Erasmus University Rotterdam - Erasmus School of Economics
Master Thesis MSc Operations Research & Quantitative Logistics

Dynamic Discretization Discovery for the Multi-Depot Vehicle Scheduling Problem with Trip Shifting

Supervisor: R. N. van Lieshout
Second assessor: D. Huisman

D.T.J. van der Schaft
431738

February 27, 2022

Abstract

The solution of the Multi-Depot Vehicle Scheduling Problem (MDVSP) can often be improved substantially by incorporating Trip Shifting (TS) as a model feature. By allowing departure times to deviate a few minutes from the original timetable, new combinations of trips may be carried out by the same vehicle, thus leading to more efficient scheduling. However, explicit modelling of each potential trip shift quickly causes the problem to get prohibitively large for current solvers. Researchers are obligated to resort to heuristic methods to solve large instances. In this thesis, we develop a Dynamic Discretization Discovery algorithm that guarantees an optimal continuous-time solution to the MDVSP-TS without explicit consideration of all trip shifts. It does so by iteratively solving and refining the problem on a partially time-expanded network until the solution can be converted to a feasible vehicle schedule on the fully time-expanded network. Computational results demonstrate that this algorithm outperforms the explicit modelling approach by a wide margin and is able to solve the MDVSP-TS even when many departure time deviations are considered.

Contents

1 Introduction

Public transportation companies aim to provide service to passengers in a certain region by assigning buses to take these passengers from one place to another. The field of vehicle scheduling deals with optimally combining a predefined set of timetables into a schedule, adhering to the availability of buses at the right moments in time at specified locations. When vehicles can be pulled out of multiple depots, the problem of minimizing operating costs whilst covering every bus trip is called the Multi-Depot Vehicle Scheduling Problem (MDVSP).

Although the provision of timetables beforehand heavily reduces the complexity of the problem, it limits the opportunities to improve the solution quality. Even slight alterations of the starting times of bus trips may allow connections to be made that were impossible at first, potentially enhancing the schedule and reducing operating cost. Allowing the timetables to deviate from its original values is referred to as trip shifting, transforming the problem into a Multi-Depot Vehicle Scheduling Problem with Trip Shifting (MDVSP-TS).

Ideally, we would like to consider all starting times within a certain margin from the timetable. Unfortunately, the more deviations we consider from the original departure time, the larger the set of possible timetables becomes, up to a point where the mathematical model is prohibitively large for current solvers. In practice, heuristic methods are applied to large instances instead at the cost of guaranteed optimality.

A promising exact solution method that may alleviate the reliance on heuristic approaches for the MDVSP-TS is Dynamic Discretization Discovery (DDD; Boland et al., 2017). This recently developed technique for time-dependent problems operates on a partially time-expanded network, overcoming the need to explicitly model all possible trip shifts, yet yielding optimal continuous-time solutions. In an iterative fashion, the outcome of an IP-model based on a partial discretization is used to discover new time points and update the network accordingly. Boland et al. (2017) apply the algorithm to the Service Network Design problem, but as the time-expanded network structure of this problem is prevalent in many transportation problems, they expect that many of the fundamental ideas underlying their approach are widely applicable.

Apart from the theoretical interest of developing a new solution approach for the MDVSP-TS, the computational study conducted by Boland et al. (2017) shows the potential value in practical sense as DDD outperforms existing exact methods for solving the Service Network Design problem. Moreover, after its introduction, DDD has been investigated in the context of fundamental optimization problems such as the shortest path problem (He et al., 2021) and the traveling salesman problem (Vu et al., 2020), time after time proving to be highly effective. Therefore, the goal of this thesis is to apply DDD to the MDVSP-TS and investigate its potential.

The MDVSP-TS will be modelled on a time-space network, as DDD is specifically designed to solve problems defined on a time-expanded network. Moreover, Kliewer, Mellouli, & Suhl (2006) have shown that modelling the MDVSP on a time-expanded network yields a substantial reduction of the problem size, allowing larger instances to be solved compared to any of the state-of-the-art techniques at that point. Ever since, most researchers that investigated the MDVSP have used a time-expanded structure for the problem.

In this thesis, we develop a DDD algorithm that is specifically designed for the MDVSP-TS. A partially time-expanded network is constructed that models all potentially beneficial deviations of a trip as one arc of underestimated length. Then, in iterative fashion, the problem is solved on this network and converted to a feasible solution that adheres to the actual travel times of the arcs. As long as such a conversion cannot be made, we lengthen and duplicate the underestimated arc and restore the network around it. When the solution does not violate travel times anymore, optimality has been reached. Within this solution structure, two refinement strategies and three upper bound heuristics are introduced and compared.

To test the performance of DDD, we conduct a series of computational experiments on instances generated according to the data generation process introduced by Carpaneto et al. (1989). The results show that DDD is able to find an optimal solution to the MDVSP-TS in a much shorter amount of time than the explicit modelling technique that operates on a fully time-expanded network. This allows for a wider range of departure times to be considered for the trips in our problem instances, resulting in a significant reduction in objective value. The performance of DDD is heavily affected by the selected refinement strategy, favoring minimal refinement when only small deviations are considered and aggressive refinement otherwise. Furthermore, even though the running time of DDD increases in the maximum allowed deviation, strong upper bounds can be found early on in the algorithm.

The thesis is organized as follows: Chapter 2 provides an overview of the existing research on the MDVSP-TS and DDD. Then, Chapter 3 gives a description of the MDVSP-TS and dives deeper into the construction of the time-space network, whereas Chapter 4 introduces the DDD algorithm that is developed for the specific context of the MDVSP-TS. The computational results of the algorithm on generated instances are presented in Chapter 5. Finally, conclusions on the performance of DDD are drawn in Chapter 6.

2 Literature Review

2.1 Multi-Depot Vehicle Scheduling and Timetabling

The class of vehicle scheduling problems has been extensively researched for more than 50 years in Operations Research (Bunte & Kliewer, 2009). In short, it revolves around finding a feasible assignment of timetabled trips to vehicles such that each trip is covered once and operating costs are minimized. The consideration of multiple depots where vehicles must start and finish their route through the network heavily increases the complexity of the problem. Whereas multiple polynomial time algorithms have been developed for the single depot case, the problem considering multiple depots is proven to be NP-hard (Bertossi et al., 1987).

Modelling the multiple depot case is generally done in either of the following three ways: by means of a single-commodity, multi-commodity or set partitioning model. First of all, in the single-commodity representation of the problem, both trips and vehicles are represented as nodes. The goal is to construct a set of circuits containing one vehicle node and one or more trip nodes that minimizes the operating costs. Carpaneto et al. (1989) were the first to develop a branch-and-bound algorithm that solves the MDVSP using such a single-commodity approach. Mesquita & Paixão (1992) improved upon the original representation of the problem by aggregating all vehicle nodes corresponding to the same depot to arrive at a more comprehensive network structure and a considerable reduction in the amount of variables and constraints. Secondly, the multi-commodity formulations are based on a layered network where each layer represents the network with regards to one of the depots. A distinction is this approach can be made between so-called connection-based networks and time-space networks. In the former, each possible connection between two compatible trips is explicitly modelled. Solution methods that employ the connection-based representation include a branch-and-bound algorithm by Forbes et al. (1994) and two heuristic approaches by Haghani & Banihashemi (2002) reducing the problem size by imposing fuel consumption constraints. In the time-space network representation, the number of arcs in the network connecting compatible trips can be heavily reduced by aggregation of deadheading opportunities (97% to 99% on real-world instances) without compromising the feasible solution space (Kliewer, Mellouli, & Suhl, 2006). Finally, set partitioning models aim to select a subset of feasible routes such that all trips are covered exactly once at minimal cost. The routes are chosen from a set of feasible routes that is either constructed a priori or gradually by means of column generation. The set partitioning approach was first introduced by Ribeiro & Soumis (1994) who use column generation to iteratively add reduced cost vehicle routes to the mathematical model.

The opportunity to alter departure times of scheduled trips in the MDVSP was first examined by Mingozzi et al. (1995), who employed a branch and bound algorithm to solve a set partitioning formulation of the MDVSP including time windows (resulting in a MDVSP-TW). Desaulniers et al. (1998) used a multi-commodity network flow representation with a non-linear objective function to more realistically interpret waiting costs. Their column generation method was later adopted by Hadjar & Soumis (2009) and improved through tactical reduction of the width of the time windows. Note that the notion of time windows in all these papers is different from the concept of trip shifting, since the latter reduces the choice of departure times to a discrete set of deviations. Although the terms are often used interchangeably, Kliewer, Bunte, & Suhl (2006) were the first to consider trip shifting in the time-space network formulation of the MDVSP - we will refer to this as the MDVSP-TS.

Extending the MDVSP with time windows or trip shifting for all trips quickly deems exact solution methods intractable for medium- to large-sized instances. Therefore, heuristic approaches have been developed to deal with alterations to the departure time. Examples include heuristics that construct a small set of trip shifts with high cost saving potential (Kliewer, Bunte, & Suhl, 2006) and a two-phase matheuristic that builds bus timetables in the first stage and constructs an optimal schedule given these timetables thereafter (Desfontaines & Desaulniers, 2018). Despite the computational efficiency of such methods, the researchers of the latter underline the relevance of integrating both trip shifting and vehicle scheduling into an iterative algorithm.

Van den Heuvel et al. (2008) have investigated the integration of timetabling and bus scheduling and proposed a simulated annealing heuristic, under the condition that the timetable remains clock-face. That is, the timetable can only be changed by shifting the departure time on an entire line forward or backward with the same number of minutes. Ceder (2011) developed a similar method that safeguards the presence of even headways and even loads for vehicles in the optimal timetable. Petersen et al. (2013) considered the quality of the timetable from a passenger's perspective by partly evaluating solutions based on the waiting time between different lines in a large neighbourhood search algorithm. Schmid & Ehmke (2015) employed a similar approach and decomposed the problem into a scheduling and a balancing component. They applied large neighbourhood search to the scheduling phase of the problem, whereas departure times are balanced through solving a linear program. The importance of timetable quality from the viewpoint of the passenger has been further integrated by Laporte et al. (2017) and Liu & Ceder (2017) by taking passenger connections into account besides timetabling and vehicle scheduling. Both studies obtain a set of Pareto-efficient solutions through an ϵ-constraint method and a deficit function-based sequential search method, respectively. Fonseca et al. (2018) developed a matheuristic that iteratively adds modified departure times to the integrated timetabling and vehicle scheduling problem by either one of four selection strategies, allowing a wider set of timetable adjustments to be considered.

