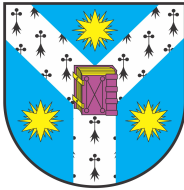

Alexandru Ioan Cuza University of Iaşi
Faculty of Computer Science
Str. General Berthelot 16, Iaşi 700483, România
Tel. +40-32-201090, email: tr@info.uaic.ro

TECHNICAL REPORT

Multiple-Depot Vehicle Scheduling Problem Heuristics

Emanuel Florentin Olariu,
Cristian Frăsinaru

TR 20-02, April 2020

ISSN 2668-1765

ISSN-L 1224-9327

Faculty of Computer Science, Alexandru Ioan Cuza University Iaşi
Technical Report TR 20-02, April 2020

Multiple-Depot Vehicle Scheduling Problem Heuristics

Emanuel Florentin OLARIU, Cristian FRĂSINARU
Faculty of Computer Science, Alexandru Ioan Cuza University Iaşi,
General Berthelot 16, 700483 Iasi, Romania,
Email: olariu@info.uaic.ro, acf@info.uaic.ro

Abstract

The Multiple-Depot Vehicle Scheduling Problem (MDVSP) is very important in the planning process of transport systems. It consists in assigning a set of trips to a set of vehicles in order to minimize a certain total cost. We introduce three fast and reliable heuristics for MDVSP based on a classical integer linear programming formulation and on graph theoretic methods of fixing the infeasible subtours gathered from an integer solution. Extensive experimentations using a large set of benchmark instances show that our heuristics are faster and give good or even better results compared with other existing heuristics.

Keywords: *MDVSP*, *linear programming*, *relaxation heuristic*.

1 Introduction

The multiple depot vehicle scheduling problem (MDVSP) is a well-known and important problem combinatorial and optimization problem. MDVSP arises in public transport and trucking industry being part of a large class of problems that includes vehicle routing and scheduling problems. Since

The publisher does not claim any copyright for the technical reports. The author keeps the full copyright for the paper, and is thus free to transfer the copyright to a publisher if the paper is accepted for publication elsewhere.

these problems are usually NP-hard the exact employed methods cannot solve medium or large MDVSP instances to optimality.

MDVSP aims to assign a set of timetabled tasks (trips) to a set of homogeneous vehicles provided by several depots in order to minimize a given (linear) objective function.

We are given a set of *trips* $T_0, T_1, \ldots, T_{n-1}$, each trip T_i having a starting time σ_i and an ending time τ_i, along with a set of *depots*, $D_0, D_1, \ldots, D_{m-1}$, each depot D_j has a number of r_j available vehicles. We are given also the time, θ_{ij} needed for a vehicle to travel from the end location of trip T_i to the start location of trip T_j - these values are useful for deciding if the vehicle performing the trip T_i can be used after that to perform the trip T_j. An ordered pair of trips (T_i, T_j) is *feasible* if $\tau_i + \theta_{ij} \leq \sigma_j$

A vehicle schedule can be described as an ordered sequence of trips such that any two consecutive trips is a feasible pair. Usually the cost of scheduling includes the sum of the traveling and/or waiting costs between two consecutive trips, the cost of pulling-out the vehicle from the depot to its first trip, and the cost of pulling-in the vehicle from its last trip to the depot.

A solution to MDVSP is an assignment of trips to vehicles that minimizes the sum of costs such that: each trip must be covered, each vehicle schedule must start and end its duty in the same depot, and the number of vehicles available in each depot is not exceeded.

When the number of depots is at least two MDVSP is known to be NP-hard ([2]). Several approaches have been proposed for this problem; among them: meta-heuristics like neighborhood search and Tabu search ([13]), iterated local search ([12]), and integer linear programming approaches which are the most frequent used methods.

Basically there are three models for MDVSP linear programming formulation: the single-commodity network flow model (introduced in [4]), the multi-commodity network flow model and the set partitioning model with side constraints. The multi-commodity model has two different flavors: the classical multi-commodity network flow formulation where the vehicles from different depots are viewed as different commodities ([14]) and the time-space network flow formulation ([9], [10]). The set partitioning model with side constraints ([14], [3], [8]) was derived using Dantzig-Wolfe decomposition.

The paper is organized as follows: section 2 describes the linear programming model and two relaxations of this model, section 3 describes the ways we fix a solution that contains the so-called *infeasible subtours*, section

4 contains the numerical results, and section 5 concludes the study.

2 LP model

We use the classical model of single-commodity network flow ([4]) because of its reasonable number of variables - in a multi-commodity flow formulation the number of variables is multiplied by the number of depots. The background of this model is the digraph $G = (V, E)$ (for graph notations and other graph related concepts used in this paper see [7]) where $V = \{0, 1, \ldots, m + n - 1\}$. The depots are the vertices in $V_d = \{0, 1, \ldots, m - 1\}$, while the remaining vertices represent the trips, $V_t = V \setminus V_d$.

