristic for the vehicle routing problem. Management Science 40, 1276–1290.

Golden, B.L., Wong, R.T., 1981. Capacitated arc routing problems. Networks 11, 305–315.

Golden, B.L., DeArmon, J.S., Baker, E.K., 1983. Computational experiments with algorithms for a class of routing problems. Computers and Operations Research 10, 47–59.

Greistorfer, P., 1994. Computational experiments with heuristics for a capacitated arc routing problem. In: Operations Research Proceedings 1994. Physica-Verlag, Wurzburg, pp. 185–190.

Greistorfer, P., 2000. Tabu scatter search metaheuristic for the arc routing problem. Applied Intelligence (in press).

Hertz, A., Laporte, G., Mittaz, M., 2000. A Tabu search heuristic for the capacitated arc routing problem. Operations Research 48, 129–135.

Hertz, A., Laporte, G., Nanchen Hugo, P., 1999. Improvement procedures for the undirected rural postman problem. INFORMS Journal on Computing 11, 53–62.

Hirabayashi, R., Saruwatari, Y., Nishida, N., 1992. Tour construction algorithm for the capacitated arc routing problem. Asia-Pacific Journal of Operational Research 9, 155–175.

Johnson, D.S., McGeoch, L.A., 1997. The traveling salesman problem: A case study. In: Aarts, E., Lenstra, J.K. (Eds.), Local Search in Combinatorial Optimization. John Wiley and Sons, Chichester, pp. 215–310.

Kilby, P., Prosser, P., Shaw, P., 1999. Guided local search for the vehicle routing problem. In: Voss, S., Martello, S., Osman, H., Roucairol, C. (Eds.), Meta Heuristics: Advances and Trends in Local Search Paradigms for Optimization. Kluwer Academic Publishers, Boston, pp. 473–486.

Kiushi, M., Shinano, Y., Hirabayashi, R., Saruwatari, Y., 1995. An exact algorithm for the capacitated arc routing problem using parallel branch and bound method. Abstracts of the 1995 Spring National Conference of the Operational Research Society of Japan, pp. 28–29.

Lin, S., 1965. Computer solutions of the traveling salesman problem. Bell System Technical Journal 44, 2245–2269.

Muyldermans, L., Beullens, P., Cattrysse, D., Van Oudheusden, D., 2001a. The *k*-opt approach for the general routing problem. Working paper 01/18, Center for Industrial Management, Katholieke Universiteit Leuven.

Muyldermans, L., Cattrysse, D., Van Oudheusden, D., 2001b. Evaluation of districting procedures for arc routing. Talk presented at EURO XVIII, Rotterdam, The Netherlands.

Savelsbergh, M.W.P., 1988. Computer aided routing. PhD Thesis. Centrum voor Wiskunde en Informatica, Amsterdam.

Voudouris, C., Tsang, E., 1996. Partial constraint satisfaction problems and guided local search. In: Proceedings of the Second International Conference on Practical Application of Constraint Technology (PACT'96), London, pp. 337–356.

Voudouris, C., Tsang, E., 1999. Guided local search and its application to the traveling salesman problem. European Journal of Operational Research 113, 469–499.