universität wien

MASTERARBEIT | MASTER'S THESIS

Titel | Title

Dynamic Discretization Discovery

verfasst von | submitted by

Melika Dulancic BSc

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien | Vienna, 2026

Studienkennzahl lt. Studienblatt | Degree programme code as it appears on the student record sheet:	UA 066 915
Studienrichtung lt. Studienblatt | Degree programme as it appears on the student record sheet:	Masterstudium Betriebswirtschaft
Betreut von | Supervisor:	Ass.-Prof. Dr. Christian Tilk

To my mum for her endless love and support

Table of Contents

Abstract

The main aim of this thesis is to explore the Dynamic Discretization Discovery (DDD) algorithm designed to solve complex optimization problems in continuous-time networks. The main research motive is to do a comprehensive literature analysis of all the available articles on the topic of DDD and examine the algorithm's versatility across different fields, including freight transport, vehicle routing and industrial scheduling. By analyzing different mathematical formulations and computational studies, the thesis demonstrates how DDD provides great scalability and faster convergence than commercial solvers and overall effective cost minimization and smaller operational gaps in large-scale logistical problems.

Keywords: Dynamic Discretization Discovery, time-expanded networks, time discretization, dual bounds, problem relaxation, refinement procedure

1. **Introduction**

In todays' world, carriers operate last-mile shipments in a less-than-truckload sector where the increase demand shows that 80% of those shipments need to be delivered in two days (Boland et al. 2017). This trend puts carriers in an unsatisfying position of having to deliver good in the shortest possible time with the lowest possible cost, meaning that large delivery companies are constantly experimenting with different ways to enable next or same day shipments (Boland et al. 2019). In order to be able to achieve these goals, carriers need to consolidate shipments in both time and space (Boland et al. 2017). Timing of different activities has become increasingly more important and the researchers approach to solving the problems with timing decisions was to represent the problem on a time-expanded network and solve it using a mixed integer program (Boland et al. 2019). Some examples of how important timing and coordination can be can be shown on an example from the airline industry, where flights, crew and airplanes must be schedules to meet the demand of customers, where a slight deviation from departure time can lead to huge impact on legal connection possible for the crew and the passengers (Boland et al. 2019b). Innovative logistics practices driven by the market demand create a need for time dependent models that can be executed efficiently. These models are very difficult to solve in practice, since they have weak linear programming relaxations which limits their use with current solver technology to only a small number of instances (Boland et al. 2019b). In order to overcome this issue, it's not uncommon to use time discretization instead of deciding on the exact time at which an activity should occur, the model decided on a time interval during which the activity occurs (Boland et al. 2017). Formulations with binary variables indexed by time have stronger relaxations at a cost of having a huge number of variables and they strongly rely on time discretization (Boland et al. 2019b). Time expanded network is constructed from a static network by partitioning the planning horizon into discrete intervals, where a duplicated node in a time expanded network represents a location at a point in time and an arc represents a transportation from one location to another including the transit duration and dispatch time (Boland et al. 2019). Time expanded networks are often used due to their flexible modeling and IP formulations that can be solved with a powerful software packages, but that still opens a question about choosing the right discretization (Boland et al. 2019). While fine discretization yields near-optimal or optimal solutions, their IP formulations are too large for current solver capabilities, which leads to the usage of coarser discretization in practice at the cost of solution quality (Boland et al. 2019b, Boland et al. 2019). The granularity of discretization clearly impacts the computation tractability where e.g. an hourly discretization of a week-long planning horizon leads to 168 and a five-minute discretization lead to 2016 timed copies of a node, where the later most probably yields an impossible to solve network design problem (Boland et al. 2017). Refining the granularity from 60-minute discretization to a 1-minute discretization leads to a factor 15 increase in program size. Since the time is discretized, parameters involving time also need to be mapped to discrete intervals as well (Boland et al. 2019). Huge part of the challenge associated with choosing the right discretization come from doing it in an a priori fashion when we still don't know which points are to be modeled. In order to overcome the mentioned challenges of balancing the discretization granularity and the difficulty of the formulation, a novel paradigm has emerged