An arc ij with $i, j \in V_t$ exists only if (T_{i-m+1}, T_{j-m+1}) is a pair of feasible trips and has an associated cost c_{ij} representing the traveling and/or waiting costs between these trips. An arc ij with $i \in V_d$ and $j \in V_d$ (respectively, $i \in V_t$ and $j \in V_d$) exists only if the starting (ending) duty of a vehicle from depot D_i (respectively, D_j) is possible, and its cost c_{ij} is the incurred cost of starting (ending) the duty from D_i (respectively to D_j) with the trip T_{j-m+1} (T_{i-m+1}).

The decision variables are x_{ij} and $x_{ij} = 1$ if and only if the arc ij is used in the optimal solution of the MDVSP. Defining r_i to be 1, for $i \in V_t$ and x_{jj} to be the the number of unused vehicles in the depot D_j, for $j \in V_d$, we have the following equivalent linear programming problem:

$$min \quad \left(\sum_{i=0}^{m+n-1} \sum_{j=0}^{m+n-1} c_{ij} x_{ij} \right) \tag{1}$$

$$\sum_{i=0}^{m+n-1} x_{ij} = r_j, 0 \le j \le m + n - 1 \tag{2}$$

$$\sum_{j=0}^{m+n-1} x_{ij} = r_i, 0 \le i \le m + n - 1 \tag{3}$$

$$\sum_{ij \in E(P)} x_{ij} \le |E(P)| - 1, P \in \Pi \tag{4}$$

$$x_{ij} \in \mathbb{Z}_+, 0 \le i, j \le m + n - 1 \tag{5}$$

where Π is the set of the (inclusion-wise minimal) infeasible paths, that is the paths connecting two different depots. We can linear relax this integer

problem by replacing integrality constraints (5) with

$$x_{ij} \geq 0, 0 \leq i, j \leq m+n-1 \tag{5'}$$

Since Π is very large, even for a reasonable number of depots, the constraints (4) could be difficult be difficult to implement, since using an enumerative technique for finding Π would be very costly. Our approaches are based on relaxing by reducing the number of constraints of type (4) and, than, apply a technique of repairing the integer solutions.

2.1 First relaxation - Circulation

The first relaxation of our problem is (1 - 3), (5) which can be solved efficiently by modeling it as a *minimum-cost circulation problem* [1], a generalization of the minimum-cost flow problem with node capacities and lower bounds on the edges.

For a given transportation network $R = (G, c, l, u, a)$, where $G = (V, E)$ is a digraph, $c : V \to \mathbb{R}$ represents the *capacities* of the nodes, $l, u : E \to \mathbb{R}_+$ represent *lower* and *upper* bounds for the edges and $a : E \to \mathbb{R}$ is a *cost* function interpreted as the cost of "sending" an unit of flow on a specific arc. If $c(v) > 0$, v is called a *supply* node, if $c(v) < 0$, v is a *demand* node, the remaining nodes being *transit* nodes. The sum of all node capacities must be zero, meaning that the supply must equal the demand. Let $S = \{v \in V \mid b(v) > 0\}$ be the *source* nodes and $T = \{v \in V \mid b(v) < 0\}$ be the *sink* nodes.

The minimum-cost circulation problem is to find a feasible circulation (a function that satisfies equations (7 - 8)) that minimizes the total cost:

$$min \quad \left(\sum_{ij \in E} c_{ij} x_{ij} \right) \tag{6}$$

$$l_{ij} \leq x_{ij} \leq u_{ij}, \forall ij \in E \tag{7}$$

$$\sum_{ij \in E} x_{ij} - \sum_{ji \in E} x_{ji} = b_i, \forall i \in V \tag{8}$$

The problem can be solved in a pseudo-polynomial time $O(nU(m + n) \log n)$ using the successive shortest path algorithm with capacity scaling [1], where $n = |V|$, $m = |E|$, and U is the maximum edge capacity.

In order to represent our scheduling problem as a minimum-cost circulation problem we define the following transportation network $R = (G', b, l, u, c')$, based on the initial digraph G and cost matrix c.

- For each depot D_i, add two nodes in $V(G')$, representing a source and a sink. The capacities of these nodes are $c(i) = r_i$ (the supply) and $c(i') = -r_i$ (the demand).
- For each trip T_{j-m+1}, add two nodes j^- and j^+ in G' with zero capacity (they will be transit nodes), and the arc j^-j^+ in G', having the cost 0 (we are transforming the trip nodes in G into arcs in G'); both the lower and the upper bounds of these arcs are set to 1, as we are looking for a solution that saturates all trips.
- For each arc ij connecting the depot D_i to a trip T_{j-m+1} in G, add the arc ij^- in G', having the cost c_{ij}, the lower bound 0, and the upper bound 1.
- For each arc ji connecting the trip T_{j-m+1} to a depot D_i in G, add the arc j^+i in G', having the cost c_{ji}, the lower bound 0, and the upper bound 1.
- For each arc ij connecting two trips T_{i-m+1} and T_{j-m+1} in G, add the edge i^-j^+ in G', having the cost c_{ij}, the lower bound 0, and the upper bound 1.
- For each depot D_i in G, add the arc ii' in G', having the cost 0, the lower bound 0, and the upper bound r_i (this arc is necessary when some vehicles in the depot D_i are not used).

