J Comb Optim (2008) 15: 368–386
DOI 10.1007/s10878-007-9090-4

A reactive tabu search algorithm for the vehicle routing problem with simultaneous pickups and deliveries

Niaz A. Wassan · A. Hameed Wassan · Gábor Nagy

Published online: 8 June 2007

Abstract The vehicle routing problem with pickups and deliveries (VRPPD) extends the vehicle routing problem (VRP) by allowing customers to both send and receive goods. The main difficulty of the problem is that the load of vehicles is fluctuating rather than decreasing as in the VRP. We design a reactive tabu search metaheuristic that can check feasibility of proposed moves quickly and reacts to repetitions to guide the search. Several new best solutions are found for benchmark problems.

Keywords Vehicle routing · Pickups and deliveries · Heuristic · Reactive tabu search

1 Introduction

The *Vehicle Routing Problem with Pickups and Deliveries (VRPPD)* is an extension to the vehicle routing problem (VRP) where the vehicles are not only required to deliver goods from a depot to customers but also to pick some goods up at customer locations for returning to the depot. From a *practical* point of view, the VRPPD falls within the field of *reverse logistics*, a research area of increasing importance. A very common application area is postal logistics, as letters and parcels may be delivered and collected on the same tour. From a *mathematical* point of view, the VRPPD is a *combinatorial optimization* problem; it is NP–hard, being a generalization of the classical VRP. It can be formulated as a mixed ILP (see Nagy and Salhi 2005) and solved by exact methods for small problems. Our focus, however, will exclusively be on heuristics.

In the remainder of this section, we define the problem and put it in the context of similar problems. Section 2 presents a review of the literature. The reactive tabu

N.A. Wassan · A.H. Wassan · G. Nagy (✉)
Centre for Heuristic Optimisation, Kent Business School, The University of Kent,
Canterbury CT2 7PE, UK
e-mail: g.nagy@kent.ac.uk

heuristic is described in Sect. 3. We provide the computational results in Sect. 4. Finally, we present our conclusions and outline some suggestions for future work.

1.1 Assumptions and models

It is normally assumed that goods stored at some customer location cannot directly be transported to another customer. In other words, all goods have to either originate from, or end up, at a depot. Thus, problems such as the dial-a-ride problem are excluded from our consideration, although some authors do refer even to these as pickup-and-delivery problems. It is often assumed that the vehicle fleet is homogeneous and its size is not fixed in advance. The *objective function* of the VRPPD is to minimize the total distance traveled by the vehicles, subject to maximum time (or maximum distance) and maximum capacity constraints on the vehicles. Furthermore we note that this paper will not consider time windows, as the time window requirements are usually more restrictive than the constraints arising from the existence of pickups and deliveries and hence including such problems would deviate us from the focus of the research.

Within the above assumptions, three important VRPPD models may be distinguished. In the following, we briefly describe these models and then present and explain our choice of model.

Delivery—first, pickup—second VRPPD A significant proportion of researchers make the assumption that customers can be divided into *linehauls* (customers receiving goods) and *backhauls* (customers sending goods); furthermore vehicles can only pick up goods after they have finished delivering their entire load. One reason for this is that it may be difficult to re-arrange delivery and pickup goods on the vehicles. Other assumptions sometimes made for this problem class are a fixed vehicle fleet and a prohibition of backhaul-only routes. A typical way of solving such problems is creating open vehicle routes for delivery and pickup segments and *matching* them. (This problem class is often called as the *vehicle routing problem with backhauling (VRPB)*.)

Mixed pickups and deliveries A VRPPD where linehauls and backhauls can occur in any sequence on a vehicle route is referred to as a *mixed* VRPPD. This is a conceptually harder problem class, since the vehicle load may go up or down at a customer location, depending on whether the customer is a linehaul or a backhaul. (In the VRP, the vehicle load is always decreasing, while in the VRPB, it decreases to zero and then it increases.) Hence, checking feasibility is a harder task. (Some researchers refer to both mixed VRPPD problems and delivery-first pickup-second problems as *backhauling (VRPB)* problems.) A common way of solving this problem class is by creating routes for linehauls only and then *inserting* the backhauls.

Simultaneous pickups and deliveries In this model, customers may *simultaneously* receive and send goods. We note that mixed and simultaneous VRPPD problems can be modeled in the same framework. Mixed problems can be thought of as simultaneous ones with either the pickup or the delivery load being nil; while the customers of simultaneous problems can be divided into pickup and delivery entities to arrive at a mixed model. (In the simultaneous problem, there may be a further restriction on serving the pickup and delivery of a customer at the same time.) The nature of this problem class requires an *integrated* solution approach.

The aim of this study is to produce an efficient metaheuristic for the *simultaneous* VRPPD. As explained above, this algorithm would also be able to solve the mixed VRPPD. However, we do not address the delivery-first pickup-second problem, as we believe that this assumption is unnecessarily restrictive. It will thus not be possible to compare our results with those of delivery-first pickup-second algorithms, as our solutions may violate that restriction. In practice, it may be a useful exercise to find the solution to a problem both with and without the delivery-first pickup-second assumption; this can help in evaluating whether the cost reduction justifies the possible inconvenience of rearranging loads. We also point to the research of Wade (2002), who proposed a model that encompasses the above types in a unified framework.

1.2 Problem description

Due to the number of different problems versions and minor variations within these, it is important to set out clearly the particular problem that we are addressing in this paper.

Given are a number of customers, each with a known demand and supply. There is a single depot that can supply the customers and receive their goods. The goods supplied from the depot to the customers are different from the goods supplied by the customers to the depot. Hence, it is not possible to satisfy a customer's demand from another customer's supply. All distances are known. A homogeneous vehicle fleet is available, of a sufficiently large size that it does not constrain the solution, but not every vehicle needs to be used. Each vehicle can travel no longer than a given limit (called *maximum time*) and carry no more load than a given quantity (called *maximum capacity*). Customers strictly require that their delivery and pickup is to be serviced in a single visit by a vehicle. This further implies that demands and supplies cannot be split.

The problem is to design a number of vehicle routes, originating and terminating at the depot, such that all customers are visited and the above maximum time and maximum capacity constraints are satisfied. The objective is to minimize the total distance traveled by the vehicles. (We note that in practice the number of vehicles used may also form part of the objective function.)

