Optim Lett (2016) 10:1449–1461
DOI 10.1007/s11590-015-0944-x

ORIGINAL PAPER

Simple and efficient heuristic approach for the multiple-depot vehicle scheduling problem

Pablo Cristini Guedes[1] · **William Prigol Lopes**[1] · **Leonardo Rosa Rohde**[3] · **Denis Borenstein**[1,2]

Received: 9 April 2015 / Accepted: 2 September 2015 / Published online: 11 September 2015

Abstract In this paper, a fast heuristic approach is proposed for solving the multiple depot vehicle scheduling problem (MDVSP), a well-known NP-hard problem. The heuristic is based on a two stage procedure. The first one applies two state space reduction procedures towards reducing the problem complexity. One procedure is based on the solutions of the single-depot vehicle scheduling for each depot, while the other uses the solution of a relaxed formulation of the MDVSP, in which a vehicle can finish its task sequence in a different depot from where it started. Next, the reduced problem is solved by employing a truncated column generation approach. The heuristic approach has been implemented in several variants, through different combinations of the reduction procedures, and tested on a series of benchmark problems provided by Pepin et al. (J Sched 12:17–30, 2009). The heuristic variants found solutions with very narrow gaps (below 0.7 %, on average) to best-known solutions (Pepin et al., J Sched 12:17–30, 2009), decreasing the required CPU time by an overall average factor of 17 in comparison with reported results in the literature (Otsuki and Aihara, J Heuristics 1–19, 2014).

✉ Denis Borenstein
denis.borenstein@gmail.com; denisb@ea.ufrgs.br

Pablo Cristini Guedes
pcguedes@ea.ufrgs.br

William Prigol Lopes
wplopes@gmail.com

Leonardo Rosa Rohde
lrohde@gmail.com

1 Management School, Universidade Federal do Rio Grande do Sul, Porto Alegre, RS, Brazil

2 Universidad de Cuenca, Cuenca, Azuay, Ecuador

3 Industrial Engineering Department, Federal University of Pelotas, Pelotas, RS, Brazil

Keywords Heuristics · Vehicle scheduling · Multi-depots · Column generation

1 Introduction

The multi-depot vehicle scheduling problem (MDVSP) is a classical problem in operations research, arising in applications such as public transport systems, and appearing as a subproblem in the more complex crew scheduling and management disruption problems [6]. The MDVSP has been shown by Bertossi et al. [1] to be a NP-hard problem.

Initially, the problem was solved using heuristic methods, given the complexity of the problem and the computational power of the 70s and 80s [4]. Carpaneto et al. [2] developed the first optimization method, employing an "additive lower bounding" scheme. Bertossi et al. [1] employed Lagrangean relaxation, based on a multicommodity formulation. Column generation (CG) was also used to solve the MDVSP [5,13,16]. Löbel [10] combined Lagrangian relaxation and CG to solve large instances based on data from three public transportation companies in Germany. Kliewer et al. [8] introduced a new vehicle scheduling network called time–space network to solve the MDVSP. The use of this network led to a reduction in the associated mathematical models, allowing the solution of large instances by direct application of commercial integer programming solvers. Metaheuristics were also employed to solve the MDVSP. Pepin et al. [15] developed two metaheuristic algorithms based on large neighborhood search (LNS) and tabu search, while Laurent and Hao [9] presented an iterated local search (ILS) heuristic. More recently, Otsuki and Aihara [12] developed a variable depth search framework which utilizes pruning and deepening techniques to speedup the CPU time required to find a good solution.

Concerning the quality and efficiency of the solutions for the problem, Pepin et al. [15] analyzed the performance of five different approaches to solve the MDVSP, namely: truncated branch-and-cut, Lagrangian relaxation, column generation, LNS, and tabu search. The comparison showed that column generation is the best method when powerful computational resources are available, while LNS is the best option in the presence of limited resources (this result was recently ratified by Otsuki and Aihara [12]). However, the most important conclusion of Pepin et al.'s [15] paper is the difficulty of these heuristics to find good solution in a reasonable time for large instances. Given a time limit of one hour, the methods could only find solution for instances up to 1500 tasks and eight depots. Löbel [10] and Kliewer et al. [8] succeeded to solve real-world public transit instances to optimality involving up to 20,000 and 7000 tasks, respectively. These instances, however, have a particular structure that ease their solution process [15], being previously validated in practice.

This paper describes a simple and fast heuristic procedure to solve efficiently very large instances of the MDVSP. The heuristics consist of two sequential steps. The first step is based on a state space selection process intended to reduce the number of variables in the problem. The second step employs a CG approach to solve the reduced problem to find good solutions very quickly. The performance of variants of the proposed heuristics were assessed on the set of testbed instances by Pepin et al. [15]. All variants found very good solutions, with average gaps from the best-known solutions below 0.7 %, requiring, on average, 18 times less CPU usage. The major contributions in this paper are as follows: (i) the introduction of effective state space reduction techniques to decrease the problem complexity; and (ii) the development of a heuristic approach, based on a truncated CG approach, capable of solving very efficiently and effectively the MDVSP.

