J Sched (2009) 12: 17–30
DOI 10.1007/s10951-008-0072-x

A comparison of five heuristics for the multiple depot vehicle scheduling problem

Ann-Sophie Pepin · Guy Desaulniers · Alain Hertz · Dennis Huisman

Received: 15 April 2007 / Accepted: 26 May 2008 / Published online: 12 July 2008

Abstract Given a set of timetabled tasks, the multi-depot vehicle scheduling problem consists of determining least-cost schedules for vehicles assigned to several depots such that each task is accomplished exactly once by a vehicle. In this paper, we propose to compare the performance of five different heuristics for this well-known problem, namely, a truncated branch-and-cut method, a Lagrangian heuristic, a truncated column generation method, a large neighborhood search heuristic using truncated column generation for neighborhood evaluation, and a tabu search heuristic. The first three methods are adaptations of existing methods, while the last two are new in the context of this problem. Computational results on randomly generated instances show that the column generation heuristic performs the best when enough computational time is available and stability is required, while the large neighborhood search method is the best alternative when looking for good quality solutions in relatively fast computational times.

A.-S. Pepin
Giro Inc., 75 Port-Royal East, Suite 500, Montreal, Canada H3L 3T1
e-mail: annsophie.pepin@giro.ca

G. Desaulniers (✉) · A. Hertz
Department of Mathematics and Industrial Engineering, École Polytechnique and GERAD, P.O. Box 6079, succ. Centre-ville, Montreal, Canada H3C 3A7
e-mail: guy.desaulniers@gerad.ca

A. Hertz
e-mail: alain.hertz@gerad.ca

D. Huisman
Econometric Institute and ECOPT, Erasmus University Rotterdam, P.O. Box 1738, 3000 DR Rotterdam, The Netherlands
e-mail: huisman@few.eur.nl

Keywords Vehicle scheduling · Multiple depot · Heuristics · Branch-and-cut · Column generation · Lagrangian heuristic · Tabu search · Large neighborhood search

1 Introduction

The class of vehicle routing and scheduling problems is an important class of problems studied by many researchers in Operations Research. Because most of these combinatorial optimization problems are NP-hard, exact methods often cannot solve large instances encountered in practice. For tackling these instances, various heuristic approaches have been developed, ranging from local search methods to methods based on mathematical programming decomposition techniques, including metaheuristics. On the one hand, heuristics based on decomposition techniques such as column generation and Lagrangian relaxation can often provide very good quality solutions when sufficient computational time is available. On the other hand, metaheuristics such as tabu search are known to be able to find good solutions rather rapidly, especially when solving very large-scale instances. Therefore, both types of approaches which are very popular nowadays have different advantages and disadvantages.

To our knowledge, no single study comparing decomposition-technique-based heuristic methods to metaheuristics for solving the same vehicle routing and scheduling problem has been published in the literature. In this paper, we propose such a study and compare the performance of five heuristic methods for solving the multiple depot vehicle scheduling problem (MDVSP). These heuristics are a truncated branch-and-cut method (relying on the CPLEX MIP

solver), a Lagrangian heuristic, a truncated column generation method, a large neighborhood search heuristic using truncated column generation for neighborhood evaluation, and a tabu search heuristic. The first and third heuristics are adaptations of the exact methods proposed by Kliewer et al. (2006) and Ribeiro and Soumis (1994), the second heuristic is a variant of the heuristic approach developed by Lamatsch (1992), while the other two heuristics are, to our knowledge, the first proposed metaheuristics for the MDVSP. These metaheuristics rely on similar methods developed by Ropke and Pisinger (2004) and Cordeau et al. (2001), which are state-of-the-art methods for different vehicle routing problems.

The MDVSP is a well-known problem that has several applications in different fields such as public transit and the trucking industry. It can be defined as follows. Given a set T of timetabled tasks and a fleet of vehicles housed in a set K of depots, find least-cost feasible vehicle schedules such that each task is accomplished exactly once by a vehicle and the number v_k of vehicles available in each depot $k \in K$ is not exceeded. Each task $i \in T$ is defined by a start location s_i, an end location e_i (which might be the same as s_i), a start time a_i, and a duration δ_i that includes travel time between s_i and e_i. A vehicle schedule must start and end at the same depot and is composed of an ordered sequence of tasks. It is feasible if, for every pair i and j of consecutive tasks it contains, the relation $a_i + \delta_i + t_{ij} \leq a_j$ holds, where t_{ij} is the travel time between locations e_i and s_j. The cost of a schedule assigned to a vehicle in depot k is given by the sum of the traveling and waiting costs incurred between two of its consecutive tasks, between the pull-out of the depot and its first task, and between its last task and the return to the depot. The cost of a schedule may also include a fixed vehicle cost. Note that additional constraints restricting for each task the set of depots that can supply the vehicle to accomplish it might be imposed. Such constraints, which simplify the problem by reducing the number of feasible solutions, are not considered in this paper although they could easily be handled by the proposed heuristics.

The literature on the MDVSP is abundant (see the surveys of Odoni et al. 1994; Desrosiers et al. 1995; Desaulniers and Hickman 2007). From the 1970s to the early 1990s, several heuristic solution approaches have been proposed. For instance, Bodin et al. (1983) developed a two-phase heuristic that first computes vehicle schedules by solving a single-depot vehicle scheduling problem and assigns them to the depots by solving a transportation problem; Bertossi et al. (1987), Mesquita and Paixão (1992), and Lamatsch (1992) designed Lagrangean heuristics; and Dell'Amico et al. (1993) introduced a heuristic that first solves a sequence of shortest path problems to build a good quality solutions before applying different reoptimization procedures on this solution. To our knowledge, no metaheuristics have been developed for the MDVSP. Indeed, research on this problem changed its focus towards exact solution approaches at the beginning of the 1990s, that is, before the metaheuristics became very popular. Exact branch-and-bound methods have been developed by Carpaneto et al. (1989) and Kliewer et al. (2006), while exact column generation methods have been proposed by Ribeiro and Soumis (1994), Löbel (1997, 1998), Hadjar et al. (2006). Note that Löbel (1997, 1998) and Kliewer et al. (2006) succeeded to solve real-world public transit instances to optimality involving up to 7000 tasks. These instances, however, have a particular structure that ease their solution process. Indeed, for most trips (tasks), it is easy to determine the successor trip in optimal vehicle schedules covering them because, in general, when a trip ends at a terminal, another trip starts from there a few minutes later. For less structured MDVSP instances such as the randomly generated ones proposed in Carpaneto et al. (1989) and used subsequently by many other researchers, instances involving only up to 800 tasks can be solved to optimality (see Hadjar et al. 2006).

Although exact methods are able to solve certain classes of real-world MDVSP instances, there are several reasons to pursue the development of efficient heuristics for the MDVSP and to compare the performance of such heuristics. First, MDVSPs need to be solved in very short computational times in certain situations. This is required for instance when a planned solution must be reoptimized in an operational context or when the MDVSP appears as a subproblem in more complex problems. In this latter case, decomposition methods such as the one proposed by Huisman et al. (2005) for the multiple depot vehicle and crew scheduling problem solve the MDVSP subproblem several times during the overall solution process. Good heuristics can therefore help to improve algorithms for such complex problems as well. Second, many software packages used in public and freight transportation solve MDVSPs. Most often, heuristics are used to rapidly compute relatively good solutions. Consequently, comparing different classes of heuristics for the MDVSP can certainly help the practitioners developing these softwares to choose which heuristics are the most effective.

The contributions of this paper are as follows. We develop two metaheuristics for the MDVSP that are based on state-of-the-art methods for other vehicle routing problems. These are the first metaheuristics for the MDVSP. We also adapt three mathematical programming approaches previously proposed for the MDVSP. For these five heuristics, we provide computational results that allow to compare their performance. To our knowledge, this comparison is the first involving decomposition-technique-based heuristics and metaheuristics applied in exactly the same setting. The results also show the effectiveness of most of these algorithms. It should be noted that the primary goal of this paper is not to propose totally new heuristics for the MDVSP, but rather to compare heuristics based on trendy methodologies.

This paper is organized as follows. In Sect. 2, we formally define the MDVSP and provide two mathematical formulations for it that will be used by some of the solution approaches. The following five sections present the five heuristic solution approaches. Computational results on random instances generated as in Carpaneto et al. (1989) are then reported and discussed in Sect. 8. Finally, we draw some conclusions in Sect. 9.

2 MDVSP formulations

In this section, we present two mathematical formulations for the MDVSP: an integer multi-commodity formulation and a set partitioning type formulation. The former formulation is at the basis of the branch-and-cut and the Lagrangian heuristics, while the column generation heuristic relies on the latter formulation.