Let us address in more detail the issues of feasibility. The maximum time constraint can be viewed as an extension to the concept of maximum distance. The total time traveled by a vehicle equals the total distance traveled multiplied by its speed plus the time spent waiting at customer locations (known as the drop time). In the instances we consider a unit speed is assumed hence the total time is numerically equivalent to the total distance plus the total waiting time. (As the drop time is assumed to be the same for every customer, the latter equals to the drop time multiplied by the number of customers on the route.) We note that checking for time feasibility for the VRPPD can be done in exactly the same way as for the VRP. Hence, the presence of pickups and deliveries present no additional difficulty in this aspect and many of the benchmark instances do not have a maximum time limit.

This is not true for the maximum capacity constraint—it is much harder to check in the case of the simultaneous VRPPD and this is where the added difficulty of solving such problem lies. Hence, this aspect deserves some detailed discussion. First, let us show why this constraint is tricky. In the VRP, the vehicle leaves the depot loaded with the total demand of the customers on the route. If this does not exceed the vehicle capacity, the route will be feasible, as at each customer, the load on the vehicle decreases. Even in the delivery-first, pickup-second VRPPD load feasibility can be easily achieved by ensuring that neither the total delivery nor the total pickup of a vehicle exceeds the vehicle capacity. (The vehicle load first monotonously decreases, then on reaching zero it monotonously increases.) However, the load on a simultaneous VRPPD vehicle fluctuates—at each customer it may increase (if the customer's demand is less than its supply) or decrease (if the customer's demand is more than its supply). Any algorithm to solve this problem must tackle this difficulty.

In order to make the description of our subsequent solution algorithm clearer and easier, let us define here some variables relating to the issue of load feasibility. In the following, no index is used to indicate the route. This is done to make the notation easier. No confusion shall result from this, as in the subsequent discussions, we always talk about a single route.

- Let q_i denote the demand of customer i.
- Let p_i denote the pickup (supply) of customer i.
- Let MC denote the maximum capacity of the vehicle.

If S is the set of customers on the route, then:

- $P = \sum_{i \in S} p_i$ is the total load collected on the route and
- $Q = \sum_{i \in S} q_i$ is the total load delivered on the route.

We note that $P \leq MC$ and $Q \leq MC$ are necessary but not sufficient conditions of load feasibility. A very important measure is the level of load on the vehicle at any particular point on its route. This can be calculated for any arc fg as follows:

- $load_{fg} = \sum_{i \in F} p_i + \sum_{i \in G} q_i$ where F is the set of customers on the route up to f and G is the set of customers on the route from g.

Another way to calculate the *load* function is by observing that the load of the first arc is Q and at each customer i the load increases by $p_i - q_i$. (Note that the load of the last arc is P.) A route is feasible with respect to the maximum capacity constraint if for all its arcs, $load_{ij} \leq MC$. It may be of interest to know the highest and lowest level of load on a route. If R is the set of arcs on the route, then:

- $maxload = \max_{ij \in R} load_{ij}$ and
- $minload = \min_{ij \in R} load_{ij}$.

Thus, an easy way of establishing route feasibility is to compare *maxload* to *MC*. (Although the variable *minload* does not appear very useful, it will prove its use in our algorithm.)

2 Literature review

As discussed in the previous section, the VRPPD literature can be classified into the following three main categories:

- Simultaneous pickups and deliveries.
- Mixed pickups and deliveries.
- Deliveries-before-pickups.

We exclude from this review works that do not comply with our assumptions made in the previous section. The interested reader is referred to Nanry and Barnes (2000) and Fabri and Recht (2006) for dial-a-ride/courier problems, Drezner (1982) for the round-trip location problem and Thangiah et al. (1996), Irnich (2000), Nanry and Barnes (2000) and Fabri and Recht (2006) for problems involving time windows.

2.1 Literature on simultaneous version of the VRPPD

Min (1989) is the first to study this problem version. He solves a small-scale real-world problem. Customers are initially clustered into groups and then for each group a TSP is solved. The resulting infeasible arcs are penalized by setting their lengths to infinity, and the TSPs are solved again. Clearly, this procedure is unlikely to work for large-scale problems, as there may be several infeasible arcs, as shown by Nagy's (1996) experiments.

Halse (1992) studies the simultaneous VRPPD along with other variants of the VRPPD and the VRP. In order to solve the simultaneous and mixed versions of the VRPPD, he uses a cluster-first route-second method. The method consists of two phases: firstly customers are assigned to the vehicles, followed by a routing procedure based on the 3-opt method.

Nagy (1996), see also Nagy and Salhi (2005), develops a composite heuristic approach for the simultaneous VRPPD. This methodology can also cater for multiple depots. The author modifies a number of VRP routines. His methodology combines the power of different routines in an organized way. These routines include standard VRP routines such as 2-Opt, 3-Opt, Shift, Exchange, Perturb but also some specially developed for the VRPPD such as Reverse (reverses the direction of a route) and Neck (allows customers to be visited twice, once for delivery and then for pickup). An insertion-based method is also developed for comparison purposes, see Salhi and Nagy (1999). This is based on the concept of inserting more than one customer at a time.

Gendreau et al. (1999) solve the traveling salesman problem with pickup and deliveries (TSPPD) using a heuristic method. The method solves the TSP first without taking into account the pickups and deliveries. The order of pickup and deliveries is then achieved on the TSP tour.

Dethloff (2001) uses an insertion-based algorithm more typically adopted for mixed problems. Customers are inserted into emerging routes according to three criteria: travel distance (as in the basic VRP), capacity remaining and distance from the depot. No improvement routines are presented.

Crispim and Brandão (2005) are the first to present a metaheuristic approach for the simultaneous VRPPD. Their method is a hybrid of tabu search and variable neighborhood search. Initial solutions are generated using a sweep method. If any route within this solution is infeasible due to intermediate arcs being overloaded, the order of customers on the route is exchanged until feasibility was established. The improvement phase is built on the moves "insert" and "swap". Infeasible solutions are allowed but are penalized according to the level of overload as in the strategic oscillation method of Nagy (1996).

Tang and Galvão (2006) use a tabu search framework. Initial solutions are found using a number of methodologies (sweep, tour partitioning, and extensions of the TSPPD heuristics proposed by Mosheiov 1994; Anily and Mosheiov 1994 and Gendreau et al. 1999). The neighborhood is built on the moves insert, exchange, crossover (splitting and splicing two routes) and 2-opt. The tabu search structure relies on both short- and long-term memory.

Chen and Wu (2006) also use tabu search but find initial feasible solutions by an insertion method, which relies on both distance- and load-based criteria. The neighborhood for the improvement phase is built on the moves 2-exchange, swap, shift, 2-opt and Or-opt.