The documen is organized as follows. In the next section, we present the problem and the formulations related to our solution method. Section 3 describes the development and the features of the heuristic procedure. The computational experiments and comparison with previous results are presented in Sect. 4. Finally, Sect. 5 presents some final remarks.

2 Problem

Before defining a formal mathematical formulation for the MDVSP, we introduce some useful notation. We define the movement of empty vehicles as a deadheading trip. We are given a set of N service trips, each trip $i \in N$ starting at time st_i and ending at time et_i, along with a set of K depots, in which v_k vehicles are stationed. Let t_{ij} be the deadheading transportation time between the ending point of trip i and the starting point of trip j. An ordered pair of trips (i, j) is said to be compatible iff satisfies the relation $et_i + t_{ij} \leq st_j$. The MDVSP objective is to find the minimal cost fleet schedule for executing all the service trips. A vehicle schedule is defined as a feasible sequence of service trips to perform, using the same depot as the starting and ending point.

There are several existing mathematical formulations for this problem [15]. It is beyond the scope of this article to discuss them in detail. We briefly present the set partitioning formulation, as introduced by Ribeiro and Soumis [16]. Define the MDVSP network $G^k = \langle V^k, A^k \rangle$ for each depot $k \in K$, where V^k is the set of nodes, and A^k denotes the set of arcs. Set $V^k = \{o(k), d(k)\} \cup N$ contains a node for each trip $i \in N$ and a pair of nodes, $o(k)$ and $d(k)$, representing the depot k as the initial and final nodes of the allocated schedule of vehicle v_k. Set $A^k = \{(o(k) \times N) \cup (N \times d(k)) \cup E\}$ is the set of deadheading trips, where $(N \times d(k))$ is the set of pull-in arcs to depot k, and $E = \{(i, j) | (i, j)$ is a compatible pair of trips, $i, j \in N\}$, $(o(k) \times N)$ is the set of pull-out arcs from depot k. Let c_{ij} be the cost of transversing arc $(i, j) \in A^k$, which is dependent on traveling and waiting costs. A path from $o(k)$ to $d(k)$ represents a feasible scheduling for a vehicle in depot k. The MDVSP network can be represented by its adjacency matrix $\mathbf{MDVSP}_{|K|+|N|\times|K|+|N|}$, with $MDVSP[i][j] = c_{ij}$ if $(i, j) \in A^k$ is a compatible pair of trips, and $MDVSP[i][j] = -1$ otherwise.

Define Ω^k as the set of all feasible set of paths in $G^k, k \in K$. Let $p \in \Omega^k$ be a feasible schedule with cost c_p. For each path $p \in \Omega^k$, $a_{ip} = 1$ iff node $i \in N$ is visited in path p, $a_{ip} = 0$ otherwise. Using binary variable θ_p, with $\theta_p = 1$ if schedule p is in the solution, $\theta_p = 0$ otherwise, the MDVSP can be formulated as a set partitioning type problem as follows:

$$\min \sum_{k \in K} \sum_{p \in \Omega^k} c_p \theta_p \tag{1}$$

st

$$\sum_{k \in K} \sum_{p \in \Omega^k} a_{ip}\theta_p = 1 \qquad \forall i \in N \tag{2}$$

$$\sum_{p \in \Omega^k} \theta_p \leq \upsilon_k \qquad \forall k \in K \tag{3}$$

$$\theta_p \in \{0, 1\} \qquad \forall p \in \Omega^k, \ k \in K \tag{4}$$

The objective function (1) seeks to minimize the total costs involved. Constraints (2) assure that each trip $i \in N$ is visited by only one schedule $p \in \Omega^k$, while constraints (3) ensure that the capacity of each depot is respected. Constraints (4) define the domain of the decision variable.

3 Method

The framework of the proposed heuristics can be described as follows:

Step 1: State space reduction of the problem by applying:
- Step 1.1: k-Single Depot VSP (SDVSP), $k = 1, \ldots, K$, based selection procedure (selection R1);
- Step 1.2: Relaxed-MDVSP selection procedure (selection R2);

Step 2: Solution of the reduced problem by employing a modified truncated CG procedure.

The method is basically a two-step approach. The first step is a state space reduction method intended to reduce the set of variables to a smaller, but relevant, subset of variables, decreasing the complexity of the instances. The second step solves the reduced state space problem using an improved truncated column generation approach towards accelerating the CG stabilization, including the use of an initialization procedure based on the solutions of the $|K|$-SDVSPs.

