# Comparison of heuristic approaches for the multiple depot vehicle scheduling problem

**Ann-Sophie Pepin,[¶] Guy Desaulniers,[†] Alain Hertz,[†]**

**Dennis Huisman[‡]**

[¶]*Giro Inc.*
*Montreal, Canada*

annsophie.pepin@giro.ca

[†]*École Polytechnique and GERAD*
*Department of mathematics and industrial engineering*
*Montreal, Canada*

{Guy.Desaulniers, Alain.Hertz}@gerad.ca

[‡] *Econometric Institute and ECOPT,*
*Erasmus University Rotterdam,*
*Rotterdam, the Netherlands*

huisman@few.eur.nl

1st November 2006

## Abstract

Given a set of timetabled tasks, the multi-depot vehicle scheduling problem is a well-known problem that 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 heuristic approaches for this problem, namely, a heuristic MIP solver, a Lagrangian heuristic, a column generation heuristic, a large neighborhood search heuristic using column generation for neighborhood evaluation, and a tabu search heuristic. The first three methods are adaptations of existing methods, while the last two are novel approaches for 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 a compromise between computational time and solution quality.

**Keywords**: Vehicle scheduling, multiple depot, heuristics, column generation, Lagrangian heuristic, tabu search, large neighborhood search.

## 1 Introduction

The class of vehicle routing/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/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 the CPLEX MIP solver applied heuristically, a Lagrangian heuristic, a column generation heuristic, a large neighborhood search heuristic using 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 novel approaches for the MDVSP.

The MDVSP is a well-known problem that has several applications in different fields such as public transit and the trucking industry. It consists of determining least-cost schedules for vehicles housed in several depots such that each timetabled task of a given set is accomplished exactly once by a vehicle. The literature on this problem is abundant (see the surveys of Bodin *et al.*, 1983, Odoni *et al.*, 1994, Desrosiers *et al.*, 1995, and Desaulniers and Hickman, 2007). From the 1970s to the early 1990s, several heuristic solution approaches have been proposed, for instance, by Bodin *et al.* (1978), Bodin *et al.* (1983), Bertossi *et al.* (1987), Mesquita and Paixão (1992), Lamatsch (1992), and Dell'Amico *et al.* (1993). Then, exact solution approaches have been developed, among others, by Carpaneto *et al.* (1989), Ribeiro and Soumis (1994), Löbel (1997), Löbel (1998), Hadjar *et al.* (2006), and Kliewer *et al.* (2006). In particular, Löbel (1997), Löbel (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).

The contributions of this paper are as follows. We develop two novel heuristic approaches for the MDVSP, including the first metaheuristics for this problem, and adapt three other approaches. For these 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.

This paper is organized as follows. In Section 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 Section 8. Finally, we draw some conclusions in Section 9.

## 2 MDVSP definition and formulations

The MDVSP can be formally 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 $\delta_i$ 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.

Next, we present two mathematical formulations for the MDVSP : an integer multi-commodity formulation which is used by the heuristic CPLEX MIP solver and the Lagrangian heuristic, and a set partitioning type formulation which is at the basis of the column generation heuristic.

## 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 $X_{ij}^k$, $(i,j) \in A^k$, $k \in K$. Such a variable indicates the flow of commodity $k$ on the arc $(i,j)$. Using this notation, the MDVSP can be modeled as:

$$\textit{Minimize} \qquad \sum_{k\in K} \sum_{(i,j)\in A^k} c_{ij} X_{ij}^k \tag{1}$$

$$\text{subject to:} \quad \sum_{k\in K} \sum_{j\,:\,(i,j)\in A^k} X_{ij}^k = 1, \qquad \forall\, i \in T, \tag{2}$$

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

$$\sum_{j\,:\,(j,i)\in A^k} X_{ji}^k - \sum_{j\,:\,(i,j)\in A^k} X_{ij}^k = 0, \qquad \forall\, i \in V^k \setminus \{o(k), d(k)\},\ k \in K, \tag{4}$$

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

The objective function (1) seeks at minimizing 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 $\mathcal{G}^k = (\mathcal{V}^k, \mathcal{A}^k)$ can be 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, $\mathcal{V}^k = \{o(k), d(k)\} \cup T \cup E$. 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 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 $a_i + \delta_i + t_{ij'} \leq a_{j'}$. With such a time-space network structure, model (1)–(5) remains valid for the MDVSP after replacing $A^k$ by $\mathcal{A}^k$ and constraints (2) and (5) by

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

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

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

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.*, 1998). Let $\Omega^k$ 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 $\theta_p$ that takes value 1 if $p$ is retained in the solution and 0 otherwise.