2.2 Literature on mixed version of the VRPPD

This version seems to be relatively better studied in the literature. As the focus of this paper is the simultaneous case, we review these papers only briefly and refer the reader to Nagy and Salhi (2005) for further details. (That study also reviews some early papers omitted here.)

Halse's (1992) cluster-first route-second method, described in the previous section, is also applied to mixed problem instances.

Mosheiov (1994) deals with the Traveling Salesman Problem with Pickups and Deliveries (TSPPD), the single-vehicle version of the VRPPD. The central solution concept is reinserting the depot to a different location in the route. Mosheiov (1995) combines the TSPPD with the traveling salesman location problem to create a location-routing type problem.

Anily and Mosheiov (1994) find a solution to the TSPPD based on a minimum spanning tree. This tree is then scanned from leaves to roots to sort customers according to their delivery/pickup loads, then scanned backwards to build a tour.

Salhi and Nagy (1999) present a cluster-based insertion approach to solve the mixed version of the VRPPD. The heuristic is based on the idea of inserting more than one backhaul customer at a time.

Dethloff (2002) applies his insertion-based method, developed for the simultaneous case, to the mixed VRPPD. The author notes the pitfalls of comparing mixed and simultaneous algorithms.

Wade and Salhi (2002) introduce a generalized VRPPD problem. Their model suggests a compromise between the delivery-first pickup-second and the mixed cases. A mixture of linehauls and backhauls is permitted on routes, but subject to a restriction on having some space on the vehicle to allow maneuvering the loads. An insertion-based procedure is presented, allowing either a single backhaul or a pair of backhauls to be inserted into a route.

Wade and Salhi (2003) use an ant colony approach that treats linehauls and backhauls in an integrated manner. The approach focuses on enhancements to the standard ant system approach, such as a site-dependent candidate list, a look-ahead based visibility and efficient trail updating strategies.

Crispim and Brandão (2005) use a hybrid metaheuristic to solve the mixed VRPPD. This approach is already detailed in the previous section.

2.3 Literature on the delivery-first pickup-second version of the VRPPD

This VRPPD model has attracted the most attention. As it is not central to our work, we refer the reader to the excellent review of Toth and Vigo (2002) and just mention a few recent papers.

Osman and Wassan (2002) develop a reactive tabu search algorithm for the VRPB. This algorithm uses two route construction methods based on the savings methodology of Clarke and Wright (1964). These methods are named as *saving-insertion* and *saving-assignment*. In the saving-insertion procedure delivery routes are built using modified version of the savings method of Gaskell (1967) at first and then the backhauls are inserted at the end of each route using the best insertion criterion. In the savings-assignment procedure delivery and pickup routes are built separately using again the same modified savings approach at first. These routes are then optimally merged using an assignment heuristic procedure. This solution is then transferred to their reactive tabu search phase of the algorithm for further improvement.

As mentioned in the previous section, Wade and Salhi (2002) propose a generalized new class of VRPPD that contains this problem version as a special case.

Wassan (2006b) develops a hybrid heuristic method that intelligently combines the features of reactive tabu search and the adaptive memory programming. This algorithm produced better quality solutions at a lot less computational time as compared to Osman and Wassan (2002).

3 Solution algorithm

The reactive tabu search procedure consists of two phases: constructing an initial solution and then improving on it. For ease of explanation, we separated the description of the underlying improvement moves from the overall tabu search mechanism. To the best of our knowledge, this is the first time reactive tabu search has been applied to the VRPPD, although Nanry and Barnes (2000) used reactive tabu search for a different kind of pickup and delivery problem.

3.1 Initial solution

We modified the *sweep* method proposed by Gillett and Miller (1974) to generate the initial solution. Our modification is to exclude the customers nearest to the depot from the sweep procedure. The proportion of such customers is a controllable parameter α; different values of α were tested in our experiments. These customers are directly connected to the depot on single-stop routes. The rest of the customers are formed into routes using the classical sweep method. Note that this procedure contains the classical sweep method as one extreme case (no customers excluded) and the starting point of the Clarke and Wright (1964) method as the other (all customers excluded from sweep and the initial solution consists of n single-stop routes). The reason for this modification is that it gives us flexibility to insert these excluded customers to possibly better positions during the improvement phase than their polar coordinates would give.

We sort the remaining customers in order of their polar coordinates with the depot as the origin. We then select customer 1 as the starting point and start the sweep process in anticlockwise order. Customers are added to the current route unless this would cause either a maximum time violation or a maximum load violation, in which case the current route is terminated and the customer considered becomes the first customer on a new route.

Although generally in the VRPPD checking load feasibility is more difficult than in the VRP or the VRPB, it can be done quite easily for the sweep procedure. We observe that by adding a new customer i to the end of an emerging route all preceding arc loads are increased by q_i, hence the highest load on these arcs will be $\mathit{maxload} + q_i$. The load on the new last arc will be equal to the new total pickup $P + p_i$. Thus, adding i is feasible only if $\max(\mathit{maxload} + q_i, P + p_i) \le MC$. If customer i is added, *maxload* will be updated to $\max(\mathit{maxload} + q_i, P + p_i)$. Once all customers have been considered, this procedure will yield a feasible solution.

This sweep process is repeated with customers $2, 3, 4, \dots, n$ as the starting customer points. This is called the *forward-sweep* process in the literature. We also perform the *backward-sweep* process from all n starting points, which is similar to the forward-sweep except that the sweep is performed in a clockwise direction. We choose the best solution from the forward and backward processes of the sweep algorithm and pass it on to the improvement phase of our algorithm.

3.2 Improvement procedures

The improvement phase of our algorithm is built around simple VRP operators, namely *shift* and *swap* (jointly known as 1-exchange) and a problem-specific routine we named *reverse.*

Shift This move entails moving a customer i from a route to the best possible position on another route. (It is also known as the $(1, 0)$ operator.) All customers are considered for shifting and the move with the largest decrease in total route length is implemented. The shift process can result in drastic reduction of total distance traveled. It can also be very useful in the sense that it could produce an empty route if a customer is shifted from a single customer route. The computational complexity of this move is $O(n^2)$.