2.2 Dynamic Discretization Discovery

The DDD algorithm by Boland et al. (2017) has originally been developed for the Continuous-Time Service Network Design problem, in which minimum-cost paths are constructed in both space and time for the shipments of goods and the resources necessary to execute these shipments. Because a full discretization of the problem (i.e. taking all possible dispatch times into consideration) quickly causes the network to grow too large to solve the problem in a reasonable amount of time, coarser discretizations are commonly used to ensure the computational tractability. Boland et al. (2017) were the first to prove that an optimal continuous-time solution can be found without full discretization, but through iterative refinement of a partially time-expanded network. In such a network, only those parts are fully discretized where optimality cannot be guaranteed based on a coarser degree of discretization.

Within the area of Service Network Design, Scherr et al. (2020) extended the model to the Service Network Design problem with Mixed Autonomous Fleets by adding additional properties to the partially time-expanded network, allowing themselves to manage the flow of vehicles more precisely. Moreover, they improve upon the algorithm by strengthening lower bounds through the use of valid inequalities and by introducing heuristics methods to find strong upper bounds more easily. Marshall et al. (2021) introduced an Interval-based DDD approach where the nodes in the time-expanded network represent a certain location within a time interval instead of at a specific time. The resulting solution structure is exploited in the refinement stage of the algorithm, leading to fewer iterations and smaller partially time-expanded networks. Consequently, larger instances can be solved and high-quality solution are found more quickly than in the original DDD.

The application of DDD has been extended to other types of transportation problems as well. Hewitt (2019) adapted and enhanced the algorithm for the Service Network Design problem to fit its less-than-truckload freight transportation analogue: the Continuous-Time Load Plan Design Problem. He proposed a two-phase algorithm that solves a linear relaxation in the first phase, after which a set of arc-time window inequalities enables the definition of feasible time windows for the transportation of commodities. Vu et al. (2020) solved the Time Dependent Traveling Salesman Problem with Time Windows by means of DDD. They showed that it outperforms even the strongest algorithms in existing literature and that it is robust to all instance parameters. He et al. (2021) applied DDD to the Minimum Duration Time-Dependent Shortest Path problem with piecewise linear travel times and find that the number of explored breakpoints drastically reduces, making the algorithm highly efficient and scalable.

3 Multi-Depot Vehicle Scheduling Problem with Trip Shifting

The Multi-Depot Vehicle Scheduling Problem considers a set of scheduled trips M, where each trip $m \in M$ departs from a location l^s_m at time t^s_m and arrives at a location l^e_m at time t^e_m. The aim is to optimally route a number of vehicles through a network such that all trips are covered at minimum cost. Every single vehicle route must start its route at the beginning of the time horizon at one of the depots $d \in D$ and is obligated to return to the exact same depot before the end of the time horizon. Through the introduction of trip shifting to the MDVSP, trips are allowed to depart at most δ^{max} earlier or later than originally timetabled.

3.1 Time-space network representation

The network consists of a separate layer for each depot $d \in D$. The full network $G = (N, A)$ is thus the union of $|D|$ individual network layers $G^d = (N^d, A^d)$, where N^d and A^d are the set of nodes and arcs in the network layer for depot $d \in D$, respectively. A node (l, t) represents a location $l \in L$, which is either a station or a depot, at a certain point in time t. The travel time between two locations $l, k \in L$ is denoted by tt_{lk}. The nodes in a network layer are connected through several types of arcs, namely:

- **Trip arcs** represent timetabled trips $m \in M$ and their possible deviations $\delta \in \Delta = \{-\delta^{max}, -\delta^{max}+1, \dots, \delta^{max}-1, \delta^{max}\}$, and connect node $(l^s_m, t^s_m + \delta)$ corresponding to the start location and time of the trip with node $(l^e_m, t^e_m + \delta)$ corresponding to the end location and time. In order to cover a trip, either one of its trip arcs must be traversed. The set of all trip arcs, including those with a deviated departure time, is denoted by $\bar{A}^d \subset A^d$. Set $\bar{A}^d(m) \subset \bar{A}^d$ in its turn contains the trip arcs belonging to trip $m \in M$.

- **Deadheading arcs** represent empty moves of vehicles between stations and connect the end station of one trip with the start station of another.

- **Pull-out and pull-in arcs** represent empty moves of vehicles in and out of the depot and connect the depot to the start station of a trip in case of a pull-out, or from the end station of a trip back to the depot in case of a pull-in.

- **Waiting arcs** represent vehicles standing still and waiting at the depot or a station and connect every node at a location to the consecutive node of that location in time.

The foundation of the mathematical model for the MDVSP-TS is based on the multi-commodity time-space network flow formulation proposed by Kliewer, Mellouli, & Suhl (2006) for the MDVSP. As this formulation does not consider trip shifting, we incoporate this feature in a similar fashion as Kliewer et al. (2012) did in their model for the integrated vehicle and crew scheduling problem with multiple depots and consideration of time windows for scheduled trips.

Decision variables x_{ij}^d denote the number of vehicles traversing arc $(i,j) \in A^d$ belonging to depot $d \in D$. The nature of x_{ij}^d is binary for trip arcs and their deviations (which are grouped in the set $\bar{A}^d \subset A^d$), whereas it can take on any non-negative integer value for any other arc $(i,j) \in A^d \setminus \bar{A}^d$. Altogether, the MDVSP-TW can be formulated as follows:

$$\min \quad \sum_{d \in D} \sum_{(i,j) \in A^d} c_{ij}^d x_{ij}^d \tag{1}$$

$$\sum_{j:(j,i) \in A^d} x_{ji}^d - \sum_{j:(i,j) \in A^d} x_{ij}^d = 0 \qquad \forall i \in N^d \setminus \{i_0^d, i_{end}^d\}, \forall d \in D, \tag{2}$$

$$\sum_{d \in D} \sum_{(i,j) \in \bar{A}^d(m)} x_{ij}^d = 1 \qquad \forall m \in M \tag{3}$$

$$x_{ij}^d \in \mathbb{B} \qquad \forall (i,j) \in \bar{A}^d, \forall d \in D, \tag{4}$$

$$x_{ij}^d \in \mathbb{Z}^* \qquad \forall (i,j) \in A^d \setminus \bar{A}^d, \forall d \in D, \tag{5}$$

The objective function (1) minimizes the total operating cost of the vehicles in the network. Flow conservation is ensured by constraints (2), whereas cover constraints (3) make sure that every trip is covered by exactly one vehicle by using one of the trip arcs in $\bar{A}^d(m)$.

3.2 Trip arc duplication

The idea behind trip shifting is that allowing the departure time to deviate slightly might enable new connections between trips that were not feasible given the original timetable. However, it is unnecessary to explicitly model a copy of a trip arc for a deviation $\delta \in \Delta$ that does not introduce a new connection, especially given the benefit of minimizing the size of the network. Therefore, we investigate each pair of trips (i,j) where $tt_{l_i^e l_j^s} - 2\delta^{max} \leq t_j^s - t_i^e \leq tt_{l_i^e l_j^s} + 2\delta^{max}$. In other words, these are the pairs of trips that are not guaranteed to connect regardless of the deviation of either one of them, but do have the potential to connect. Only for those pairs it may be beneficial to deviate departure times of one or both trips. In determining which deviations to incorporate in the model, two separate cases are considered sequentially:

- Case 1: $t_j^s - t_i^e < tt_{l_i^e l_j^s}$. There is no connection between trips i and j given the original timetable. We check for each pair of deviations (δ_i, δ_j) if it causes $t_j^s - t_i^e$ to be larger than $tt_{l_i^e l_j^s}$ If it does, we copy the trip arcs correspondingly. Figure 1 contains an example where two trips cannot be connected given their original departure time - trip 1 arrives at $t = 3$ and trip 2 departs at $t = 6$ with a travel time of 4 between the respective locations.

However, advancing the departure time of the first trip by 1 or delaying the departure time of the second trip by 1 would yield a feasible connection. Hence, $\delta_1 = -1$ and $\delta_2 = 1$ are considered as potentially beneficial trip shifts and trip arcs are duplicated accordingly.

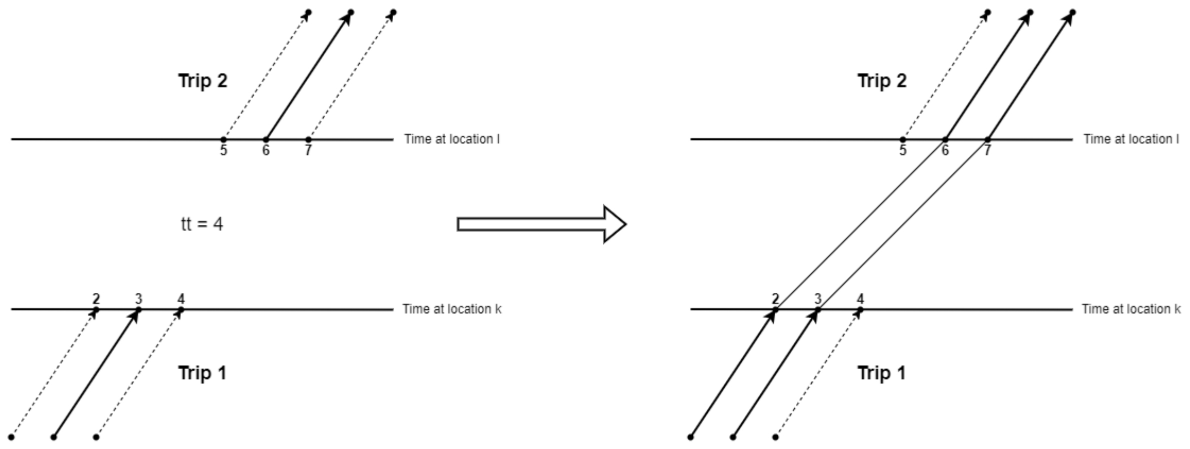

Figure 1: *Example of trip arc duplication for case 1. Trip arcs are represented by black arrows, whereas potential trip shifts are depicted as dotted arrows.*

- Case 2: $t_j^s - t_i^e \geq tt_{l_i^e l_j^s}$. Although trips i and j can be connected if they both depart at their scheduled time, either delaying the first trip or advancing the second trip may lead to the disappearance of a feasible connection between the new departure time of the deviated trip with the originally timetabled departure time of the other. It must therefore be checked whether any deviations are included in the model for either of the two trips that cause infeasibility. If this is the case, a new deviation must be included that ensures a connection between i and j for all deviations. An example is included in Figure 2. The travel time between the arrival station of trip 1 and the departure station of trip 2 can exactly be covered when adhering to the original timetable, but does not allow a connection if $\delta_1 = 1$ is modelled. To ensure the possibility of executing trip 1 and 2 consecutively despite traversing the delayed arc, we must add the trip arc corresponding to $\delta_2 = 1$.