It is straightforward to prove that a solution for the minimum-cost circulation problem defined for the network $R = (G', b, l, u, c')$ represents an optimal solution for the relaxation (1 - 3), (5). The total cost is the same for both problems, the supply/demand constraints ensure that the number of vehicles used from a depot is not exceeded and the lower bound constraints imposed for the arcs j^-j^+ ensure that all trips are saturated.

This model is a guarantee that the relaxed MDVSP problem can be solved in an efficient manner. From a practical point of view, the polynomial complexity guarantee usually translates in an easy resolution when it comes to dedicated MIP solvers, such as Gurobi.

2.2 Second relaxation

Our second method of relaxing the original problem consists in replacing Π by a smaller set of infeasible subtours (paths that link different depots)

by adding one by one constraints of type (4) and re-optimizing until the new problem has the same optimum as (1 - 4), (5'). This procedure of re-optimizing is based on the interpretation of graph theoretic properties of a fractional solution.

At a certain step during the algorithm we have a particular set of infeasible subtours Π' and (4) is replaced in the current problem by

$$\sum_{ij \in E(P)} x_{ij} \quad \leq \quad |E(P)| - 1, P \in \Pi' \tag{4'}$$

Consider $\mathbf{x}^* = (x^*_{ij})$, a solution to problem (1 - 3), (4'), (5'), and define a weight on the edges of the underlying digraph: $\alpha_{ij} = 1 - x^*_{ij}$, for all arcs ij. (4) is equivalent with

$$\alpha(P) \geq 1, P \in \Pi \tag{4''}$$

since

$$\sum_{ij \in E(P)} x^*_{ij} \leq |E(P)| - 1 \Leftrightarrow \sum_{ij \in E(P)} (1 - x^*_{ij}) \geq 1 \Leftrightarrow \sum_{ij \in E(P)} \alpha_{ij} \geq 1.$$

Hence, $\mathbf{x}^*$ is an optimum solution to (1 - 4), (5') if and only if the underlying digraph doesn't contain paths between different depots of sub-unitary weight. We will test this by using an algorithm for finding shortest paths in a weighted graph, like Floyd-Warshall or Bellman-Ford-Moore.

Therefore the first step is to relax the problem (1 - 4), (5') to (1 - 3), (4' - 5') for a certain known set of infeasible paths $\mathcal{D}'$ such that the two problems have the same optimum. The process of building this problem is given below.

$\Pi' \leftarrow \varnothing$;
solve problem (1 - 3), (4' - 5') and let $\mathbf{x}^*$ be an optimum solution;
while (there exists a path D with $\alpha(D) < 1$) **do**
  add D to $\mathcal{D}'$;
  solve the problem (1 - 3), (4' - 5') and let $\mathbf{x}^*$ be an optimum solution;
end while
return $\mathbf{x}^*$.

The aim of the above procedure is to build a problem that has a larger (but known) set of feasible solutions but the same optimum with (1 - 4), (5')

2.3 Column generation perspective for the second relaxation

Writing the original relaxed problem as a maximum one means to replace (1) by (1') (ignoring the minus in front of *max*)

$$max \quad \left(\sum_{i=0}^{m+n-1} \sum_{j=0}^{m+n-1} -c_{ij} x_{ij} \right) \tag{1'}$$

The dual of the problem (1'), (2) - (4), (5') is

$$min \quad \left(\sum_{j=0}^{m+n-1} r_j y_j + \sum_{i=0}^{m+n-1} r_i z_i + \sum_{P \in \Pi} (|E(P)| - 1) u_P \right) \tag{9}$$

$$z_i + y_j + \sum_{P \in \Pi : ij \in E(P)} u_P \geq -c_{ij}, \forall ij \in E(G) \tag{10}$$

$$u_P \geq 0, \forall P \in \Pi \tag{11}$$

We can replace (10) by (10'), and (11) by (11')

$$z_i + y_j + \sum_{P \in \Pi : ij \in E(P)} u_P - v_{ij} = -c_{ij}, \forall ij \in E(G) \tag{10'}$$

$$v_{ij} \geq 0, \forall ij \in E(G), u_P \geq 0, \forall P \in \Pi \tag{11'}$$

and get the dual in equations form: (9) (10'), (11'). An initial feasible basic solution to this problem could be $v_{ij} = c_{ij}$, $\forall ij \in E(G)$. Now, this dual problem has a very large number of variables and we can solve it by using the column generation method (see for example [5], [6]). We start with a small set of variables (that contains a feasible basis) - this is the restricted master problem - and in each step - by solving the corresponding sub-problem - find a variable with the minimum negative reduced cost that would be added to the current problem. When such variables doesn't exists we have an optimum solution to the dual problem.