The main difference from the VRP version of this operator is in checking the feasibility of proposed moves. While in the VRP (or the VRPB) it is enough to check if the total customer demand (and also the total customer pickup) is less than the vehicle capacity, for the VRPPD the load can vary from arc to arc. Calculating the arc load for each arc for each possible customer insertion would be unrealistically time-consuming. A crucial observation is that inserting customer i into a route increases then load of each arc prior to the insertion point by q_i and the load of each arc after the insertion point by p_i. We can now distinguish four cases:

1. $\mathit{maxload} + \max(p_i, q_i) \le MC$. In this case customer i can be inserted anywhere on the route. (Let *kl* be the arc with the highest load after insertion, it will have previously had a load no greater than *maxload* and it was increased by no more than $\max(p_i, q_i)$.)

2. $maxload + p_i \leq MC$ and $maxload + q_i > MC$. This implies that there must be at least one arc on the route whose load is greater than $MC - q_i$. Let the first such arc on the route be *kl*. Then i cannot be inserted into *kl* or any arc after *kl*. This is because by inserting i into a subsequent arc fg the load of each arc preceding *fg*, including that of *kl*, will increase by q_i. Thus, the set of arcs feasible for insertion is a continuous sequence of arcs beginning from the depot. Furthermore, checking for feasibility can be done by simply comparing the load against $MC - q_i$ (until we find an infeasible arc).
3. $maxload + p_i > MC$ and $maxload + q_i \leq MC$. In this case, the set of arcs feasible for insertion is a continuous sequence of arcs ending at the depot and checking for feasibility can be done by simply comparing the load against $MC - p_i$. (We check this working backward on the route until we come across the first feasibility violation.) This can be shown similarly to case 2.
4. $maxload + \min(p_i, q_i) > MC$. In this case customer i cannot be inserted anywhere on the route. (Let *kl* be the arc with the highest load before insertion, its new load is at least $maxload + \min(p_i, q_i)$.)

The above observations make checking the feasibility of insertions much easier. We also note that these feasibility checks do not increase the computational complexity of the neighborhood.

Swap This move involves reallocating two customers, say i and j, which are currently on different routes. (It is also known as the $(1, 1)$ operator.) Customer i is removed from its route and inserted to a position on the route from which j is removed. Customer j is moved to a position on the route formerly containing i. All pairs of customers are considered for this operation within an iteration, and the best move is executed. The computational complexity of this move is $O(n^4)$. In terms of feasibility, removing customer i from a route decreases the load of each preceding arc by q_i and it decreases the load of each subsequent arc by p_i. Inserting customers is similar to the *shift* procedure. Hence, checking feasibility of *swap* moves can be achieved similarly to *shift* moves.

Local-shift Further to the above neighborhood, we have also implemented an intra-route move called *local-shift* that relocates a customer to a different position within the route, if this improves the solution quality. The computational complexity of this move is $O(n^2)$. Checking feasibility can be done similarly to the *shift* procedure above.

Reverse We also included a routine that does not, in itself, change the solution quality, but was found to be helpful for guiding the search. Routine *reverse* entails simply reversing the direction of a route, if this results in a decrease in the *maxload* on that route. This then makes it easier for customer insertion into the route later on. The computational complexity of this move is $O(n)$. We state (but omit the proof here) that reversing a route changes its *maxload* to $Q + P - minload$. Thus, in this routine we reverse every route for which $Q + P - minload < maxload$.

3.3 The tabu search framework

Neighborhood search algorithms can fall into the local optima trap. This can be avoided by using a metaheuristic that allows non-improving moves. Tabu search (Glover and Laguna 1997) is a well-known metaheuristic and is considered by some to be the best approach for solving VRP problems, see Cordeau et al. (2002) for further information. The general structure of our RTS-VRPSPD algorithm is presented in the following pseudo-code, while its "reactive" aspects are discussed in the next subsection.

Step 1: Initial Solution

- Generate an initial solution S using *Modified Sweep* procedure
- Set $S_{\text{best}} = S$

Step 2: Initialization

- Initialize the tabu search data structures
- Set number of iterations: $iter = 3000$
- Initialize the *RTS* parameter values (shown in Sect. 3.4)

Step 3: Search Framework

- Evaluate the neighborhood of S defined by the moves *shift* and *swap* and choose the best neighboring solution, S'
- Apply post-optimization routines *local-shift* and *reverse* to the two affected routes; let the new solution be S''
- Set $S = S''$ as the current solution and update tabu list
- Update tabu list size *tls* using the RTS *mechanism* (see in Sect. 3.4)
- If $\text{Cost}(S) < \text{Cost}(S_{\text{best}})$ then set $S_{\text{best}} = S$

Step 4: Termination

- Perform Step 3 until *iter* is reached

3.4 Reactive tabu search—dynamically controlling the tabu list size

One of the ways our method is different from previous approaches is the use of reactive tabu search (RTS). This approach was introduced by Battiti and Tecchioli (1994) and focuses on a component of tabu search called *tabu list size* (*tls*), also known sometimes as tabu tenure, which determines how long a move can be locked up before it is allowed to reappear. The RTS scheme uses an analogy with the theory of dynamical systems, where the tabu list size depends on the repetition of solutions and consequently *tls* is determined dynamically, as opposed to the standard version of tabu search where *tls* is fixed. RTS employs two mechanisms, both mechanisms react to the repetitions. The *first mechanism* is used to produce a balanced tabu list size and consists of two reactions. The *fast reaction* increases the list size when solutions are repeated, the *slow reaction* reduces the list size for those regions of the search space that do not need large list lengths. The *second mechanism* provides a systematic way to diversify the search when it is only confined to one portion of the solution space. RTS seems to be robust enough and there may be a little effect of parameter changing, as found in the experiments of Battiti and Tecchioli (1994). This observation is also supported by Osman and Wassan (2002) and Wassan (2006a). Moreover, the experiments of Battiti and Tecchioli (1994) and Wassan and Osman (2002) showed the superiority of RTS compared to other tabu search schemes.

In the following, we give the particulars of our implementation of the RTS scheme that dynamically updates the value of tabu list size *tls* automatically reacting to repetitions. (Note that we only use the first mechanism of RTS.)

The following counters and parameters used in our RTS-VRPSPD algorithm, given in Sect. 3.3, are defined as follows and initialized to the following values in Step 2.

initial tabu list size value: $tls = 1$
counter for the often-repeated sets of solutions: $Chaotic = 0$
moving average for the detected repetitions: $MovAvg = 0$
gap between two consecutive repetitions: $GapRept = 0$
number of iterations since the last change in *tls* value: $LastChange = 0$
iteration number when last time an identical solution was noticed: $LasTimeRept = 0$
iteration number of the most recent repetition: $CurTimeRept = 0$

maximum limit for the often-repeated solutions: $REP = 3$
maximum limit for the sets of often-repeated solutions: $Chaos = 6$
percentage increase for the tabu tenure value: $Increase = 1.1$
percentage decrease for the tabu tenure value: $Decrease = 0.9$
constant used to compare with GapRept to get the moving average: $GapMax = 50$

The constant values for the above parameters are the same as in the standard RTS scheme of Battiti and Tecchioli (1994) except for the value of *Chaos* which in our case was found after some initial experiments.