2.1 Multi-commodity formulation

Several authors, including Bodin et al. (1983), Bertossi et al. (1987), Forbes et al. (1994), Ribeiro and Soumis (1994), and Kliewer et al. (2006), have formulated the MDVSP as an integer multi-commodity network flow model, where a commodity is defined for each depot in K. Here, we begin by presenting the model described in Bodin et al. (1983) and Ribeiro and Soumis (1994). Consider a network $G^k = (V^k, A^k)$ for each depot $k \in K$, where V^k and A^k denote its node and arc sets, respectively. Set V^k contains one node for each task $i \in T$ and one pair of nodes, $o(k)$ and $d(k)$, representing the start and the end of a vehicle schedule associated with depot k, respectively. Thus, $V^k = \{o(k), d(k)\} \cup T$. Set A^k contains three types of arcs: pull-out, pull-in, and connection arcs. There is a pull-out arc $(o(k), i)$ for each task node $i \in T$. Symmetrically, there is a pull-in arc $(i, d(k))$ for each task node $i \in T$. Finally, there is a connection arc (i, j) for each pair of task nodes, i and j in T, such that $a_i + \delta_i + t_{ij} \leq a_j$. The cost of an arc $(i, j) \in A^k$, denoted c_{ij}, is equal to the travel and waiting costs associated with it. If $i = o(k)$ and a fixed cost must be paid for each vehicle used, c_{ij} also includes this cost. It is easy to see that there is a one-to-one correspondence between the paths from $o(k)$ to $d(k)$ in G^k and the feasible vehicle schedules for depot k. In the following, we refer to this type of network as a connection network.

The proposed formulation involves the binary variables X^k_{ij}, $(i, j) \in A^k$, $k \in K$. Such a variable is equal to the flow of commodity k on the arc (i, j). Using this notation, the MDVSP can be modeled as:

$$\text{Minimize} \sum_{k \in K} \sum_{(i,j) \in A^k} c_{ij} X^k_{ij} \tag{1}$$

subject to:

$$\sum_{k \in K} \sum_{j:(i,j) \in A^k} X^k_{ij} = 1, \quad \forall i \in T, \tag{2}$$

$$\sum_{j:(o(k),j) \in A^k} X^k_{o(k),j} \leq v_k, \quad \forall k \in K, \tag{3}$$

$$\sum_{j:(j,i) \in A^k} X^k_{ji} - \sum_{j:(i,j) \in A^k} X^k_{ij} = 0,$$

$$\forall i \in V^k \setminus \{o(k), d(k)\},\ k \in K, \tag{4}$$

$$X^k_{ij} \in \{0, 1\}, \quad \forall (i, j) \in A^k,\ k \in K. \tag{5}$$

Function (1) minimizes total costs. Constraints (2) ensure that each task is executed exactly once by a vehicle. Constraints (3) limit the number of vehicles that can be used from each depot, while constraints (4) are flow conservation constraints which define a multiple-path structure for each depot. Finally, variable binary requirements are provided by (5).

Very recently, Kliewer et al. (2006) proposed a similar multicommodity model based on a different network structure, namely, a time–space network structure. This network structure can be quite advantageous with respect to the connection network structure when the number of start and end task locations is small compared to the number of tasks. For depot $k \in K$, the time–space network $\mathcal{G}^k = (\mathcal{V}^k, \mathcal{A}^k)$ is defined as follows Its node set $\mathcal{V}^k$ contains the nodes $o(k)$ and $d(k)$, the nodes $i \in T$ that in this case only represent the starts of the tasks, and finally nodes that represent the ends of the tasks. The set of these end tasks nodes is denoted E. Thus, $\mathcal{V}^k = \{o(k), d(k)\} \cup T \cup E$. Figure 1 illustrates the network $\mathcal{G}^k$ for an example that involves six tasks linking three locations A, B, and C. In this figure, time increases from left to right along the horizontal axis, while the locations are distributed along the vertical axis. For this example, $T = \{1, 2, \ldots, 6\}$ and $E = \{1', 2', \ldots, 6'\}$.

Each node in $T \cup E$ is associated with a time and a location (start or end time and location of the corresponding task). Let W be the set of these start and end locations and assume that the nodes associated with each location $w \in W$ are sorted in chronological order (end nodes before start nodes in case of equality). Denote by f_w and l_w the first and last nodes at location $w \in W$, respectively. Arc set $\mathcal{A}^k$ contains five arc types: pull-out, pull-in, task, wait, and deadhead arcs. There is a pull-out arc $(o(k), f_w)$ and a pull-in arc $(l_w, d(k))$ for each location $w \in W$. For each task in T, there is a task arc (i, j) linking its start node $i \in T$ to its end node $j \in E$. For each location $w \in W$, there is a chain of wait arcs linking the consecutive nodes associated with this location. Deadhead arcs allow to reposition a vehicle from one location where a task just ended to a different location where a task is about to start. Instead of considering

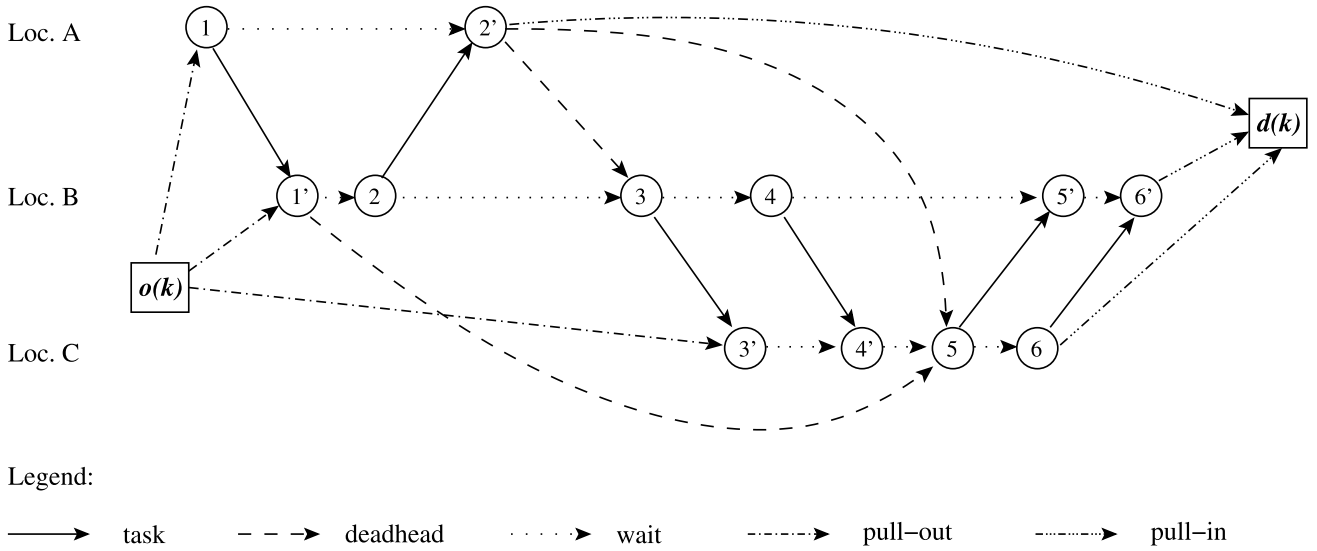

Fig. 1 Example of a time-space network $\mathcal{G}^k$

all possible deadhead arcs, Kliewer et al. (2006) proposed to aggregate them for drastically reducing their number. With this aggregation procedure, there is a deadhead arc linking an end node $i \in E$ (associated with time $a_i + \delta_i$ at location e_i) to a start node $j \in T$ (associated with time a_j and location s_j) if and only if there is no task i' ending at e_i after time $a_i + \delta_i$ such that $a_{i'} + \delta_{i'} + t_{i'j} \leq a_j$ and no task j' starting at s_j before time a_j such that $a_i + \delta_i + t_{ij'} \leq a_{j'}$. For the example, this aggregation procedure allows to omit deadhead arcs $(1', 6)$, $(2', 4)$, and $(2', 6)$.

With such a time–space network structure, model (1)–(5) remains valid for the MDVSP after replacing A^k by $\mathcal{A}^k$, V^k by $\mathcal{V}^k$, and constraints (2) and (5) by

$$\sum_{k \in K} \sum_{j:(i,j) \in \mathcal{A}_T^k} X_{ij}^k = 1, \quad \forall i \in T, \tag{6}$$

$$X_{ij}^k \in \{0, 1\}, \quad \forall (i, j) \in \mathcal{A}_T^k, \ k \in K, \tag{7}$$

$$X_{ij}^k \geq 0, \text{ integer}, \quad \forall (i, j) \in \mathcal{A}^k \setminus \mathcal{A}_T^k, \ k \in K, \tag{8}$$

where $\mathcal{A}_T^k$ is the subset of task arcs in $\mathcal{A}^k$, $k \in K$. Indeed, with a time–space network structure, the flow on all arcs except the task arcs can exceed one.

2.2 Set partitioning type formulation

Ribeiro and Soumis (1994) also formulated the MDVSP as a set partitioning model with side constraints which can be derived from model (1)–(5) using Dantzig–Wolfe decomposition (Dantzig and Wolfe 1960; Desaulniers et al. 1998a). Let Ω^k be the set of all feasible vehicle schedules for depot $k \in K$. For each schedule $p \in \Omega^k$, define the following parameters: its cost c_p and, for each task $i \in T$, a binary parameter a_{ip} equal to 1 if schedule p includes task i and 0 otherwise. Furthermore, with each schedule $p \in \Omega^k$, define a binary variable θ_p that takes value 1 if p is retained in the solution and 0 otherwise.