In our case the sub-problem would be

$$\arg\min_{P \in \Pi} \left(|E(P)| - 1 - \sum_{ij \in E(P)} x_{ij} \right) < 0 \tag{10'}$$

This holds because the variables z_i and y_j cannot enter the basis - they cannot have negative reduced cost based on (2) and (3).

Adding a new variable (column) to the dual problem means adding a new constraint to the primal, i. e., finding a new path between different depots of α sub-unitary cost. Hence solving the problem with the algorithm from the subsection 2.2 is equivalent with solving the dual using the column generation.

3 Building feasible solutions - Path repairing methods

The next step is to solve one of the two problems (1) - (3), (5) or (1) - (3), (4'), (5) which are integer linear programming problems. Because of their reasonable size these problems can be easily solved with existing LP solvers - for the first one there exist even combinatorial algorithms (see subsection 2.1). Solutions to this problems may not be feasible for (1) - (5), thus a process of fixing the infeasible subtours (that is a path that starts in a depot and ends in a different depot) must follow.

In the remaining of this section we suppose that we have a solution, $\mathbf{x}^*$, to one of the above two integer linear programming problems.

First we define an auxiliary digraph $H = (V_d, A_d)$, where $ij \in A_d$ if and only if there is an infeasible subtour between the depots i and j; we add also a weight on this arc w_{ij} which represents the number of such subtours. By inspecting $\mathbf{x}^*$ we extract the infeasible tours and memorize them, build H and the weight w.

3.1 Repairing one subtour

Suppose that we have the infeasible subtour (described by its arcs):

$$P : it_1, t_1t_2, \ldots, t_{p-1}t_p, t_pj,\ i, j \in V_d,$$

where $i, j \in V_d$ and $t_h \in V_t, \forall h = \overline{1, p}$; this subtour can be replaced by

$$P' : it_1, t_1t_2, \ldots, t_{p-1}t_p, t_pi$$

with cost penalty $c'(P) = c\,(t_pi) - c\,(t_pj)$, or by

$$P\text{"} : jt_1, t_1t_2, \ldots, t_{p-1}t_p, t_pj,$$

with cost penalty $c''(P) = c\,(jt_1) - c\,(it_1)$.

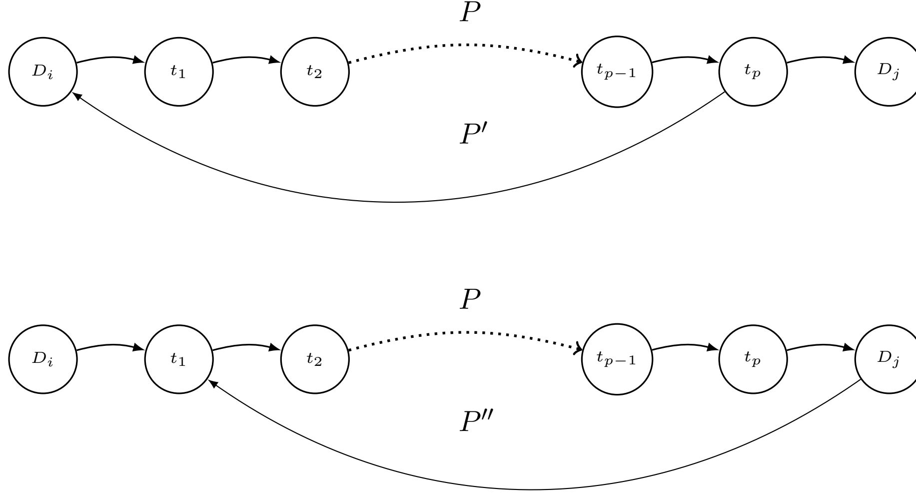

Figure 1: Repairing one subtour.

3.2 Repairing a pair of subtours

Suppose now that we have two infeasible subtours

$$P_1 : it_1^1, t_1^1 t_2^1, \ldots, t_{p-1}^1 t_p^1, t_p^1 j \text{ and } P_2 : jt_1^2, t_1^2 t_2^2, \ldots, t_{q-1}^2 t_q^2, t_q^2 i,$$

where $i, j \in V_d$ and $t_h^1, t_k^2 \in V_t, \forall h = \overline{1,p}, k = \overline{1,q}$. Such a pair will be called *compatible*.