3.1 State space reduction

In VSPs, it is natural that some trips are geographically more distant from others. In addition to the geographical issue, there are also incompatibility issues. This means that trips that are close geographically can be far apart due to the time in which they must start, while others may be compatible in terms of timing, but geographically infeasible. Several variables in the problem become too costly to be considered as viable alternatives in an optimal or close to optimal solution, both due to distance or timing issues. As a consequence, although the complete space is very large, the set of "relevant" variables (with high chance of being in the final solution) is much smaller. In fact, very few variables from the complete space state will be in the final solution. Rooted on this observation, we developed two selection procedures, which main objective is to identify a smaller, but representative, solution space state. Pull-in and pull-out arcs cannot be reduced, since this led to the CG solution process to become unstable, not finding feasible solutions for some problem instances. Next, the selection procedures developed are discussed in details.

3.1.1 k-SDVSP based selection procedure (selection R1)

The goal of this procedure is to identify "relevant" variables that are selected by any solution of all $|K|$-SDVSPs. The reasoning behind this procedure is quite intuitive. If a variable is not chosen as a solution considering $|K|$-SDVSPs, then it would have small chance to be chosen as a candidate solution when considering the MDVSP. Based on this reasoning, this selection procedure consists in efficiently solving $|K|$ individual SDVSPs. The arcs found in any of the solved SDVSPs configures the reduced connection graph, represented by matrix $\mathbf{MDVSP}_r$, with a similar structure to matrix **MDVSP**, in which $MDVSP_r[i][j] = MDVSP[i][j]$ if arc (i, j) is in the solution of any k-SDVSP, $k \in K$; and $MDVSP_r[i][j] = -1$, otherwise. Matrix VSP_k is easily obtained from **MDVSP** by taking into consideration only the pull-in and pull-out arcs to and from depot k, respectively. This structure enables to solve each SDVSP as an assignment problem (AP) following Paixão and Branco [14]. There are several specially designed algorithms to solve linear AP. Based on results reported by Dell'Amico and Toth [3], the algorithm LAPJV developed by Jonker and Volgenant [7] was employed to solve each SDVSP. Algorithm 1 outlines this selection procedure. The LAPJV algorithm has a complexity of $O(n^2. \log n)$. As Algorithm 1 is executed $|K|$ times, the complexity of the selection procedure is $O(|K|n^2. \log n)$.

Algorithm 1 *k*-SDVSP Based Selection Procedure

```
$MDVSP_r[i][j] \leftarrow MDVSP[i][j], i = 1, \ldots, |K|, j = 1, \ldots, |K| + |N|$
$MDVSP_r[i][j] \leftarrow MDVSP[i][j], i = |K| + 1, \ldots, |K| + |N|, j = 1, \ldots, |K|$
for all $k \in K$ do
  Generate matrix $\mathbf{SDVSP}_k$ from matrix MDVSP
  $Solution \leftarrow$ LAPJV($\mathbf{SDVSP}_k$)
  if arc $(i, j) \in Solution$ then
    $MDVSP_r[i][j] \leftarrow MDVSP[i][j]$
  end if
end for
```

3.1.2 Relaxed-MDVSP selection procedure (selection R2)

This procedure uses the solution of a relaxed formulation of the MDVSP, relaxed-MDVSP, as the selection criterion of the variables to be included in the reduced state space. Basically, the relaxed-MDVSP is a relaxation of the multicommodity formulation of the MDVSP [10], in which vehicles are allowed to end its sequence of trips in different depots from where they started from. In order to present the formulation of the relaxed-MDVSP, it is necessary to define the associated underlying network. Let $G = \langle V, A \rangle$ be a digraph, where $V = 1, \ldots, n$ is the vertex set containing a node for each service trip $i \in N$, and A is the set of arcs representing the deadheading trips between compatible service trips. Let $G^* = \langle V^*, A^* \rangle$ be the graph with nodes in $V^* = V \cup K$, and arcs in $A^* = A \cup A_1 \cup A_2$, where $A_1 = \{(i, j)|i \in K, j \in V\}$

represents the set of pull-out arcs from depots, and $A_2 = \{(i, j) | i \in V, j \in K\}$ the set of pull-in arcs to depots. Introducing the decision variable x_{ij}, representing the flow in arc (i, j), the formulation for the relaxed-MDVSP is as follows:

