Solving the multiple-depot vehicle scheduling problem

Bachelor Thesis
Econometrics & Operations Research

Nemanja Milovanović [1]
Supervisor: Dr. D. Huisman
Co-reader: Prof. Dr. A.P.M. Wagelmans

Erasmus School of Economics
Erasmus University Rotterdam

July 5, 2015

[1]Studentnumber: 378244

Abstract

The Multiple-Depot Vehicle Scheduling Problem (MDVSP) is the problem where one wants to assign timetabled tasks to vehicles which are housed in multiple depots. This problem is highly relevant for the public transport domain, as competition grows fierce and companies need to stay competitive to survive. As solving the MDVSP optimally is difficult in practice, the aim of this thesis is to give a detailed description for two heuristics discussed in Pepin et al. (2009), namely truncated branch-and-cut and the Lagrangian heuristic. Our implementation of the truncated branch-and-cut heuristic has very poor behavior, which is probably caused by CPLEX' root node heuristics. Also, in comparison with the heuristics in Pepin et al. (2009), our implementation of the Lagrangian heuristic is rather lackluster, only outperforming their tabu search heuristic. This is probably due to poor code-optimization. Nevertheless, our implementation still provides solutions within 1% optimality gap for all the test instances considered.

Contents

1 Introduction

As public transport nowadays is getting more and more privatized, firms seek to operate as efficiently as possible in order to attain a better market position. Bus companies and railway operators, for example, all need to create schedules for their vehicles in which is described which trip they cover. This problem is called the *vehicle scheduling problem* (VSP). In this thesis we will assume that the vehicles are stationed in multiple depots, which is appropriately called the *multiple-depot vehicle scheduling problem* (MDVSP).

The motivation for this thesis is to provide a detailed account on two heuristics discussed in Pepin et al. (2009): Truncated branch-and-cut, and a Lagrangian heuristic. Especially for the Lagrangian heuristic, there are lots of ways to implement the various features. This thesis aims to give the reader an example of which methods can be chosen, and also aims to give a detailed description such that the reader may be able to implement the heuristics him- or herself. As such, this thesis is more practically oriented than theoretically. Basic computer and mathematical programming knowledge is assumed.

This thesis is organized as follows. In Chapter 2, we describe the MDVSP thoroughly. In Chapter 3, we provide a short overview of the necessary literature used in this thesis and for further reading. We give a mathematical programming formulation of the MDVSP in Chapter 4. Following in Chapter 5, we give an overview of the heuristics which are treated in this paper and we provide the details that are needed to implement them. The computational results of the heuristics are then presented in Chapter 6, after which we conclude this thesis with some remarks in Chapter 7.

2 Problem description

In this thesis, we base our notation on Pepin et al. (2009). The MDVSP can be described as follows. Given a set T of timetabled tasks and a set of depots K, produce least-cost vehicle schedules in which every task is performed by exactly one vehicle, and the depot capacity v_k for $k \in K$ is not exceeded. In our case, we assume that all tasks can be performed by vehicles in all depots. This is, for example, the case when the fleet is homogeneous.

For every task $i \in T$ there exist start and end locations which we model as vertices s_i and e_i, respectively. It is allowed that $s_i = e_i$, but this need not be the case. Furthermore, a task is characterized by a start time $a_i \geq 0$, and a duration $\delta_i \geq 0$. A vehicle schedule is feasible if all tasks are performed by exactly one vehicle, and for every pair of tasks (i, j) it holds that $a_i + \delta_i + t_{ij} \leq a_j$, where t_{ij} is the travel time from e_i to s_j. This implies that task i starts and ends before j does. Total schedule costs are the sum of the total travel and total waiting costs. The total travel costs consists of costs made by traveling from the depot to the first task, traveling in between tasks, and traveling from the last task to the depot. The total waiting costs consist of the sum of the costs of waiting in between tasks.

We now introduce some terminology which is quite common in the MDVSP literature. A *pull-out trip* is a trip from the depot to the start location s_i of some task i. On the other hand, a *pull-in trip* is a trip from the end location e_i of some task i to the depot. Lastly, a *deadhead trip* is a trip from the end location e_i of task i to the start location s_j of task j, for some $i \neq j$.

For an example of an MDVSP (taken from Kliewer et al. 2006), see Figure 1. Here, an instance is depicted with $|K| = 2$ depots and $|T| = 3$ tasks, where task 1 starts and ends before task 2, and task 2 starts and ends before task 3.

Task i	s_i	e_i	a_i	$a_i + \delta_i$
1	A	B	1	2
2	B	A	3	4
3	B	A	5	6

Table 1: *Example of an MDVSP with* $|K| = 2$ *depots and* $|T| = 3$ *tasks.*

There exist several extensions of the MDVSP. A closely related problem is the *crew scheduling problem* (CSP), which consists of the scheduling of so-called *duties*. All tasks are assigned to exactly one duty in a cost-minimizing manner, and duties can have additional labor constraints. An extension of the MDVSP is the *multiple-depot integrated vehicle- and crew-scheduling problem*

(MD-VCSP) (see Huisman et al. 2005), which integrates the MDVSP with the CSP (normally, CSP and MDVSP are done sequentially). Also, one could consider tasks with *time-windows* (TW) in the MDVSP or the MD-VCSP, which can considerably lower the necessary amount of vehicles and duties (Kliewer et al., 2012) which in turn lowers the total costs. These extensions will not be considered in this thesis, however.

3 Literature overview

A great starting point in the MDVSP literature is Löbel (1997), which is a dissertation on optimal vehicle scheduling as applied to public transit. This dissertation focuses primarily on the MDVSP, more specifically, it provides a short overview of the necessary mathematics involved after which several techniques are discussed to solve the MDVSP exactly, including branch-and-bound and variants thereof. Other mathematical programming approaches can be found in Hadjar et al. (2006), and Ribeiro and Soumis (1994).

Although the MDVSP is often formulated as a connection network, Kliewer et al. (2006) apply the time-space network model to the problem. Also, because of this modeling approach, they are able to aggregate many deadhead arcs. The authors report a tremendous reduction in variables due to this aggregation.