If we can find a pair (h, k), $1 \leq h \leq p$ and $1 \leq k \leq q$ such that $t_h^1 t_k^2, t_{k-1}^2 t_{h+1}^1 \in E(G)$, then we can replace the above pair of infeasible but compatible subtours by the following pair of feasible subtours

$$\begin{aligned} P_1' &: it_1^1, t_1^1 t_2^1, \ldots, t_{h-1}^1 t_h^1, t_h^1 t_k^2, t_k^2 t_{k+1}^2, \ldots, t_{q-1}^2 t_q^2, t_q^2 i \\ P_2' &: jt_1^2, t_1^2 t_2^2, \ldots, t_{k-2}^2 t_{k-1}^2, t_{k-1}^2 t_{h+1}^1, t_{h+1}^1 t_{h+2}^1, \ldots, t_{p-1}^1 t_p^1, t_p^1 j. \end{aligned}$$

The cost penalty is

$$\gamma_{hk} = c\left(t_h^1 t_k^2\right) + c\left(t_{k-1}^2 t_{h-1}^1\right) - c\left(t_i^1 t_{i+1}^1\right) - c\left(t_{k-1}^2 t_k^2\right).$$

We will choose the pair (h, k) for which the cost penalty is minimum, i. e.,

$$c(P_1, P_2) = \min_{1 \leq h \leq p, 1 \leq k \leq q} \gamma_{hk}.$$

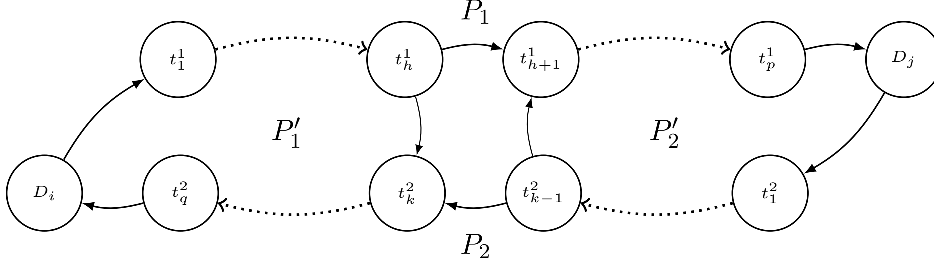

Figure 2: Repairing two subtours.

3.3 Fixing all subtours

The first way of repairing all subtours is to iterate the first method for all infeasible tours. A better way is to use both methods above: we match infeasible subtours or choose to repair a subtours by itself.

Since for any specific infeasible subtour P_1 there may be more than one compatible subtour P_2, we must match infeasible subtours in a manner that minimizes the overall penalty of the repairing. To implement this method we create a bipartite graph $\mathcal{H} = (\mathcal{S}, \mathcal{T}; \mathcal{E})$ that models the compatibility relation between infeasible subtours. The partition classes are

$$\mathcal{S} = \{P \,:\, P \text{ is an infeasible subtour with respect to } \mathbf{x}^*\}$$
$$\mathcal{T} = \{P' \,:\, P \text{ is an infeasible subtour with respect to } \mathbf{x}^*\}.$$

The set of edges is

$$\mathcal{E} = \{P_1P_2' \,:\, (P_1, P_2) \text{ is a compatible pair of infeasible subtours}\} \cup \cup \{PP' \,:\, P \text{ is an infeasible subtour}\} \quad (12)$$

We define on these edges two weight functions $\gamma', \gamma'' : \mathcal{E} \to \mathbb{R}$: $\gamma'(P_1P_2') = \gamma''(P_1P_2') = c(P_1, P_2)$, if P_1 and P_2 are compatible infeasible subtours, and $\gamma'(PP') = c'(P)$, $\gamma''(PP') = c''(P)$ if P is an infeasible subtour.

Since $\mathcal{H}$ has perfect matchings (due to the edges of the form PP'), we can find two γ' - and γ'' - minimum weight perfect matchings using the Kuhn-Munkres (Hungarian) algorithm in $O(|\mathcal{S}|^3)$ time complexity. We choose the matching having the smaller weight for repairing the solution $\mathbf{x}^*$; applying this method will fix all the infeasible subtours. In our numerical experiments the number of infeasible subtours is quite small, hence this method will work fast in practice.

In this way the method of repairing just one subtour is used by this second method that fixes all the infeasible tours. Iterating the first method cannot improve the result of the second since the latter already takes account the edges PP' (for all infeasible subtours P).

3.4 Heuristics

We developed three heuristics based on these linear integer problems and fixing subtours method. The first heuristic(H_1), after solving (6 - 8) (or, equivalently, (1 - 3), (5)), fixes all the infeasible subtours by finding the minimum weight matching in the corresponding bipartite graph. The second heuristic (H_2) requires a given number (a pool) of integer solutions and then builds the bipartite graphs for all of these solutions, fixes the subtours and chooses the best repaired solution.

The third heuristic (H_3) first builds the set of infeasible subtours Π' by repeatedly finding paths of sub-unitary weight in the subjacent digraph, then solves the ILP (1 - 3), (4'), (5); the resulting solution is then fixed using the above method.