The MDVSP can then be modeled as:

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

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

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

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

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 Heuristic MIP approach

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 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, we set the CPLEX MIP emphasis parameter 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

For the MDVSP, Bertossi *et al.* (1987) and Kokott and Löbel (1996) proposed heuristic solution approaches based on Lagrangian relaxation (see Geoffrion, 1974) in which the task covering constraints are relaxed. Alternatively, Mesquita and Paixão (1992), Lamatsch (1992), and Kokott and Löbel (1996) developed Lagrangian relaxation approaches where the flow conservation constraints are relaxed. In this section, we present a heuristic Lagrangian relaxation approach similar to those of Lamatsch (1992) and Kokott and Löbel (1996) for computing lower bounds and that can be used to derive feasible solutions throughout the solution process. This approach relies on the connection networks and model (1)–(5) augmented by the redundant constraints:

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

To obtain a lower bound on the optimal value, the availability constraints (3) are omitted and the flow conservation constraints (4) are relaxed in a Lagrangian way using Lagrangian multipliers $\lambda_i^k$, $i \in T$, $k \in K$. The Lagrangian subproblem is then formulated as follows:

$$\phi(\boldsymbol{\lambda}) \quad = \quad \text{Minimize} \quad \sum_{k \in K} \sum_{(i,j) \in A^k} (c_{ij} + \lambda_j^k - \lambda_i^k) X_{ij}^k \tag{14}$$

$$\text{subject to:} \quad \sum_{k \in K} \sum_{j \,:\, (i,j) \in A^k} X_{ij}^k = 1, \qquad \forall\, i \in T, \tag{15}$$

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

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

The Lagrangian subproblem 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 $X_{ij}^k$ by a single variable corresponding to the variable $X_{ij}^k$ with the lowest reduced cost, i.e., with the smallest value $\lambda_j^k - \lambda_i^k$ over all valid $k$. The other variables are set to 0. To solve the SDVSP, we use the auction algorithm as described in Freling *et al.* (2001).

In this way, for each set of Lagrangian multipliers $\boldsymbol{\lambda}$, we obtain a lower bound. Moreover, we can use the values of the variables $X_{ij}^k$ to construct a feasible solution for the overall problem. Indeed, after assigning to each arc $(i, j)$ the depot $k$ for which $X_{ij}^k = 1$ in the Lagrangian subproblem solution, 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 for the MDVSP can 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 $c_p^k$ (see Section 2.2). The problem of finding the best such solution corresponds to a transportation problem, which can be solved in polynomial time using the Hungarian method (see Ahuja *et al.*, 1993).

Since the Lagrangian dual function $\phi(\boldsymbol{\lambda})$ yields a lower bound for each vector $\boldsymbol{\lambda}$, the best lower bound can be obtained by solving the Lagrangian dual problem: $\max_{\boldsymbol{\lambda}} \Phi(\boldsymbol{\lambda})$. We use subgradient optimization to obtain a good lower bound. Furthermore, an upper bound is computed at each iteration of the subgradient algorithm. 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, we construct a feasible solution in the way described above. This solution provides an upper bound on the objective value. The best upper bound $UB$ is then updated every time we find an improvement. 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 $\alpha$. This parameter is updated in step 4 in order to ensure the convergence of the subgradient algorithm, that is, $\alpha$ is halved after a certain number of iterations ($\gamma$) without improvement in the lower bound. Finally, the procedure terminates when the best lower bound is found, i.e., the subgradient vector is the 0-vector and/or the upper bound is equal to the lower bound, $\alpha$ 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: $n_{\max} = 10000$, $\alpha^0 = 1.0$, $\gamma = 10$, and $\epsilon = 0.000001$.