$$\min \sum_{(i,j)\in A} c_{ij} x_{ij}$$

st

$$\sum_{j:(i,j)\in A^*} x_{ij} = 1 \qquad \forall i \in V \tag{5}$$

$$\sum_{j:(k,j)\in A_1} x_{kj} \leq v_k \qquad \forall k \in K \tag{6}$$

$$\sum_{i:(i,k)\in A_2} x_{kj} \leq v_k \qquad \forall k \in K \tag{7}$$

$$\sum_{j:(j,i)\in A^*} x_{ji} - \sum_{j:(i,j)\in A^*} x_{ij} = 0 \qquad \forall i \in V \tag{8}$$

$$x_{ij} \geq 0 \qquad \forall (i, j) \in A^* \tag{9}$$

The objective of this formulation is to minimize the total deadheading costs. Constraint (5) ensure that each task is executed exactly once by a vehicle. Constraints (6) and (7) limit the number of vehicles that can be used from each depot, while constraints (8) are flow conservation constraints which define a multiple-path structure for each depot. Constraint (9) defines the range of the decision variable. This problem can be seen as a minimum cost flow problem. Since all capacities and demands are integral in the relaxed-MDVSP, the solution of this problem is composed of integer variables [17].

The selection procedure based on the solution of the relaxed-MDVSP is very similar to Algorithm 1, in which the relaxed model is solved just once by a commercial linear programming solver. If arc (i, j) is in the solution of the relaxed-MDVSP then $MDVSP_r[i][j] = MDVSP[i][j]$, $MDVSP_r[i][j] = -1$ otherwise.

3.2 Modified truncated column generation

Column generation is a well-known method to solve MDVSP [13,16]. As the number of paths is huge, they are generated dynamically based on a Dantzig–Wolfe decomposition, in which the restricted master problem (RMP) is a linear relaxation of model (1)–(4) and the subproblems are the shortest path problems on the network G^k, $k \in K$. Dual variables π_i and β_k are associated with constrains (2) and (3), respectively. At iteration t, the modified costs of arc $(i, j) \in A^k$ is computed by $c_{ij} - \pi_i^t$, if $i \in N$, and $c_{ij} - \beta_i^t$, if $i = o(k)$. The RMP is solved for each iteration to a subset of variables θ_p. The dual variables of the RMP are used in the subproblems to both test the optimal solution (if the reduced costs of all variables are all non-negative) and to generate new columns (paths found in the subproblems which reduced cost is negative).

Our CG approach toward solving the reduced MDVSP problem is based on the truncated CG algorithm described in Pepin et al. [15]. The algorithm requires three predefined parameters, namely Z_{min}, I, and Ω_{min}. The algorithm terminates early if the optimal value of the RMP has not decreased by more than Z_{min} in the last I iterations. Parameter Ω_{min} is a threshold value in the rounding of variables θ_p toward an integer solution. However, we introduced some changes to this algorithm. The modified algorithm uses two different (but similar) formulations to solve the reduced MDVSP problem. One problem is the traditional RMP as presented by Ribeiro and Soumis [16]. The second formulation, called the relaxed restricted master problem (RRMP), replace constraints (2) by the following constraints:

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

keeping the same objective function and remaining constraints. These two problems have different dual solutions. The rationale behind this change is to restrict the dual variables in sign [11]. Let π_i and $\pi_i^{'}, \forall i \in N$ be the dual variables of constraints (2) and (10), respectively. In the standard CG procedure, the optimal solution of the RMP is obtained when the reduced costs of the corresponding path variables are non-negative. This cost is computed by the expression $c_{ij} - \pi_i$ at each iteration of the method. Since π_i can assume negative values given the equality in constraints (2), the reduced cost may be positive for several iterations. As a consequence, the convergence of the method can be rather slow, since the RMP objective function can decrease very slowly. As $\pi_i^{'}$ can only assume positive values, we expect that the reduced costs of paths p obtained in the subproblems, $c_{ij} - \pi_i^{'}$, will be often disturbed when compared with the standard CG. Better bounds are often found, since the dual solutions are more frequently changed. Since the solution of the RRMP does not respect all restrictions in the MDVSP, this model can be used until the problem becomes "stagnated", e.g., when the subproblems were not able to generate new columns, since all reduced costs are positive. At this stage, we change the RRMP to the standard RMP. The solution process continues until an integer solution of variables θ_p is found.

Another change introduced was in the way the columns are inserted in the master problem. The algorithm described by Pepin et al. [15] insert at most $|K|$ columns at each iteration of the CG procedure, depending on the sign of the reduced cost of each $k \in K$ subproblem. The master problem, with these new columns, is solved once at the beginning of the next iteration of the CG procedure. Our approach uses the traditional way of selecting columns to insert in the current master problem. However, it solves the master problem right after a new column is inserted. During experimentation, we noticed that the inclusion of several columns at each instance was generating many similar columns, unnecessarily solving several subproblems. This problem was observed in several CG applications [11]. Although this procedures solves initially a higher number of master problems, the use of updated duals helps to stabilize the CG, resulting in better convergence.