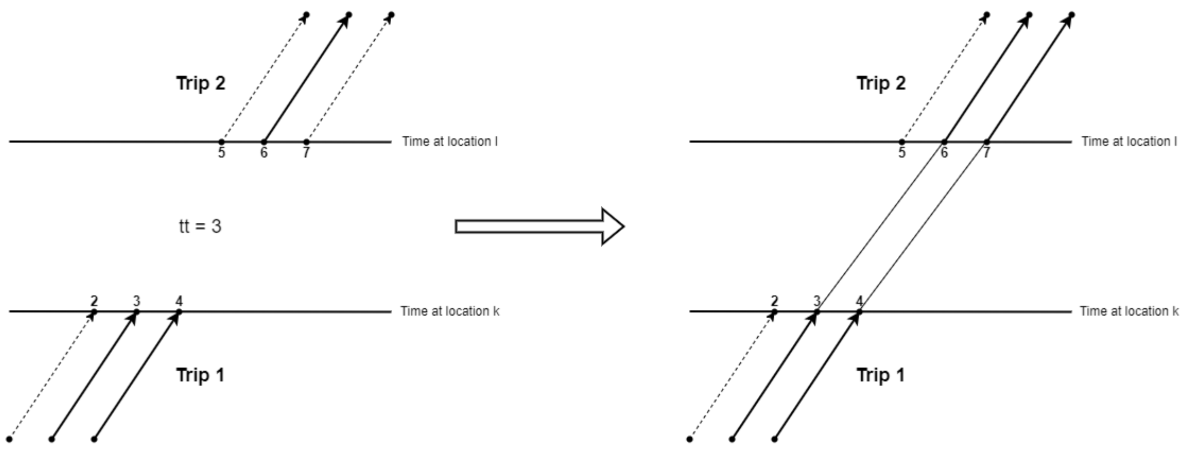

Figure 2: *Example of trip arc duplication for case 2. Trip arcs are represented by black arrows, whereas potential trip shifts are depicted as dotted arrows.*

Note that if a pair of deviations (δ_i, δ_j) does not result in a feasible connection, the deviations may still be included in the model if they enable connections with other trips.

3.3 Deadheading arc aggregation

Key to the ability of the time-space representation to perform well on large instances is the aggregation of deadheading arcs in the network (Kliewer, Mellouli, & Suhl, 2006). Instead of explicitly modelling any possible connection between two trips at any point in time, arcs that are redundant are removed. By aggregating all arcs that enable two trips to be executed consecutively, the size of the network and the corresponding number of variables in the mathematical model is heavily reduced without compromising the feasible region of solutions. Aggregation is performed by means of the following two-step procedure:

First stage aggregation. For each timetabled trip m arriving at location k, the earliest compatible trip is considered at each location l other than k. Then, a deadheading arc is constructed to the so-called *first-match* of trip m with location l, departing from the end node of trip m and arriving at the starting node of the first-match. All later matches of trip m are disregarded, as they do not enable new connections to be made. Trips at location l, $l \neq k$ that are not first-matches can be reached by consecutively traversing the first-match arc to l and the waiting arcs at l.

Second stage aggregation. The set of first-match deadheading arcs is further reduced in the second stage of the aggregation process. Consider a trip m at location k and let F_l be the set of trips arriving at any location l, $l \neq k$ that consider trip m as their first-match at location k. Out of set F_l, only the trip with the latest arrival time keeps its direct deadheading connection with trip m, which we call the *latest-first-match*. All the other arcs are removed from the model as their connection can be replaced by traversing waiting arcs at location l combined with the latest-first-match arc.

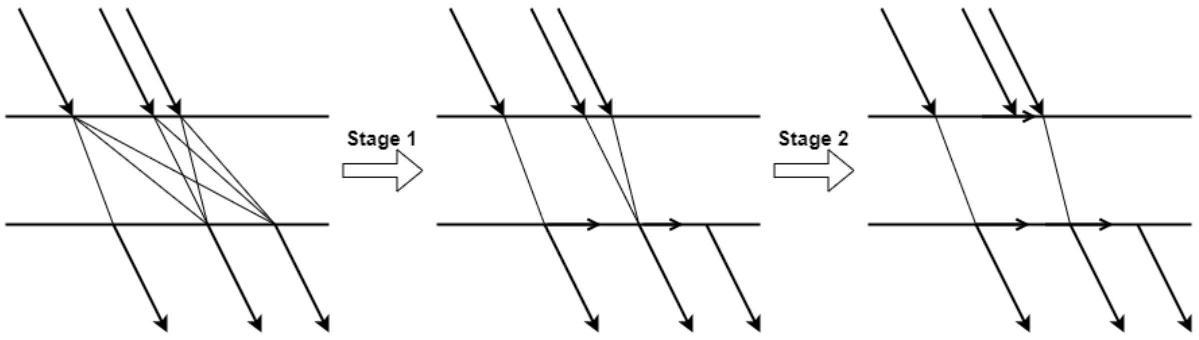

Figure 3: *Network reduction through the two-stage aggregation process.*

Figure 3 shows how deadheading arcs are aggregated between two locations in the network. At first, all feasible deadheading possibilities from the location with incoming trips to the location with outgoing trips are modelled explicitly. After the first stage, only deadheading arcs to the *first-matches* of each of these incoming trips remain. In the second stage of aggregation, we switch our attention to the outgoing trips. Whenever an outgoing trip has multiple incoming *first-match* arcs from the same location, all but the *latest-first-match* are deleted.

4 Dynamic Discretization Discovery

DDD is an iterative refinement algorithm introduced by Boland et al. (2017) that uses a partially time-expanded network as a starting point. Algorithm 1 gives a high-level overview of the procedure. The partially time-expanded network serves as an optimistic representation of the problem. It does not explicitly model each potential trip shift, but underestimates the length of trip arcs instead to decrease the number of nodes and arcs in the network. In each iteration, the problem is solved on the relaxed network, yielding a lower bound to the optimal objective value. If the solution can be converted to a feasible solution on the full network with the same cost, an optimal solution for the MDVSP-TS has been obtained. Otherwise, the partially time-expanded network is refined and a new iteration of the algorithm is entered. Chapter 4.1 - 4.3 dive deeper into the implementation of each of the steps within the algorithm.

Algorithm 1 Dynamic Discretization Discovery

```
Require: Time-space network $G = (N, A)$
 1: Create a partially time-expanded network $G_T$
 2: while not solved do
 3:     Solve MDVSP-TS($G_T$)
 4:     Convert the solution on the partial network $G_T$ to a feasible solution on the full
 5:     network $G$
 6:     if optimality gap $< \epsilon\%$ then
 7:         Terminate, optimal solution found
 8:     end if
 9:     Refine $G_T$ by lengthening and duplicating the arcs corresponding to at least one trip
10: end while
```

4.1 Creating a partially time-expanded network G_T

A partially time-expanded network can be regarded as a relaxation of the full network and aims to find a valid lower bound to the MDVSP-TS. Using such a partially time-expanded network within the DDD framework requires proving the following proposition:

Proposition 1. *The optimal objective value $z_T^\star$ of a solution to the model on the partially time-expanded network G_T is always less than or equal to the optimal objective value $z^\star$ of the full network.*

The proposition can be proven by showing that any feasible vehicle route in the full network can be converted to a feasible vehicle route in the partially time-expanded network at the same or lower cost. Careful construction of the initial partially time-expanded network is key and can be done as described in Algorithm 2.

Algorithm 2 CREATE-PARTIAL-NETWORK

```
Require: Fully time-expanded network $G = (N, A)$
1: for $\forall d \in D$ do
2:     Add nodes $(d, 0)$ and $(d, t^{end})$ to $N_T$
3:     for $\forall m \in M$ do
4:         Add nodes $(l^s_m, t^s_m + \delta^{max})$ and $(l^e_m, t^e_m - \delta^{max})$ to $N_T$
5:         Add trip arc $((l^s_m, t^s_m + \delta^{max}), (l^e_m, t^e_m - \delta^{max}))$ to $A_T$
6:         Add pull-out arc $((d, 0), (l^s_m, t^s_m + \delta^{max}))$ and pull-in arc $((l^e_m, t^e_m - \delta^{max}), (d, t^{end}))$ to $A_T$
7:         Add deadheading arcs to $A_T$ according to the aggregation procedure of Chapter 3.3
8:     end for
9: end for
```

To ensure that not only the timetabled trips themselves, but also their potential shifts are incorporated in the partially time-expanded network, the nodes $(l^s_m, t^s_m + \delta^{max})$ and $(l^e_m, t^e_m - \delta^{max})$ are created for each trip $m \in M$, representing the latest departure time and earliest arrival time. This way, a vehicle route traversing any deviation of a trip arc in the fully time-expanded network can be converted to a feasible solution in the partially time-expanded network, simply by traversing the underestimated trip arc instead. Using the underestimation, none of the potential connections are neglected as in reality, the trip can never depart later or arrive earlier than in the relaxation. As any route in the fully time-expanded network can be converted to a solution in the partially time-expanded network in this manner, the partial objective value z^*_T can never be higher than the full objective value z^*.

For each network layer, a pull-out arc connects each starting node to the starting depot node and a pull-in arc connects each ending node to the ending depot node. Moreover, waiting arcs are created for every location to connect nodes at the same station or depot throughout the time horizon. In addition, deadheading arcs are added to the network following the aggregation process described in Chapter 3.3, considering the nodes $(l^s_m, t^s_m + \delta^{max})$ and $(l^e_m, t^e_m - \delta^{max})$ as the departure and arrival node of each trip m. Note that the partially time-expanded network is identical to the fully time-expanded network when $\delta^{max} = 0$.

4.2 Converting the solution to MDVSP-TS(G)

The solution to the MDVSP-TS on the partially time-expanded network does not guarantee feasibility on the full network, as vehicle routes may consist of one or more arcs with underestimated length. Such an arc may connect two trips that cannot be executed consecutively when actual travel times are considered. However, the presence of underestimated arcs does not automatically deem a schedule infeasible. For example, if an arc that is 10 minutes too short is preceded by a waiting arc of at least that length, a feasible solution can be constructed by simply departing earlier.