**Algorithm 1** Lagrangian heuristic

```
Step 0: Initialization
  Choose parameters $n_{max}$, $\alpha^0$, $\gamma$, $\epsilon$
  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 $UB = \phi(\boldsymbol{\lambda}^n)$, $\sum_{i\in T}\sum k \in K(Y_i^{k,n})^2 \leq \epsilon$, $\alpha^n \leq \epsilon$ or $n \geq n_{max}$ then STOP
  else return to Step 1
```

## 5 Heuristic 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.*, 1998). 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.* (1998) for the MDVSP with time windows and waiting costs.

## 5.1 Column generation

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 and one subproblem per depot. At iteration $n$, the restricted master problem is simply the master problem restricted to a subset of the variables $\theta_p$, 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 restricted master problem 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 $p \in \bigcup\limits_{k \in K} \left(\Omega^k \setminus \Omega_n^k\right)$) 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 $\pi_n^i$, $i \in T$, and $\beta_n^k$, $k \in K$, the values of the dual variables associated with constraints (10) and (11) of the restricted master problem 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 $\theta_p$. Hence, by solving this subproblem at iteration $n$, we can identify the schedule $p \in \Omega^k$ with the smallest reduced cost. The current restricted master problem solution is thus optimal for the overall master problem if the optimal value of each subproblem is non-negative.

An exact column generation method proceeds as follows. Starting the first iteration with an initial set of columns and/or artificial variables in the restricted master problem, the method solves at each iteration the current restricted master problem using a linear programming algorithm and then each shortest path subproblem using a label-setting algorithm (see Ahuja *et al.*, 1993). If the optimal value of each subproblem is non-negative, the solution process stops and the optimal primal solution of the current restricted master problem is declared also optimal for the master problem. Otherwise, columns associated with the negative reduced cost paths identified by the subproblems are added to the restricted master problem and another iteration is performed.

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 restricted master problem 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 restricted master problem has not decreased by more than $Z_{min}$ in the last $I$ iterations, where $Z_{min}$ and $I$ are predefined parameters. To obtain different computational times and solutions of varying qualities, we ran tests using different values for these parameters. The first parameter varied from 0 to 500000, while the second parameter was set to either 2 or 5.

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.

## 5.2 Rounding procedure

The following iterative rounding procedure is applied to compute rapidly an integer solution for the MDVSP. At each iteration, the column generation method described in the preceding section is applied for solving the master problem, modified according to the decisions made in the previous iterations. Then, all variables $\theta_p$ taking a fractional value greater than or equal to a predetermined threshold value (0.7 for our experiments) in the computed master problem solution are rounded up to 1. If no such variables exist but there are fractional-valued variables, then the variable with the highest fractional value is rounded up to 1. Otherwise, the procedure ends with an integer solution. Note that this procedure might fail to generate an integer solution when vehicle availability is tight. In this case, the master problem may become infeasible after rounding up several variables. However, this situation did not occur in our experiments.

Note also that, in order to reduce the size of the master problem to solve at each iteration, instead of rounding up the variables in the master problem, we rather remove from the master problem these variables and the covering constraints (10) of the tasks covered in the schedules associated with them. Also, we remove all other variables covering these tasks, as well as the nodes representing these tasks in the networks $G^k$, $k \in K$, and their incident arcs. Finally, the schedules associated with the rounded up variables together with their assigned depots are kept in memory and the right-hand side members of the availability constraints (11) are updated accordingly.

## 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 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 Section 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 2, 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, we use a variant of the two-phase heuristic approach introduced by Bodin *et al.* (1983). In the first phase, the MDVSP is transformed into an SDVSP by

**Algorithm 2** Large neighborhood search algorithm

```
1: Assign a weight of 1 to each selection strategy
2: Build an initial solution $s$
3: Set $s^* \leftarrow s$ and $z^* \leftarrow z(s)$
4: while no stopping criterion is satisfied do
5:     Choose a schedule selection strategy according to their weights
6:     Apply this strategy to select $r$ schedules to reoptimize
7:     Reoptimize these $r$ schedules using the column generation heuristic
8:     Update the current solution $s$
9:     if $z(s) < z^*$ then
10:        Set $s^* \leftarrow s$ and $z^* \leftarrow z(s)$
11:    Update the weight of the chosen selection strategy
```

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 a 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 we solve heuristically using a greedy procedure. In this procedure, each schedule is assigned to the depot yielding the least cost for this schedule and for which there is still at least one vehicle available.