As the MDVSP is NP-hard, several heuristic approaches have been proposed. An overview of some heuristics can be found in Pepin et al. (2009). Also, they propose a *large neighborhood search* (LNS) heuristic which uses column generation to evaluate the neighborhood. More recently, other local search heuristics have been proposed by Laurent and Hao (2009) and Otsuki and Aihara (2014). The former propose an *iterated local search* heuristic (ILS) which uses the notion of *ejection chains* in the construction of their neighborhood. The latter present a *variable-depth local search* heuristic (VDS), which utilizes the previous ILS in a variable-depth context. Both articles use the same test instances as presented in Pepin et al. (2009) and report superior results as compared to their LNS (still inferior to their column generation approach), but fail to compare computation times.

The Lagrangian heuristic as discussed in this paper uses three key features that require additional literature. First and foremost, it uses subgradient optimization to optimize the Lagrangian multipliers. Although this thesis uses the subgradient method proposed by Pepin et al. (2009), the reader may find it useful to try a different method. We therefore point the reader to a dissertation by Guta (2003), in which subgradient optimization is treated extensively. The second and third features are obtaining the lower and upper bounds, respectively. For these components, we use auction algorithms as proposed by Freling et al. (2001) and Bertsekas and Castañon (1992) for the quasi-assignment and asymmetric assignment problems, respectively. Alternatives to these auction algorithms can also be used. Some of these can be found in the book by Bertsekas (1998), which is an excellent account on network optimization in general.

4 Model formulation

The MDVSP can be formulated in different ways. However, in this thesis we only use one formulation, namely the connection network variant of the multi-commodity flow network, which can be described as follows. Let $G^k = (V^k, A^k)$ for each depot $k \in K$. Here V^k is the set of vertices, which contains each task vertex $i \in T$ and also $o(k)$ and $d(k)$, respectively the start and end vertices of depot k. Thus, it holds that $V^k = \{o(k), d(k)\} \cup T$. The set of arcs A^k contains three types of arcs: pull-out arcs, connection arcs, and pull-in arcs. Pull-out arcs are defined as pairs $(o(k), j)$ for all $j \in T$. This means that every task has a pull-out arc connected with it. Similarly, all tasks also have a pull-in arc which is defined as $(i, d(k))$ for all $i \in T$. Finally, there are connection arcs. These arcs are characterized as (i, j), where $i, j \in T$, connecting two task i and j such that $a_i + \delta_i + t_{ij} \leq a_j$. The last parameter in the model is the cost of an arc, which we denote by c_{ij}. This cost is usually equal to the travel costs between tasks i and j, but other costs are also allowed. For instance, if $i = o(k)$ then it is possible that a certain fixed-cost is incurred.

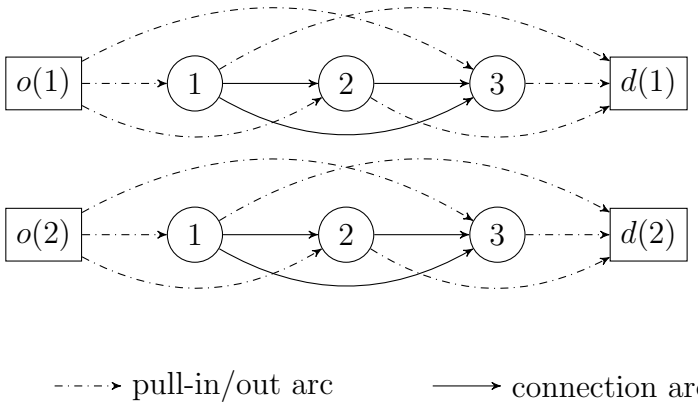


Figure 1: *Example from Table 1 modeled as a connection network. Here, circular vertices represent tasks and rectangular vertices are associated to depots.*

For an example of the connection network approach, see Figure 1, which extends the example from Table 1. Note that every depot has its own network layer and that the cost of arc (i, j) does not necessarily have to be the same between depots.

We are now ready to present the connection network flow formulation as

presented by Pepin et al. (2009) for the MDVSP, which is as follows:

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

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

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

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

$$X_{ij}^k \in \mathbb{B}, \quad \forall (i,j) \in A^k, k \in K, \tag{5}$$

where $\mathbb{B} = \{0, 1\}$, and v_k is the vehicle capacity of depot k. Also, X_{ij}^k are the decision variables of this model which equal 1 if arc $(i, j) \in A^k$ has flow coming from depot k and 0 otherwise. Flow in this context can be seen as a vehicle traveling from task i to task j. Objective function (1) minimizes the total costs. Constraints (2) ensure that every task is performed by exactly one vehicle, while constraints (3) keep the number of dispatched vehicles from depot k below the corresponding capacity. Lastly, constraints (4) conserve the flow through all nodes, as all incoming flow through a certain task vertex must also go out.

5 Overview of heuristics

5.1 Truncated branch-and-cut

When one incorporates cutting planes (see Gomory, 1963) in a brand-and-bound algorithm, the result is known as *branch-and-cut*. As this subject is not in the scope of this text, we refer the interested reader to Mitchell (2002).

As letting the branch-and-cut algorithm run to completion is too time consuming for large problem instances, we could simply terminate the algorithm after the first feasible integer solution we find. This results in a procedure which is called *truncated branch-and-cut*. This procedure is actually a heuristic, as we are not guaranteed that the solution we find is optimal.

For the branch-and-cut implementation we use the connection network formulation, which was discussed in Chapter 4, and we implement it using the commercial MIP solver CPLEX (version 12.6, 32-bits), with default settings unless stated otherwise.

5.2 Lagrangian heuristic

A Lagrangian heuristic is a heuristic which uses a Lagrangian relaxation to obtain lower bounds for the problem at hand. Usually, an upper bound is obtained by "transforming" the lower bound solution to a feasible one. Lagrangian relaxation transfers certain constraints to the objective function by introducing new variables called *Lagrangian multipliers*. The problem of assigning these multipliers values such that the lower bound is as tight as possible is called the *Lagrangian dual problem*. Additionally, the objective function to the Lagrangian dual problem is called the *Lagrangian dual function*. As the Lagrangian dual function is non-smooth, a popular choice in solving the dual problem is subgradient optimization. For a more detailed text on subgradient optimization for the Lagrangian dual problem with applications, we refer the reader to Guta (2003). This section outlines the heuristic proposed by Pepin et al. (2009). In the following, we provide the reader with details regarding every aspect of the Lagrangian heuristic.