The MDVSP can then be modeled as:

$$\text{Minimize} \sum_{k \in K} \sum_{p \in \Omega^k} c_p \theta_p \tag{9}$$

subject to:

$$\sum_{k \in K} \sum_{p \in \Omega^k} a_{ip} \theta_p = 1, \quad \forall i \in T, \tag{10}$$

$$\sum_{p \in \Omega^k} \theta_p \leq v_k, \quad \forall k \in K, \tag{11}$$

$$\theta_p \in \{0, 1\}, \quad \forall p \in \Omega^k, \ k \in K. \tag{12}$$

In this model, the objective function (9) aims at minimizing total costs. Set partitioning constraints (10) impose that each task be accomplished by exactly one vehicle, while inequalities (11) express the availability constraints per depot. Finally, binary requirements on the variables are given by (12).

3 Truncated branch-and-cut

The first heuristic approach for the MDVSP that we propose is to solve the integer multi-commodity model (1), (3), (4), (6)–(8) relying on time–space networks using a heuristic branch-and-cut method, namely, the CPLEX MIP solver, version 9.0.1. The implemented method is a heuristic because

it stops as soon as it finds a first integer solution. To favor the obtention of good quality solutions in relatively fast solution times, the CPLEX MIP emphasis parameter was set to "balance optimality and feasibility". Preliminary tests also showed that it was preferable to use the barrier algorithm at the root node of the search tree. All other CPLEX parameters were left to their default values.

Notice that preliminary tests on the instances used for experiments were also performed using the connection networks and model (1)–(5). The results of these tests indicated that solution times were reduced by approximately 30 to 50% using the time–space networks instead of the connection networks, while maintaining the same level of solution quality.

4 Lagrangian heuristic

A Lagrangian heuristic uses Lagrangian relaxation (see Geoffrion 1974) to compute lower bounds. Lagrangian relaxation transfers some of the constraints into the objective function using Lagrangian multipliers to define a so-called Lagrangian subproblem. For any given values of the multipliers, the optimal value of this subproblem provides a lower bound. The problem of finding multiplier values that yield the largest lower bound is called the Lagrangian dual problem. In a Lagrangian heuristic, this problem is solved heuristically using an iterative non-smooth optimization algorithm such as a subgradient algorithm. Since, in most (often all) iterations, the optimal solution of the Lagrangian subproblem is infeasible for the original problem, feasible solutions must be derived using an ad hoc heuristic procedure. This procedure is invoked at each iteration if it is not too time consuming, or only at the last one otherwise.

For the MDVSP, Bertossi et al. (1987) and Kokott and Löbel (1996) proposed Lagrangian heuristics in which the task covering constraints are relaxed. Mesquita and Paixão (1992), Lamatsch (1992), and Kokott and Löbel (1996) also developed Lagrangian relaxation approaches where the flow conservation constraints are relaxed instead. In this section, we present a Lagrangian heuristic similar to those of Lamatsch (1992) and Kokott and Löbel (1996) for computing lower bounds and that derives feasible solutions throughout the solution process. This heuristic relies on the connection networks and model (1)–(5) augmented by the redundant constraints:

$$\sum_{k\in K}\sum_{j:(j,i)\in A^k} X^k_{ji} = 1, \quad \forall i \in T. \tag{13}$$

The Lagrangian subproblem is obtained by omitting the availability constraints (3) and relaxing in a Lagrangian way the flow conservation constraints (4) using multipliers λ^k_i, $i \in T$, $k \in K$. This subproblem is formulated as follows:

$$\phi(\boldsymbol{\lambda}) = \text{Minimize} \sum_{k\in K}\sum_{(i,j)\in A^k} (c_{ij} + \lambda^k_j - \lambda^k_i) X^k_{ij} \tag{14}$$

subject to:

$$\sum_{k\in K}\sum_{j:(i,j)\in A^k} X^k_{ij} = 1, \quad \forall i \in T, \tag{15}$$

$$\sum_{k\in K}\sum_{j:(j,i)\in A^k} X^k_{ji} = 1, \quad \forall i \in T, \tag{16}$$

$$X^k_{ij} \in \{0,1\}, \quad \forall (i,j) \in A^k,\ k \in K. \tag{17}$$

Subproblem (14)–(17) is equivalent to a single-depot vehicle scheduling problem (SDVSP), which can be solved in polynomial time. This can be seen by replacing, for each arc (i, j), the variables X^k_{ij} by a single variable corresponding to the variable X^k_{ij} with the lowest reduced cost, i.e., with the smallest value $\lambda^k_j - \lambda^k_i$ over all k. The other variables are set to 0. To solve the SDVSP, the auction algorithm of Freling et al. (2001) is used. This algorithm starts with an empty set of schedules and an arbitrary set of dual variable values for the constraints (15) and (16). Then it switches between forward and backward auction iterations to adjust these dual values until an optimal dual solution is found. In the forward iterations, every task without a forward assignment (that is, it has no successor yet) has to bid for such an assignment, which is either a successor task or the sink. The backward auction is a similar procedure.

The Lagrangian dual function $\phi(\boldsymbol{\lambda})$ yields a lower bound for each vector $\boldsymbol{\lambda}$. We use an iterative subgradient algorithm to obtain a good lower bound. Furthermore, an upper bound is computed at each iteration of this algorithm. It corresponds to the value of a feasible solution for the MDVSP that is constructed from the values of the variables X^k_{ij} in the subproblem optimal solution. Indeed, this solution can be seen as a set of disjoint paths from a source node to a sink node, but where the arcs in the same path can be assigned to different depots. A feasible solution can then be obtained by assigning a unique depot to each of these paths, where at most v_k paths can be assigned to depot k. The cost of assigning a path p to a depot k is again given by c^k_p (see Sect. 2.2). The problem of finding the best assignment of paths to depots corresponds now to the classical transportation problem, which can be solved in polynomial time.

The complete method is detailed in Algorithm 1, where the index n is an iteration counter. The algorithm is initialized by setting the Lagrangian multipliers equal to 0. In Step 1, the Lagrangian subproblem is solved and a subgradient is calculated. In Step 2, a feasible solution is constructed in the way described above. This solution provides an upper bound on the objective value. The best upper bound UB is

Algorithm 1 Lagrangian heuristic

Step 0: Initialization

Choose parameters $n_{\max}$, α^0, γ, and ϵ
Set $UB \leftarrow \infty$, $LB \leftarrow -\infty$, $n \leftarrow 0$, $m \leftarrow 0$, and
$\boldsymbol{\lambda}^0 \leftarrow 0$

Step 1: Lower bound and subgradient

Solve the Lagrangian subproblem (14)–(17) to obtain a solution $\boldsymbol{X}^n$
and a lower bound $\phi(\boldsymbol{\lambda}^n)$
Compute the subgradient components
$Y_i^{k,n} \leftarrow \sum_{j:(j,i)\in A^k} X_{ji}^{k,n} - \sum_{j:(i,j)\in A^k} X_{ij}^{k,n}$

Step 2: Upper bound

Compute an upper bound UB^n by solving a transportation problem
If $UB^n < UB$ **then** set $UB \leftarrow UB^n$

Step 3: Lagrangian multipliers update

Set $\lambda_i^{k,n+1} \leftarrow \lambda_i^{k,n} + \alpha^n \frac{UB-\phi(\boldsymbol{\lambda}^n)}{\sum_{k\in K}\sum_{i\in T}(Y_i^{k,n})^2} Y_i^{k,n}$

Step 4: Parameters update

If $\phi(\boldsymbol{\lambda}^n) > LB$ **then** set $m \leftarrow 0$ and $LB \leftarrow \phi(\boldsymbol{\lambda}^n)$
else set $m \leftarrow m + 1$
If $m = \gamma$ **then** set $\alpha^{n+1} \leftarrow \alpha^n/2$
else set $\alpha^{n+1} \leftarrow \alpha^n$

Step 5: Termination criterion

If $UB = LB$, $\sum_{i\in T}\sum_{k\in K}(Y_i^{k,n})^2 \le \epsilon$, $\alpha^n \le \epsilon$ or $n \ge n_{\max}$ **then** STOP
else set $n \leftarrow n + 1$ and return to Step 1

then updated every time that an improvement is found. In Step 3, the subgradient vector is used to update the multipliers in the direction of the subgradient. The step size is determined by the difference between the lower and the best upper bound found so far, the norm of the subgradient, and a certain parameter α. This parameter is updated in Step 4 in order to ensure the convergence of the subgradient algorithm, that is, α is halved after a certain number of iterations (γ) without improvement in the lower bound. Finally, the procedure terminates when the upper bound is equal to the lower bound, the norm of the subgradient vector is close to 0, α is very small, or a maximum number of iterations ($n_{\max}$) is reached. Note that, for our tests, the parameters were set to the following values that were selected based on preliminary tests: $n_{\max} = 10000$, $\alpha^0 = 1.0$, $\gamma = 10$, and $\epsilon = 0.000001$. These values were chosen after an extensive computational study.