## 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, we keep in memory the number of times that it was selected for reoptimization.

**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, we select $r - 1$ other schedules 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 $\omega$ 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 $\frac{w_i}{\sum_i w_i}$ of being chosen where $w_i$ is a positive weight assigned to strategy $i$. Such a weight $w_i$ is updated to the value $\rho w_i + (1-\rho)(z(s_{prev}) - z(s))$ at every iteration that strategy $i$ is selected. In this expression, $\rho$ is a constant in the interval $[0, 1[$ ($\rho = 0.5$ for our tests), $z(s_{prev})$ is the value of the solution before, and $z(s)$ the value of the solution after reoptimization. Hence, these weights are weighted averages 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 restricted master problem and the parameter $Z_{min}$ was set to 0 (column generation was not halted prematurely).

**Algorithm 3** Tabu search algorithm

```
1: Generate an initial feasible solution $s$
2: Set $TL \leftarrow \emptyset$ and $s^* \leftarrow s$
3: while no stopping criterion is satisfied do
4:     Determine a solution $s' \in N(s)$ with minimum value $f(s')$ such that either $s'$ is obtained from $s$ by performing a move $m \notin TL$ or $s'$ is feasible and $f(s) < f(s^*)$
5:     if $s'$ is feasible and $f(s) < f(s^*)$ then
6:         Set $s^* \leftarrow s'$
7:     Set $s \leftarrow s'$ and update $TL$
```

## 7 Tabu search

Let $S$ be a set containing all feasible, and possibly also non-feasible solutions to a combinatorial optimisation problem. 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 in 1986, and follows the general scheme of Algorithm 3, 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 tabu search algorithm developed by Cordeau *et al.* (2001) for vehicle routing problems with time windows. 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, we never exceed the number of available vehicles at each depot, 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, we denote $z(s)$ the usual total cost of the vehicle schedules, $w(s)$ the total delay in $s$, $\rho_{ik}$ the number of solutions visited by the tabu search in which task $i$ is accomplished by vehicle $k$, and $\alpha^s_{ik}$ 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:

- $\gamma$ is a parameter which gives more or less importance to the penalty due to the delays. Parameter $\gamma$ 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 $\gamma$ is divided by $1 + \mu$, else $\gamma$ is multiplied by $1 + \mu$, where $\mu$ 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. Parameter $\sigma$ 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 $\sigma$ is chosen randomly in $[0, \sqrt{\mid T \mid\mid K \mid}\,]$.

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 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 taks $i$ 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 $[0, \sqrt{t \mid T \mid}\,]$, at each iteration, where $t$ is the total number of vehicles used in the current solution.

As explained above, neigbhor 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 we check if we can reduce the costs for this vehicle by assigning the modified schedule to a different depot having at least one available vehicle.

## 8 Computational results

In this section, we describe the results of the experiments that we conducted for comparing the five heuristic solution methods described above. For those tests, we used random MDVSP instances generated as in Carpaneto *et al.* (1989) (class A instances). 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. To our knowledge, the largest of these instances reported 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 5 instances of the same size that were generated with different random seeds. All tests were run on an Intel Xeon 2.66 GHz workstation with 1 Gb of memory.

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 (MIP and column generation) stop when the first feasible solution is obtained. For the MIP 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 Figures 1 to 3 and in Table 1. 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 $t$