in the recent years called dynamic discretization discovery that tries to efficiently find optimal solutions to the time dependent models (Boland et al. 2019b). Dynamic discretization discovery finds solution on a fine discretization without ever fully constructing the full time expanded network and instead the time indexed IP models are based on a partial discretization of time that are solvable in theory and practice and yield either provable lower and upper bounds or exact solutions (Boland et al. 2019b). The paradigm also has a designed algorithm that allows dynamic discovery and refinement of a partial time discretization that strengthens the quality of a time-indexed IP model, efficiently solving the IP models (Boland et al. 2019b). DDD's key approach is that it discovers exactly which times are needed to obtain an optimal, continuous time solution by efficiently solving a sequence of small integer programs rather than one big program (Boland et al. 2019). One study shows the efficiency of DDD on a network that consists of 30 nodes, 700 arcs and 400 commodities, with a 1-minute discretization over an 8 hour period, and is able to solve it in around 30 minutes with IP that have more than 1,500,000 variables and 1,400,000 constraints, where 97% of instances are solved in less than 15 minutes (Boland et al. 2017).

In order to explain how DDD works in a most understandable manner, we will use Boland et al. (2017) as an example to explain the algorithm in simple terms. As previously mentioned, companies are tried to save money by consolidating as many shipments as possible in one truck and coordinating where and when each package has to arrive on time. Usually, algorithms solve this issue by breaking time into fixed, unified boxes prior to solving the algorithm like e.g. time boxes of one hour or 10 minutes, but with the issue that if these time boxes are too big the problem produces an inaccurate time plan with missed opportunities of combining shipments and if the blocks are too small the mathematical problem becomes large and unsolvable with millions of variables and constraints (Boland et al. 2017). The researches then took a different approach where instead of starting with a detailed schedule for every minute of the day, the algorithm starts with a partially time-expanded network and never creates a full time-expanded network. Partially time-expanded network can be imagined as a rough map that only includes few time points, such as the time and place when a package become available and when it's due at destination (Boland et al. 2017). The algorithm repeatedly solves the problem defined on a partially time-expanded network and refines it based on the analysis of a previously obtained solution, where each resulting problem is a relaxation of the original problem. The algorithm solves a relaxed version of the original problem where it's allowed to slightly "cheat" on the actual travel times, assuming that if the truck needs to get somewhere, it could arrive at the next available time point on the map, even if that time point is sooner than the truck can physically arrive there (Boland et al. 2017). Since the underestimation of travel times is easier, the algorithm finds a best-case scenario for the cost very quickly. The algorithm then takes a look into its best-case scenario and assess if it's actually possible in the real world, by looking for any parts of the plan where the truck is scheduled to arrive earlier than physically possible due to a thing called short arc (Boland et al. 2017). If the truck is supposed to move faster than the set speed limit to arrive in time for consolidation, the algorithm concludes that this specific part of the schedule is invalid. Instead of making the whole plan more detailed, the algorithm only adds new time points at those exact places where the plan broke the rules and it lengthens those specific short trips to match the real-world times (Boland et all. 2017). This way by refining only parts of the map and plan, the problem stays small and manageable and the plan gets more detailed where it matters. The algorithm repeats this process of solving new, slightly more detailed map and checking the time "cheats" again, only stopping when it finds a plan

where every truck follows real-world travel times and the costs are as low as possible (Boland et al. 2017). In summary, the algorithm starts with a simple outline and strategically adds detail only to the specific moments necessary to create a realistic and low-cost schedule rather than trying to plan every possible minute of the day at once.

An easier way to understand this DDD algorithm is with the following analogy of planning a road trip across the country. Instead of overwhelming ourselves by looking at the whole map showing every street in every city, we start our planning with a map that only shows major highways. We will only zoom in and take a look at the surrounding street when we realize that we are getting of a highway and we need to choose the side streets in a specific neighborhood to reach our accommodation. We only choose to bother with the detailed street plan specifically when we need it and keep the rest of the plan simple.