5.2.1 Computing the lower bound

As described earlier, the Lagrangian dual problem is obtained when one relaxes certain constraints. However, there is no limit on the choice of which constraints to relax. The Lagrangian heuristic is based on the connections network formulation given by (1)–(5) augmented by the redundant

constraints:

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

The Lagrangian dual problem is then obtained by relaxing constraints (3) by deleting them, and relaxing constraints (4) in a Lagrangian fashion, as follows:

$$\phi(\lambda) = \min \quad \sum_{k \in K} \sum_{(i,j) \in A^k} \tilde{c}_{ij} X_{ij}^k \tag{7}$$

$$s.t. \quad \sum_{k \in K} \sum_{j:(i,j) \in A^k} X_{ij}^k = 1, \quad \forall i \in T, \tag{8}$$

$$\sum_{k \in K} \sum_{j:(j,i) \in A^k} X_{ji}^k = 1, \quad \forall i \in T, \tag{9}$$

$$X_{ij}^k \in \mathbb{B}, \quad \forall (i,j) \in A^k, \forall k \in K, \tag{10}$$

where $\tilde{c_{ij}}$ is defined as

$$\tilde{c}_{ij} = \begin{cases} c_{ij} + \lambda_j^k - \lambda_i^k & \forall i,j \in T \\ c_{ij} + \lambda_j^k & \forall j \in T, i = o(k) \\ c_{ij} - \lambda_i^k & \forall i \in T, j = d(k) \end{cases}. \tag{11}$$

This dual problem is equivalent to a single-depot vehicle scheduling problem (SDVSP), which can be seen by replacing X_{ij}^k with a single variable corresponding to the arc (i,j) for which $c_{ij} + \lambda_j^k - \lambda_i^k$ is the lowest. This SDVSP can be solved in a manner of ways (see Freling et al. 2001), one of which is an auction algorithm.

An auction algorithm in the SDVSP sense is an algorithm in which tasks bid for other tasks or the start/end vertices of the depot. The auction algorithm Freling et al. (2001) proposed for the SDVSP consists of *forward* and *reverse* auction iterations. In a forward iteration, trips bid for successor trips or the end vertex of the depot, and in a reverse iteration trips bid for predecessor trips or the start vertex of the depot. The pseudo-code for both types of iteration can be found in Appendix A.

In the combined forward and reverse auction algorithm as proposed by Freling et al. (2001), one switches between forward and reverse iterations. Freling et al. (2001) also guarantees convergence when one refrains from switching between the iterations until at least one more task is forward or backward assigned. Also, the start values of the dual variable vectors π and p may be chosen arbitrarily.

In theory, the previously described algorithm should always terminate. However, in practice we see that if $\varepsilon < \frac{1}{|T|}$ remains constant during the auction algorithm, it may take many iterations before the algorithm converges. Luckily, one can remedy this by means of *ε-scaling*. ε-scaling is a technique where the original problem is solved several times for decreasing values of ε according to some (monotonically) decreasing function $f(\varepsilon, i)$, where i is the number of the iteration, until finally $\varepsilon < \frac{1}{|T|}$, in which case the solution is optimal. Freling et al. (2001) proposes the following updating scheme. First, for all (i, j) we define new costs $a'_{ij} = |T+1|a_{ij}$. Then, we define the ε-scaling function to be

$$f(\varepsilon, i) = \max\{1, \Delta/\theta^i\}, \tag{12}$$

with $\Delta = |T| \max_{(i,j)} |a'_{ij}|$ and $\theta = 4$. The new optimality criterion for the auction algorithm then becomes $\varepsilon = 1$. The entire auction procedure (including ε-scaling) can be found in Algorithm 1.

Algorithm 1 *The entire auction algorithm for the SDVSP.*

```
1:  procedure AUCTION
2:      ε ← initial value
3:      π ← 0, p ← 0
4:      i ← 1
5:      do_forward ← true
6:      while ε > 1 do
7:          S ← ∅
8:          while ∃i, j ∈ T : (i, j) ∉ S do
9:              if ∃i ∈ T : (i, j) ∉ S and do_forward is true then
10:                 forward(p, π, ε, S)
11:                 if we forward assigned, do_forward is false
12:             end if
13:             if ∃j ∈ T : (i, j) ∉ S and do_forward is false then
14:                 reverse(p, π, ε, S)
15:                 if we backward assigned, do_forward is true
16:             end if
17:         end while
18:         i ← i + 1
19:         ε ← f(ε, i)
20:     end while
21:     return S
22: end procedure
```

5.2.2 Obtaining a feasible solution and an upper bound

The next step in the heuristic is obtaining an upper bound to the original problem. Although several options are possible, usually the solution corresponding to the lower bound is made feasible using some procedure. In our case, we have obtained a number of paths which are assigned to a single depot. By assigning these paths to different depots and keeping in mind the capacity constraints, we obtain a feasible solution for the MDVSP. Formally, we have the following problem:

$$\min \quad \sum_{k \in K} \sum_{p \in P} c_p^k Y_p^k \tag{13}$$

$$s.t. \quad \sum_{k \in K} Y_p^k = 1, \quad \forall p \in P, \tag{14}$$

$$\sum_{p \in P} Y_p^k \leq v_k, \quad \forall k \in K, \tag{15}$$

$$Y_p^k \in \mathbb{B}, \quad \forall p \in P, \forall k \in K, \tag{16}$$

where P is the set of all paths obtained by solving the Lagrangian subproblem, and K is the set of depots. Also, c_p^k is the cost of letting path p be serviced by a vehicle from depot k, and v_k is the capacity for depot k. Furthermore, the decision variable Y_p^k is equal to 1 if path p is serviced from depot k, and 0 otherwise. The objective function (13) minimizes the total costs, constraints (14) make sure that every path is serviced by exactly one depot, and constraints (15) guarantee that the depot capacities are respected.