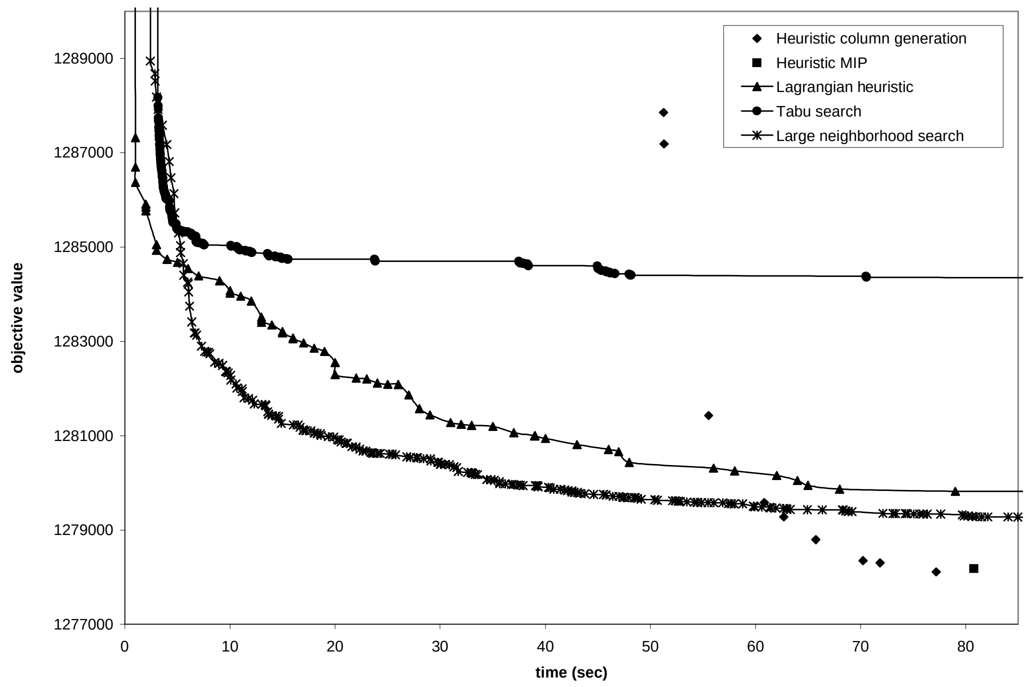

Figure 1: Results for the 4-depot, 500-task instances

for instance $i$ by the corresponding method. Each point along such a curve 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 Figure 1, a single point is shown to illustrate the average result of the MIP 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.

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 are highlighted in bold.

From these results, one can notice that the behavior of the different methods are quite

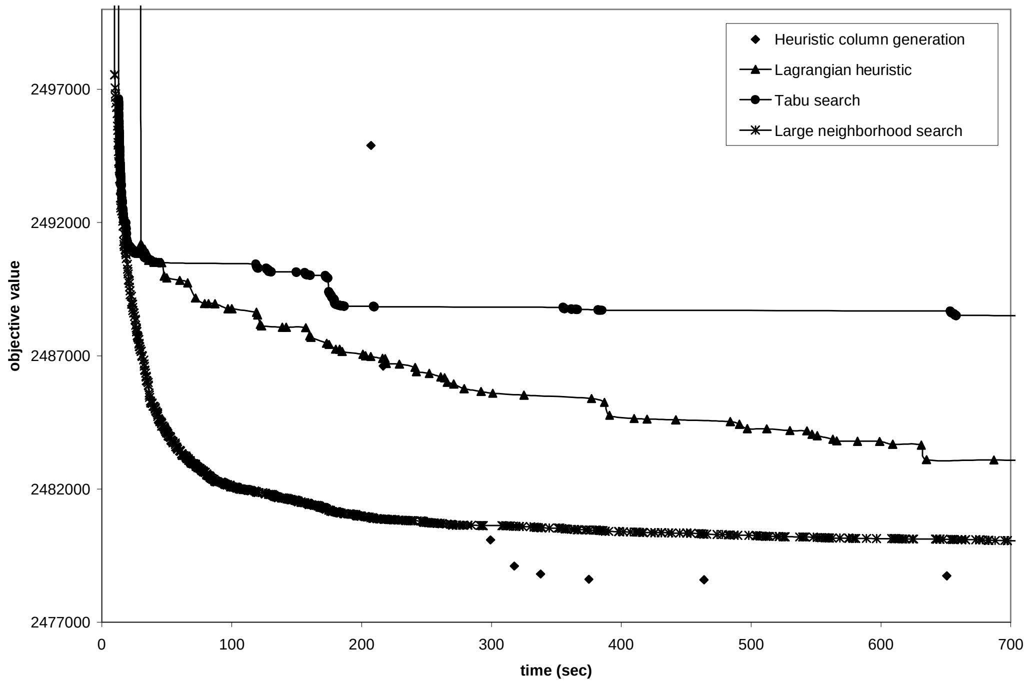

Figure 2: Results for the 4-depot, 1000-task instances

similar for all instance sizes, except for the MIP 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 MIP 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 MIP 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 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.