4 Computational results

In this section we describe the results of our numerical experiments using the heuristics described above. Our test bed is composed of the instances used in ([13]) (see the Huisman's website `https://personal.eur.nl/huisman/instances.htm`) and most of the instances in ([10]) (see ([11])), all generated with the method from ([4]).

All computational results from below were obtained using an Intel(R) Core (TM) i5-7500 CPU 3.40GHz computer with 8GB RAM, under Ubuntu 18.04.4 LTS.

The linear programming problems were solved using Gurobi 9.0 under an Academic License.

The experiments were performed using seventy different instances. Results are very close to the best known results; for instances with at most 1500 trips and at most 8 depots all the results but three are within 1% relative error from the best known corresponding solutions. The fastest of the three heuristics is, as expected, H_1; the best results are obtained with H_2 or H_3. The differences among the three heuristics are very small.

The results reported in the tables below use the following performance

measure:

$$\text{percent error (\%)} = 100 \cdot \frac{z - z^0}{z^0},$$

where z^0 the the best known objective function value and z is the heuristic's obtained value.

Tables 1 and 2 show the best known objective function values, the objective values for our heuristics, the CPU times spent, and the percent errors. For the larger instances (table 2) the second heuristic (H_2) proved to be too time costly, H_1 remains the faster and H_3 gives the best results in terms of gap.

5 Concluding remarks

The MDVSP is very important in the management of public transport systems. Our paper introduces three heuristics based on a classical integer linear programming formulation and on graph theoretic methods of fixing the infeasible subtours gathered from an integer solution.

The effectiveness of our heuristics was proved by extensive experimentations using a large set of benchmark instances. Our heuristics are fast and give good results compared with other existing solutions; the running times are much lower (but we performed the experiments four years later) and the results are better than in [8] for the benchmarks from Huisman's website. Compared with the results in [10], the running times for larger benchmark instances are lower and the results remain within 1.6% percent error, while we worked on a mainstream desktop PC instead of a dedicated server.

Future work will be directed towards a truncated branch and bound (and branch and cut) algorithm based on adding constraints like in the second relaxation - corresponding with a column generation procedure for the dual problem. Our heuristics can be used for providing initial upper bounds for such an algorithm.

References

[1] Ahuja, R. K., Magnanti, L. T., Orlin, J. B. (1993) "Network Flows: Theory, Algorithms, and Applications", Prentice-Hall.

[2] Bertossi, A., Carraresi, P., Gallo, G. (1987) "On some Matching Problems Arising in Vehicle Scheduling Models." *Networks* **17** (3): 271–281.

[3] Bianco, L., Mingozzi, A., Ricciardelli, S. (1994) ”A set partitioning approach to the multiple depot vehicle scheduling problem.” *Org. Methods and Soft.* **3**: 163–194.

[4] Carpaneto, G., Dell'Amico, M., Fischetti, M., Toth, P. (1989) “A Branch and Bound Algorithm for the Multiple Depot Vehicle Scheduling Problem.” *Networks* **19** (5): 531–548.

[5] Desrosiers, J., Lubbecke, M. E. (2005) ”A primer in column generation.” in *Column Generation* Desaulniers, G., J. Desrosiers, M.M. Solomon (Eds.): 1–32.

[6] Desrosiers, J., Lubbecke, M. E. (2004) ”Selected Topics in Column Generation.” Technical Report, Technische Universitat Berlin, 2004/08.

[7] Diestel, R. (2000) ”Graph Theory”, electronic edition, Springer.

[8] Guedes, P. C., Lopes, W. P., Rohde, L. R., Borenstein, D. (2016) ”Simple and efficient heuristic approach for the multiple-depot vehicle scheduling problem.” *Optimization Letters*, **10**: 1449–1461.

[9] Kliewer, N., Mellouli, T., Suhl, L. (2006) ”A time-space network based exact optimization model for multi-depot bus scheduling.” *European Journal of Operations Research* **175** (3): 1616–1627.

[10] Kulkarni, S., Krishnamoorthy, M., Ranade, A., Ernst, A. T., Patil, R. (2018) ”A new formulation and a column generation-based heuristic for the multiple depot vehicle scheduling problem.” *Transportation Research Part B Methodological* **B** (118): 457–487.

[11] Kulkarni, S., Krishnamoorthy, M., Ranade, A., Ernst, A. T., Patil, R. (2019) ”A benchmark dataset for the multiple depot vehicle scheduling problem.” *Data in Brief* **B** (22): 484–487.

[12] Laurent, B., Hao, J.-K. (2009) ”Iterated local search for the multiple depot vehicle scheduling problem.” *Computers and Industrial Engineering* (57): 277–286.

[13] Pepin A.-S., Desaulniers, G., Hertz, A., Huisman, D. (2009) ”Comparison of heuristic approaches for the multiple depot vehicle scheduling problem”, *Journal of Scheduling* **12** (17): 17–30.

[14] Riberio, C., Soumis, F. (1994) "A column generation approach to the multiple-Depot vehicle scheduling problem." *Operations Research* **42** (1): 41–52.