The RTS *mechanism*, called in Step 3 of our RTS-VRPSPD algorithm, is given in the pseudo-code below.

```
(1) Search for repetition of $S$, if found then,
        $GapRept = CurTimeRept - LasTimeRept$, and go to (2);
        Else go to (4).
(2) If repetition of $S > REP$, then set $Chaotic = Chaotic + 1$;
        Else go to (3).
        If $Chaotic > Chaos$, then set $Chaotic = 0$, clear and rebuild tabu search data
        structures and go to Step 3 of the RTS algorithm;
        Else go to (3).
(3) If $GapRept < GapMax$, then
        $tls = tls \times Increase$, set $LastChange = 0$,
        $MovAvg = 0.1 \times GapRept + 0.9 \times MovAvg$; and go to (5);
        Else go to (4).
(4) If $LastChange > MovAvg$, then
        $tls = tls \times Decrease$, and set $LastChange = 0$;
        Else set $LastChange = LastChange + 1$, store the solution $S$ and go to (5).
(5) Stop the RTS mechanism.
```

4 Computational results

4.1 Data sets, benchmarks and implementation particulars

We use the most commonly accepted benchmark data for the simultaneous VRPPD, namely the CMT data set adapted by Nagy (1996). This consists of 28 instances, derived from the 14 problem instances given in Christofides et al. (1979) and contain between 50 and 199 customers. Nagy (1996) created two VRPPD problem instances ("X" and "Y") for each VRP instance, see Nagy and Salhi (2005) for an explanation of how this was done. We note that the 14 VRP instances can be divided into two subsets: those without a maximum time limit and those with such a limit. (Instances 6, 7, 8, 14, 13, 9 and 10 correspond to instances 1, 2, 3, 12, 11, 4 and 5, with a time limit imposed on them.) For these instances a "drop time" (time spent at a customer location) is included. As stated in Sect. 1.2, total time on the road is numerically the same as total route length; to which the drop times are added to find the total journey time.

We compare our algorithm against five sets of benchmark solutions, namely those found by Nagy and Salhi (2005), Dethloff (2001), Crispim and Brandão (2005), Chen and Wu (2006) and Tang and Galvão (2006). All these authors have solved the non-time-constrained problem instances, but only Nagy and Salhi (2005), Dethloff (2001) and Tang and Galvão (2006) solved those with time constraints. Furthermore, there is a difference of interpretation between these studies. The former two incorporated the "drop time", while the last one did not. Hence, we solved these instances both with the drop times given in Christofides et al. (1979) and with drop times of zero. Finally, lower bounds were produced for most instances by Tang and Galvão (2006).

The algorithm was implemented in Fortran and the experiments executed on a Sun-Fire-V440 with UltraSPARC-IIIi processor, CPU speed 1062 MHz, running Solaris 9. Each instance is solved five times, starting from five different initial solutions, corresponding to $\alpha = 0\%$, 10%, 20%, 30% and 50% in the modified sweep method.

4.2 Comparison against benchmarks

First, let us compare our results to those found previously in the literature. For the instances without limits on maximum time, there are a number of previous results available. Best solutions to date were found by Tang and Galvão (2006) or Chen and Wu (2006). Table 1 shows the results. Our algorithm improves the best results found so far in 10 out of 14 instances. (In the other four cases, we are on average 1.8% away from the best.) The average gap from the lower bound is 15.72%, better than Chen and Wu (2006) (18.03%) and Tang and Galvão (2006) (19.10%) and considerably better than Crispim and Brandão (2005) (27.15%). However, this is still a significant gap, and future work should be directed to see if either the solutions or the lower bounds could be improved. In terms of the number of vehicles required, our method was able to reduce this by one vehicle for four instances. In one instance, however, it requires one vehicle more than the solution of Dethloff (2001), although this is offset by a significant (20%) saving in routing cost. However, our algorithm is somewhat slower than previous methods. We do report here both total computing times (time for 3000 iterations) and time till the best solution was found. This is normally a significantly

Table 1 Comparison between different algorithms for the CMT problems (without maximum time constraints)

		Nagy & Salhi			Dethloff		Crispim & Brandão			Chen & Wu			Tang & Galvão			RTS-VRPSPD				Lower
Problem	Size	Sol	v	T	Sol	v	Sol	v	T	Sol	v	T	Sol	v	T	Sol	v	TB	TT	bound
CMT1X	50	525	5	0.3	501	3	477	3	11.2	478.59	3	7.74	472	3	3.7	**468.30**	3	0.3	48	454.68
CMT1Y	50	525	5	0.3	501	3	485	3	9.1	480.78	3	7.81	470	3	4.37	**458.96**	3	2	69	455.52
CMT2X	75	841	10	0.8	782	7	710	6	24.3	688.51	6	24.86	695	7	6.91	**668.77**	6	5	94	617.01
CMT2Y	75	839	10	0.7	782	7	715	6	26.4	679.44	6	12.02	700	7	7.61	**663.25**	6	6	102	617.64
CMT3X	100	829	8	1.1	847	5	744	5	37.3	744.77	5	94.06	**721**	5	11.04	729.63	4	65	294	646.73
CMT3Y	100	829	8	1.5	847	5	742	5	33.5	723.88	5	120.66	**719**	5	12.01	745.46	4	15	285	648.04
CMT12X	100	820	10	1.3	804	6	731	5	37.1	678.46	6	46.83	675	6	12.23	**644.70**	5	22	242	568.79
CMT12Y	100	825	10	1.4	825	5	860	5	33.7	676.23	6	56.35	689	6	12.80	**659.52**	6	12	254	573.53
CMT11X	120	1087	7	2.4	959	4	944	4	32.4	**858.57**	4	321.08	900	4	18.17	861.97	4	10	504	663.38
CMT11Y	120	1075	7	2.6	1070	4	1035	4	29.6	859.77	5	230.72	910	5	18.08	**830.39**	4	129	325	662.84
CMT4X	150	1053	12	3.8	1050	7	915	7	58.1	887.00	7	501.95	880	7	24.60	**876.50**	7	69	558	714.18
CMT4Y	150	1047	12	3.3	1050	7	996	7	47.6	**852.35**	7	406.32	878	7	29.09	870.44	7	73	405	715.67
CMT5X	199	1334	16	7.4	1348	11	1136	10	89.4	1089.22	10	1055.83	1098	11	51.50	**1044.51**	9	16	483	858.14
CMT5Y	199	1334	16	7.2	1348	11	1129	10	77.1	1084.27	10	771.71	1083	10	56.21	**1054.46**	9	132	533	856.59
Average gap		41.28%			38.18%		27.15%			18.03%			19.10%			**15.72%**				