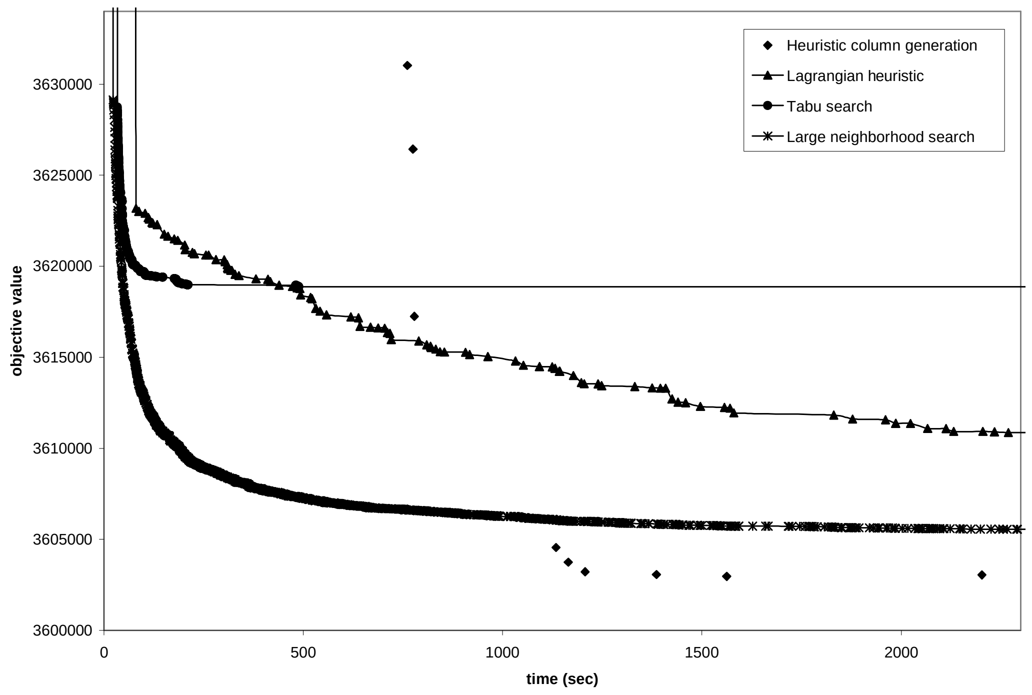

Figure 3: Results for the 4-depot, 1500-task instances

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 in Table 2. No results are given for the MIP heuristic applied to the 1500-task instances because this heuristic could not find a feasible solution within ten hours of computational time for one of the five test instances. This lack of solution for the MIP heuristic combined with high average computational times indicate that this heuristic is relatively unstable.

Finally, let us mention that all methods compute solutions involving a minimum number of vehicles. Indeed, the procedure used to compute an initial solution for the tabu search and LNS methods ensures a minimum number of vehicles in the initial solution. For the other three methods, the use of powerful mathematical programming tools allows to reach this minimum.

To conclude this section, we present in Table 3 the average optimality gaps (in percentage) obtained by each heuristic for all instance sizes (except again by the MIP heuristic for the 1500-task, 8-depot instances). As proposed in Carpaneto *et al.* (1989) and Dell'Amico *et al.* (1993), these gaps have been computed without taking into account the large fixed vehicle

| Heuristic | 500 tasks |  | 1000 tasks |  | 1500 tasks |  |
|---|---|---|---|---|---|---|
|  | time (s) | Sol val | time (s) | Sol val | time (s) | Sol val |
| MIP | 81 | 1278181.6 | 1287 | **2478545.6** | 4149 | **3602758.0** |
| Lagrangian heuristic | 85 | 1279816.9 | 700 | 2483082.4 | 2300 | 3610871.4 |
| Column generation | 77 | **1278107.6** | 651 | 2478739.0 | 2203 | 3603044.0 |
| LNS | 85 | 1279275.7 | 700 | 2480065.3 | 2300 | 3605551.3 |
| Tabu search | 85 | 1284342.8 | 700 | 2488500.3 | 2300 | 3618857.8 |