Table 1: Solutions and computational times for different heuristics.

Instance	Best solution	Heuristics solutions			CPU time (s)			Percent error (%)		
		H_1	H_2	H_3	H_1	H_2	H_3	H_1	H_2	H_3
m4n500s0	1,289,114	1,296,409	**1,295,671**	1,295,678	0.7	7.9	6.5	0.56%	0.50%	0.50%
m4n500s1	1,241,618	1,247,438	**1,246,655**	1,247,173	0.6	11.8	5.4	0.46%	0.40%	0.44%
m4n500s2	1,283,811	1,292,079	1,291,745	**1,290,891**	0.5	17.4	5.6	0.64%	0.61%	0.55%
m4n500s3	1,258,634	1,263,624	**1,263,045**	1,264,473	0.4	10.2	4.7	0.39%	0.35%	0.46%
m4n500s4	1,317,077	1,322,535	1,322,306	**1,321,138**	0.5	11.7	4.9	0.41%	0.39%	0.30%
m4n1000s0	2,516,247	2,528,728	**2,527,966**	2,528,299	2.8	41.4	48.7	0.49%	0.46%	0.47%
m4n1000s1	2,413,393	2,421,735	2,421,735	**2,420,440**	2.5	37.4	29.2	0.34%	0.34%	0.29%
m4n1000s2	2,452,905	2,461,985	2,461,787	**2,461,347**	2.7	38.4	17.9	0.37%	0.36%	0.34%
m4n1000s3	2,490,812	2,498,319	**2,498,046**	2,498,423	2.5	65.3	28.8	0.30%	0.29%	0.30%
m4n1000s4	2,519,191	2,525,357	2,525,004	**2,524,898**	2.6	45.6	13.5	0.24%	0.23%	0.22%
m4n1500s0	3,830,912	3,847,046	3,846,785	**3,846,761**	6.5	151.6	136.6	0.41%	0.41%	0.41%
m4n1500s1	3,559,176	3,566,055	3,565,962	**3,564,918**	7.0	177.1	56.5	0.19%	0.19%	0.16%
m4n1500s2	3,649,757	3,662,319	3,661,323	**3,661,344**	7.2	232.1	115.4	0.34%	0.31%	0.31%
m4n1500s3	3,406,815	3,419,905	3,419,810	**3,417,225**	5.9	289.2	118.7	0.38%	0.38%	0.30%
m4n1500s4	3,567,122	3,583,176	3,582,852	**3,581,059**	6.3	208.8	122.5	0.45%	0.44%	0.39%
m8n500s0	1,292,411	1,304,837	1,302,517	**1,301,395**	0.5	9.4	8.1	0.96%	0.78%	0.69%
m8n500s1	1,276,919	1,289,875	1,288,006	**1,289,407**	0.6	42.4	5.4	1.01%	0.86%	0.97%
m8n500s2	1,304,251	1,316,965	1,316,108	**1,313,993**	0.5	18.5	5.8	0.97%	0.90%	0.74%
m8n500s3	1,277,838	**1,290,397**	**1,290,397**	1,290,852	0.5	9.4	6.4	0.98%	0.98%	1.01%
m8n500s4	1,276,010	1,289,435	**1,287,919**	1,288,606	0.6	11.8	7.1	1.05%	0.93%	0.98%
m8n1000s0	2,422,112	2,441,490	**2,439,817**	2,439,893	2.7	143.5	57.6	0.80%	0.73%	0.73%
m8n1000s1	2,524,293	2,542,668	**2,542,668**	2,545,417	3.0	31.3	33.2	0.72%	0.72%	0.83%
m8n1000s2	2,556,313	2,581,639	2,580,507	**2,579,511**	2.6	182.0	45.4	0.99%	0.94%	0.90%
m8n1000s3	2,478,393	2,499,109	2,495,968	**2,494,389**	2.8	217.5	38.0	0.83%	0.70%	0.64%
m8n1000s4	2,498,388	2,518,121	2,517,631	**2,516,357**	2.9	34.3	42.1	0.79%	0.77%	0.71%
m8n1500s0	3,500,160	3,527,083	**3,527,083**	3,530,381	5.8	114.6	106.9	0.76%	0.76%	0.86%
m8n1500s1	3,802,650	3,821,483	3,819,634	**3,818,617**	6.0	335.2	393.9	0.49%	0.44%	0.42%
m8n1500s2	3,605,094	3,640,171	**3,635,622**	3,636,799	5.5	268.3	155.4	0.97%	0.84%	0.87%
m8n1500s3	3,515,802	3,537,090	**3,536,906**	3,536,931	5.8	283.8	139.1	0.60%	0.60%	0.60%
m8n1500s4	3,704,953	3,733,572	3,733,572	**3,730,221**	5.5	101.5	120.7	0.77%	0.77%	0.68%

Table 2: Solutions and computational times for different heuristics.