When we speak about this specific paper, its approach is a literature research of the topic of dynamic discretization discovery based on all of the relevant papers published since 2017 when the algorithm was first introduced. For this paper, the process of finding literature encompassed a detailed research of the topic on several different online platforms like Informs, ResearchGate, Google Scholar, Science Direct, Optimization Online and others with the selection of relevant literature always containing the topic of dynamic discretization discovery. Papers that only briefly mentioned time discretization without tackling DDD algorithm itself were disregarded. At the time of submission of this paper, the total amount of the available literature found and used to write this paper was 22 papers, published from 2017-2025. From the researched literature, the biggest chuck of the papers covered DDD in relation to solving the service network design problem since it is the type of problem the whole algorithm was introduced on by Boland et al. (2017). Apart from SND, DDD algorithm was also adapted to cover problems like shortest path, traveling salesman problem, inventory-routing problem, scheduling problem, heuristic based on DDD and papers about the algorithm efficiency in general. A big chunk of the literature also tackled time dependent variant of said problems and a combination of different problems in one. The aim of the algorithm is finding good solutions fast, the time frame that it covers is mainly operational with a maximum planning horizon considered over the given literature being a week.

In this paper, through the following chapters we will go over the algorithm in detail, from the initial network, mathematical formulations, different ways to achieve relaxations, dual bounding system and algorithm refinement, convergence to optimality, some enhancements of the algorithm and a computational study. This paper is a work over the last couple of months, with the factual information taken strictly from the cited papers and with information that can be traced in the mentioned articles.

2. **Problem description**

Service Network Design Problems are important transportation problems in the freight industry that study the effect of consolidating orders to save on transportation cost (Shu et al. 2024). The carrier needs to carry a shipment from origin to destination, through a network of terminals that consolidate multiple shipments, which are small compared to the trailer size, into one vehicle, while also respecting the time at which these shipments need to arrive at the customer locations (Medina et al. 2019). SNDP decides on paths for the shipments and the supporting services and resources necessary for the execution (Boland et al. 2017). Hence why, for all variants of the service network design problem (SNDP), the problem itself is defined first as a "flat" network $\mathcal{D}$=($\mathcal{N}$, $\mathcal{A}$), with a node set $\mathcal{N}$ and directed arc set $\mathcal{A}$, where nodes in $\mathcal{N}$ represent physical locations. The arcs of the form α=(i,j) in arc set $\mathcal{A}$ are associated with a travel time τ_{ij}, per unit cost c_{ij} , fixed cost f_{ij} and capacity u_{ij}. In $\mathcal{K}$ commodity set each commodity k with its capacity q_k has a source o_k and sink d_k nodes, which are related to the times the commodity k is first available, e_k, and the time it is due at its destination, l_k .
The goal of the SNDP is to ensure that the cost is kept at minimum with the respect of the time window constraints (Boland et al. 2017; Hewitt 2019; Scherr et al. 2020; He et al. 2023; Van Dyk and Koenemann 2024; Medina et al. 2019; Marshall et al. 2021, Shu et al. 2024).
Although the previously presented notation for "flat" network for the most general SNDP problem is more or less the same for all papers covering the SNDP, there are still some differences in the notation in some of the cases which primarily stemming from different variants of the SNDP problem which they study. In the following section, these differences and additions will be quickly discussed.
Without any additions to the notation of the previously defined general "flat" network $\mathcal{D}$=($\mathcal{N}$, $\mathcal{A}$), Hewitt (2019) defines his Continuous Time Load Plan Design Problem (CT-LPDP) as a Scheduled Service Network Design Problem (SSNDP) with some additional restrictions where a solution must follow a path from a predefined set of candidate paths and the paths that have the same destinations must form a tree like structure.
On the other hand, Boland et al. (2017), He at al. (2023), Shu et al. (2024) and Marshall et al. (2021) study the variant of SNDP where continuous time is also considered (CTSNDP). While the notation for "flat" network $\mathcal{D}$ remains the same as previously defined in the paper from Boland et al. (2017) ,He et al. (2023) and Marshall et al. (2021) use a slightly different notation using a directed graph, $\mathcal{G}$=($\mathcal{N}$, $\mathcal{A}$) instead of $\mathcal{D}$=($\mathcal{N}$, $\mathcal{A}$). He et al. (2023) also note that there is an additional set of predefined routes r∈ $\mathcal{R}$ which visits a predefined sequence of nodes $\mathcal{N}r$ through predefined sequence of arcs $\mathcal{A}r$. Shu et al. (2024) in their paper study the incorporation of holding costs alongside the original CTSNDP and add following additions to their notation: in-transit costs c^k_{ij} and in-storage holding costs h^k_i. He et al. (2023) studies a combined SNDP and Vehicle Routing Problem (SNDRP) where their "flat" network is called a route-expanded network for which they add to their notation a set of hubs $\mathcal{H}$, per-pallet transfer costs c_h and transfer time σ_h, as well as distinguish three different types of nodes (route copies, hub nodes and origin and destination nodes for each commodity) and three different types of arcs (routing, transfer arcs in between hubs and pick-up and delivery arcs). Medina et al. (2019) also study a combined SNDP and VRP problem and distinguish between customer, breakbulk and supplier nodes, N_c, N_{bb} and N_s, respectively, where the arc set $\mathcal{A}$ contains arcs that connect customers with their respective breakbulk and arcs that connect two breakbulks. Scherr et al. (2020) study a variant of SNDP with mixed autonomous fleet, where some autonomous vehicles (AV) may not travel without human operated manual