It can easily be checked whether such alterations to the schedule can be made to ensure feasibility, by going through each vehicle route in the solution and picking the earliest possible departure time for each arc that respects actual travel times. Let set R contain all arcs of a specific vehicle route in visiting order in the current solution on the partially time-expanded network. The departure time of arc $i \in R$ is denoted by π_i. Commencing at the depot at $t = 0$, the departure time of an arc i is computed as $\pi_i = \max\{\pi_{i-1} + tt_{i-1,i}, t^s_{m(i)} - \delta^{max}\}$ when i is a trip arc, and $\pi_i = \pi_{i-1} + tt_{i-1,i}$ otherwise. If $\pi_i > t^s_{m(i)} + \delta^{max}$ for any of the trip arcs, the vehicle route is infeasible. Chapter 4.3 explains how to identify which trip arc must be refined in the partially time-expanded network to avoid running into similar feasibility issues in further iterations of DDD.

In case the vehicle route turns out to be feasible, a new mathematical model will be solved to minimize deviations from the original timetable. Given the set of locations that are visited on the vehicle routes that are selected in the optimal solution, this model chooses departure times at the visited locations such that all trips on the route can be executed within the prespecified time window whilst adhering to the actual travel times between locations.

Alongside the departure time variable π_i, decision variables δ_i^+ and δ_i^- are introduced, representing the potential delay or advancement of the departure, respectively. Both δ_i^+ and δ_i^- are integer-valued and range from 0 to δ^{max}. If i is a trip arc (i.e. $i \in \bar{R} \subset R$), π_i is set equal to the sum of the original departure time of the corresponding trip $m(i)$ (i.e. $t^s_{m(i)}$), the delay δ_i^+ and the advancement -δ_i^-. Otherwise, it can take on any non-negative value, as long as actual travel times are not violated. This results in the following formulation:

$$
\begin{aligned}
\min \quad & \sum_{i\in\bar{R}} c_p(\delta_i^+ + \delta_i^-) && && (6)\\
\text{s.t.} \quad & tt_{i,i+1} + \pi_i \leq \pi_{i+1} && \forall i \in R \setminus \{r^{last}\} && (7)\\
& \pi_i = t^s_{m(i)} + \delta_i^+ - \delta_i^- && \forall i \in \bar{R} && (8)\\
& \delta_i^+, \delta_i^- \in \{0, 1, \dots, \delta^{max}\} && \forall i \in \bar{R} && (9)\\
& \pi_i \geq 0 && \forall i \in R \setminus \bar{R} && (10)\\
& \delta_i^+, \delta_i^- = 0 && \forall i \in R \setminus \bar{R} && (11)
\end{aligned}
$$

The objective function (6) minimizes the sum of deviations from the timetable. A penalty cost c_p is incurred for every minute of deviation from the scheduled trip departure times to avoid unnecessary alterations to the timetable. For every set of consecutive locations on the vehicle routes, the feasibility of the routes is protected by the travel time constraints (7). If location j is visited after location i, the departure time at j has to be larger than or equal to the sum of the departure time at location i and the travel time (or trip execution time) between i and j.

4.2.1 Upper bound computation

Whenever a trip on a vehicle route in the partially time-expanded network solution cannot be executed respecting actual travel time and time window constraints, the solution as a whole is infeasible. However, by focusing only on the infeasible aspects of the solution, we can often convert it to a feasible solution relatively quickly. Three techniques that find a feasible upper bound to such infeasible routes are developed: a cutting heuristic and two matheuristics.

- The purpose of the **cutting heuristic** is to find an upper bound in a split second. Anytime the departure station of a trip cannot be reached in time, the route is cut in two right after the previous trip. The first half of the route is then extended by a pull-in arc to the depot, whereas the second half starts off with a new pull-out arc before continuing its route through the network from the bottleneck trip onward. Each cut requires the use of an additional vehicle, as the infeasible route was originally assigned to a single vehicle. This process is repeated until there are no infeasible connections left. The result is a set of feasible routes, yielding an upper bound to the optimal objective value.

- The second approach is a **multi-depot matheuristic** that re-optimises the infeasible part of the solution. A time-expanded network with $\delta^{max} = 0$ is initialised as outlined in Chapter 3 using the set of trips executed on infeasible vehicle routes as its input. The lower the cardinality of this set, the smaller the size of the newly created network. To speed up the upper bound computation, trip shifting is disregarded. After initialisation, a MDVSP is solved, providing a feasible vehicle schedule covering all trips previously covered on infeasible routes. Combining this re-optimised schedule with part of the solution that was feasible in the first place provides us with a stronger upper bound than the cutting heuristic does.

- The idea of the **single-depot matheuristic** is identical to that of its multi-depot counterpart, except that the network on which the problem is solved only consists of one layer for the depot that is on average the closest in travel time to the infeasible trips. The problem now boils down to an ordinary Vehicle Scheduling Problem, whose absence of multiple depot layers leads to a much smaller model than that of the MDVSP for the same problem instance.

Whereas the cutting heuristic is guaranteed to find an upper bound with minimal computational time, the speed of the matheuristic heavily depends on the degree of infeasibility of the solution. It can be expected to perform relatively well for smaller values of δ^{max} as the possibility to construct infeasible routes in the partially time-expanded network is limited, and in later iterations of the problem where most of the underestimated trip arcs that cause infeasibility have already been refined.

4.3 Refining the partially time-expanded network G_T

When the optimality gap between the solution on the partially time-expanded network and the upper bound solution is not satisfactorily small, the network must be refined for the next iteration. In Chapter 4.2 we have established how to identify which trips on a route cannot be executed in time. In order to avoid that the same infeasible connections to these trips return in later iterations of DDD, they must be infeasible in the partially time-expanded network as well. This can be done by lengthening the last underestimated trip arc prior to the bottleneck trip. We know that such an arc exists, since a sequence of trip arcs with the correct length will never cause infeasibility. Given the set of trip arcs that should be lengthened, we deploy two different refinement strategies:

- The first is an **aggressive refinement** strategy, where all arcs that require refinement are lengthened to their actual travel times. Whenever a trip arc corresponding to trip $m \in M$ is too short, Algorithm 3 is executed for each depot $d \in D$. The underestimated trip arc is replaced by trip arcs with the correct length. Following the technique described in Chapter 3.2, only those time-shifted arcs are added that enable new connections between trips that could not be made using the original arc (the corresponding deviations δ are contained in set Δ_m). Figure 4 provides an example of aggressive refinement.

Algorithm 3 AGGRESSIVE-REFINEMENT

```
Require: Trip $m \in M$ and depot $d \in D$
1: Remove trip arc $((l_m^s, t_m^s + \delta^{max}), (l_m^e, t_m^e - \delta^{max}))$ and corresponding pull-out and pull-in arc from $A_T$
2: for $\forall \delta \in \Delta_m$ do
3:     Add nodes $(l_m^s, t_m^s + \delta)$ and $(l_m^e, t_m^e + \delta)$ to $N_T$
4:     Add trip arc $((l_m^s, t_m^s + \delta), (l_m^e, t_m^e + \delta))$ and corresponding pull-out and pull-in arc to $A_T$
5:     RESTORE-DESTINATION$(l_m^e, t_m^e + \delta)$ and RESTORE-ORIGIN$(l_m^s, t_m^s + \delta)$
6: end for
```

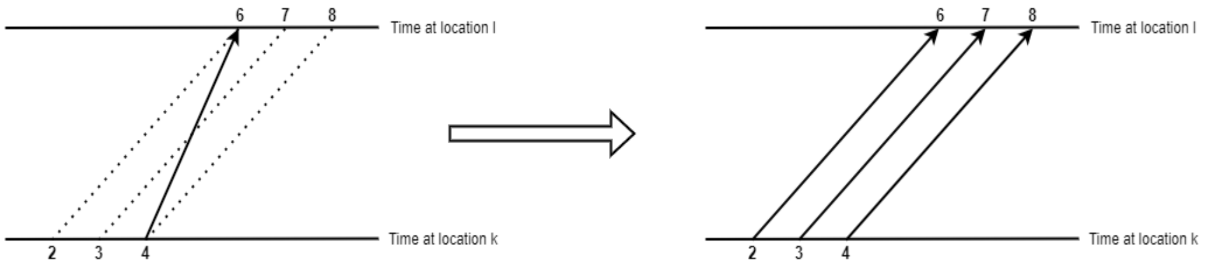

Figure 4: *Example of aggressive refinement for a trip departing from location k at $t = 3$ and arriving at location l at $t = 7$ with $\delta^{max} = 1$. The original underestimated trip arc is replaced by trip arcs of the correct length.*

- The second strategy is one of **minimal refinement**, which refines arcs to the minimal length that avoids the sequence of trips that causes infeasibility. Let ω be the degree of refinement, i.e. the minimal number of minutes that a trip arc should be lengthened by. Consider the subroute starting at the trip arc that must be lengthened, up to and including the last arc before the bottleneck occurs. By taking the difference between the combined length of the subroute's arcs and the amount of time between its departure and the bottleneck trip arc's departure, we can find the maximal lengthening that still allows a feasible connection in the partially time-expanded network. The value of ω can then be computed by adding 1 minute. Once ω is determined, Algorithm 4 is deployed for each depot $d \in D$. An example of minimal refinement is included in Figure 5.