5 Truncated column generation

Column generation (Dantzig and Wolfe 1960, and Gilmore and Gomory 1961) embedded in a branch-and-bound scheme is a well-known approach for solving a wide variety of vehicle routing and crew scheduling problems (see Barnhart et al. 1998, and Desaulniers et al. 1998a). Such an approach, also called branch-and-price, was first introduced for the exact solution of the MDVSP by Ribeiro and Soumis (1994). Here, we present a heuristic version of this approach similar to the one proposed by Desaulniers et al. (1998b) for the MDVSP with time windows and waiting costs.

Column generation is used for solving the linear relaxation of model (9)–(12), called the master problem, which typically contains a huge number of variables. It is an iterative method that avoids enumerating all variables by decomposing the problem into two parts: a restricted master problem (RMP) and one subproblem per depot. At iteration n, the RMP is simply the master problem restricted to a subset of the variables θ_p, that is, those for which schedule p belongs to a subset $\Omega_n^k \subseteq \Omega^k$, $k \in K$, of the schedules. Solving the RMP provides a primal solution and a dual solution. To prove that this primal solution is optimal for the overall master problem, one must verify that the reduced costs of the variables not yet generated (those for which $p \in \bigcup_{k\in K}(\Omega^k \setminus \Omega_n^k)$) are all non-negative. The role of the subproblems is to verify if this condition is satisfied and, if not, to propose one or several variables (columns) with a negative reduced cost. For the MDVSP, the subproblem for depot $k \in K$ is a shortest path problem from $o(k)$ to $d(k)$ in the connection network G^k with a modified cost structure. Denoting by π_n^i, $i \in T$, and β_n^k, $k \in K$, the values of the dual variables associated with constraints (10) and (11) of the RMP at iteration n, the modified cost of arc $(i, j) \in A^k$ is given by $c_{ij} - \pi_n^i$ if $i \in T$ and $c_{ij} - \beta_n^k$ if $i = o(k)$. With these altered costs, the cost of a path p in G^k corresponds to the reduced cost of the variable θ_p. Hence, by solving this subproblem at iteration n, the schedule $p \in \Omega^k$ with the smallest reduced cost can be identified. The current RMP solution is thus optimal for the overall master problem if the optimal value of each subproblem is non-negative.

It is well-known that the convergence of the column generation method is rather slow at the end of the solution process, that is, the RMP objective value does not decrease much in the last iterations. To avoid such a tailing-off which is, in general, useless in a heuristic approach, the column generation process is halted when the optimal value of the RMP has not decreased by more than $Z_{\min}$ in the last I iterations of a linear relaxation, where $Z_{\min}$ and I are predefined parameters. Once the column generation process is stopped, the computed solution can be integer or fractional. In the former case, the algorithm stops. In the latter case, a rounding procedure rounds up to 1 some of the fractional-valued

Algorithm 2 Truncated column generation

Step 0: Initialization

Choose parameters $Z_{\min}$, I, and $\theta_{\min}$
Set $n \leftarrow 1$, $n_0 \leftarrow 1$, and $\Omega_n^k \leftarrow \emptyset$
Add artificial variables Λ^i to (10) with a big-M coefficient in (9)

Step 1: Restricted master problem

Solve the RMP restricted to the subsets Ω_n^k, $k \in K$ to obtain a primal solution $(\boldsymbol{\theta}_n, \boldsymbol{\Lambda}_n)$
and a dual solution $(\boldsymbol{\pi}_n, \boldsymbol{\beta}_n)$ of cost z_n^{RMP}

Step 2: Column generation early termination test

If $\boldsymbol{\Lambda}_n = 0$, $n - n_0 > I$ and $z_{n-I}^{\text{RMP}} - z_n^{\text{RMP}} < Z_{\min}$ **then** goto to Step 6

Step 3: Subproblems

For all depot $k \in K$ **do**
Update the cost of the arcs in A^k according to $(\boldsymbol{\pi}_n, \boldsymbol{\beta}_n)$
Solve the shortest path problem on G^k
Denote its optimal value by $z_n^{\text{SUB},k}$

Step 4: Column generation optimality test

If $z_n^{\text{SUB},k} \geq 0$, $\forall k \in K$ **then** goto to Step 6

Step 5: Update restricted master problem

Get from the subproblems sets S_n^k of computed schedules with negative reduced costs
Set $\Omega_{n+1}^k \leftarrow \Omega_n^k \cup S_n^k$ and $n \leftarrow n + 1$
Return to Step 1

Step 6: Feasibility test

If $\boldsymbol{\Lambda}_n \neq 0$ **then** STOP: no feasible solution has been found

Step 7: Integrability test

If $\boldsymbol{\theta}_n$ is integer **then** STOP: $\boldsymbol{\theta}_n$ is a feasible solution

Step 8: Rounding

For all fractional-valued variables θ_p such that $\theta_{p,n} \geq \theta_{\min}$ **do**
Set $\theta_p \leftarrow 1$
If no such variables exist **then**
Set $\theta_{p^*} \leftarrow 1$, where θ_{p^*} is the variable with the highest fractional value in $\boldsymbol{\theta}_n$.
Set $n \leftarrow n + 1$ and $n_0 \leftarrow n$
Return to Step 1

variables θ_p and the column generation process is re-started. The details of the truncated column generation method are presented in Algorithm 2.

To start the column generation process, artificial variables Λ_i, $i \in T$, are added in constraints (10) of the RMP. These variables bear a very large cost in the objective function (9). In Step 1, the current RMP is solved using a linear programming algorithm. Step 2 verifies if the column generation process needs to be truncated in order to avoid tailing-off. In Step 3, the shortest path subproblems are solved using a label-setting algorithm (see Ahuja et al. 1993). If the optimal value of each subproblem is non-negative (Step 4), the column generation process stops as the current RMP solution is optimal for the current linear relaxation. Otherwise, columns associated with the negative reduced cost paths identified by the subproblems are added to the RMP in Step 5 and another column generation iteration is performed. When column generation is halted, tests for feasibility (Step 6) and integrability (Step 7) are performed. A positive test stops the overall method either with or without a feasible solution. When both tests are negative, Step 8 executes the following rounding procedure. All variables θ_p taking a fractional value in the current RMP solution greater than or equal to a predetermined threshold value $\theta_{\min}$ are rounded up to 1. If no such variables exist, then the variable with the highest fractional value is rounded up to 1. These rounded-up variables modify the current RMP which is then reoptimized in Step 1.

Note that this rounding procedure might fail to generate an integer solution when vehicle availability is tight. However, this situation did not occur in our experiments. Note also that, to speed up the overall solution process, all the nodes in the networks G^k, $k \in K$, representing the tasks covered by the rounded-up variables and their incident arcs can be removed from these networks after the rounding procedure.

The column generation heuristic was implemented using version 4.5 of the GENCOL software package commercialized by Kronos Inc. This package relies on version 9.0.1 of the CPLEX solver for solving the restricted master problems. Based on preliminary test results, we opted for the primal simplex algorithm for the 500-task instances, and the barrier algorithm for the 1000- and 1500-task instances. To obtain different computational times and solutions of varying qualities, we ran tests using different values for the parameters $Z_{\min}$ and I. The first parameter varied from 0 to 500000, while the second parameter was set to either 2 or 5. Finally, the parameter $\theta_{\min}$ was set to 0.7 for our experiments.

6 Large neighborhood search

Introduced by Shaw (1998), large neighborhood search (LNS) is a metaheuristic that starts with an initial solution and destroys, at each iteration, a part of the current solution before reoptimizing it to obtain hopefully an improved overall solution. For the MDVSP, we propose to destroy at each

Algorithm 3 Large neighborhood search algorithm

```
Assign a weight of 1 to each selection strategy
Build an initial solution s
Set s* ← s and z* ← z(s)
while no stopping criterion is satisfied do
    Choose a schedule selection strategy according to their weights
    Apply this strategy to select r schedules to reoptimize
    Reoptimize these r schedules using the column generation heuristic
    Update the current solution s
    if z(s) < z* then
        Set s* ← s and z* ← z(s)
    end if
    Update the weight of the chosen selection strategy
end while
```

iteration r schedules from the current solution and reoptimize the MDVSP restricted to the tasks contained in these schedules using the column generation heuristic described in Sect. 5. When r is not too large, the column generation approach typically computes a very good quality solution for the restricted MDVSP in a very reasonable time. Hence, this approach can generate feasible solutions regularly throughout the solution process. To diversify the search, three different strategies for selecting the schedules to reoptimize are used. The strategy applied at a given iteration is chosen randomly according to weights that are dynamically adjusted throughout the solution process.