vehicles (MV), hence why in their notation there is distinction between nodes for external zones and nodes for satellites, N_E and N_S respectively , as well as A_A arcs through which AV are able to travel alone and A_M which represent arcs where AV vehicles need to be pulled by MVs. Van Dyk and Koenemann (2024) study a SNDP with restricted routes (SNDP-RR) with an additional subgraph $\mathcal{D}^K$ in the notation for each commodity where the trajectory of the commodity k must be fully contained in $\mathcal{D}^K$ in order to be feasible.

Since the SNDP problem is typically modeled using a time-expanded network, the defined "flat" network $\mathcal{D}=(\mathcal{N},\mathcal{A})$ representing physical connections between nodes needs to be expanded to model the time component as well (Boland et al. 2017; Van Dyk and Koenemman 2024). In the time-expanded network $\mathcal{D}^T$ locations are replicated in discrete time steps $t\in T=\{0,\ldots,t_{MAX}\}$ over the whole planning horizon of t_{MAX} (Scherr et al. 2020). He et al. (2023) argue that the time-expanded network lays the basis for the mathematical model of the problem which will be the base of our static network in the beginning.

Time-expanded network $\mathcal{D}_T=(\mathcal{N}_T,\mathcal{H}_T \cup \mathcal{A}_T)$ is derived from the physical network $\mathcal{D}$ and a set of time points T such that every node in node set $\mathcal{N}_T$ has a form of (i,t) for each $i\in\mathcal{N}$ and $t\in T_i$. The arc set $\mathcal{H}_T$ contains holdover arcs of the form $((i, t^i_k),(i, t^i_{k+1}))$ and arc set $\mathcal{A}_T$ of the form $((i, t),(j, t'))$ where $i,j\in\mathcal{A}$, $t\in T_i$ and $t'\in T_j$ (Hewitt 2019; Boland et al. 2017). Arcs of the form $((i, t),(j, t'))$ model the possibility to dispatch freight from location i at time point t, which arrives at location j at time point t', where it would be common to choose such t' such that $t'-t\geq \tau_{ij,}$ although going forward we will see that assumption does not have to hold in the DDD algorithm (Hewitt 2019). The arc set $\mathcal{H}_T$ with arcs of the form $((i, t^i_k),(i, t^i_{k+1}))$ model the possibility of holding freight in the location i, which uses the possibility to consolidate freight at the later point in time (Medina et al. 2019; Boland et al. 2017). The initial algorithm presented by Boland et al. (2017) strongly relies on the assumption that the consolidation is possible and that holding freight at terminals doesn't induce any additional cost. Shu et al. (2024) argue that indeed DDD algorithm can be successfully extended such that the holding cost, also known as consolidation penalty costs, are taken into account by computing cost of a solution as a function of holding cost and consolidation plans. Their developed DDD algorithm firstly proves the existence of a time-expanded network which accounts for the holding cost and incorporates a new upper bound heuristic and refinement strategy to solve the CTSNDP-HC. Another slightly different time-expanded network can be observed in the work of He et al. (2023) where they expand their route-expanded network by introducing time such that a complete route-time-expanded network contains for all route sets all timed routes that start and end within a pre-defined planning horizon.