Table 1: Average best solution values for the 4-depot instances

| Heuristic | 500 tasks |  | 1000 tasks |  | 1500 tasks |  |
|---|---|---|---|---|---|---|
|  | time (s) | Sol val | time (s) | Sol val | time (s) | Sol val |
| MIP | 612 | 1285640.2 | 6206.63 | **2495918.4** | - | - |
| Lagrangian heuristic | 125 | 1287924.1 | 900 | 2501648.9 | 3200 | 3637341.8 |
| Column generation | 119 | **1285575.4** | 857 | 2496005.6 | 3085 | **3625731.8** |
| LNS | 125 | 1286820.9 | 900 | 2498458.9 | 3200 | 3629288.8 |
| Tabu search | 125 | 1293769.6 | 900 | 2514100.8 | 3200 | 3650643.0 |

Table 2: Average best solution values for the 8-depot instances

cost (10000). The formula used is: $Gap = \frac{z_{IP}^{oc} - z_{LP}^{oc}}{z_{LP}^{oc}}$, where $z_{IP}^{oc}$ and $z_{LP}^{oc}$ 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 $Z_{min} = 0$). The very small gaps for the column generation heuristic clearly highlight the effectiveness of this heuristic to produce very high quality solutions. Based on the results presented in Figures 1 to 3, we can also say that the LNS approach is effective at generating good quality solutions in relatively small computational times.

## 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

| | 4 depots | | | 8 depots | | |
|---|---|---|---|---|---|---|
| Heuristic | 500 tasks | 1000 tasks | 1500 tasks | 500 tasks | 1000 tasks | 1500 tasks |
| MIP | 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 |

Table 3: Average optimality gaps in percentage

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, embedding mathematical programming tools in a metaheuristic framework seems to guarantee success when looking for a compromise between computational time and solution quality.

**Acknowledgments** The authors would like to thank Tsjitske Groen for implementing the Hungarian method in the Lagrangian heuristic, and Guillaume Dereu for developing preliminary versions of the column generation, large neighborhood search, and tabu search heuristics.

## References

Ahuja, R.K., T.L. Magnanti, and J.B. Orlin (1993). *Network Flows: Theory, Algorithms, and Applications.* Prentice-Hall, Englewood Cliffs, New-Jersey.

Barnhart, C., E.L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance (1998). Branch-and-Price: Column Generation for Solving Huge Integer Programs. *Operations Research* 46, 316–329.

Bertossi, A.A., P. Carraresi, and G. Gallo (1987). On Some Matching Problems Arising in Vehicle Scheduling Models. *Networks* 17, 271–281.

Bodin, L., D. Rosenfield, and A. Kydes (1978). UCOST: A Micro Approach to a Transit Planning Problem. *Journal of Urban Analysis* 5, 47–69.

Bodin, L., B. Golden, A. Assad, and M. Ball (1983). Routing and Scheduling of Vehicles and Crews: The State of the Art. *Computers and Operations Research* 10, 63-211.

Carpaneto, G., M. Dell'Amico, M. Fischetti, and P. Toth (1989). A Branch and Bound Algorithm for the Multiple Vehicle Scheduling Problem. *Networks* 19, 531–548.

Cordeau, J.-F., G. Laporte, and A. Mercier (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. and P. Wolfe (1960). Decomposition Principle for Linear Programs. *Operations Research* 8, 101–111.

Dell'Amico, M., M. Fischetti, and P. Toth (1993). Heuristic Algorithms for the Multiple Depot Vehicle Scheduling Problem. *Management Science* 39, 115–125.

Desaulniers, G., J. Desrosiers, I. Ioachim, M.M. Solomon, and F. Soumis (1998). A Unified Framework for Deterministic Time Constrained Vehicle Routing and Crew Scheduling Problems. In: T.G. Crainic and G. Laporte (eds.), *Fleet Management and Logistics*, Kluwer, Norwell, MA, 57–93.