The proposed algorithm is described in Algorithm 3, where $z(s)$ denotes the value of solution s and s^* the current best solution. In the following paragraphs, we present the procedure building the initial solution, the schedule selection strategies, and the procedure choosing the schedule selection strategy.

6.1 Initial solution

To build an initial solution, a variant of the two-phase heuristic approach introduced by Bodin et al. (1983) was used. In the first phase, the MDVSP is transformed into an SDVSP by replacing all depots $k \in K$ by a single fictitious depot. The SDVSP is defined over a single network $G = (V, A)$. Node set V contains a start of schedule node o, an end of schedule node d, and a node for each task in T. Arc set A contains all connection arcs of any set A^k, $k \in K$, (these arcs exist for all depots). It also contains a pull-out arc (o, j) for each task node $j \in T$ corresponding to the cheapest pull-out arc $(o(k), j)$, $k \in K$, and a pull-in arc (i, d) for each task node $i \in T$ corresponding to the cheapest pull-in arc $(i, d(k))$, $k \in K$. Hence, a path from o to d in G represents a schedule that can start and end at different depots. Furthermore, the SDVSP definition does not take into account the number of available vehicles per depot. The SDVSP is thus a relaxation of the MDVSP which guarantees finding the minimum number of vehicles to cover all the tasks in T when a sufficiently large fixed cost is comprised in the cost of the pull-out arcs. The solution of the SDVSP provides a set of vehicle schedules unassigned to the depots. The second phase thus consists of assigning these schedules to the depots using their real costs while respecting vehicle availability per depot. This problem is a transportation problem that can easily be solved.

6.2 Schedule selection strategies

At each iteration of the large neighborhood search metaheuristic, r schedules are selected using one of the following three strategies.

Random schedules: The r schedules are chosen at random.

Less frequent schedules: This strategy selects the r schedules that have been the least frequently chosen to be reoptimized (ties are broken randomly). Hence, for each feasible schedule that appeared in a solution, the number of times that it was selected for reoptimization is kept in memory.

Closest schedules: A first schedule p_1 is selected among those that have not been chosen as the first schedule in the last J iterations ($J = 20$ for our tests). Then, $r - 1$ other schedules are selected, namely, those that are the "closest" to p_1 in time and in space according to the following measure: $\min_{i \in T_{p_1}, j \in T_p} \{\omega c_{ij} + t_{ij}, \omega c_{ji} + t_{ji}\}$, where $p \neq p_1$ is a schedule in the current solution, T_p is the set of tasks contained in schedule p, $t_{ij} = \infty$ if task j cannot be followed by task i, and ω is a weighting factor ($\omega = 10$ for our tests).

As in Ropke and Pisinger (2004), the schedule selection strategy to use at a given iteration is randomly chosen. A strategy i has probability $\frac{w_i}{\sum_j w_j}$ of being chosen where w_i is a positive weight assigned to strategy i. Such a weight w_i is updated to the value $\rho w_i + (1 - \rho)(z(s_{\text{prev}}) - z(s))$ at every iteration that strategy i is selected. In this expression, ρ is a constant in the interval $[0, 1[$ ($\rho = 0.5$ for our tests), $z(s_{\text{prev}})$ is the value of the solution before, and $z(s)$ the value of the solution after reoptimization. Hence, these weights are convex combinations of the gains (losses) yielded by the corresponding strategy. They assign a high probability of being selected to the most efficient strategies.

6.3 Implementation details

In our experiments, the number r of schedules to reoptimize at each iteration was set to 30 for the 500- and 1000-task instances and to 40 for the 1500-task instances. Given the small size of the reoptimization problems, the column generation approach used for these reoptimizations relied on the primal simplex algorithm to solve the RMPs and the parameter $Z_{\min}$ was set to 0 (column generation was not halted

prematurely). All these parameter values and those mentioned above were determined after performing a series of preliminary tests.

7 Tabu search

Let S be a set containing all feasible solutions to a combinatorial optimization problem and possibly some infeasible solutions (for instance, those made up of vehicle schedules that do not respect all task start times). Let f be a function to be minimized over the set of feasible solutions in S. For a solution $s \in S$, let $N(s)$ denote the neighborhood of s which is defined as the set of solutions in S obtained from s by performing a local change, called move. Local search techniques visit a sequence $s_0, \ldots, s_t$ of solutions, where s_0 is an initial solution and $s_{i+1} \in N(s_i)$ $(i = 1, \ldots, t-1)$. Tabu search is one of the most famous local search techniques. It was introduced by Glover (1986), and follows the general scheme of Algorithm 4, where TL is a list of forbidden moves. A more detailed description of the method and its concepts can be found in Glover and Laguna (1997).

For the MDVSP, we use an adaptation of the state-of-the-art tabu search algorithm developed by Cordeau et al. (2001) for vehicle routing problems with time windows, that are akin to the MDVSP. We define a solution as a set of vehicle schedules which satisfy all constraints, except that tasks are possibly accomplished too late. Hence, each vehicle starts and ends at the same depot, the number of available vehicles at each depot is never exceeded, and each task is accomplished exactly once by a vehicle. No task is performed too early, which means that the vehicle performing task $i \in T$ waits if it arrives at the start location before time a_i. Tasks can, however, be accomplished too late, and this constraint violation is penalized in f. Function f also penalizes solutions which are visited too often. More precisely, let $z(s)$ be the usual total cost of the vehicle schedules, $w(s)$ the total delay in s, ρ_{ik} the number of solutions visited by the tabu search in which task i is accomplished by vehicle k, and α_{ik}^s a variable taking value 1 if task i is accomplished by vehicle k in solution s, and 0 otherwise. The cost $f(s')$ of a neighbor solution $s' \in N(s)$ is defined as $z(s') + \gamma w(s') + p(s')$, where:

- γ is a parameter which gives more or less importance to the penalty due to the delays (Taillard 1993). Parameter γ is initially set equal to 1 and is then adjusted every iteration: if the current solution s is feasible (i.e., $w(s') = 0$) then γ is divided by $1 + \mu$, else γ is multiplied by $1 + \mu$, where μ is a random number in the open interval $(0, 1)$.
- $p(s') = \sigma z(s') \sum_{(i,k)} \alpha_{ik}^{s'} \rho_{ik}$ is a penalty factor that helps to diversify the search (Gendreau et al. 1994). Parameter σ is equal to 0 if $z(s') + \gamma w(s') \leq z(s) + \gamma w(s)$ or s' is feasible and $z(s') < z(s^*)$; otherwise σ is chosen randomly in $[0, \sqrt{|T||K|}]$.

Notice that if s' is a feasible solution and $z(s') < z(s^*)$ then $f(s') = z(s')$, since $w(s')$ and $p(s')$ are both equal to 0.

We use the same initial solution as for the LNS algorithm. A neighbor solution $s' \in N(s)$ is obtained from s by using two kinds of moves:

1-move: In a 1-move, a task i is moved from a vehicle k to a vehicle $k' \neq k$. The position of i in k' is chosen so that the cost of the new vehicle schedule is minimized. When performing such a move, the pair (i, k) is introduced in the tabu list TL, with the meaning that it is forbidden for several iterations to move i back into k.

Swap-move: Let i and i' be two tasks accomplished by two different vehicles k and k', respectively. A swap-move consists in moving i from k to k' and i' from k' to k. The positions of i in k' and i' in k are chosen so that the cost of the new vehicle schedules are minimized. When performing such a move, the pairs (i, k) and (i', k') are introduced in the tabu list TL, with the meaning that it is forbidden for several iterations to move i back into k and i' back into k'.

The duration of the tabu status of a move is chosen randomly in $[0, \sqrt{t|T|}]$, at each iteration, where t is the total number of vehicles used in the current solution.

As explained above, neighbor solutions are obtained by modifying the set of tasks for two vehicles. If there is a change in the first or the last task of a vehicle, then a check is made to verify if the costs for this vehicle can be reduced by assigning the modified schedule to a different depot having at least one available vehicle.

Algorithm 4 Tabu search algorithm

```
1: Generate an initial feasible solution s
2: Set TL ← ∅ and s* ← s
3: while no stopping criterion is satisfied do
4:    Determine a solution s' ∈ N(s) with minimum value
      f(s') such that either s' is obtained from s by per-
      forming a move m ∉ TL or s' is feasible and f(s') <
      f(s*)
5:    if s' is feasible and f(s) < f(s*) then
6:       Set s* ← s'
7:    end if
8:    Set s ← s' and update TL
9: end while
```

8 Computational results

In this section, we describe the results of the experiments that we conducted for comparing the five heuristics described above. For those tests, we used random MDVSP instances generated as in Carpaneto et al. (1989) (class A instances). In these instances representing daily MDVSPs, the

task start and end locations are selected from a restricted set of random points in a square so that each point is chosen approximately five times on average as a task start or end location. The depots are randomly located in this square. The task start times are chosen randomly so as to simulate morning and evening peak hours. Approximately 40% of the tasks have short durations (less than 125 minutes), while the rest have longer durations (between 180 and 360 minutes). The number of available vehicles in each depot is also generated randomly in such a way that vehicles from several depots are necessary to cover all tasks. However, the overall number of vehicles is far from being constraining. In these instances, the objective consists of minimizing first the number of vehicles used and second the total operational costs. In fact, a large fixed cost (10000) is incurred for each vehicle used to put a very high priority on the first objective. In an optimal solution that uses a minimal number of vehicles, a vehicle performs 4 to 5 tasks on average. To our knowledge, the largest of these instances to be solved to optimality, involves 800 tasks and 6 depots (see Hadjar et al. 2006).