Algorithm 4 Minimal-Refinement

```
Require: Trip $m \in M$, depot $d \in D$ and refinement degree $\omega$
1: Remove all trip arcs in $\bar{A}_T^d(m)$ and corresponding pull-out and pull-in arcs from $A_T$
2: for $\forall i \in \{0, \ldots, \omega\}$ do
3:     Add nodes $(l_m^s, t_m^s + \delta^{max} - i)$ and $(l_m^e, t_m^e - \delta^{max} + \omega - i)$ to $N_T$
4:     Add trip arc $((l_m^s, t_m^s + \delta^{max} - i), (l_m^e, t_m^e - \delta^{max} + \omega - i))$ and corresponding pull-out
       and pull-in arc to $A_T$
5:     Restore-Destination$(l_m^e, t_m^e - \delta^{max} + \omega - i)$ and Restore-Origin$(l_m^s, t_m^s + \delta^{max} - i)$
6: end for
```

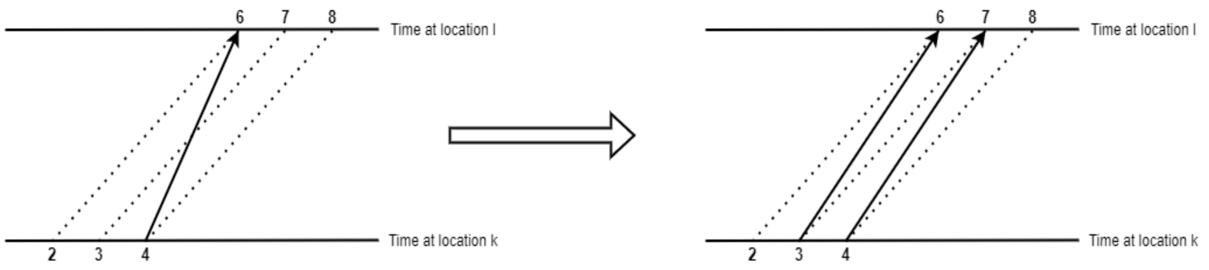

Figure 5: *Example of minimal refinement for a trip departing from location k at $t = 3$ and arriving at location l at $t = 7$ with $\delta^{max} = 1$ and $\omega = 1$. The original underestimated trip arc is replaced by trip arcs of length 3.*

For all new trip arcs that are added in Algorithm 3 and 4, nodes are added accordingly. Each departure node is connected to the starting depot node through a pull-out arc and each arrival node is connected to the ending depot node through a pull-in arc. After the arcs that were considered too short are replaced, the network is restored at the newly added nodes according to the procedure outlined in Algorithms 5 and 6. Together, these methods ensure that these nodes are connected to the existing nodes in the network in such a way that all deadheading arcs in the network remain latest-first-matches as elaborated upon in Chapter 3.3. Waiting arcs are adjusted to the new time points and deadheading arcs are updated in such a way that all possible connections between trips and their deviations are preserved whilst retaining the latest-first-match property. For the destination nodes of the introduced trip arcs, it is examined which outgoing deadheading arcs should be constructed and whether deadheading arcs departing from the previous node at that location must be removed (see Algorithm 5). For the origin nodes, incoming deadheading arcs are added and existing deadheading arcs arriving at the next node at that location are removed if necessary (see Algorithm 6).

Algorithm 5 RESTORE-DESTINATION

```
Require: Node $(i, t^i_{new}) \in N_T$ with $t^i_k < t^i_{new} < t^i_{k+1}$
1: Remove arc $((i, t^i_k), (i, t^i_{k+1}))$ from $A_T$ and add arcs $((i, t^i_k), (i, t^i_{new}))$ and $((i, t^i_{new}), (i, t^i_{k+1}))$ to $A_T$
2: for $\forall((i, t^i_k), (j, t)) \in A_T$ do
3:     if $t^i_{new} + tt_{ij} \leq t$ then
4:         Remove arc $((i, t^i_k), (j, t))$ from $A_T$ and add arc $((i, t^i_{new}), (j, t))$ to $A_T$
5:     else
6:         Set $t' = \text{argmin}\{s \in T_i | s \geq t^i_{new} + tt_{ij}\}$
7:         if $\exists!((i, t^*), (j, t')) \in A_T$ such that $t^* \geq t^i_{new}$ then
8:             Add arc $((i, t^i_{new}), (j, t'))$ to $A_T$
9:         end if
10:    end if
11: end for
```

Algorithm 6 RESTORE-ORIGIN

```
Require: Node $(i, t^i_{new}) \in N_T$ with $t^i_k < t^i_{new} < t^i_{k+1}$
1: Remove arc $((i, t^i_k), (i, t^i_{k+1}))$ from $A_T$ and add arcs $((i, t^i_k), (i, t^i_{new}))$ and $((i, t^i_{new}), (i, t^i_{k+1}))$ to $A_T$
2: for $\forall((j, t), (i, t^i_{k+1})) \in A_T$ do
3:     if $t + tt_{ji} \leq t^i_{new}$ then
4:         Remove arc $((j, t), (i, t^i_{k+1}))$ from $A_T$ and add arc $((j, t), (i, t^i_{new}))$ to $A_T$
5:     else
6:         Set $t' = \text{argmax}\{s \in T_i | s + tt_{ji} \leq t^i_{new}\}$
7:         if $\exists!((j, t'), (i, t^*)) \in A_T$ such that $t^* \leq t^i_{new}$ then
8:             Add arc $((j, t'), (i, t^i_{new}))$ to $A_T$
9:         end if
10:    end if
11: end for
```

The DDD algorithm is guaranteed to find an optimal solution within a finite number of iterations. Because at least one trip arc is lengthened in each iteration, at most $|M|$ iterations are needed to reach optimality when aggressive refinement is applied. In case of minimal refinement, the number of iterations is capped by $2\delta^{max}|M|$. Note that when all trip arcs are fully lengthened, the original fully time-expanded network is retrieved.

5 Results

This chapter dives into the computational results of the DDD algorithm. First, a description is given of the data generating process and the resulting problem instances, followed by a detailed analysis of the performance of DDD.

5.1 Problem instances

Datasets for the MDVSP will be generated according to the data generating procedure introduced by Carpaneto et al. (1989) and later employed by, amongst others, Pepin et al. (2009), Desfontaines & Desaulniers (2018) and Kulkarni et al. (2018). These problem instances contain $|L|$ locations that serve as departure or arriving locations for trips, spread out randomly over a square grid, and $|D|$ depots. The first four depots are always located in the corners of the grid, additional depots are assigned randomly as well. The time horizon spans a 24-hour period and consist of both short trips and long trips.

- **Short trips** start and end at different, randomly chosen locations. The execution time of a short trip exceeds the travel time between two locations by at most 45 minutes. Starting times during peak hours (7:00 - 8:00 and 17:00 - 18:00 are chosen with a higher probability than off-peak departures. In the default setting, short trips comprise 40% of the total number of trips $|M|$.
- **Long trips** are round trips departing and arriving at the same, randomly chosen location. They are distributed randomly over the time horizon and take between three and five hours to complete.

The cost c_{ij} of traversing an arc $(i, j) \in A$ in the network is calculated as follows:

- If (i, j) is a trip arc, $c_{ij} = 0$.
- If (i, j) is a deadheading arc. $c_{ij} = 8 * tt_{ij} + 2 * (t_j - t_i)$.
- If (i, j) is a waiting arc, $c_{ij} = 2 * (t_j - t_i)$.
- If (i, j) is a pull-in or pull-out arc, $c_{ij} = 5000 + tt_{ij}$.

For this research, the benchmark problem instances are of type 4-500M. These datasets contain 4 depots, 500 trips and consist of a mix ('M') of 40% short trips and 60% long trips. To investigate the effect of the network size and structure on the performance of DDD, problem instances are considered of type 4-250M, 4-500S (only short trips), 4-500L (only long trips) and

4-750M as well. The comparison with the full discretization approach is based on the 4-250M instances, as the instances with 500 trips quickly become too large to solve for high values of δ^{max}. For each problem type, ten different instances are generated. By default, the number of locations in the network is one tenth of the number of trips. All mathematical programs are solved by means of CPLEX 12.6.3 with default settings.

5.2 Comparison with full discretization

Figure 6 compares the running time of the DDD algorithm on the 4-250M instances to that of the exact MDVSP-TS model which operates on a fully time-expanded network. It clearly illustrates the computational efficiency of a tailored algorithm for the MDVSP-TS: DDD is not only faster for every value of δ^{max}, but its relative speed increases in the value of δ^{max}. In the most complex case of a 5-minute maximum deviation, naively modelling all potential trip shifts explicitly leads to 13.5 times the running time of DDD.

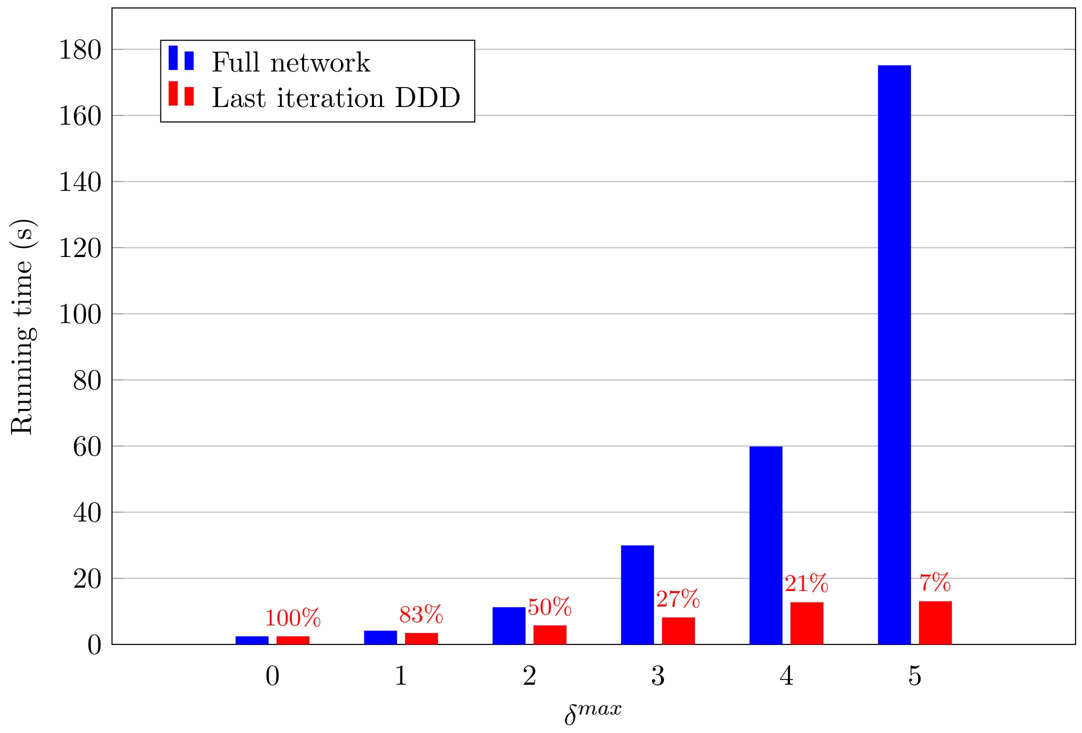

Figure 6: *The total number of variables in the fully time-expanded network and the partially time-expanded network in the last iteration of DDD with aggressive refinement for* $\delta^{max} \in \{0, 1, 2, 3, 4, 5\}$*, averaging over the 10 instances of 4-250M. The labels above the bars indicate the relative size of the DDD network compared to the full network.*

The large difference in running time between DDD and the full discretization approach can be explained by the gap in network size. Although the problem gets increasingly difficult to solve when the iterations progress due to the growing size of the network, we greatly benefit from not having to explicitly model each potential trip shift. Especially for a maximum deviation of 1 or 2 minutes, which is most common when trip shifting is considered as a model feature, 0.3% and 1.7% of trip arcs are refined respectively (for the 4-250M case), barely increasing the total number of variables and proving the computational efficiency of DDD. The gap between the size of the partially time-expanded network in the final DDD iteration and the fully time-expanded network size, which is depicted in Figure 7, underlines how much we benefit from only refining those parts of the partially time-expanded network that lead to infeasible connections being made. Whereas for $\delta^{max} = 0$, both networks are identical, the fully time-expanded network grows to a size four and a half times as large as the partially time-expanded network for $\delta^{max} = 5$.