4 Computational results

The performance of the heuristic approach was evaluated in a set of instances available at the public site http://people.few.eur.nl/huisman/instances.htm, referred to as

Huisman's website. The instances comprise problems sizes with 4 and 8 depots and 500, 1000, and 1500 trips, totaling 30 test problems. The number of depots (m), the number of trips (n) and the instance id (s0, s1, s2, s3, and s4) are indicated in the name of the instance. We have set the fixed costs of the depots pull-in and pull-out arcs high enough (a total of 10,000) to allow the minimum cost flow problem determine the optimal number of vehicles required for the whole journey, following Pepin et al. [15]. The proposed heuristics were implemented in C++ and used CPLEX 12.5 to solve the linear programming models involved in the CG and in selection R2. All experiments were performed on a computer Intel Core I5-3210 processor running at 2.80 GHz under Linux kernel 3.12 (64 bits) with 8 GB of RAM.

CG requires an initial solution. Initially, we replicated the use of artificial variables penalized by a big-M cost as defined in Pepin et al. [15]. However, we faced a solution process that led to poor convergence and very high CPU times for all instances. CG required excessive time to find solutions without artificial variables in the basic solution of the RRMP. Although a good initial solution cannot guarantee a good convergence process [11], we decided to use the paths obtained by Algorithm 1 as an initial set of columns to the RRMP, regardless of applying selection R1. Its primal optimal solution is used to compute an initial solution for the MDVSP, offering an upper bound on the integer optimal value. Its dual solution provides a lower bound on the RRMP.

The following notation is used to indicate the implemented algorithms: (i) MTCG—the modified truncated column generation using the initialization routine, (ii) R1—state space reduction of the problem by applying only selection R1; (iii) R2—state space reduction of the problem by applying only selection R2; (iv) R1 + R2—state space reduction of the problem by applying selection R1 and R2. The last three algorithms are variants of the developed heuristic and use MTCG to solve the reduced space state in the second phase of the heuristic approach.

The settings of CG parameters were carried out during the experiments. The best quality results were obtained with small values of Z_{min} and large values of I and Ω_{min}. The best solutions were found with $Z_{min} = 0$. Concerning the number of minimum iterations, MTCG obtained good solutions with $I = 5$. As we reduced the state space of an instance to be solved by the CG procedure, it was necessary to increase the value of this parameter to $I = 30$, as a compensation factor. Regarding Ω_{min}, good solutions (in terms of quality and efficiency) were obtained with this value either set to 0.8 or 0.9.

Table 1 presents a detailed comparison of the developed algorithms for the 30 instances in Huisman's website, concerning solution quality. For each algorithm, Table 1 displays the optimal solution found and the solution gap (in %). The gaps were computed using $GAP = 100 \times \frac{(V-VB)}{VB}$, where V is the value computed by the evaluated method and VB is the solution value informed in Huisman's website. Average and maximum gaps (in %) in relation to Huisman's best available solutions (using CPLEX, truncated branch-and-cut, or TCG) were also reported in this table. As all developed algorithm have succeeded to find the same optimal number of vehicles as presented by Huisman's website for all instances, we decided to omit these values for economy's sake. We refer to this website for the obtained values.

MTCG obtained very similar results to the ones reported by Huisman's site, with gaps below 0.16 % for all instances. These gaps could be decreased for some instances, if a loss of efficiency is tolerated, since the solutions presented in Table 1 are good