Sol: Solution value; v: number of vehicles used in the solution

T: CPU time in seconds; TB: CPU time to best; TT: Total CPU time (for 3000 iterations)

Best solutions are highlighted in bold

Average gap is with respect to the lower bound (found by Tang and Galvão) smaller figure than the former; unfortunately in literature it is not always made clear which figure is reported. However, our running times are not much longer than those of Chen and Wu (2006).

For the instances with a maximum time constraint, the picture is not so clear. These instances, proposed by Nagy (1996), were only solved subsequently by Dethloff (2001) and Tang and Galvão (2006). The former included drop times, the latter not. A further complication is that Nagy and Salhi (2005) allow two visits to customers (one for delivery and one for pickup). Thus, we cannot make a straightforward comparison here. The RTS-VRPSPD algorithm is implemented both with the drop times given by Christofides et al. (1979) and with no drop time; however the case of two visits to customers was not considered. Table 2 reports the results. The RTS-VRPSPD algorithm improves the results found by Dethloff (2001) for every instance, in most cases with the same number of vehicles required. Although Nagy and Salhi (2005) allow for a larger feasible set and the chance of finding better solutions, in four instances we improved on their solutions and in one instance found the same solution. Comparing to Tang and Galvão (2006), we have again improved on 12 out of 14 of their results and in many cases reduced the number of vehicles needed by one. The time taken by RTS-VRPSPD is longer than needed by Nagy and Salhi (2005) and by Tang and Galvão (2006); Dethloff (2001) does not give computing times.

4.3 Other observations

In order to carry out a more in-depth analysis of our algorithm RTS-VRPSPD, Table 3 shows the comparison between initial solutions, single runs and the best of five runs. It also gives lower bounds and previous best solutions. The two initial solutions are those found by normal sweep (i.e. $\alpha = 0\%$) and modified sweep with $\alpha = 10\%$; the two single-run solutions derive from these initial solutions.

Table 3 clearly shows the dramatic improvement of tabu search on the initial solution. The gap between the initial solutions and lower bounds is 134%, while between the final solutions and lower bounds it is only 17%. Thus tabu search reduced the optimality gap by 87%. (The above observations relate only to the unconstrained instances for which we have proper lower bounds and use the averages between normal sweep and modified sweep.)

We decided to modify the widely-used sweep method to exclude some customers from the sweep process and let them form their own (single-stop) routes, as explained in Sect. 3.1. Our experiments justify this modification: out of the 28 instances, the modified sweep method outperforms the normal sweep method in 23 cases, improving the solution by 2.0% on average. For some instances, significant improvements (up to nearly 8%) can be achieved by using this modification. In particular, $\alpha = 10\%$ appears a good choice: this setting gives best results for 13 instances and provides on average a 1.1% better solution than normal sweep. We also note that a better initial solution often, but not always, leads to a better final solution.

In our experiments, we used a number of different initial solutions. This contributed to better solutions as the best result of five runs is on average 0.7% better than a single run with the best α parameter (*i.e.* 10%) and 2.0% better than a single run with normal sweep. Moreover, on some instances, considerable improvement (up to 4.5%) can be found by running five runs rather than just one. However, even single

Table 2 Comparison between different algorithms for the CMT problems (with maximum time constraints)

		Dethloff		Nagy & Salhi			RTS-VRPSPD (with $\delta > 0$)				Tang & Galvão			RTS-VRPSPD (with $\delta = 0$)				Lower
Problem	Size	Sol	v	Sol	v	T	Sol	v	TB	TT	Sol	v	T	Sol	v	TB	TT	bound
CMT6X	50	584	6	555	6	0.3	556.06	6	0.3	32	476	3	3.7	**471.89**	3	3	65	436.63
CMT6Y	50	584	6	555	6	0.3	558.17	6	0.5	33	474	3	4.37	**467.70**	3	9	60	437.11
CMT7X	75	961	11	910	11	0.7	903.05	11	2	49	695	7	6.91	**663.95**	6	6	86	607.97
CMT7Y	75	961	11	910	11	0.7	903.36	11	2	47	700	6	7.61	**662.50**	6	6	80	606.38
CMT8X	100	928	9	873	9	1.8	879.60	9	40	131	**720**	5	11.04	726.88	5	4	254	649.25
CMT8Y	100	936	9	867	9	1.6	917.42	10	2	113	**721**	5	12.01	741.96	5	15	315	655.53
CMT14X	100	871	10	879	11	1.5	823.95	10	28	134	675	6	12.23	**644.70**	5	22	244	558.86
CMT14Y	100	871	10	879	11	1.5	823.34	10	4	122	689	6	12.80	**659.52**	6	12	255	568.48
CMT13X	120	1576	11	1557	11	2.3	1647.51	12	10	163	918	5	18.17	**858.48**	5	156	319	–
CMT13Y	120	1576	11	1546	11	2.3	1647.04	11	29	173	910	5	18.08	**880.56**	4	224	546	–
CMT9X	150	1299	15	1188	14	3.5	1220.00	15	48	201	885	7	24.60	**880.61**	7	532	561	–
CMT9Y	150	1299	15	1188	14	3.4	1213.11	15	79	171	900	8	29.09	**886.84**	7	87	562	–
CMT10X	199	1571	19	1420	18	7.5	1464.58	19	157	301	1100	11	51.50	**1079.99**	10	19	529	–
CMT10Y	199	1571	19	1420	18	7.5	1419.79	18	80	318	1083	11	56.21	**1058.09**	10	271	524	–

Sol: Solution value; v: number of vehicles used in the solution

T: CPU time in seconds; TB: CPU time to best; TT: Total CPU time

δ: drop time (positive for Dethloff and Nagy & Salhi, but zero for Tang & Galvão). Lower bounds also assume no drop time