The problem described by (13)–(16) is called a *transportation problem* and can be solved by a variety of methods. In this thesis, we propose to solve the transportation problem by transforming it into an *asymmetric assignment problem* (AAP), which we solve by means of a combined forward and reverse auction algorithm proposed by Bertsekas and Castañon (1992). More information about the forward/reverse iterations can be found in Appendix B. The transformation is simple, we split every depot $k \in K$ into v_k artificial depots. We denote the set of artificial depots by K', which has

cardinality $|K'| = \sum_{k \in K} v_k$. The resulting AAP then becomes:

$$\min \quad \sum_{k \in K'} \sum_{p \in P} c_p^k Y_p^k \tag{17}$$

$$s.t. \quad \sum_{k \in K'} Y_p^k = 1, \quad \forall p \in P, \tag{18}$$

$$\sum_{p \in P} Y_p^k \leq 1, \quad \forall k \in K', \tag{19}$$

$$Y_p^k \in \mathbb{B}, \quad \forall p \in P, \forall k \in K'. \tag{20}$$

Note that instead of having $|K||P|$ variables using the transportation problem, we now have $\sum_{k \in K} v_k |P| \geq |K||P|$ variables. This shows that the complexity of the auction algorithm (which is dependent on the number of variables) will be heavily influenced by the depot capacities. However, transformations done on the data instances which will be introduced in Chapter 6 showed that the number of paths $|P|$ remains relatively small, and so the transformed transportation problem, even for rather large instances ($|T| = 1500$), is solved swiftly.

As with the SDVSP auction algorithm, it is possible to achieve convergence faster with ε-scaling. We let the scaling function $f(\varepsilon, i)$ be the same as before, where we define Δ as $\Delta = |P| \max_{(p,k)} |a'_{pk}|$ and $a'_{pk} = |P + 1| a_{pk}$. The complete auction procedure can be found in Algorithm 2.

5.2.3 An algorithm for the Lagrangian heuristic

Now for the final ingredient for the Lagrangian heuristic: the subgradient optimization procedure. As already mentioned, the Lagrangian dual function is non-smooth, so optimization by means of gradient descent methods is out of the question. For this reason, one often turns to subgradient optimization, as subgradients are available even for non-differentiable points.

The basic form of a subgradient procedure is

$$\lambda_i^{k,n+1} = \lambda_i^{k,n} + \delta_n s_i^{k,n}, \tag{21}$$

where $\lambda_i^{k,n}$ is the Lagrangian multiplier of iteration n, δ_n is a scalar, and $s_i^{k,n}$ is a subgradient. Note that subgradient optimization comes in many flavors, of which in this thesis we only focus on the one suggested by Pepin et al.

Algorithm 2 *The complete auction procedure for solving the AAP.*

procedure AsymmetricAuction
  $\varepsilon \leftarrow initialvalue$
  $\pi \leftarrow 0, p \leftarrow 0$
  $\lambda \leftarrow 0, i \leftarrow 0$
  while $\varepsilon > 1$ **do** $S \leftarrow \emptyset$
    forward()
    $\lambda \leftarrow \min_{k:(p,k)\in S} p_k$
    do_forward $\leftarrow$ **true**
    while there is an unassigned $p \in P$ and $\exists k \in K' : p_k \leq \lambda$ **do**
      if do_forward is **true then**
        forward(p, π, ε, S, λ)
        if we forward assigned, do_forward is **false**
      end if
      if do_forward is **false** and $\exists k \in K' : p_k \leq \lambda$ **then**
        reverse(p, π, ε, S, λ)
        if we backward assigned, do_forward is **true**
        if all $p \in P$ are assigned, do_forward is **false**
      end if
    end while
    $i \leftarrow i + 1$
    $\varepsilon \leftarrow f(\varepsilon, i)$
  end while
end procedure

(2009), namely:

$$\delta_n = \alpha_n \frac{UB - \phi(\lambda_n)}{\sum_{k \in K} \sum_{i \in T} (s_i^{k,n})^2}, \tag{22}$$

$$s_i^{k,n} = \sum_{j:(j,i) \in A^k} X_{ji}^{k,n} - \sum_{j:(i.j) \in A^k} X_{ij}^{k,n}, \tag{23}$$

where UB is the best upper bound found so far, α_0 is left as a parameter to be chosen by the user, and the rule for updating α_{n+1} given α_n will be given shortly.

The entire Lagrangian heuristic (as suggested by Pepin et al. 2009) is given in Algorithm 3. First, we compute a lower bound $\phi(\lambda_n)$ by solving the Lagrangian subproblem given a vector λ_n of Lagrangian multipliers. Then, we make this (often unfeasible) solution feasible for the MDVSP by transforming it. We do this by means of solving a transportation problem. Then, we update the lower and upper bounds, and the Lagrangian multipliers using a subgradient optimization procedure. Finally, we check if we fulfilled a stopping criterium, and if so, we terminate the heuristic. If not, we begin by computing a new lower bound, and so on. Also, we use the following updating rule for α_n. Every γ iterations in which we have not improved our lower bound, we halve α_n.

The stopping criteria we use are the following. If the upper and lower bounds collide, we stop as we have found the optimal solution. Also, if α_n is too small, say smaller than ε, or after n_{max} iterations, we terminate the procedure.

Algorithm 3 *Lagrangian heuristic for solving the MDVSP.*

procedure LagrangianHeuristic(n_{max}, γ, ε)
  $UB \leftarrow \infty$, $LB \leftarrow -\infty$
  $n \leftarrow 0$, $m \leftarrow 0$, $\lambda_0 \leftarrow 0$
  while true **do**
    solve Lagrangian subproblem (7)–(10) to obtain X_n and $\phi(\lambda_n)$
    compute a subgradient $s_i^{k,n}$
    compute an upper bound UB_n by solving (13)–(16)
    if $UB_n < UB$ **then**
      $UB \leftarrow UB_n$
    end if
    $\lambda_i^{k,n+1} = \lambda_i^{k,n} + \delta_n s_i^{k,n}$
    if $\phi(\lambda_n) > LB$ **then**
      $m \leftarrow 0$
      $LB \leftarrow \phi(\lambda_n)$
    else
      $m \leftarrow m + 1$
    end if
    if $m = \gamma$ **then**
      $\alpha_{n+1} \leftarrow \alpha_n/2$
      $m \leftarrow 0$
    else
      $\alpha_{n+1} \leftarrow \alpha_n$
    end if
    if $UB = LB, \alpha_n \leq \varepsilon$, or $n \geq n_{max}$ **then**
      stop
    else
      $n \leftarrow n + 1$
    end if
  end while
end procedure

6 Computational results

All results were obtained with a computer running Windows 7 (64-bit) with Intel Core i7-4770 CPU at 3.40GHz and 8GB RAM. Additionally, we used Java (version 1.8.0u45, 32-bits) to program (parts of) the heuristics. In case a MIP solver was necessary, we used the commercial MIP solver CPLEX (version 12.6, 32-bits). All user-created code runs on a single thread.