Desaulniers, G., and M. Hickman (2007). Public Transit. In: G. Laporte and C. Barnhart (eds), *Transportation*, Handbooks in Operations Research and Management Science, Elsevier Science, Amsterdam. (To appear.)

Desaulniers, G., J. Lavigne, and F. Soumis (1998). Multi-Depot Vehicle Scheduling Problems with Time Windows and Waiting Costs. *European Journal of Operational Research* 111, 479–494.

Desrosiers, J., Y. Dumas, M. M. Solomon, and F. Soumis (1995). Time Constrained Routing and Scheduling. In: M. O. Ball, T.L. Magnanti, C.L. Monma, and G.L. Nemhauser (eds.), *Network Routing*, Handbooks in Operations Research and Management Science 8, Elsevier Science, Amsterdam, 35–139.

Forbes, M.A., J.N. Holt, and A.M. Watts (1994). An Exact Algorithm for Multiple Depot Bus Scheduling. *European Journal of Operational Research* 72, 115–124.

Freling, R., A.P.M. Wagelmans, and J.M.P. Paixão (2001). Models and Algorithms for Single-Depot Vehicle Scheduling. *Transportation Science* 35, 165–180.

Geoffrion, A. (1974). Lagrangian Relaxations for Integer Programming. *Mathematical Programming Study* 2, 82–114.

Gilmore, P.C. and R.E. Gomory (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. and M. Laguna (1997). *Tabu Search.* Kluwer Academic Publishers, Boston.

Hadjar, A., O. Marcotte, and F. Soumis (2006). A Branch-and-Cut Algorithm for the Multiple Depot Vehicle Scheduling Problem. *Operations Research* 54, 130–149.

Kliewer N., T. Mellouli, and L. Suhl (2006). A Time-Space Network Based Exact Optimization Model for Multi-Depot Bus Scheduling. *European Journal of Operational Research.* (To appear.)

Kokott, A. and A. Löbel (1996). Lagrangean Relaxations and Subgradient Methods for Multiplie-Depot Vehicle Scheduling Problems. ZIB-Report 96-22, Konrad-Zuse-Zentrum für Informationstchnik, Berlin, Germany.

Lamatsch, A. (1992). An Approach to Vehicle Scheduling with Depot Capacity Constraints. In: M. Desrochers and J.-M. Rousseau (eds.), *Computer-Aided Transit Scheduling*, Lecture Notes in Economics and Mathematical Systems 386, Springer-Verlag, Berlin, 181–195.

Löbel, A. (1997). Optimal Vehicle Scheduling in Public Transit. Ph.D. thesis, Technische Universität Berlin, Berlin, Germany.

Löbel, A. (1998). Vehicle Scheduling in Public Transit and Lagrangian Pricing. *Management Science* 44, 1637–1649.

Odoni, A.R., J.-M. Rousseau, and N.H.M. Wilson (1994). Models in Urban and Air Transportation. In: S.M. Pollock, M.H. Rothkopf, and A. Barnett (eds.), *Operations Research and the Public Sector*, Handbooks in Operations Research and Management Science 6, North-

Holland, Amsterdam, 107–150.

Mesquita, M. and J. Paixão (1992). Multiple Depot Vehicle Scheduling Problem: A New Heuristic Based on Quasi-Assignment Algorithms. In: M. Desrochers and J.-M. Rousseau (eds.), *Computer-Aided Transit Scheduling*, Lecture Notes in Economics and Mathematical Systems 386, Springer-Verlag, Berlin, 167–180.

Ribeiro, C. and F. Soumis (1994). A Column Generation Approach to the Multiple Depot Vehicle Scheduling Problem. *Operations Research* 42, 41–52.

Ropke, S. and D. Pisinger (2004). An Adaptative Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. Technical report, Copenhagen University, Denmark. To appear in *Transportation Science*.

Shaw, P. (1998). Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In: M. Maher, J.-F. Puget (eds.), *Principles and Practice of Constraint Programming - CP98*, Lecture Notes in Computer Science, Springer-Verlag, New-York, 417–431.