Best results are highlighted in bold only for the case of zero drop time

Table 3 Detailed results of the RTS-VRPSPD algorithm

		Previous best result		Lower bound	Initial solution, normal sweep		Initial solution, modified sweep		Single-run, normal sweep				Single-run, modified sweep				Best of 5 runs, modified sweep			
Problem	Size	Solution	v		Solution	v	Solution	v	Solution	v	TB	TT	Solution	v	TB	TT	Solution	v	TB	TT
CMT1X	50	472^{T}	3	454.68	741.41	3	676.28	8	479.75	3	0.4	80	468.30	3	0.2	48	468.30	3	0.3	48
CMT1Y	50	470^{T}	3	455.52	729.72	3	667.80	8	473.72	3	1	103	465.47	3	2	59	458.96	3	2	69
CMT2X	75	688.51^{C}	6	617.01	1101.63	6	1017.37	13	668.77	6	49	95	680.34	6	1	82	668.77	6	5	94
CMT2Y	75	679.44^{C}	6	617.64	1106.35	6	1028.81	13	663.25	6	29	102	675.95	6	1	79	663.25	6	6	102
CMT3X	100	721^{T}	5	646.73	1317.77	5	1281.01	12	762.96	5	340	343	729.63	4	4	294	729.63	4	65	294
CMT3Y	100	719^{T}	5	648.04	1302.53	5	1293.10	12	746.56	5	338	341	745.46	4	11	285	745.46	4	15	285
CMT12X	100	675^{T}	7	568.79	869.50	6	1054.16	14	655.03	6	69	254	649.14	5	3	220	644.70	5	22	242
CMT12Y	100	676.23^{C}	7	573.53	874.96	6	1061.80	14	662.99	6	186	245	668.43	6	33	182	659.52	6	12	254
CMT11X	120	858.57^{C}	10	663.38	2818.52	4	2609.15	15	934.43	4	727	730	861.97	4	69	504	861.97	4	10	504
CMT11Y	120	859.77^{C}	10	662.84	2793.44	4	2584.85	15	870.93	4	719	722	869.59	4	14	583	830.39	4	129	325
CMT4X	150	880^{T}	4	714.18	1929.55	7	1783.76	16	876.50	7	502	560	883.87	7	95	474	876.50	7	69	558
CMT4Y	150	852.35^{C}	5	715.67	1937.09	7	1791.14	16	893.34	7	304	595	870.44	7	25	448	870.44	7	73	405
CMT5X	199	1089.22^{C}	6	858.14	2418.61	10	2286.03	21	1071.90	10	596	681	1061.69	9	62	487	1044.51	9	16	483
CMT5Y	199	1084.27^{C}	6	856.59	2449.02	10	2335.48	21	1063.11	11	637	686	1054.46	9	40	533	1054.46	9	132	533
CMT6X	50	584^{D}	6	–	872.35	8	815.53	28	558.81	6	7	44	556.06	6	0.3	32	556.06	6	0.3	32
CMT6Y	50	584^{D}	6	–	872.35	8	815.53	28	558.81	6	9	39	558.17	6	0.5	33	558.17	6	0.5	33
CMT7X	75	961^{D}	11	–	1421.10	16	1403.61	12	903.51	11	2	45	906.57	11	31	42	903.05	11	2	49
CMT7Y	75	961^{D}	11	–	1421.10	16	1403.61	12	903.36	11	2	53	905.52	11	12	52	903.36	11	2	47
CMT8X	100	928^{D}	9	–	1548.67	13	1561.75	21	913.16	10	80	119	889.10	9	39	139	879.60	9	40	131
CMT8Y	100	936^{D}	9	–	1548.67	13	1561.75	21	919.60	10	50	119	925.08	10	123	129	917.42	10	2	113
CMT14X	100	871^{D}	10	–	1070.50	11	1170.01	17	847.74	11	17	111	831.77	10	2	111	823.95	10	28	134
CMT14Y	100	871^{D}	10	–	1070.50	11	1172.60	17	841.32	10	9	109	823.34	10	4	122	823.34	10	4	122
CMT13X	120	1576^{D}	11	–	3217.67	14	2979.38	25	1675.24	11	54	158	1647.51	12	10	163	1647.51	12	10	163
CMT13Y	120	1576^{D}	11	–	3217.67	14	2979.38	25	1655.16	11	43	163	1647.04	11	29	173	1647.04	11	29	173
CMT9X	150	1299^{D}	15	–	2407.61	22	2322.07	34	1272.46	16	88	170	1220.00	15	48	201	1220.00	15	48	201
CMT9Y	150	1299^{D}	15	–	2407.61	22	2322.07	34	1213.11	15	79	171	1250.86	16	11	185	1213.11	15	79	171
CMT10X	199	1571^{D}	19	–	3087.12	28	2871.83	43	1517.77	20	93	269	1466.33	19	157	301	1464.58	19	157	301
CMT10Y	199	1571^{D}	19	–	3087.12	28	2871.83	43	1427.00	18	184	286	1419.79	18	80	318	1419.79	18	80	318

Previous best result is from Chen & Wu (C), Tang & Galvão (T) or Dethloff (D)

v: number of vehicles used in the solution. TB: CPU time (in seconds) to best; TT: Total CPU time

Lower bounds are given only for non-time-constrained instances runs of our algorithm are pretty good, as all of them improved on the previous best results for the majority of instances. Furthermore, running our algorithm just once (with $\alpha = 10\%$) gives in most (26 out of 28) cases results that are within 2% from the overall best result of five runs.

Finally, we would like to briefly comment on other experimentation not shown in tables. We looked at the effect of the post-optimizer *local-shift* and the *reverse* routines. We saw that in several cases *local-shift* was able to make further improvements to the 1-exchange moves in the main reactive tabu search framework. While it is difficult to see just when a *reverse* operation bears fruit (as it does not yield an immediate cost improvement), it was executed several times during the run of the RTS-VRPSPD algorithm, and as it has an insignificant computational cost, there is no sense in not using it. (We carried out some preliminary tests with and without the *reverse* routine and we found that including it led to cost improvements.)

5 Conclusions and suggestions

5.1 Conclusions

We designed a heuristic to solve the vehicle routing problem with pickups and deliveries. We identified the main difficulty in this problem as that of checking load feasibility. Hence, the *inner core* of our algorithm is a quick and elegant way of *checking feasibility* for neighborhood moves. In fact, the feasibility checks do not increase the computational complexity of the neighborhood. The *outer shell* of our algorithm is a *reactive tabu search* framework. This forcefully guides the search towards better solutions by dynamically maintaining the tabu list size. We found several new best solutions to well-known test problems. In particular on those instances for which several results are available, our algorithm yielded 10 new best results (and came within 4% of the best result on the remaining four instances) and reduced the gap from the lower bound by 17%.