The test instances we used in this thesis to evaluate the heuristics were created by Pepin et al. (2009) and are available from `http://people.few.eur.nl/huisman/instances.htm`. Optimal costs and optimal amount of vehicles can also be found here.

The data vary in two dimensions, namely number of depots and number of tasks. Also, for every depot-task pair, there are five different cases available. Table 2 gives the reader an insight in how big the test instances are. Here, we have summed the number of arcs across all depots.

Table 2: *Number of arcs for the 500, 1000, and 1500 task test instances, for both 4 and 8 depots.*

Instance	500 tasks	1000 tasks	1500 tasks
0	304,620	1,221,640	2,684,272
1	307,728	1,224,572	2,763,400
2	311,772	1,218,576	2,744,656
3	307,676	1,194,052	2,757,136
4	301,868	1,207,288	2,750,936

(a) *4-depot case*

Instance	500 tasks	1000 tasks	1500 tasks
0	615,488	2,435,368	5,453,944
1	624,744	2,477,024	5,459,040
2	620,208	2,412,984	5,486,792
3	611,640	2,422,160	5,514,720
4	615,672	2,389,192	5,534,336

(b) *8-depot case*

6.1 Truncated branch-and-cut

Before we present the results obtained from the truncated branch-and-cut heuristic, we should mention that using the 32-bits version of CPLEX has

severe drawbacks regarding solving the MDVSP. Due to the relatively large amount of memory CPLEX needs for the MDVSP, it is impossible to solve the larger test instances; we were only able to solve the instances with 4 depots and 500 tasks. Also, selecting any algorithm other than the barrier algorithm at the root node lead to memory problems.

Table 3: *Results of the truncated branch-and-cut heuristic for the test instances of category "4 depots, 500 tasks". The optimality gap is calculated using the optimal costs.*

Instance	Gap (%)	Time (s)	Absolute costs
0	49.33	20.45	2,544,110
1	50.44	21.34	2,505,449
2	46.10	22.12	2,381,940
3	50.24	20.74	2,529,575
4	44.30	22.51	2,364,573

The results of the truncated branch-and-cut are shown in Table 3. The reported optimality gap is calculated with

$$Gap = \frac{100(UB - LB)}{UB}. \tag{24}$$

We see from the table that using the first obtained solution results in optimality gaps which are quite large. This can be explained by CPLEX' default behavior. When left default, CPLEX uses heuristics at the root node which give quick solutions. The quality of these quick solutions, however, may turn out be poor, as is likely our case. To investigate this, one could turn off these root node heuristics. Unfortunately, when doing so, we ran into memory problems, which prohibited us to investigate this issue any further.

6.2 Lagrangian heuristic

6.2.1 Sensitivity analysis

As there is quite some freedom in choosing the parameters for the subgradient procedure described in Chapter 5.2.1, we begin with a small sensitivity analysis. As a reference point, we take the following parameter values: $n_{max} = 500$, $\alpha_0 = 1.0$, $\gamma = 10$, $\varepsilon = 0.000001$. Also, due to the fact that sensitivity analysis requires many runs of the algorithm, we limit ourselves to the 4-depot instances with 500 tasks and assume that the larger instances comply in behavior.

We vary the parameters α_0 and γ, which both give us some control over the size of the step we take in the direction of the subgradient. Setting the parameter α_0 relatively large (small) results in taking relatively big (small) steps in the direction of the subgradient, whereas assigning γ a large (small) value leads to relatively slow (fast) decline of the step size. In this sensitivity analysis we consider the values $\alpha_0 = 0.5, 0.8, 1.0, 1.5$ and $\gamma = 1, 3, 5, 7, 10, 15$. The results of which can be found in Figure 2 and Figure 3. From the two figures it is obvious that there exist trade-offs for both parameters, where the trade-off for α_0 is the most apparent. In both cases we find that an increase in solution quality requires an increase in computation time.

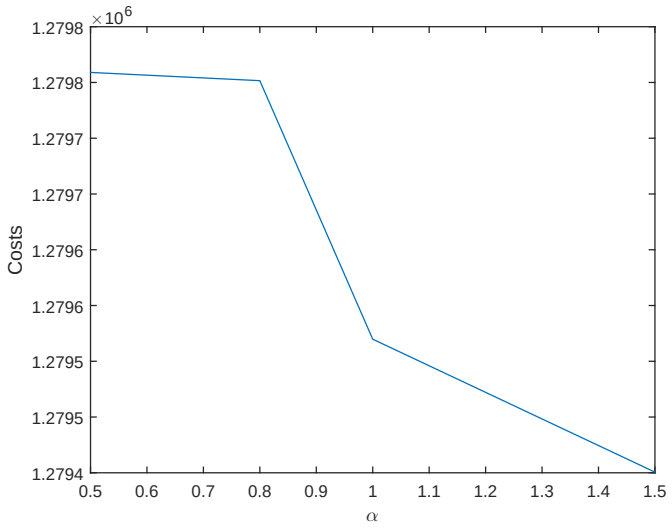
6.2.2 Heuristic results

Considering the results of the sensitivity analysis, we ultimately decide upon the following parameter values: $n_{max} = 500$, $\alpha_0 = 1.0$, $\gamma = 7$, $\varepsilon = 0.000001$. Next, we would like to visually summarize the proposed Lagrangian heuristic by means of Figure 4. In this figure, which was made by applying the heuristic to an instance with four depots and 1500 tasks, we see that the lower bound increases non-monotonically, which is typical for subgradient methods. Nevertheless, the lower bound does seem to converge to a particular value. Also interesting to note is that the upper bound seems to have a tendency to decrease. Small experiments have shown that the quality of the upper bound tends to the quality of the lower bound. Which means that if the lower bound is of poor quality, the upper bound also tends to be poor. Our last observation is that looking at the best available upper bounds, we see that after a certain number of iterations, it is not very rewarding to let the heuristic continue.