For our tests, we used MDVSP instances where $|T| \in \{500, 1000, 1500\}$ and $|K| \in \{4, 8\}$. All reported results correspond to averages over five instances of the same size that were generated with different random seeds. All tests were run on a Linux i686 PC machine equipped with an Intel Xeon processor clocked at 2.66 GHz and 1 Gb of memory. All tested instances and the best individual results that we obtained for these instances are available at http://people.few.eur.nl/huisman/instances.htm.

Recall that three of the five implemented solution methods (Lagrangian heuristic, LNS, and tabu search) produce integer solutions throughout the solution process, while the other two (truncated branch-and-cut and truncated column generation) stop when the first feasible solution is obtained. For the truncated branch-and-cut approach, we have no control on the computational time required for obtaining this first solution. For the column generation approach, this computational time can be adjusted by modifying the values of the parameters I and $Z_{\min}$, giving us the possibility of producing different solutions in different times.

The computational results for the 4-depot instances are reported in Figs. 2, 3, 4. These figures present the results for the 500-, 1000-, and 1500-task instances, respectively, using curves and points in a two-dimensional space (objective value versus computational time). There is a curve for each of the three methods that regularly produce integer solutions. Such a curve represents the average function $(\sum_{i=1}^{5} z_i(t))/5$, where 5 is the number of random instances and $z_i(t)$ is the value of the best solution found before time t for instance i by the corresponding method. Each point on the curves shows when an improved solution was found for one of the 5 instances. For the column generation heuristic, the figures exhibit several points, each of them indicating the average result for one setting of the parameters I and $Z_{\min}$. The rightmost point corresponds to the case when column generation is not halted prematurely ($Z_{\min} = 0$), providing in general the best quality solution. Finally, in Fig. 2, a single point is shown to illustrate the average result of the branch-and-cut heuristic. Such a point is not shown in the other two figures because the corresponding average computational time is much higher than that required by the column generation method.

Let us mention that, in our experiments, all methods succeeded to compute solutions involving the minimum number of vehicles. Indeed, the procedure used to compute an initial solution for the tabu search and the LNS method ensures a minimum number of vehicles in the initial solution. Given the large value of the fixed vehicle cost, only solutions with the same number of vehicles can be considered as best solutions in these algorithms. For the other three methods, the use of powerful mathematical programming tools allowed to reach this minimum number of vehicles. Consequently, as proposed in Carpaneto et al. (1989) and Dell'Amico et al. (1993), we pursue our analysis of the results without taking into account the large fixed vehicle cost (10000).

For each heuristic, Table 1 indicates the average computational time and the average best solution value (obtained in a limited amount of time for the Lagrangian, LNS, and tabu search heuristics). These time limits were set at 85, 700, and 2300 seconds for the 500-, 1000-, and 1500-task instances, respectively, in order to slightly exceed the times required by the column generation heuristic. The best solution values (without vehicle fixed costs) are scaled for each problem size so that the minimum value obtained by the heuristics is equal to 1.

From these results, one can note that the behavior of the different methods are quite similar for all instance sizes, except for the branch-and-cut approach which requires long computational times (compared to the times needed by the column generation method) for the 1000- and 1500-task instances. One can also make the following observations. First, the branch-and-cut heuristic and the column generation heuristic with $Z_{\min} = 0$ provide the best solutions when enough computational time is available. It seems, however, difficult to substantially reduce the computational time needed by the latter approach without significantly deteriorating solution quality. Nevertheless, average solution times remain reasonable (less than 35 minutes) with this approach compared to the solution times required by the branch-and-cut approach. Second, the LNS method is the best method when the available computational time is restricted. In fact, it succeeds to rapidly improve the initial solution and continues afterwards to produce small improvements at a slower pace until reaching a plateau. Third, the Lagrangian heuristic consistently improves the quality of the solutions found until reaching good quality solutions. Finally, the tabu

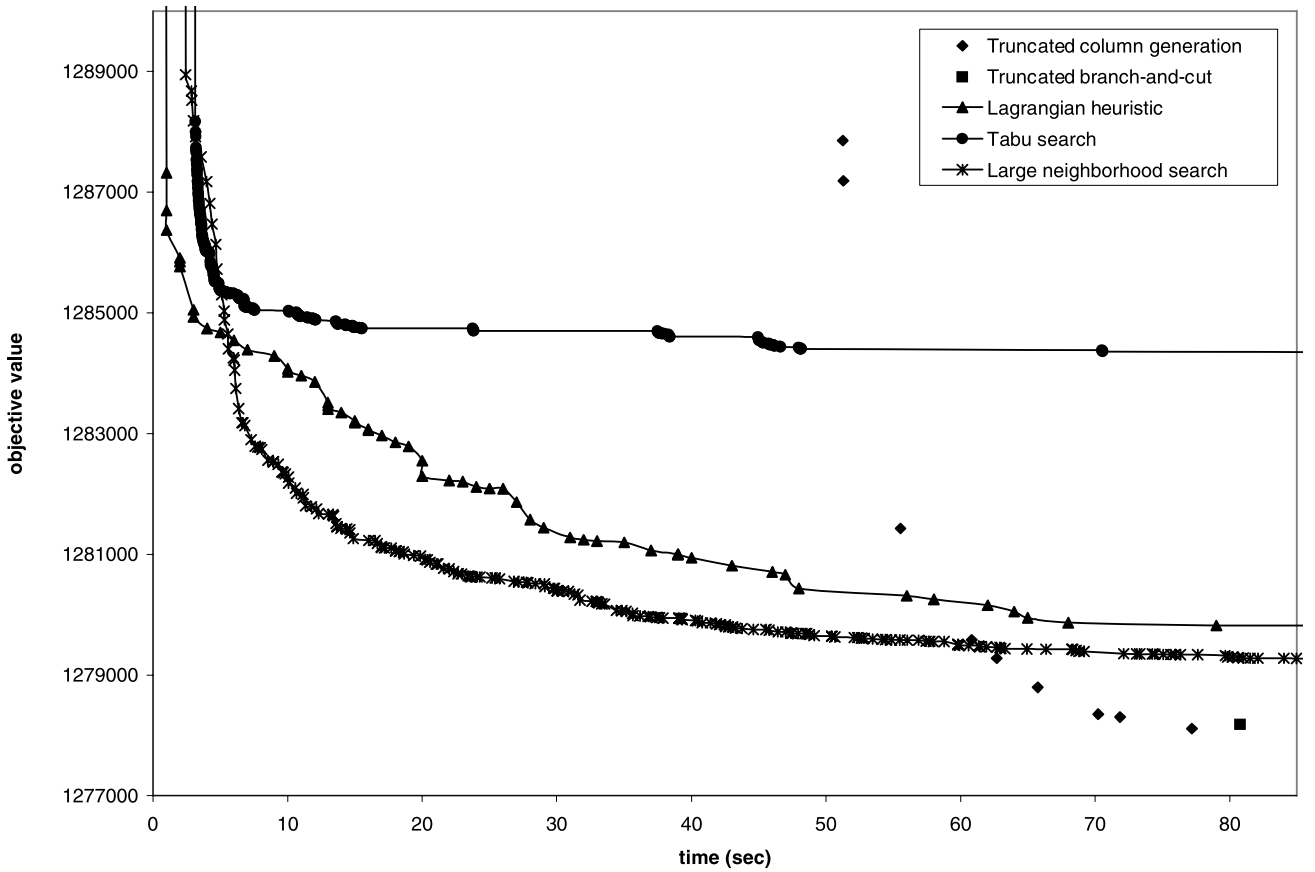

Fig. 2 Results for the 4-depot, 500-task instances

search rapidly improves the initial solution at the very beginning of the solution process before struggling to yield improvements afterwards. The quality of the solutions obtained with this heuristic is far from the quality reached by the other methods. This poor performance can be explained by the fact that most solutions selected throughout the algorithm are infeasible: for instance, on average 89% of them are infeasible for the 1500-task, 4-depot instances. Indeed, since task starting times are fixed in the MDVSP, the tabu search algorithm has much more difficulty to find feasible solutions for the MDVSP, than for the vehicle routing problems with time windows considered by Cordeau et al. (2001).

Because the results for the 8-depot MDVSP instances are very similar to those obtained for the 4-depot instances, we only report the average solution times and the average best solution values (without the vehicle fixed cost) in Table 2. No results are given for the branch-and-cut heuristic applied to the 1500-task instances because this heuristic could not find a feasible solution within ten hours of computational time for three of the five test instances. This lack of solution for the branch-and-cut heuristic combined with high average computational times indicate that this heuristic is relatively unstable.