Since time-expanded formulations can become very large quickly, by using interval-based approach it is possible to keep the time-expanded network as small as possible while also converging to the optimal solution quickly resulting in an algorithm that can solve large instances quickly. (Marshall et al. 2021). In this case, the time-expanded network for a given discretization introduces node-intervals and timed-arcs, where the node-intervals are a set of triples representing the interval of time (t,t') at every node n and the timed-arcs are divided between dispatch and holding-arcs.

Now that we've introduced the classical SND problem and other variants, in this section we will go over a classical combinatorial problem called Traveling Salesman Problem, which was also studied by some authors who tried to solve it with the DDD algorithm. TSP is a type of problem where the salesman departs from home to visit a set of other cities and then returns home, all with the objective of minimizing the total travel cost (Vu et al. 2019; Boland et al. 2017b). In logistics context the traditional salesman is substituted by the vehicle, the home

becomes depo and cities are then customers. In the modern-day world, most TSP problems are extended with a time component where time windows for each customer are introduced to TSP, requiring the vehicle to visit the customer during a pre-specified period (Vu et al. 2019). This classical problem was also extended in the context of solving TSPTW under Makespan objective, where the time at which the vehicle returns to the depot is minimized (Vu et al. 2022; Vu et al. 2019). Another extension to the problem are time dependent travel times that introduce a scheduling dimension to a routing problem (Vu et al. 2022). Authors like Vu et al. (2022) in their work study TD-TSPTW with the objective function that requires the departure time from the depot to be optimized, specifically, solving both the TD Minimum Tour Duration Problem and the TD Delivery Man Problem. In TSPTW, if the vehicle arrives before the time window at the customer begins, it must wait which in the TD-MTDP, because of the minimum duration objective function the optimal solution may exist if it doesn't involve waiting at any location unless it is for time window to open (Vu et al.2022). Similar characterization is for the other extension, namely the TD-DMP, where the problem considers the amount of time a customer waits for the vehicle since its departure from the depot and it tries to minimize the sum of the waiting times. For the time dependent variants one important assumption is made regarding travel times. The assumption commonly used is called First-In-First-Out or for short FIFO, meaning that when traveling between two locations, later departure cannot lead to earlier arrival since travel time is a piecewise linear function of departure time (Vu et al. 2019). Without this property in place, vehicles could wait at the depot until the travel time function hits its minimum to depart from the depot and still arrive before other vehicles who departed earlier. Having said all that, all of the TSPTW have a high significance and practical importance because they appear as a subproblem in various real-life routing and scheduling applications, but nonetheless are very difficult to solve due to the fact that finding a feasible customer sequence in an NP-hard problem (Boland et al. 2017b).