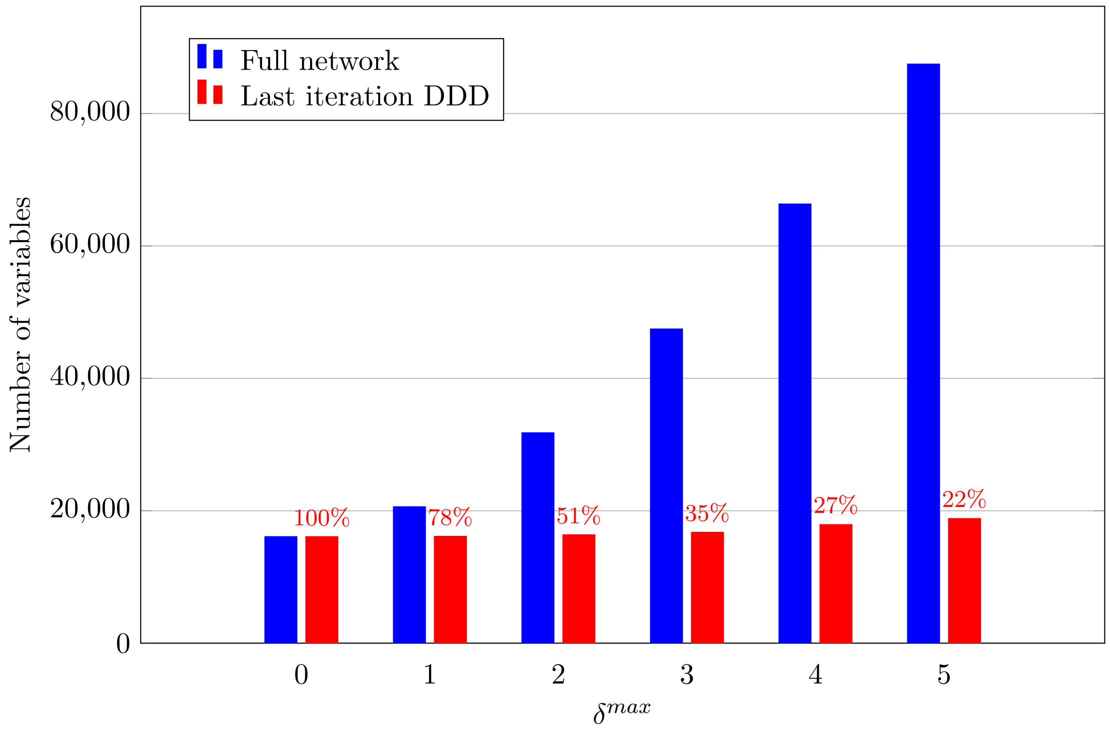

Figure 7: *The total number of variables in the fully time-expanded network and the partially time-expanded network in the last iteration of DDD with aggressive refinement for* $\delta^{max} \in \{0, 1, 2, 3, 4, 5\}$*, averaging over the 10 instances of 4-250M. The labels above the bars indicate the relative size of the DDD network compared to the full network.*

5.3 Detailed performance of DDD

The computational results of DDD for the instances of type 4-500M are denoted in Table 1. The reported numbers are an average over 10 randomly generated instances for each value of $\delta^{max} \in \{0, 1, 2, 3, 4, 5\}$. From left to right, the table shows the optimal objective value of the problem instance, the number of shifted trips in the optimal solution, the refinement strategy (M for minimal refinement, A for aggressive refinement), the total running time, the number of iterations of the DDD algorithm, the number of variables in the last iteration of DDD and the total number of trip arcs that are refined throughout the iterations. For our base case $\delta^{max} = 0$, the algorithm boils down to solving the mathematical model of the MDVSP without trip shifting.

Table 1: *Average performance of the DDD algorithm over 10 randomly generated datasets of type 4-500M*

Instance	δ^{max}	Obj. value	Trip shifts	Refinement	Running time	Iterations	Variables last iteration	Trip arcs refined
4-500M	0	1,326,328	0	N/A	31.8s	1	55,713	0
	1	1,316,062 (-0.77%)	62.3	M	40.5s	1.8	55,914	1.3
				A	42.0s	1.8	55,945	1.5
	2	1,293,496 (-2.48%)	86,7	M	192.9s	4.3	56,600	7.5
				A	155.7s	3.2	56,662	6.3
	3	1,270,952 (-4.18%)	125.6	M	209.7s	4.7	57,774	16.8
				A	215.1s	4.6	58,329	14.7
	4	1,254,587 (-5.51%)	160.8	M	613.6s	7.2	60,152	32.5
				A	495.9s	6.1	61,344	25
	5	1,235,364 (-6.86%)	191.7	M	3014.4s	13.2	65,773	65.8
				A	1646.1s	7.1	67,080	40.9

Table 1 indicates that there is a clear trade-off between the maximum deviation δ^{max} and the running time of DDD. On the one hand, allowing a larger maximum deviation causes more trips to be shifted from their original departure time, ranging from 12% of trips for $\delta^{max} = 1$ to over 38% for $\delta^{max} = 5$. This leads to a reduction in objective value of 0.8% to 6.9%, respectively. On the other hand, computational time appears to increase exponentially. Not only does the number of iterations that is required to solve the MDVSP-TS to optimality increase in the maximum deviation from the departure time, but the partially time-expanded network also grows larger in each iteration. It can be seen that more and more arcs are refined (both per iteration and in total) when departure times get more flexible, which enlarges the network by adding additional trip arcs and deadheading opportunities to it.

5.3.1 Refinement strategies

The choice of refinement strategy for DDD has a clear impact on the performance of the algorithm, as can be observed from Table 1. Although minimal refinement leads to smaller network sizes overall, the number of iterations necessary to reach optimality increases more steeply in the value of δ^{max} compared to the aggressive refinement strategy. The same holds true for the total number of lengthened arcs. For small values of δ^{max}, these factors are slightly outweighed by the benefit of having to solve smaller problems, resulting in a shorter or similar running time. However, when the maximum allowed deviation is 4 or 5 minutes, the many iterations and refined arcs take a heavy toll yielding a running time increase of 24% and 83%, respectively.

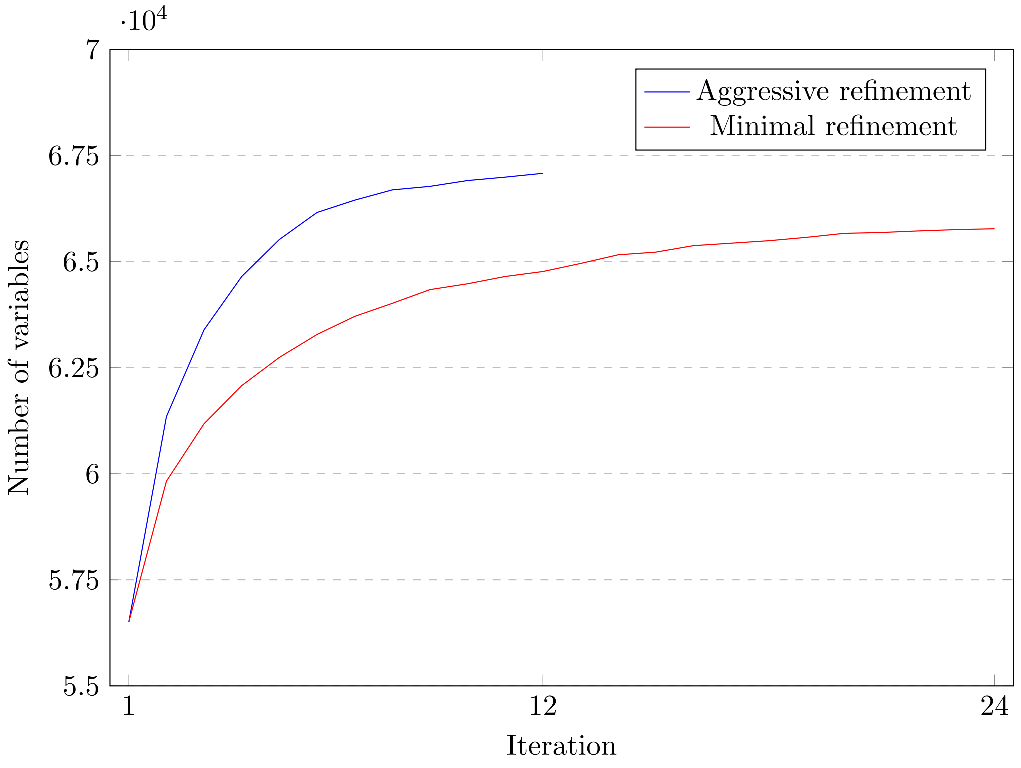

Figure 8: *Evolution of the problem size per iteration of the DDD algorithm with aggressive and minimal refinement, averaging over the 10 instances of type 4-500M with* $\delta^{max} = 5$.