Next, Table 3 presents upper bounds on the average optimality gap (in percentage and without vehicle fixed costs) obtained by each heuristic for all instance sizes (except again by the branch-and-cut heuristic for the 1500-task, 8-depot instances). The formula used is: $Bound = 100\frac{\tilde{z}_{IP}-\tilde{z}_{LP}}{\tilde{z}_{LP}}$, where $\tilde{z}_{IP}$ and $\tilde{z}_{LP}$ are the value of the best integer solution (as reported in Tables 1 and 2) and the linear relaxation optimal value (computed by the column generation heuristic with $Z_{\min} = 0$). For each size, the smallest upper bound is highlighted in bold. The very small bounds for the column generation heuristic clearly highlight the effectiveness of this heuristic to produce very high quality solutions. Based on the results presented in Figs. 2, 3, 4, we can also say that the LNS approach is effective at generating good quality solutions in relatively small computational times.

To conclude this section, we would like to mention that we also tried to solve the instances to optimality using a non-truncated version of the branch-and-cut method. We were able to solve all 4-depot, 500-task instances in an average of 1270 seconds. The average optimal value for these instances (scaled as in Table 1) is 0.9997. Hence, the solutions computed by the column generation heuristic are very

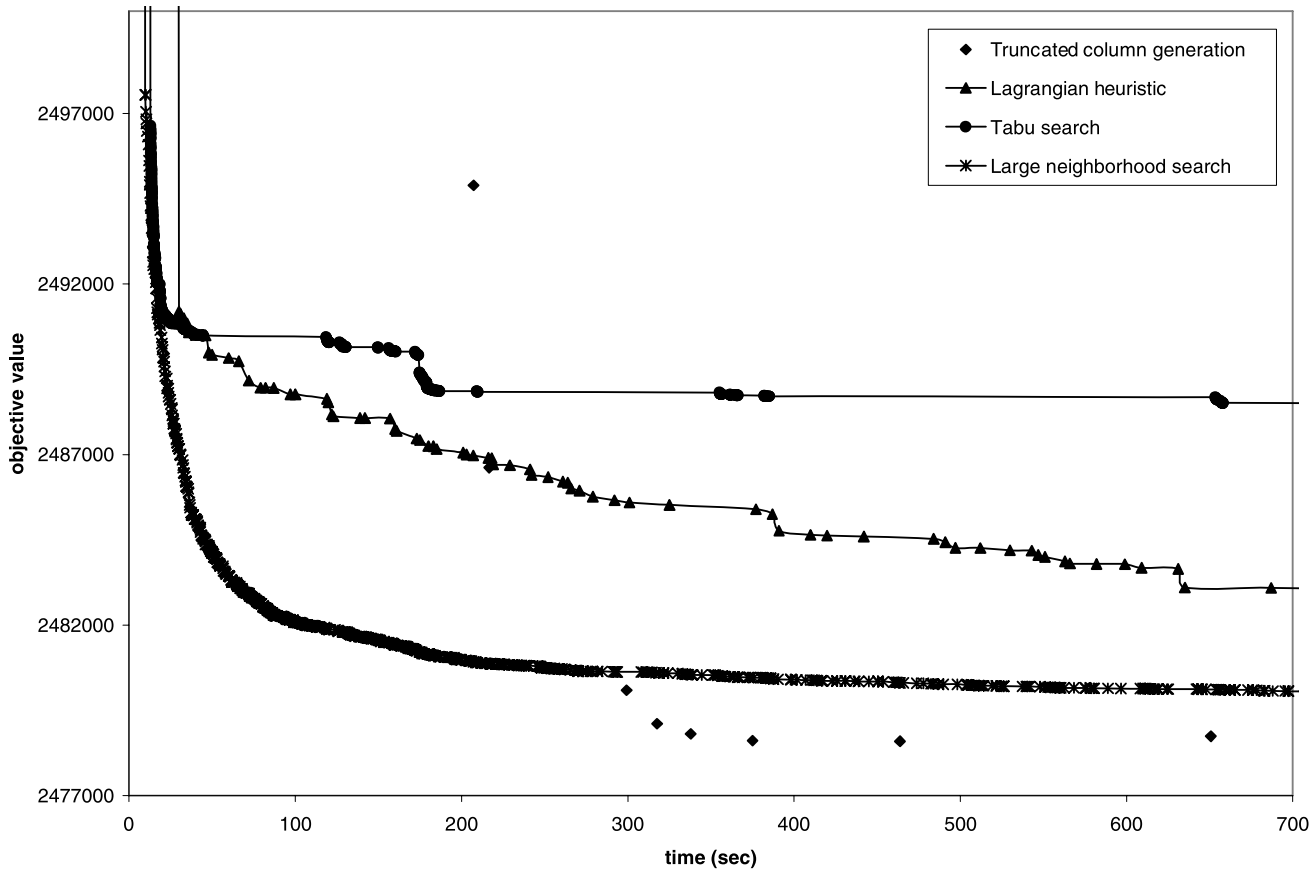

Fig. 3 Results for the 4-depot, 1000-task instances

Table 1 Average best (scaled) solution values for the 4-depot instances

Heuristic	500 tasks		1000 tasks		1500 tasks	
	Time (s)	Scaled sol val	Time (s)	Scaled sol val	Time (s)	Scaled sol val
Branch-and-cut	81	1.0013	1287	1.0000	4149	1.0000
Lagrangian heuristic	85	1.0294	700	1.0370	2300	1.0531
Column generation	77	1.0000	651	1.0016	2203	1.0019
LNS	85	1.0201	700	1.0124	2300	1.0182
Tabu search	85	1.1073	700	1.0812	2300	1.1054

Table 2 Average best (scaled) solution values for the 8-depot instances

Heuristic	500 tasks		1000 tasks		1500 tasks	
	Time (s)	Scaled sol val	Time (s)	Scaled sol val	Time (s)	Scaled sol val
Branch-and-cut	612	1.0014	6207	1.0000	–	–
Lagrangian heuristic	125	1.0494	900	1.0597	3200	1.0923
Column generation	119	1.0000	857	1.0091	3085	1.0000
LNS	125	1.0262	900	1.0265	3200	1.0283
Tabu search	125	1.1722	900	1.1896	3200	1.1981