Instance	Best solution	Heuristics solutions		CPU time (s)		Percent error (%)	
		H_1	H_3	H_1	H_3	H_1	H_3
m8n2000s0	4,916,810	4,975,718	**4,962,626**	9.8	402.9	1.19%	0.93%
m8n2000s1	4,769,442	4,819,440	**4,813,103**	9.4	807.6	1.04%	0.91%
m8n2000s2	4,897,886	4,948,430	**4,938,756**	8.9	474.7	1.03%	0.83%
m8n2000s3	5,171,924	5,231,090	**5,220,119**	9.0	513.5	1.14%	0.93%
m8n2000s4	4,761,862	4,808,420	**4,802,721**	9.1	469.5	0.97%	0.85%
m8n2500s0	5,911,824	5,981,468	**5,961,055**	14.7	879.1	1.17%	0.83%
m8n2500s1	6,296,870	6,363,706	**6,357,577**	14.8	1,347.4	1.06%	0.96%
m8n2500s2	5,835,360	5,895,176	**5,887,819**	13.6	1,095.8	1.02%	0.89%
m8n2500s3	6,046,374	6,110,906	**6,104,058**	14.5	1.015.1	1.06%	0.95%
m8n2500s4	6,021,410	6,078,364	**6,075,874**	14.0	1,238.4	0.94%	0.90%
m12n1500s0	3,621,952	3,670,642	**3,663,952**	5.1	215.9	1.34%	1.16%
m12n1500s1	3,523,474	**3,570,252**	3,570,484	5.5	189.0	1.32%	1.33 %
m12n1500s2	3,932,474	3,988,062	**3,983,324**	4.6	160.4	1.41%	1.29%
m12n1500s3	3,789,274	3,833,318	**3,831,427**	4.7	122.5	1.15%	1.10%
m12n1500s4	3,694,646	3,745,298	**3,738,872**	4.6	233.5	1.37%	1.19%
m12n2000s0	5,239,126	5,301,310	**5,294,030**	10.8	530.9	1.18%	1.04%
m12n2000s1	4,844,414	4,907,954	**4,899,575**	9.4	3,275.3	1.31%	1.13%
m12n2000s2	4,611,692	4,667,510	**4,665,170**	9.3	883.5	1.21%	1.16%
m12n2000s3	4,822,028	4,881,702	**4,871,930**	9.1	438.5	1.23%	1.03%
m12n2000s4	4,961,406	5,025,946	**5,019,364**	8.8	684.5	1.30%	1.16%
m12n2500s0	5,860,766	5,948,488	**5,937,754**	14.5	1,101.0	1.49%	1.13%
m12n2500s1	6,000,516	**6,070,994**	6,071,772	14.4	1,534.9	1.17%	1.18%
m12n2500s2	5,940,276	6,012,290	**6,005,232**	15.6	1,236.5	1.21%	1.10%
m12n2500s3	6,072,130	6,154,820	**6,140,502**	14.5	1,060.8	1.36%	1.12%
m12n2500s4	5,748,976	5,817,298	**5,819,317**	15.5	1,725.6	1.18%	1.12%
m16n1500s0	3,568,522	3,618,204	**3,616,462**	5.5	195.7	1.39%	1.34%
m16n1500s1	3,591,374	**3,637,940**	3,641,353	5.5	150.2	1.29%	1.39%
m16n1500s2	3,554,800	3,604,392	**3,601,830**	4.8	322.5	1.39%	1.32%
m16n1500s3	3,861,652	3,914,558	**3,908,222**	4.8	217.2	1.27%	1.20%
m16n1500s4	3,603,796	**3,657,334**	3,659,326	5.6	357.9	1.48%	1.54%
m16n2000s0	4,789,504	4,856,390	**4,852,389**	9.8	3,705.6	1.39%	1.31%
m16n2000s1	4,680,998	4,745,990	**4,744,248**	9.7	2,347.0	1.38%	1.35%
m16n2000s2	4,774,408	4,847,436	**4,834,352**	10.1	732.3	1.53%	1.25%
m16n2000s3	4,850,652	4,926,124	**4,920,016**	9.9	520.2	1.55%	1.43%
m16n2000s4	4,700,490	4,767,256	**4,761,708**	9.6	649.6	1.42%	1.30%
m16n2500s0	5,960,298	6,045,500	**6,038,849**	15.5	1,318.1	1.42%	1.31%
m16n2500s1	6,055,252	6,148,106	**6,138,606**	15.0	1,566.7	1.53%	1.37%
m16n2500s2	6,043,364	6,123,422	**6,118,930**	16.2	1,240.2	1.32%	1.25%
m16n2500s3	6,067,858	6,155,328	**6,148,939**	15.3	902.3	1.44%	1.33%
m16n2500s4	5,857,966	5,952,214	**5,942,387**	14.5	1,414.2	1.60%	1.44%