With regards to the behaviour of DDD throughout the iterations, similar patterns can be observed for both the aggressive and minimal refinement strategy. Figure 8 and 9 showcase that the first iteration decidedly has the largest contribution to the refinement of the network. The number of lengthened arcs quickly and steadily decreases throughout the iterations, as well as the number of variables that are added correspondingly. This implies that most infeasible connections made by underestimated trip arcs in the partially time-expanded network are cleared away early in the algorithm, yet it takes relatively many iterations to get rid of all bottleneck

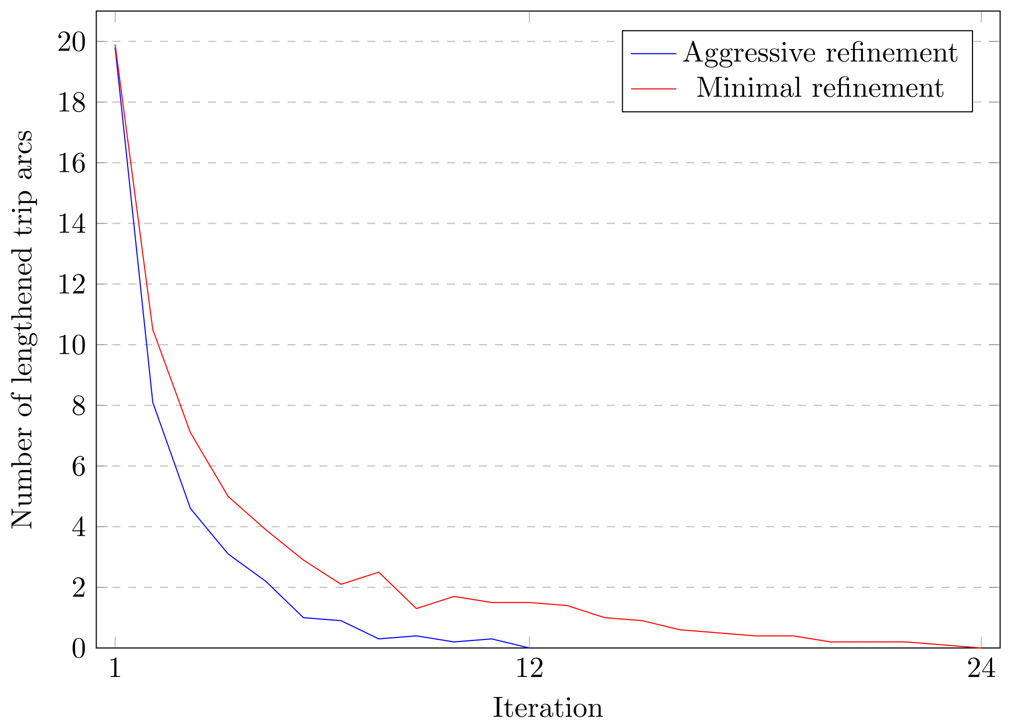

Figure 9: *Evolution of the number of lengthened trip arcs per iteration of the DDD algorithm with aggressive and minimal refinement, averaging over the 10 instances of type 4-500M with* $\delta^{max} = 5$.

trip arcs and yield a completely feasible solution.

Moreover, Figure 8 and 9 offer an explanation for the difference in performance between minimal and aggressive refinement. Because minimal refinement adds a lower number of trip arc duplicates (which can be deduced from the smaller increase in the number of variables), some of the refined arcs are still underestimated. Even though the specific connections that cause infeasibility in one iteration cannot exist in later iterations, the refined arcs can still lead to infeasible routes in combination with other arcs in the network, potentially requiring additional lengthening in the future. This is impossible in case of aggressive refinement, where trip arcs always have the correct length after refinement.

5.3.2 Different problem sizes and structures

Table 2 shows that when problem instances are considered with 250 and 750 trips, the relative performance of DDD when the maximum allowed deviation is increased is similar to the case with 500 trips. A notable difference is the flattening of the number of iterations for larger values of δ^{max} in the 4-250M case. This causes running times to grow much more slowly as opposed to the 500 trip benchmark. A possible explanation is the fact that with fewer trips to cover, the amount of connections that are enabled by trip shifting reaches its limit more quickly.

Table 2: *Average performance of the DDD algorithm over 10 randomly generated datasets of type 4-250M and 4-750M*

Instance	δ^{max}	Obj. value	Trip shifts	Refinement	Running time	Iterations	Variables last iteration	Trip arcs refined
4-250M	0	720,253	0	N/A	2.3s	1	16,082	0
	1	708,093 (-1.69%)	29.6	M	4.0s	1.5	16,132	0.5
				A	3.3s	1.4	16,134	0.5
	2	700,714 (-2.71%)	38.5	M	6.2s	2.3	16,332	2.7
				A	5.6s	2.3	16,360	2.4
	3	690,369 (-4.15%)	56.5	M	11.0s	3.8	16,692	6.1
				A	8.0s	2.5	16,741	4.1
	4	682,101 (-5.30%)	67.8	M	14.8s	5.8	17,465	11,7
				A	12.6s	4.4	17,906	9.3
	5	677,997 (-5.87%)	79.3	M	17.1s	5.5	18,280	15,9
				A	12.9s	4.0	18,817	11.4
4-750M	0	1,925,544	0	N/A	218.0s	1	119,774	0
	1	1,900,072 (-1.32%)	96.6	M	689.3s	3.1	119,077	5.2
				A	834.2s	3.2	119,098	4.7
	2	1,879,071 (-2.41%)	145.8	M	2144.3s	5.9	120,852	21.6
				A	2080.9s	5.8	121,194	18.7

Regarding the difference between minimal and aggressive refinement, similar conclusions can be drawn as for the 4-500M case. Minimal refinement performs roughly as well as its aggressive counterpart for small δ^{max}, but falls behind when larger deviations are allowed. However, Table 2 indicates that for smaller problem instances, minimal refinement never performs better, whereas it is the preferred method for larger problem instances when δ^{max} is small.

The computational results of DDD for problem instances with short trips and long trips only are depicted in Table 3. The most notable observation is that for these monotonous network structures, the algorithm runs much faster overall compared to the instances with a mix of short and long trips. Especially the instances with long trips only can be solved relatively quickly, ranging from a quarter to one fiftieth of the running time for the 4-500M case. This major difference can partly be devoted to the smaller size of the network. As on average, long trips arrive at a later moment in time than short trips, the number of deadheading options to connecting trips is smaller, resulting in a fewer variables in the network. Another consequence is that there are not as many possibilities to decrease the objective value (-4% compared to almost -7% for $\delta^{max} = 5$). The opposite is true for the instances with short trips only, where more than 11% of costs can be eliminated. Despite the abundance of deadheading opportunities and corresponding arcs in the network, the running time per iteration of the algorithm is still considerably smaller than for the mixed structure instances.

Table 3: *Average performance of the DDD algorithm over 10 randomly generated datasets of type 4-500S, and 4-500L*

Instance	δ^{max}	Obj. value	Trip shifts	Refinement	Running time	Iterations	Variables last iteration	Trip arcs refined
4-500S	0	880,140	0	N/A	16.4s	1	59,956	0
	1	860,178 (-2.27%)	60.8	M	24.6s	1.6	60,184	0.8
				A	28.2s	1.7	60,187	0.8
	2	842,584 (-4.27%)	75.1	M	42.7s	2.8	60,552	3.4
				A	44.9s	2.6	60,563	2.8
	3	820,803 (-6.74%)	116.1	M	81.4s	4.5	61,863	13.6
				A	92.8s	5.1	62,330	12.5
	4	805,211 (-8.51%)	154	M	185.5s	8.3	64,650	34.0
				A	126.5s	5.7	65,421	24
	5	783,086 (-11.03%)	183.6	M	394.9s	12.3	69,742	61.1
				A	175.7s	7.1	71,464	40.3
4-500L	0	1,701,860	0	N/A	7.5s	1	49,989	0
	1	1,685,666 (-0.95%)	58.1	M	10.6s	1.7	50,170	0.9
				A	11.1s	1.7	50,175	0.9
	2	1,672,241 (-1.74%)	86.3	M	20.0s	2.9	50,656	4.8
				A	18.5s	2.8	50,742	4.6
	3	1,659,996 (-2.46%)	118.0	M	32.3s	4.4	51,476	10.4
				A	26.7s	3.6	51,766	9.0
	4	1,650,779 (-3.00%)	145.5	M	45.9s	5.8	53,088	20.5
				A	30.8s	4.4	53,849	16.6
	5	1,633,556 (-4.01%)	185.0	M	63.9s	8.3	55,716	36.7
				A	38.8s	5.0	57,173	26.2

5.4 Upper bound performance

Despite the fact that DDD takes increasingly many iterations to execute when larger deviations are allowed, strong upper bounds can be found from the first iteration onward as indicated by Figure 10. The average optimality gap obtained at iteration 1 by means of the matheuristics is

5.0% for the multi-depot and 5.8% for the single depot approach, slowly dropping thereafter. This implies that the solutions obtained already have a lower objective value than the optimal solution for the base case without trip shifting. The cutting heuristic performs much worse with an initial optimality gap of 17.4% but improves quickly. The optimality gaps of the three methods converge gradually, all dropping below 1% at iteration 6. Figure 10 also illustrates that the objective value of the lower bound barely increase over the course of the algorithm (in this example with only 0.07%), thus giving a good indication of the eventual optimal objective value.

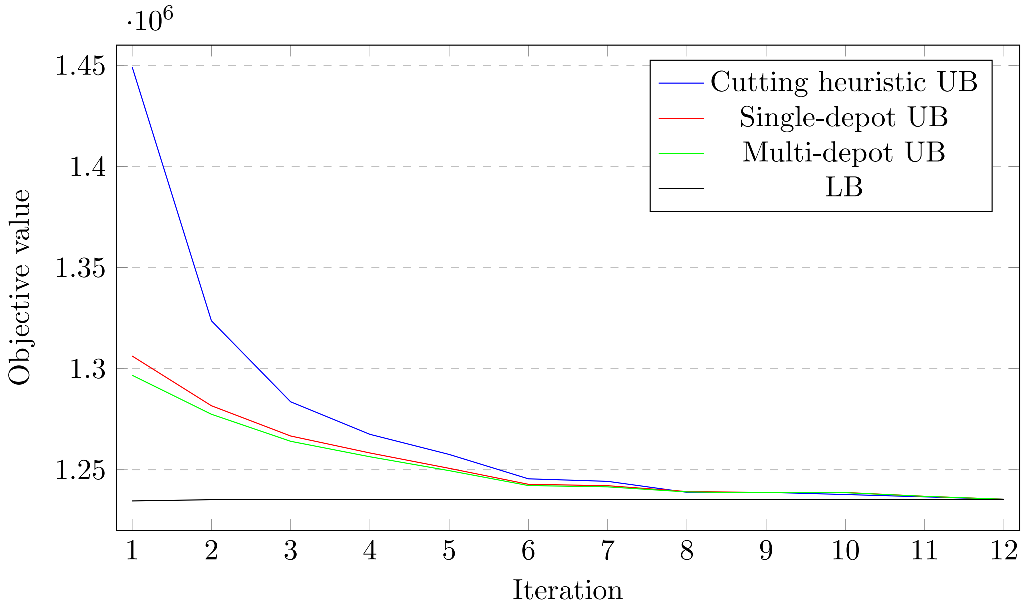

Figure 10: *Evolution of the value of the lower bound and upper bounds per iteration of the DDD algorithm with aggressive refinement, averaging over the 10 instances of 4-500M with* $\delta^{max} = 5$.