Table 1 Solution quality comparison with Huisman's results

Instance	Huisman	MTCG		Selection R1		Selection R2		Selection R1 + R2	
	Solution	Solution	Gap	Solution	Gap	Solution	Gap	Solution	Gap
m4n500s0	1,289,114	1,289,280	0.01	1,297,940	0.68	1,300,900	0.91	1,296,600	0.58
m4n500s1	1,241,618	1,241,970	0.03	1,247,940	0.51	1,247,170	0.45	1,247,960	0.51
m4n500s2	1,283,811	1,284,360	0.04	1,292,650	0.69	1,298,200	1.12	1,291,547	0.60
m4n500s3	1,258,634	1,259,290	0.05	1,267,820	0.73	1,267,260	0.69	1,266,780	0.65
m4n500s4	1,317,077	1,317,310	0.02	1,323,870	0.52	1,325,750	0.66	1,322,490	0.41
m4n1000s0	2,516,247	2,516,580	0.01	2,539,920	0.94	2,539,430	0.92	2,534,440	0.72
m4n1000s1	2,413,393	2,414,020	0.03	2,436,200	0.95	2,443,230	1.24	2,427,680	0.59
m4n1000s2	2,452,905	2,454,390	0.06	2,467,880	0.61	2,463,880	0.45	2,461,690	0.36
m4n1000s3	2,490,812	2,491,950	0.05	2,506,060	0.61	2,507,770	0.68	2,503,240	0.50
m4n1000s4	2,519,191	2,523,100	0.16	2,525,980	0.27	2,524,680	0.22	2,524,420	0.21
m4n1500s0	3,830,912	3,832,930	0.05	3,855,030	0.63	3,861,470	0.80	3,853,490	0.59
m4n1500s1	3,559,176	3,560,920	0.05	3,571,280	0.34	3,570,820	0.33	3,570,820	0.33
m4n1500s2	3,649,757	3,650,790	0.03	3,676,580	0.73	3,669,560	0.54	3,662,380	0.35
m4n1500s3	3,406,815	3,408,510	0.05	3,436,530	0.87	3,428,770	0.64	3,435,040	0.83
m4n1500s4	3,567,122	3,567,740	0.02	3,591,240	0.68	3,588,810	0.61	3,589,170	0.62
m8n500s0	1,292,411	1,292,590	0.01	1,302,270	0.76	1,300,370	0.62	1,300,810	0.65
m8n500s1	1,276,919	1,277,300	0.03	1,285,350	0.66	1,284,500	0.59	1,284,080	0.56
m8n500s2	1,304,251	1,304,530	0.02	1,310,880	0.51	1,309,920	0.43	1,309,930	0.44
m8n500s3	1,277,838	1,278,030	0.02	1,286,140	0.65	1,284,550	0.53	1,284,590	0.53
m8n500s4	1,276,010	1,276,210	0.02	1,283,890	0.62	1,282,690	0.52	1,282,760	0.53
m8n1000s0	2,422,112	2,422,410	0.01	2,440,000	0.74	2,439,170	0.70	2,438,330	0.67
m8n1000s1	2,524,293	2,524,640	0.01	2,536,730	0.49	2,534,950	0.42	2,535,240	0.43
m8n1000s2	2,556,313	2,556,320	0.00	2,571,010	0.57	2,578,960	0.89	2,569,170	0.50
m8n1000s3	2,478,393	2,478,390	0.00	2,488,730	0.42	2,488,060	0.39	2,487,920	0.38
m8n1000s4	2,498,388	2,499,840	0.06	2,509,520	0.45	2,509,960	0.46	2,508,890	0.42
m8n1500s0	3,500,160	3,500,580	0.01	3,522,880	0.65	3,520,360	0.58	3,520,520	0.58
m8n1500s1	3,802,650	3,802,480	0.00	3,815,900	0.35	3,840,640	1.00	3,813,860	0.29
m8n1500s2	3,605,094	3,605,680	0.02	3,627,980	0.63	3,625,970	0.58	3,625,950	0.58
m8n1500s3	3,515,802	3,516,100	0.01	3,534,960	0.54	3,531,310	0.44	3,531,480	0.45
m8n1500s4	3,704,953	3,705,220	0.01	3,729,960	0.67	3,726,310	0.58	3,726,490	0.58
Average			0.03		0.62		0.63		0.51
Maximum			0.16		0.95		1.24		0.83

trade-off values between running time and quality. Better quality solutions could be obtained by increasing CG parameters I and Ω_{min}. In terms of efficiency, MTCG also presented competitive values with the results reported by Pepin et al. [15], on average reducing the CPU times in around 32 %. This reduction was expected, considering the use of more modern computational resources. MTCG efficiency is justified by the different path structures in the subproblems of the CG procedure, resulting from the introduced changes described in Sect. 3.2. Given the obtained results, MTCG was considered a valid and competitive CG implementation for the MDVSP.

Table 2 compares the optimality gap of our developed algorithms with the following state-of-the-art solutions reported in the literature [9,12,15] for each category instance: (i) Lagrange relaxation (LR); (ii) large neighborhood search combined with CG (LNS); (iii) Tabu search (TS); (iv) Ejection chain based approach (EC); and (v) variable depth search algorithm (VDS). The optimality gaps were computed as follows:

$$\text{Optimality GAP (in \%)} = 100 \times \frac{(VA - V_{best})}{V_{best}}$$

where V_{best} is the best known solution, and VA is the solution value obtained by the algorithm being compared, say algorithm A. The values presented in this table refer to the best solution found for each instance category.