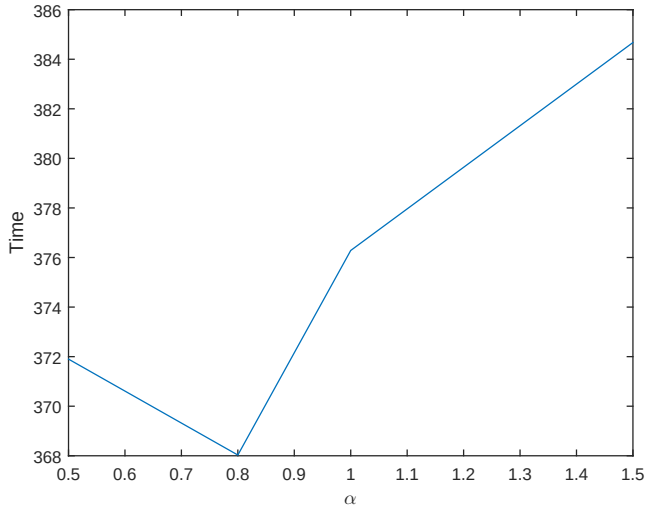
Finally, using the aforementioned parameter values, we run the Lagrangian heuristic for all test instances and compare the results with the heuristics in Pepin et al. (2009). In accordance to Pepin et al. (2009), we subtract the fixed vehicle costs from the objective value and use the result for comparison purposes.

Table 4 and Table 5 show the results of the comparison for the 4-depot and 8-depot instances. The abbreviations for the heuristics are as follows:

- CLGR: Column generation;
- LGRH: Lagrangian heuristic. Note that Pepin et al. (2009) use an unknown method to solve the transportation problem;
- LNS: Large neighborhood search;
- TBnC: Truncated branch-and-cut;

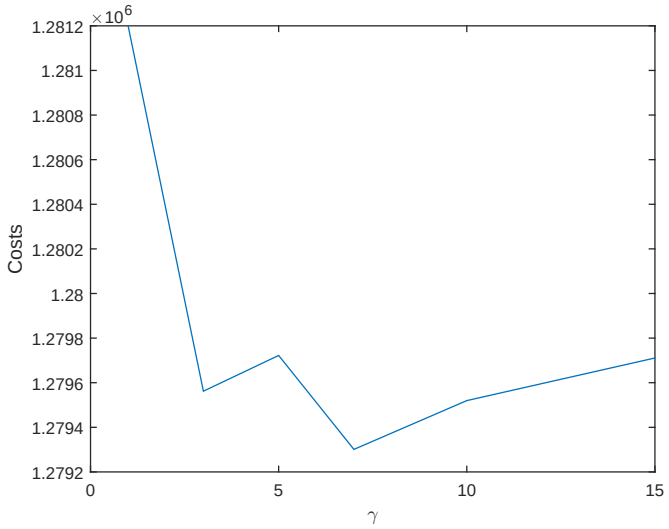


(a)

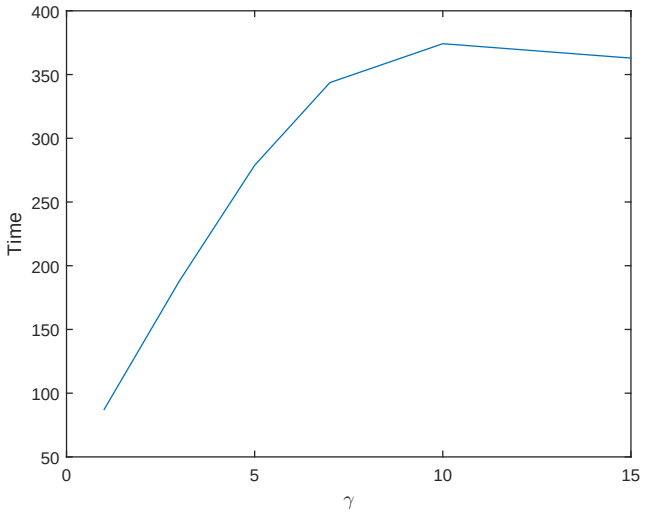


(b)

Figure 2: *Graphs where various values for α_0 are plotted against (a) mean total costs, and (b) mean computation time in seconds. For both graphs, the mean is taken over the five instances with four depots and 500 tasks.*



(a)



(b)

Figure 3: *Graphs where various values for γ are plotted against (a) mean total costs, and (b) mean computation time in seconds. For both graphs, the mean is taken over the five instances with four depots and 500 tasks.*

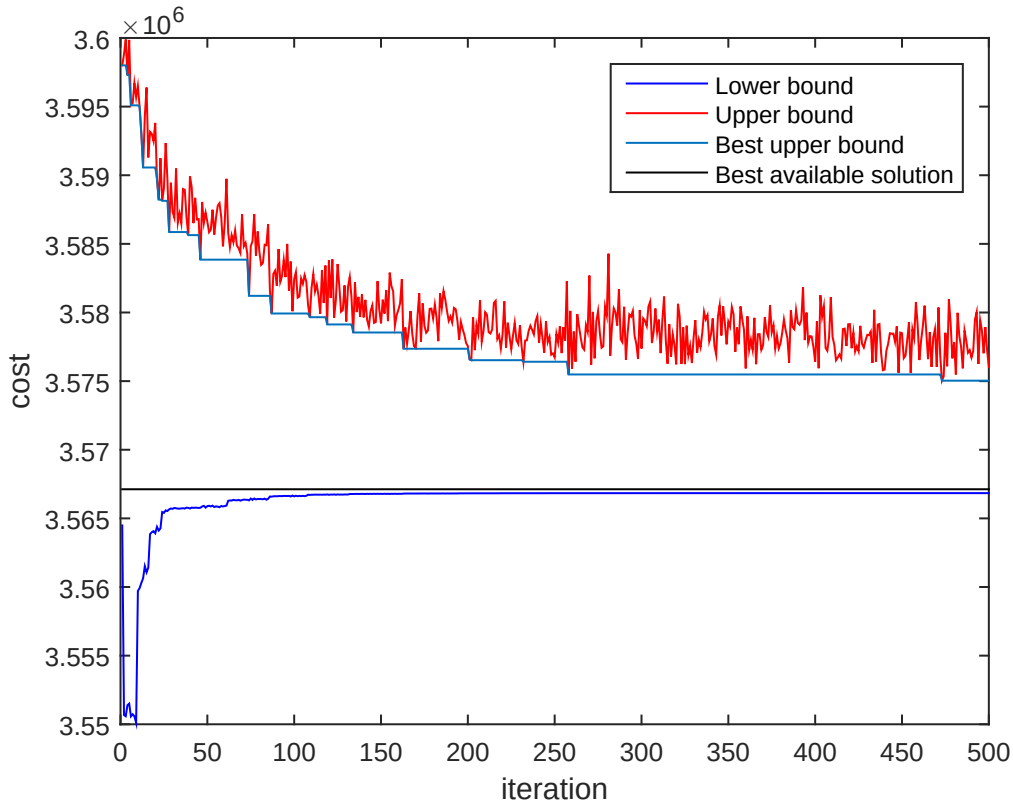


Figure 4: *Typical behavior of the Lagrangian heuristic. This graph was made applying the heuristic on an instance with four depots and 1500 tasks.*