All TSPTW are defined with following characteristics similar to those of the SND problems. $(\mathcal{N}, \mathcal{A})$ denotes a directed graph, with a node set $\mathcal{N}$ that includes depot at point zero and a set of locations that must be visited (Vu et al. 2019; Vu et al. 2022; Boland et al. 2017b). Same as in SND problems, each location also has a predetermined time window during which it must be visited which we denote as (e_i, l_i), where it is allowed for the vehicle to arrive to the location before the time window opening. For the depot, a time window (e_0, l_0) is also determined where the vehicle can't depart before the e_0 opening time and return after the l_0 closing time (Vu et al. 2019; Vu et al. 2022; Boland et al. 2017b). The arc set $\mathcal{A} \subseteq \mathcal{N} \times \mathcal{N}$ represents the physical travel arcs between two locations. With each travel arcs (i,j) is associated time t at which travel can begin and τ_{ij} which represents the travel time. With the addition of time dependency, the travel time is a function of time that commences at time t, which we denote as follows $\tau_{ij}(t)$ (Vu et al. 2019; Vu et al. 2022). For the earlier mentioned FIFO property, in mathematical terms, $t+ \tau_{ij}(t) \leq t'+ \tau_{ij}(t')$, where $t \leq t'$, must hold.

As it was the case with SNDP, the TSPTW problems are also formulated as integer programs defined on a time-expanded network, hence why the previously introduced flat network has to be extended to account for the time component. The time expanded network denoted as $\mathcal{D}=(\mathcal{N}, \mathcal{A})$ in TSP problems, is characterized by the timed arcs of the form $((i, t),(j, t'))$, where $i,j \in \mathcal{A}$ and $i \neq j$ (Vu et al. 2019; Vu et al. 2022; Boland et al. 2017b). We model the property that the vehicle cannot visit early and imbeds any waiting time as $t \geq e_i$, $t'=\max\{ e_j, t+ \tau_{ij}(t)\}$ and the property that it cannot arrive late as $t' \leq l_j$ (Vu et al. 2019; Vu et al. 2022). With this in mind we can go over the actual mathematical formulations of the TSP problem.

Along the SND and TSP problems, in this subsection we will present the scheduling problems more closely. There are different types of scheduling problems, but the ones solved with the means of DDD are train rescheduling (Croella et al. 2024), cross-dock trailer scheduling with workforce constraint (Ojha and Erera, 2025) and scheduling with time dependent resource consumption (Pottel and Goel, 2022). Starting with the introduction of train rescheduling problem, immediately it's real life importance can be seen since the problem is in wide use. When trains move through the railway, they occupy a sequence of rail that no other train can occupy at the same time and in case of any readjustments, it needs to be done in real time to adjust to any disruptions or delays (Croella et al. 2024). This problem is a generalization of a job-shop problem that tries to minimize the deviation of the actual schedule from the official timetable computed in a very short time no longer than 10 s for all trains covering the next couple of hours (Croella et al. 2024). With the TI formulation MILP contains only binary variables, which makes the conversion of the problem into a MaxSAT problem easy. For this problem, the authors consider that for each train i and a given resource r, time is partitioned into intervals of different width, hence constructing an Interval Assignment Problem with binary variable x^{ir} that is one when train i enters resource r at some point t^{ir} during the p^{th} interval (Croella et al. 2024). Costs are, as well, chosen in such a way that the value of the optimal solution of DDD is a lower bound on the optimal solution of the original problem. For each train $i \in I$ we have R_i as a set of track segments of the rail path with $l^r{}_i$ being the minimum amount of time necessary for a train to traverse the segment (Croalla et al. 2024). For each segment r we also have the earliest time the train can enter it and for each pair of trains i and j a set of rail paths pairs $\mathcal{D}ij \subseteq R_i x R_j$, such that either i enters r before j enters q or j enter q before i enter r. For each pair we hence have $t^{jq} \geq t^{ir} + l^{rq}$ or $t^{ir} \geq t^{jq} + l^{qr}$ that makes sure the precedence is not violated. Contrary to previous SND and TSP problems, the 'flat network' isn't formulated the same way for the TRP problem and TI formulation calls for feasibility intervals represented as time intervals for each train i and rail segment r, in which the train can enter the track segment r in its planning horizon (Croella et al. 2024). The binary variable x^{ir} in the flat network that determines if the train i enters rail segment r is then extended to account for the time component to $x^{ir}{}_p$, which determines if train i enters rail segment r at some time p. With this alteration in mind, the authors present the full TI formulation for the problem. So far in previous subsections we've discussed problems with LTL freight transportation, some of which also used at least one warehouse to ship in freight, consolidate it before shipping it out to the customers, but Ojha and Erera (2025) also present a DDD formulation for the cross-dock trailer scheduling problem with workforce constraints, that covers the best assignment of workers at individual loading doors at the consolidation terminal. The loading activities at each loading door, done by dock workers using forklifts, have to be completed by the target due date that enables the outbound trailers to depart to their final destination in time. For this problem, each cross-dock has a set of unloading doors U and a set of loading doors L with each loading door having a specific loading due date d_l (Ojha and Erera, 2025). With M being a set of permissible worker allocations and Q the total number of workers available, once assigned to the trailer the workers have to finish that unloading process until the end. With n as the total number of shipments on trailer i to the cross-docked loading door l, τ_{ul} represents the total time to transport the shipment (Ojha and Erera, 2025). Based on the flat network, the authors present a time expanded network denoted as $\check{G}=(\check{N},\check{A})$, where $\check{N}$ represents the set of nodes (u,t) over the planning horizon T with a discretization at the minute-level (Ojha and Erera, 2025). Additionally, all parameters that measure time are measured in units of Δ. Arcs of the form $((u,t),(u, t+\rho^{im}{}_u)$ represent the