5.2 Suggestions for future work

The reactive tabu concept could be enhanced by the addition of Adaptive Memory Programming. A hybrid of these two was previously used successfully by Wassan (2006b) to solve the VRP with backhauls.

Another attractive way of enhancing tabu search is by incorporating it in a variable neighborhood framework, as in Crispim and Brandão (2005). Combining reactive tabu search with variable neighborhood descent is likely to yield a powerful algorithm.

The search could be allowed to traverse through infeasible solutions as this may find a "shortcut" to better quality feasible solutions, as in the strategic oscillation method of Nagy (1996) and in Crispim and Brandão (2005).

The model can be extended for multiple depots as in Nagy and Salhi (2005). As many of the routines in our method are capable of handling multiple depots, this should be a reasonably straightforward extension. A more challenging extension would be to allow for a heterogeneous fleet.

Mixed models have often been solved using an insertion-based method, although we argued previously that even these models could be solved using a method that considers linehaul and backhaul customers in an integrated manner. We propose to use the reactive tabu search methodology described in this paper to solve mixed instances. (This is in line with Dethloff 2002, who applied his simultaneous method to the mixed case with good results.)

Acknowledgements The authors are grateful for the constructive and helpful comments of two anonymous referees.

References

Anily S, Mosheiov G (1994) The traveling salesman problem with delivery and backhauls. Oper Res Lett 16:11–18

Battiti R, Tecchioli G (1994) Reactive tabu search. ORSA J Comput 6:126–140

Chen J-F, Wu T-H (2006) Vehicle routing problem with simultaneous deliveries and pickups. J Oper Res Soc 57:579–587

Christofides N, Mingozzi A, Toth P (1979) The vehicle routing problem. In: Christofides N, Mingozzi A, Toth P, Sandi C (eds) Combinatorial optimization. Wiley, Chichester, pp 315–338

Clarke G, Wright J (1964) Scheduling of vehicles for a central depot to a number of delivery points. Oper Res 12:568–581

Cordeau J-F, Gendreau M, Laporte G, Potvin J-Y, Semet F (2002) A guide to vehicle routing heuristics. J Oper Res Soc 53:512–522

Crispim J, Brandão J (2005) Metaheuristics applied to mixed and simultaneous extensions of vehicle routing problems with backhauls. J Oper Res Soc 56:1296–1302

Dethloff J (2001) Vehicle routing and reverse logistics: the vehicle routing problem with simultaneous delivery and pick-up. OR Spektrum 23:79–96

Dethloff J (2002) Relation between vehicle routing problems: an insertion heuristic for the vehicle routing problem with simultaneous delivery and pick-up applied to the vehicle routing problem with backhauls. J Oper Res Soc 53:115–118

Drezner Z (1982) Fast algorithms for the round trip location problem. IEE Trans 14:243–248

Fabri A, Recht P (2006) On dynamic pickup and delivery vehicle routing with several time windows and waiting times. Transportation Res Part B 40:335–350

Gaskell T (1967) Bases for vehicle fleet scheduling. Oper Res Quart 18:281–295

Gendreau M, Laporte G, Vigo D (1999) Heuristics for the traveling salesman problem with pickup and delivery. Comput Oper Res 26:699–714

Gillett BE, Miller LR (1974) A heuristic algorithm for the vehicle-dispatch problem. Oper Res 22:340–349

Glover F, Laguna M (1997) Tabu search. Kluwer, New York

Halse K (1992) Modeling and solving complex vehicle routing problems. PhD thesis, Institute of Mathematical Statistics and Operations Research, Technical University of Denmark, Lyngby

Irnich S (2000) A multi-depot pickup and delivery problem with a single hub and heterogeneous vehicles. Eur J Oper Res 122:310–328

Min H (1989) The multiple vehicle routing problem with simultaneous delivery and pick-up points. Transp Res A 23:377–386

Mosheiov G (1994) The traveling salesman problem with pick-up and delivery. Eur J Oper Res 79:299–310

Mosheiov G (1995) The pickup delivery location problem on networks. Networks 26:243–251

Nagy G (1996) Heuristic methods for the many-to-many location-routing problem. PhD thesis, School of Mathematics and Statistics, The University of Birmingham, Birmingham

Nagy G, Salhi S (2005) Heuristic algorithms for single and multiple depot vehicle routing problems with pickups and deliveries. Eur J Oper Res 162:126–141

Nanry WP, Barnes JW (2000) Solving the pickup and delivery problem with time windows using reactive tabu search. Transp Res Part B 34:107–121

Osman IH, Wassan NA (2002) A reactive tabu search meta-heuristic for the vehicle routing problem with backhauls. J Sched 5:263–285

Salhi S, Nagy G (1999) A cluster insertion heuristic for single and multiple depot vehicle routing problems with backhauling. J Oper Res Soc 50:1034–1042

Tang FA, Galvão RD (2006) A tabu search algorithm for the vehicle routing problems with simultaneous pickup and delivery service. Comput Oper Res 33:595–619

Thangiah SR, Sun T, Potvin J-Y (1996) Heuristic approaches to vehicle routing with backhauls and time windows. Comput Oper Res 23:1043–1057

Toth P, Vigo D (2002) VRP with backhauls. In: Toth P, Vigo D (eds) The vehicle routing problem. Society for Industrial and Applied Mathematics, Philadelphia, pp 195–224

Wade AC (2002) Constructive and ant system heuristics for a class of vehicle routing problems with backhauls. PhD thesis, School of Mathematics and Statistics, The University of Birmingham, Birmingham

Wade AC, Salhi S (2002) An investigation into a new class of vehicle routing problem with backhauls. Omega 30:479–487

Wade AC, Salhi S (2003) An ant system algorithm for the mixed vehicle routing problem with backhauls. In: Resende MG, de Sousa JP (eds) Metaheuristics: computer decision-making. Kluwer, New York, pp 699–719

Wassan NA (2006a) A reactive tabu search for the vehicle routing problem. J Oper Res Soc 57:111–116

Wassan NA (2006b) Reactive tabu adaptive memory programming search for the vehicle routing problem with backhauls. J Oper Res Soc (available online)

Wassan NA, Osman IH (2002) Tabu search variants for the mix fleet vehicle routing problem. J Oper Res Soc 53:768–782