Table 4: *Average best scaled solutions for the 4-depot instances. The best scoring heuristic is used as reference.*

	500 tasks		1000 tasks		1500 tasks	
Heuristic	Time (s)	Objective	Time (s)	Objective	Time (s)	Objective
CLGR	77	1.0000	651	1.0016	2203	1.0019
LGRH*	85	1.0535	700	1.0408	2300	1.0536
LGRH	85	1.0294	700	1.0370	2300	1.0531
LNS	85	1.0201	700	1.0124	2300	1.0182
TBnC	81	1.0013	1287	1.0000	4149	1.0000
TS	85	1.1073	700	1.0812	2300	1.1054

*: Own implementation

Table 5: *Average best scaled solutions for the 8-depot instances. The best scoring heuristic is used as reference.*

	500 tasks		1000 tasks		1500 tasks	
Heuristic	Time (s)	Objective	Time (s)	Objective	Time (s)	Objective
CLGR	119	1.0000	857	1.0091	3085	1.0000
LGRH*	125	1.0770	900	1.0960	3200	1.1021
LGRH	125	1.0494	900	1.0597	3200	1.0923
LNS	125	1.0262	900	1.0265	3200	1.0283
TBnC	612	1.0014	6207	1.0000	–	–
TS	125	1.1722	900	1.1896	3200	1.1981

*: Own implementation

– TS: Tabu search.

The truncated branch-and-cut has no value for the 8-depot cases with 1500 tasks, because it failed to find a solution within 10 hours computation time for some of the instances. We have omitted our implementation of the truncated branch-and-cut heuristic because of the extremely poor performance. For all test instances we have adopted the same run times as Pepin et al. (2009), for fair comparison. As such, for the 4-depot case, our heuristic ran for 85, 125, and 700 seconds for the 500, 100, and 1500 task instances, respectively. For the 8-depot case, these run times are 125, 900, and 3200 seconds.

From Table 4 and Table 5 we can conclude that performance-wise, our implementation of the Lagrangian heuristic is outperformed by all other heuristics, except tabu search. As our implementation has quite different performance as compared to the implementation by Pepin et al. (2009), and the fact that the machine used in this thesis is more powerful, we must conclude that our implementation is poorly optimized.

7 Conclusion

In this thesis we gave a detailed account on two heuristics based on mathematical programming techniques which can be used to solve the MDVSP. Although the truncated branch-and-cut heuristic does not provide acceptable solutions, we think this is caused by the root node heuristics implemented in CPLEX.

For the Lagrangian heuristic, we have thoroughly described methods to solve the Lagrangian dual problem and how to transform a solution of such problem to a feasible solution for the MDVSP. Note that other methods may be used. For instance, one could opt to solve the Lagrangian dual problem with CPLEX, or another method could be used to transform the solution of the dual problem.

Comparing the results of our implementation of the Lagrangian heuristic to heuristics proposed by Pepin et al. (2009), we see that ours is rather lackluster, beating only the tabu search heuristic in terms of solution quality. This lackluster performance is most likely caused by poor optimization of the code. This does not mean that our implementation is not suited for practical issues, as the optimality gap (not subtracting fixed-costs) does not exceed 1% for all test instances considered.

A Forward/reverse auction iterations for the SDVSP

This appendix is dedicated to give a thorough enough description of the forward and reverse auction iterations for implementation purposes. For the validity of the upcoming algorithms we refer the reader to Freling et al. (1997).

First we introduce some terminology. Let the set of all arcs in the SDVSP be A. Let $A(i) = \{j : (i,j) \in A, j \in T \cup d\}$, where d is the end vertex of the depot, be the set of successor vertices of i. Similarly, let $B(j) = \{i : (i,j) \in A, i \in T \cup o\}$, where o is the start vertex of the depot, be the set of predecessor vertices of j. As is custom with auction terminology, we consider the SDVSP as a maximization problem by replacing the costs c_{ij} with $a_{ij} = -c_{ij}$. Also, when $(i,j) \in A$ is included in the solution, we call i and j *forward* and *backward* assigned, respectively. Next, let π and p be vectors of dual variables corresponding to the linear assignment formulation of the SDVSP. In auction terminology π_i is the vector of profits of forward assigning task i and p_j is the price of backward assigning task j. Finally, let $f_{ij} = a_{ij} - p_j$ where i and j are both tasks, and let $f_{id} = a_{id} + \varepsilon$, where $\varepsilon > 0$. Here, f_{ij} represents the amount task i bids for task j. It can be shown that when $\varepsilon < \frac{1}{|T|}$, the auction algorithm terminates with the optimal solution. Pseudo-code for the forward auction iteration can be seen in Algorithm A.1.

For the reverse auction there are some small changes, but the main idea stays the same. Task j now makes a reverse bid $r_{ij} = a_{ij} - \pi_i$ where i is a task, and $r_{oj} = a_{oj} + \varepsilon$ where o is the start vertex of the depot. The pseudo-code for the reverse auction iteration is depicted by Algorithm A.2.

Algorithm A.1 *Algorithm for the forward auction iteration.*

procedure FORWARD(p, π, ε, S)
  for $i \in T : (i,j) \notin S$ **do**
    $j_i \leftarrow \arg\max_{j \in A(i)} f_{ij}$
    $\beta_i \leftarrow f_{ij_i}$
    if $|A(i)| > 1$ **then**
      $\gamma_i \leftarrow \max_{j \in A(i), j \neq j_i} f_{ij}$
    else
      $\gamma_i \leftarrow -\infty$
    end if
    if $j_i \neq d$ **then**
      $p_{j_i} \leftarrow p_{j_i} + \beta_i - \gamma_i + \varepsilon = a_{ij_i} - \gamma_i + \varepsilon$
      $\pi_i \leftarrow a_{ij_i} - p_j$
      $S \leftarrow S \cup (i, j_i)$
      if j_i already assigned, then remove that assignment
    else
      $\pi_i \leftarrow a_{id}$
      $S \leftarrow S \cup (i, d)$
    end if
  end for
end procedure