decision to start unloading some trailer i with m workers at door u at time t, occupying the door until time $t+\rho^{im}_u$ until the unloading is complete (Ojha and Erera, 2025). Regarding another take on the scheduling problems, Pottel and Goel (2022) introduce their work based on time dependent scheduling with resource consumption, arguing that minimizing route distance could help creating routes that can be conducted without running out of the resources. The proposed time dependent activity scheduling problem combines time dependency and resource consumption possibility unrelated to activity durations, whereas the same problem TDASP with replenishments extends the problem with consumption dependent durations (Pottel and Goel, 2022). The time dependent duration and resource consumption of each activity i are given by functions $\tau_i(t_i)$ and $\rho_i(t_i)$ where each activity i must start within the time window (e_i, l_i) with the quantity given by Q. In the time expanded formulation, for each activity i and time point t_i a vertex (i,t_i) is created (Pottel and Goel, 2022). From the formulations from Ojha and Erera (2025) we see that their time expanded network with distinct node and arc sets resembles the previous SND and TSP problems and in the formulation of Pottel and Goel (2022) we once again see the return of the explicit time windows for each activity similar to the node based SND and TSP problems, whereas the TI formulation for train rescheduling problem was completely different from any other we've seen before due to the specific nature of the problem.

Another group of problems solved with DDD are routing problems, where Escobar-Vargas and Crainic (2024) solve a multi-attribute two-echelon location routing problem, Lagos et al. (2020) tackled the continuous-time inventory routing problem and then Lagos et al. (2022) extend the continuous time IRP with out-and-back routes. IRP presents a combinatorial problem that encompasses inventory management, vehicle routing and delivery scheduling, where the supplier makes replenishment decisions for product deliveries to the customers (Lagos et al. 2020). Due to continuous time, inventory and storage capacity are related to the product usage rate and initial inventory, which as a consequence of product consumption has precise scheduling and accounting the vehicle travel times. Products that need replenishments like liquid gas for example can have customers that need anywhere from one to multiple replenishments during the week which makes the use of continuous time IRP as the most appropriate and accurate representation of the system (Lagos et al. 2020). In order to represent the CIRP, the authors define for each customer i in the set N a local storage capacity C_i with a constant usage rate $\hat{u}_i$ and initial inventory I^0_i over a time horizon H. With a homogenous fleet of vehicles m with their respective capacities Q, each delivery route is set to start and finish at the depot (Lagos et al. 2020). Travel times $\tau_{ij}>0$ and costs $c_{ij}>0$ between every pair of location i and j for $i,j\in \mathcal{N}_0$ and zero inventory holding costs are known. With the assumption of instantaneous delivery and zero waiting cost, one vehicle at the time is allowed to wait at the customer location and make multiple deliveries while at the premises (Lagos et al. 2020). Over the course of the planning horizon, a vehicle can make multiple trips with the goal of finding a minimum cost delivery plan that ensures no customer ever rans out of product. Similar to the TSP problems, in routing problems trips or routes which are part of the delivery plan start and end at the