Table 3 shows the average CPU time spent for each algorithm to find the best solution values, comparing them with the CPU times reported by Pepin et al. [15] and Otsuki and Aihara [12], called as VDS in the table. It is important to notice that this efficiency comparison with the CPU times presented by Pepin et al. [15] and Otsuki and Aihara [12] should be viewed with extreme cautions, since we performed the experiments 6 years after the former research work, and using different experimental conditions of both studies. Although we employed faster hardware and softwares, the

Table 2 Solution quality comparison of the developed algorithms with other methods

Category	Optimality gaps (%)									
	LR	LNS	TS	EC	VDS	MTCG	R1	R2	R1 + R2	CG
m4n500	3.12	2.18	10.92	1.85	1.29	0.01	0.51	0.45	0.41	0.17
m4n1000	3.88	1.41	8.31	1.22	1.05	0.01	0.27	0.45	0.21	0.33
m4n1500	5.54	2.05	10.78	1.86	1.40	0.02	0.34	0.33	0.33	0.41
m8n500	5.54	2.05	17.87	3.00	2.26	0.01	0.51	0.43	0.44	0.55
m8n1000	6.59	3.25	19.65	2.47	2.26	0.00	0.42	0.39	0.38	0.67
m8n1500	10.15	3.69	20.82	2.84	2.44	0.00	0.35	0.44	0.29	0.84

Table 3 Solution efficiency comparison of the developed algorithms with other methods

Category	Average CPU time (s)					
	Pepin et al. [15]	VDS	MTCG	R1	R2	R1 + R2
m4n500	77	71	62	3	4	4
m4n1000	651	612	423	7	22	28
m4n1500	2203	2012	868	56	86	79
m8n500	119	109	175	6	13	8
m8n1000	857	787	1380	73	103	97
m8n1500	3085	2800	1813	182	208	199

direct implementation of the truncated CG presented in Pepin et al. [15], using C++ and CPLEX 12.5, took excessive CPU time. In order to obtain comparable CPU usage we needed to implement our MTCG with the initialization process described above. The CPU times reported by Pepin et al. [15] were obtained using GENCOL, a computer program specially designed to solve a broad class of routing and scheduling problems. Such softwares employ special routines to substantially reduce the CPU time to solve a problem. However, the majority of these softwares are not of public domain, with a cost beyond the budget of the average transport company in development countries. As a consequence, we would like to point out that a precise comparison can only be carried out among our developed algorithms, having the same data structure and the same CPLEX parametrization. Only rough comparisons can be made using Pepin et al. [15] and Otsuki and Aihara [12]'s results.

Tables 1, 2, and 3 show that the variants of the developed heuristics, in general, offer good quality solutions (some with very tight gaps) in comparison with other methods, very efficiently. Table 2 shows that our developed algorithms present the best solution quality among the compared methods, overcoming mathematical programming methods such as LR, metaheuristic methods such as TNS and EC, and local search methods such as LNS and VDS. As expected, the solution quality improved as the reduced state space is enlarged by the combination of selection procedures. An opposite behavior is observed concerning the efficiency of the solution, characterizing a trade-off between CPU time and the optimal value of the objective function. Algorithms R1 and R2 presented similar average solution gaps for the tested instances (around 0.62 %). However, R2 presented a slightly higher maximum gap. Variant R1 + R2 obtained the best objective function values among the reduction state space based algorithms, with average and maximum gaps of 0.51 and 0.83 %, respectively. All three variants were extremely fast in comparison with MTCG. R1 is roughly 21 times faster, on average, than MTCG; R2 is 13 time faster, on average, than MTCG; while variant R1 + R2 is 14 times faster, on average, than MTCG. Variants R1, R2, and R1 + R2 were 21, 16, and 17 times quicker, on average, than the CPU times reported by Pepin et al. [15]; and 19, 14, and 15 times quicker, on average, when compared with Otsuki and Aihara's [12] CPU times. Moreover, this latter study claimed that VDS shows the best short-term performance, since this method obtained a good feasible solution in around 300 s CPU time for category m4n1500. Variants R1, R2, and R1 + R2 obtained, for the same category, near best available solutions in one quarter of this time, on average, making our developed heuristic framework a very competitive short-term performance method for solving the MDVSP.

We could not define a pattern in the CPU times reduction based on our experiments. For instance, variant R2 was on average quicker to solve instances with four depots, while variant R1 + R2 presented an opposite behavior. The best CPU times reductions were obtained for instances with four depots and 1000 trips. Further experimentation will be required to explain both behaviors.

Overall, the developed heuristic approach offered very competitive algorithms to solve the MDVSP, specially variants R1 and R1 + R2. Both variants obtained very good solutions, with maximum gaps below 1 % with very low CPU usage in comparison to previous reported results [12, 15]. Variant R1 + R2 offered the best compromise solutions in terms of quality and efficiency. This variant can be a good option in contexts and where solution quality is an important criterion. However, variant R1 was a quite good surprise, finding best solution values with reasonable average solution gaps (0.63 %), but requiring the minimum CPU time of all variants. We recommend its application in real-time transportation logistics environments, where the MDVSP is solved several times as a subproblem, and solution quality is an important, but no essential criterion. Variant R2 was clearly dominated by the remaining variants. Nevertheless, the final selection among these three variants is case dependent, and it will depend on further experimentation, with a special attention on the settings of CG parameters.