When looking at the running time of the upper bound heuristics in Figure 11, it is clear that a stronger upper bound comes at the cost of a longer computation time. The cutting heuristic, as advertised, finds an upper bound within a split second. The matheuristics, and in particular the multi-depot matheuristic, take much longer to execute. However, it must be noted that the combined running time of either upper bound mechanism is negligible in comparison to the total running time of the DDD algorithm. The slowest method, the multi-depot matheuristic, only comprises 0.05% of the total execution time.

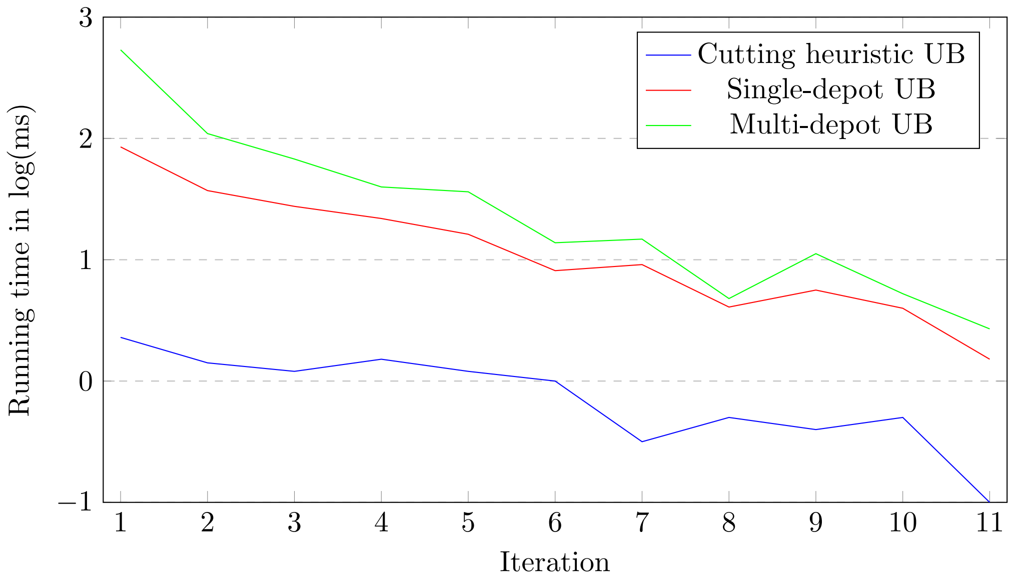

Figure 11: *Evolution of the running time of the upper bounds per iteration of the DDD algorithm with aggressive refinement, averaging over the 10 instances of 500M with* $\delta^{max} = 5$.

6 Conclusion

In this thesis, we have presented a Dynamic Discretization Discovery algorithm for the Multi-Depot Vehicle Scheduling Problem with Trip Shifting. Inspired by Boland et al. (2017), this iterative refinement method is able to find an optimal solution to the MDVSP-TS without explicitly modelling all possible trip shifts. Instead, it iteratively solves an integer program on a partially time-expanded network and extends the network only where it causes infeasibility. The core idea behind DDD is that solving the problem on a relaxed network multiple times yields an optimal solution much quicker than through using the fully time-expanded network only once.

Our computational study has shown that DDD indeed outperforms the explicit modelling approach by a wide margin. While the size of the fully time-expanded network quickly grows to a point where the model is intractable when the maximum allowed deviation increases, DDD is able to solve instances even for large deviations. Besides, a tailored multi-depot matheuristic provides strong upper bounds already in the early stages of the algorithm, whereas the lower bound is close to the optimal objective value from the first iteration onward. Combined with the observation that the problem becomes increasingly difficult to solve when the iterations progress, it may suffice to terminate the algorithm after a few iterations to obtain a good solution relatively quickly.

The choice of refinement strategy plays an important role in the performance of DDD. While minimal refinement often leads to shorter running times when there is little room for trip shifting, aggressive refinement yields much quicker solutions when the problem becomes more complex. Moreover, the structure of the network greatly impacts the running time of the algorithm. When all trips are either short or long, DDD is able to reach the optimal solution much more quickly in comparison to a combination of both short and long trips.

There are several ways to potentially improve upon the algorithm as outlined in this thesis. For example, more thorough investigation is required to find out whether a balance can be struck between both refinement mechanisms to arrive at an optimal strategy that is preferred for any value of the maximum deviation parameter. Furthermore, the computation of the solution on the partially time-expanded network might be sped up by using the solution obtained in the previous iteration as a starting point for the integer program.

All in all, DDD has proven to be a strong addition to the toolbox for solving the MDVSP-TS. Interesting ideas for future research include validating the performance of DDD on real-world datasets, especially given the influence of the network structure, or extending the application of DDD to integrated vehicle and crew scheduling problems.

References

Bertossi, A. A., Carraresi, P., & Gallo, G. (1987). On some matching problems arising in vehicle scheduling models. *Networks*, *17*(3), 271–281.

Boland, N., Hewitt, M., Marshall, L., & Savelsbergh, M. (2017). The continuous-time service network design problem. *Operations Research*, *65*(5), 1303–1321.

Bunte, S., & Kliewer, N. (2009). An overview on vehicle scheduling models. *Public Transport*, *1*(4), 299–317.

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.

Ceder, A. A. (2011). Optimal multi-vehicle type transit timetabling and vehicle scheduling. *Procedia-Social and Behavioral Sciences*, *20*, 19–30.

Desaulniers, G., Lavigne, J., & Soumis, F. (1998). Multi-depot vehicle scheduling problems with time windows and waiting costs. *European Journal of Operational Research*, *111*(3), 479–494.

Desfontaines, L., & Desaulniers, G. (2018). Multiple depot vehicle scheduling with controlled trip shifting. *Transportation Research Part B: Methodological*, *113*, 34–53.

Fonseca, J. P., van der Hurk, E., Roberti, R., & Larsen, A. (2018). A matheuristic for transfer synchronization through integrated timetabling and vehicle scheduling. *Transportation Research Part B: Methodological*, *109*, 128–149.

Forbes, M., Holt, J., & Watts, A. (1994). An exact algorithm for multiple depot bus scheduling. *European Journal of Operational Research*, *72*(1), 115–124.

Hadjar, A., & Soumis, F. (2009). Dynamic window reduction for the multiple depot vehicle scheduling problem with time windows. *Computers & Operations Research*, *36*(7), 2160–2172.

Haghani, A., & Banihashemi, M. (2002). Heuristic approaches for solving large-scale bus transit vehicle scheduling problem with route time constraints. *Transportation Research Part A: Policy and Practice*, *36*(4), 309–333.

He, E. Y., Boland, N., Nemhauser, G., & Savelsbergh, M. (2021). Dynamic discretization discovery algorithms for time-dependent shortest path problems. *INFORMS Journal on Computing*.

Hewitt, M. (2019). Enhanced dynamic discretization discovery for the continuous time load plan design problem. *Transportation Science*, *53*(6), 1731–1750.

Kliewer, N., Amberg, B., & Amberg, B. (2012). Multiple depot vehicle and crew scheduling with time windows for scheduled trips. *Public Transport*, *3*(3), 213–244.

Kliewer, N., Bunte, S., & Suhl, L. (2006). Time windows for scheduled trips in multiple depot vehicle scheduling. In *Proceedings of the EWGT2006 joint conferences* (pp. 340–346).

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

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*, *118*, 457–487.

Laporte, G., Ortega, F. A., Pozo, M. A., & Puerto, J. (2017). Multi-objective integration of timetables, vehicle schedules and user routings in a transit network. *Transportation Research Part B: Methodological*, *98*, 94–112.

Liu, T., & Ceder, A. A. (2017). Integrated public transport timetable synchronization and vehicle scheduling with demand assignment: a bi-objective bi-level model using deficit function approach. *Transportation Research Procedia*, *23*, 341–361.

Marshall, L., Boland, N., Savelsbergh, M., & Hewitt, M. (2021). Interval-based dynamic discretization discovery for solving the continuous-time service network design problem. *Transportation Science*, *55*(1), 29–51.

Mesquita, M., & Paixão, J. (1992). Multiple depot vehicle scheduling problem: A new heuristic based on quasi-assignment algorithms. In *Computer-aided transit scheduling* (pp. 167–180). Springer.

Mingozzi, A., Bianco, L., & Ricciardelli, S. (1995). An exact algorithm for combining vehicle trips. In *Computer-aided transit scheduling* (pp. 145–172). Springer.

Pepin, A.-S., Desaulniers, G., Hertz, A., & Huisman, D. (2009). A comparison of five heuristics for the multiple depot vehicle scheduling problem. *Journal of Scheduling*, *12*(1), 17–30.

Petersen, H. L., Larsen, A., Madsen, O. B., Petersen, B., & Ropke, S. (2013). The simultaneous vehicle scheduling and passenger service problem. *Transportation Science*, *47*(4), 603–616.

Ribeiro, C. C., & Soumis, F. (1994). A column generation approach to the multiple-depot vehicle scheduling problem. *Operations Research*, *42*(1), 41–52.

Scherr, Y. O., Hewitt, M., Saavedra, B. A. N., & Mattfeld, D. C. (2020). Dynamic discretization discovery for the service network design problem with mixed autonomous fleets. *Transportation Research Part B: Methodological*, *141*, 164–195.

Schmid, V., & Ehmke, J. F. (2015). Integrated timetabling and vehicle scheduling with balanced departure times. *OR spectrum*, *37*(4), 903–928.

Van den Heuvel, A., Van den Akker, J., & Van Kooten, M. (2008). Integrating timetabling and vehicle scheduling in public bus transportation. *Techical Report UU-CS-2008-003, Department of Information and Computing Sciences, Utrecht University, The Netherlands*.

Vu, D. M., Hewitt, M., Boland, N., & Savelsbergh, M. (2020). Dynamic discretization discovery for solving the time-dependent traveling salesman problem with time windows. *Transportation Science*, *54*(3), 703–720.