Algorithm A.2 *Algorithm for the reverse auction iteration.*

procedure REVERSE(p, π, ε, S)
  for $j \in T : (i, j) \notin S$ **do**
    $i_j \leftarrow \arg\max_{i \in B(j)} r_{ij}$
    $\beta_j \leftarrow f_{i_j j}$
    if $|B(j)| > 1$ **then**
      $\gamma_j \leftarrow \max_{i \in B(j), i \neq i_j} r_{ij}$
    else
      $\gamma_j \leftarrow -\infty$
    end if
    if $i_j \neq o$ **then**
      $\pi_{i_j} \leftarrow \pi_{i_j} + \beta_j - \gamma_j + \varepsilon = a_{i_j j} - \gamma_j + \varepsilon$
      $p_j \leftarrow a_{i_j j} - \pi_i$
      $S \leftarrow S \cup (i_j, j)$
      if i_j already assigned, then remove that assignment
    else
      $p_j \leftarrow a_{oj}$
      $S \leftarrow S \cup (o, j)$
    end if
  end for
end procedure

B Forward/reverse auction iterations for the AAP

The auction iterations used in this thesis for the AAP are very much alike the ones in Appendix A. We therefore refer the reader to Appendix A for the relevant notation.

The following forward and reverse auction iterations are proposed by Bertsekas (1998). They are only adapted in terms of notation. Let a forward bid be defined as $f_{pk} = a_{pk} - p_k$ where $a_{pk} = -c_p^k$, p is path, and k is a depot. This means that in this case, paths bid on depots. Similarly, a reverse bid is defined as $r_{pk} = a_{pk} - \pi_p$, where similarly depots bid on paths. Furthermore, we introduce a new scalar, defined after one forward iteration as $\lambda = \min_{k:(p,k)\in S} p_k$. Finally, the optimality criterion in this case becomes $\varepsilon < |P|$. The entire forward and reverse iterations can be found in Algorithm B.1 and Algorithm B.2

Algorithm B.1 *Forward auction iteration for the asymmetric assignment problem.*

procedure FORWARD(p, π, ε, S, λ)
  for $p \in P : (p,k) \notin S$ **do**
    $k_p \leftarrow \arg\max_{k\in A(p)} f_{pk}$
    $\beta_p \leftarrow f_{pk_p}$
    if $|A(p)| > 1$ **then**
      $\gamma_p \leftarrow \max_{k\in A(p),k\neq k_p} f_{pk}$
    else
      $\gamma_p \leftarrow -\infty$
    end if
    $p_k \leftarrow \max\{\lambda, a_{pk_p} - \gamma_p + \varepsilon\}$
    $\pi_p \leftarrow \gamma_p - \varepsilon$
    if $\lambda \leq a_{pk_p} - \gamma_p + \varepsilon$ **then**
      $S \leftarrow S \cup (p, k_p)$
      if k was already assigned, remove that assignment from S
    end if
  end for
end procedure

Algorithm B.2 *Reverse auction iteration for the asymmetric assignment problem.*

procedure REVERSE(p, π, ε, S, λ)
  for $k \in K : (p,k) \notin S, p_k > \lambda$ **do**
    $p_k \leftarrow \arg\max_{p \in B(k)} r_{pk}$
    $\beta_k \leftarrow r_{k_p p}$
    if $|B(k)| > 1$ **then**
      $\gamma_k \leftarrow \max_{p \in B(k), p \neq p_k} r_{pk}$
    else
      $\gamma_k \leftarrow -\infty$
    end if
    if $\lambda \geq \beta_k - \varepsilon$ **then**
      $p_k \leftarrow \lambda$
      continue
    else
      $\delta \leftarrow \min\{\beta_k - \lambda, \beta_k - \gamma_k + \varepsilon\}$
      $p_k \leftarrow \beta_k - \delta$
      $\pi_{p_k} \leftarrow \pi_{p_k} + \delta$
      $S \leftarrow S \cup (p_k, k)$
      if p was already assigned, remove that assignment from S
    end if
  end for
end procedure

References

D.P Bertsekas. *Network Optimization: Continuous and Discrete Models*. Athena Scientific, 1998.

D.P Bertsekas and D.A Castañon. A Forward/Reverse Auction Algorithm for Asymmetric Assignment Problems. *Computational Optimization and Applications*, 1, 1992.

R. Freling, A.P.M. Wagelmans, and J.M.P. Paixão. Models and Algorithms for Vehicle Scheduling. Working paper, 1997.

R. Freling, A.P.M. Wagelmans, and J.M.P. Paixão. Models and Algorithms for Single-Depot Vehicle Scheduling. *Transportation Science*, 35(2), 2001.

R.E. Gomory. An algorithm for integer solutions to linear programs. In R.L Graves and P. Wolfe, editors, *Recent advances in mathematical programming*. McGraw-Hill, 1963.

B. Guta. *Subgradient Optimization Methods in Integer Programming with an Application to a Radiation Therapy*. PhD thesis, Technische Universität Kaiserslautern, 2003.

A. Hadjar, O. Marcotte, and F. Soumis. A Branch-and-Cut Algorithm for the Multiple Depot Vehicle Scheduling Problem. *Operations Research*, 54 (1), 2006.

D. Huisman, R. Freling, and A.P.M. Wagelmans. Multiple-Depot Integrated Vehicle and Crew Scheduling. *Transportation Science*, 39(4), 2005.

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

N. Kliewer, B. Amberg, and B. Amberg. Multiple depot vehicle and crew scheduling with time windows for scheduled trips. *Public Transport*, 3, 2012.

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

A. Löbel. *Optimal Vehicle Scheduling in Public Transit*. PhD thesis, Technische Universität Berlin, 1997.

J.E. Mitchell. Branch-and-Cut Algorithms for Combinatorial Optimization Problems. In P.M. Pardalos and M.G.C. Resende, editors, *Handbook of Applied Optimization*, pages 65–77. Oxford University Press, 2002.

T. Otsuki and K. Aihara. New variable depth local search for multiple depot vehicle scheduling problems. *Journal of Heuristics*, 2014.

A. Pepin, G. Desaulniers, A. Hertz, and D. Huisman. A comparison of five heuristics for the multiple depot vehicle scheduling problem. *Journal of Scheduling*, 12:17–30, 2009.

C.C. Ribeiro and F. Soumis. A column generation approach to the multiple-depot vehicle scheduling problem. *Operations Research*, 42, 1994.