5 Conclusion

In this paper, we presented a two-phase heuristic to solve the MDVSP. First, the heuristics employs a combination of procedures to select, from the whole feasible solution space, a good set of arcs to compose the solution of the problem. Each procedure is based on the following selection criteria: (i) the set of paths found in the solution of $|K|$-SDVSPs; and (ii) the set of paths found in the solution of a relaxed MDVSP formulation, where the sequence of trips carried out by a vehicle can start and finish in different depots. The selected set of arcs is then solved by using a modified truncated CG algorithm. The developed approach led to very competitive method to solve the MDVSP.

Different combinations of the selection procedures were tested, resulting in three different variants. On the one hand, variant R1 achieved narrow gaps to best-known solutions (0.62 % on average) with exceptional running times (roughly 34 times faster, on average, than previous reported results). On the other hand, the variant R1+R2 offered the best trade-off between solution quality and running times. It was, on average, 16 times faster than previous reported results [12,15], obtaining solutions with an average gap of around 0.5 %.

Future research is directed toward using the heuristic method to develop novel approaches for solving both the real time vehicle recovery scheduling problem and the integrated crew and VSP. In both problems, the MDVSP needs to be solved several times as a subproblem. Moreover, we intend to alter the developed method to consider the time–space network [8] as the underlying vehicle scheduling network.

Acknowledgments The authors thank the two anonymous referees and the associate editor for their comments that greatly improved the quality of the paper. This research work was partially founded by CNPq, Brazil, grants 301453/2013-6 and 473033/2012-7.

References

1. Bertossi, A.A., Carraresi, P., Gallo, G.: On some matching problems arising in vehicle scheduling models. Networks **17**(3), 271–281 (1987)
2. Carpaneto, G., Dell'Amico, M., Fischetti, M., Toth, P.: A branch and bound algorithm for the multiple depot vehicle scheduling problem. Networks **19**(5), 531–548 (1989)
3. Dell'Amico, M., Toth, P.: Algorithms and codes for dense assignment problems: the state of the art. Discrete Appl. Math. **100**(1–2), 17–48 (2000)
4. Desaulniers, G., Hickman, M.D.: Public transit. In Barnhart, C., Laporte, B. (eds.) Handbooks in Operations Research and Management Science, Transportation, pp. 69–127. North-Holland, Amsterdam (2007)

5. Hadjar, A., Marcotte, O., Soumis, F.: A branch-and-cut algorithm for the multiple depot vehicle scheduling problem. Oper. Res. **54**(1), 130–149 (2006)
6. Huisman, D., Freling, R., Wagelmans, A.P.M.: Multiple-depot integrated vehicle and crew scheduling. Transp. Sci. **39**(4), 491–502 (2005)
7. Jonker, R., Volgenant, A.: A shortest augmenting path algorithm for dense and sparse linear assignment problems. Computing **38**(4), 325–340 (1987)
8. Kliewer, N., Mellouli, T., Suhl, L.: A time–space network based exact optimization model for multi-depot bus scheduling. Eur. J. Oper. Res. **175**(3), 1616–1627 (2006)
9. Laurent, B., Hao, J.L.: Iterated local search for the multiple depot vehicle scheduling problem. Comput. Ind. Eng. **51**, 277–286 (2009)
10. Löbel, A.: Vehicle scheduling in public transit and Lagrangean pricing. Manag. Sci. **44**(12), 1637–1649 (1998)
11. Lübbecke, M.E., Desrosiers, J.: Selected topics in column generation. Oper. Res. **53**(6), 1007–1023 (2005)
12. Otsuki, T., Aihara, K.: New variable depth local search for multiple depot vehicle scheduling problems. J. Heuristics, 1–19 (2014). doi:10.1007/s10732-014-9264-z (ISSN 1381-1231)
13. Oukil, A., Ben Amor, H., Desrosiers, J.: Stabilized column generation for highly degenerate multiple-depot vehicle scheduling problems. Comput. Oper. Res. **34**, 817–834 (2007)
14. Paixão, J.M., Branco, I.: A quasi-assignment algorithm for bus scheduling. Networks **17**(3), 249–269 (1987)
15. Pepin, A.-S., Desaulniers, G., Hertz, A., Huisman, D.: A comparison of five heuristics for the multiple depot vehicle scheduling problem. J. Sched. **12**(1), 17–30 (2009)
16. Ribeiro, C.C., Soumis, F.: A column generation approach to the multiple-depot vehicle scheduling problem. Oper. Res. **42**(1), 41–52 (1994)
17. Wolsey, L.A.: Integer Programming. Wiley, Chichester (1998)