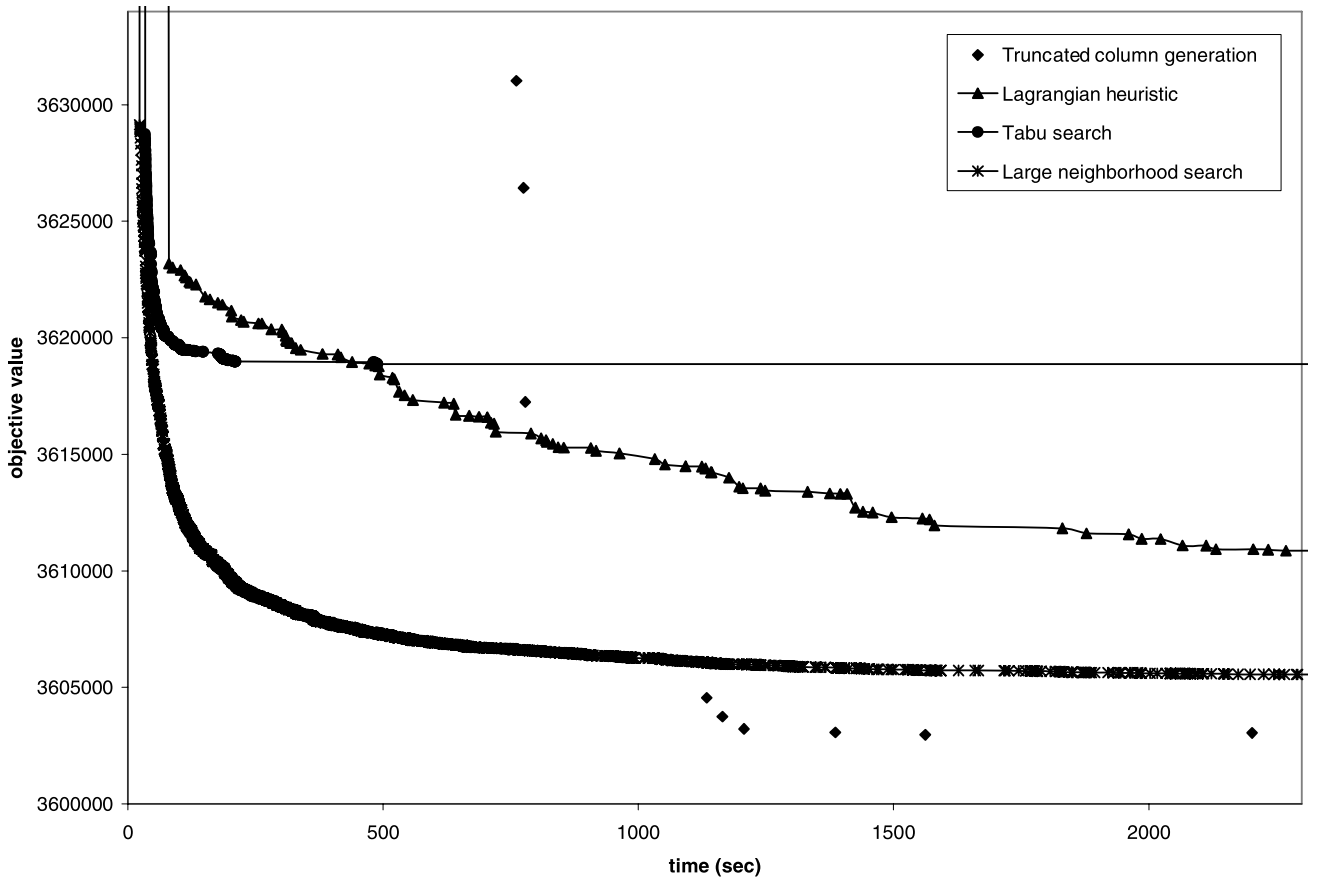

Fig. 4 Results for the 4-depot, 1500-task instances

Table 3 Upper bounds on the average optimality gap in percentage

Heuristic	4 depots			8 depots		
	500 tasks	1000 tasks	1500 tasks	500 tasks	1000 tasks	1500 tasks
Branch-and-cut	0.298	**0.171**	**0.217**	0.684	**0.581**	–
Lagrangian heuristic	3.117	3.880	5.540	5.511	6.590	10.148
Column generation	**0.170**	0.329	0.405	**0.547**	0.672	**0.837**
LNS	2.184	1.414	2.050	3.179	3.245	3.690
Tabu search	10.919	8.309	10.779	17.865	19.647	20.816

close to optimality with an average optimality gap (without fixed vehicle cost) of 0.03%. Within ten hours of computational time, the exact branch-and-cut method was able to solve three of the five 8-depot, 500-task instances (on average, in 19453 seconds), one of the five 4-depot, 1000-task instance (in 25171 seconds), and none of the other tested 1000- and 1500-task instances.

9 Conclusions

In this paper, we have presented a comparison of five different heuristic approaches for solving the MDVSP, including heuristics based on mathematical programming techniques and metaheuristics. This comparison showed that, for the tested instances, the column generation heuristic produces the best quality solutions when sufficient computational time is available and stability is required. To obtain faster solution times without deteriorating too much solution quality, our results indicate that the LNS method, which relies on the column generation heuristic for neighborhood evaluation, is the best alternative. Hence, this paper showed that embedding mathematical programming tools in a metaheuristic framework can offer a good compromise between computational time and solution quality for the MDVSP. This

conclusion might obviously be different for problems where mathematical programming models exhibit large integrability gaps. Nevertheless, this work opens up interesting perspectives for future research.

Acknowledgements The authors would like to thank Tsjitske Groen for her help in the implementation of the Lagrangian heuristic, and Guillaume Dereu for developing preliminary versions of the column generation, large neighborhood search, and tabu search heuristics.

References

Ahuja, R., Magnanti, T., & Orlin, J. (1993). *Network flows: theory, algorithms, and applications*. Englewood Cliffs: Prentice Hall.

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W. P., & Vance, P. H. (1998). Branch-and-price: column generation for solving huge integer programs. *Operations Research*, *46*, 316–329.

Bertossi, A. A., Carraresi, P., & Gallo, G. (1987). On some matching problems arising in vehicle scheduling models. *Networks*, *17*, 271–281.

Bodin, L., Golden, B., Assad, A., & Ball, M. (1983). Routing and scheduling of vehicles and crews: the state of the art. *Computers and Operations Research*, *10*, 63–211.

Carpaneto, G., Dell'Amico, M., Fischetti, M., & Toth, P. (1989). A branch and bound algorithm for the multiple vehicle scheduling problem. *Networks*, *19*, 531–548.

Cordeau, J.-F., Laporte, G., & Mercier, A. (2001). A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational Research Society*, *52*, 928–936.

Dantzig, G. B., & Wolfe, P. (1960). Decomposition principle for linear programs. *Operations Research*, *8*, 101–111.

Dell'Amico, M., Fischetti, M., & Toth, P. (1993). Heuristic algorithms for the multiple depot vehicle scheduling problem. *Management Science*, *39*, 115–125.

Desaulniers, G., & Hickman, M. (2007). Public transit. In G. Laporte & C. Barnhart (Eds.), *Handbooks in operations research and management science: Vol. 14*. *Transportation* (pp. 69–127). Amsterdam: Elsevier Science.

Desaulniers, G., Desrosiers, J., Ioachim, I., Solomon, M. M., & Soumis, F. (1998a). A unified framework for deterministic time constrained vehicle routing and crew scheduling problems. In T. G. Crainic & G. Laporte (Eds.), *Fleet management and logistics* (pp. 57–93). Norwell: Kluwer.

Desaulniers, G., Lavigne, J., & Soumis, F. (1998b). Multi-depot vehicle scheduling problems with time windows and waiting costs. *European Journal of Operational Research*, *111*, 479–494.

Desrosiers, J., Dumas, Y., Solomon, M. M., & Soumis, F. (1995). Time constrained routing and scheduling. In M. O. Ball, T. L. Magnanti, C. L. Monma, & G. L. Nemhauser (Eds.), *Handbooks in operations research and management science: Vol. 8*. *Network routing* (pp. 35–139). Amsterdam: Elsevier Science.

Forbes, M. A., Holt, J. N., & Watts, A. M. (1994). An exact algorithm for multiple depot bus scheduling. *European Journal of Operational Research*, *72*, 115–124.

Freling, R., Wagelmans, A. P. M., & Paixão, J. M. P. (2001). Models and algorithms for single-depot vehicle scheduling. *Transportation Science*, *35*, 165–180.

Gendreau, M., Hertz, A., & Laporte, G. (1994). A tabu search heuristic for the vehicle routing problem. *Management Science*, *40*, 1276–1290.

Geoffrion, A. (1974). Lagrangian relaxations for integer programming. *Mathematical Programming Study*, *2*, 82–114.

Gilmore, P. C., & Gomory, R. E. (1961). A linear programming approach to the cutting-stock problem. *Operations Research*, *9*, 849–859.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, *13*, 533–549.

Glover, F., & Laguna, M. (1997). *Tabu search*. Boston: Kluwer Academic.

Hadjar, A., Marcotte, O., & Soumis, F. (2006). A branch-and-cut algorithm for the multiple depot vehicle scheduling problem. *Operations Research*, *54*, 130–149.

Huisman, D., Freling, R., & Wagelmans, A. P. M. (2005). Multiple-depot integrated vehicle and crew scheduling. *Transportation Science*, *39*, 491–502.

Kliewer, N., Mellouli, T., & Suhl, L. (2006). A time–space network based exact optimization model for multi-depot bus scheduling. *European Journal of Operational Research*, *175*, 1616–1627.

Kokott, A., & Löbel, A. (1996). *Lagrangean relaxations and subgradient methods for multiple-depot vehicle scheduling problems* (ZIB-Report 96-22). Konrad-Zuse-Zentrum für Informationstecnik, Berlin.

Lamatsch, A. (1992). An approach to vehicle scheduling with depot capacity constraints. In M. Desrochers & J.-M. Rousseau (Eds.), *Lecture notes in economics and mathematical systems: Vol. 386*. *Computer-aided transit scheduling* (pp. 181–195). Berlin: Springer.

Löbel, A. (1997). *Optimal vehicle scheduling in public transit*. PhD thesis, Technische Universität Berlin, Germany.

Löbel, A. (1998). Vehicle scheduling in public transit and Lagrangian pricing. *Management Science*, *44*, 1637–1649.

Odoni, A. R., Rousseau, J.-M., & Wilson, N. H. M. (1994). Models in urban and air transportation. In S. M. Pollock, M. H. Rothkopf, & A. Barnett (Eds.), *Handbooks in operations research and management science: Vol. 6*. *Operations research and the public sector* (pp. 107–150). Amsterdam: North-Holland.

Mesquita, M., & Paixão, J. (1992). Multiple depot vehicle scheduling problem: a new heuristic based on quasi-assignment algorithms. In M. Desrochers & J.-M. Rousseau (Eds.), *Lecture notes in economics and mathematical systems: Vol. 386*. *Computer-aided transit scheduling* (pp. 167–180). Berlin: Springer.

Ribeiro, C., & Soumis, F. (1994). A column generation approach to the multiple depot vehicle scheduling problem. *Operations Research*, *42*, 41–52.

Ropke, S., & Pisinger, D. (2004). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, *40*, 455–472.

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In M. Maher & J.-F. Puget (Eds.), *Lecture notes in computer science*. *Principles and practice of constraint programming, CP98* (pp. 417–431). New-York: Springer.

Taillard, E. D. (1993). Parallel iterative search methods for vehicle routing problems. *Networks*, *23*, 661–673.