company facility within the planned horizon with specific departure times, sequence of customer deliveries and delivery time and quantities delivered (Lagos et al. 2020). Since the problem specifically is in continuous time, to be able to present it, the authors use time discretization as an approximation approach of the CIRP, where times are restricted to integer values while also noting that the granularity of the discretization doesn't monotonically improve the solution quality (Lagos et al. 2020). Under the assumption that there exists such time discretization that the optimal solution to

the time expanded network formulation yields a continuous time optimal solution, the problem considers time-based parameters as integer. The discretization hence has T periods of length Δ where travel times are given as τ_{ij}/Δ and usage rate as $u_i = \hat{u}_i\Delta$ (Lagos et al. 2020). For the construction of a mixed integer linear programming formulation, for every timed node in the set $\mathcal{N}_T$ ={(0,T)}U($\mathcal{N}_0$,T) and every timed arc ((i,t-τ_{ij}),(j,t)) ∈ $\mathcal{A}_T$, additional binary variables are introduced to indicate travel between locations i and j, amount of product transported and inventory levels at time t. Building on the idea of Lagos et al. (2020), Lagos et al. (2022) modify the CIRP problem by introducing a simpler variant with out-and-back routes(CIRP-OB), where a vehicle departs from depot, visits only one customer and then returns back to the depot. Same as in Lagos et al. (2020), CIRP-OB problem has the same goal of finding the minimum cost delivery plan that ensures none of the customers run out of product over the time horizon, following the same notation for problems customers, quantities and inventory (Lagos et al. 2022). Since CIRP-OB is a special case of CIRP, the optimal solution is guaranteed as long as a feasible solution exists with a sufficiently fine time discretization in which a time indexed formulation finds an optimal solution. The time indexed formulation contains a node set $\mathcal{N}_T$ and an arc set $\mathcal{A}_T$ as well as time-based binary variables representing vehicle movement from i to j in time t, the amount of product that is transported, waiting at the customer location, inventory level and delivered product quantity in time t (Lagos et al. 2022). Now that we've seen the traditional IRP problems, Escobar-Vargas and Crainic (2024) present us the slightly modified problem called the Location Routing Problem which combines not only routing but also location selection with two echelons. The problem presented also considers multiple attributes such as time-dependent multicommodity origin-to-destination demand, time windows, limited storage capacity and synchronizations at intermediate facilities which at the end combined selection of facilities at two levels, routing and allocation of OD demands using sequences of synchronized routes (Escobar-Vargas and Crainic, 2024). To address the time component of the formulation, the model is built as a hybrid formulation, where facility nodes are represented at all and customers only at relevant time points, while a continuous time representation is used to capture the timing of vehicles at intermediate facilities and customers. With $\mathcal{G}^{ph}$= ($\mathcal{V}^{ph}$,$\mathcal{A}^{ph}$) representing a physical network consisting of nodes that represent suppliers, platforms, satellites, customers and garages, and arcs that represent direct links between locations with unit costs and travel times associated with each arc (Escobar-Vargas and Crainic, 2024). Problem also models customers that have hard service time windows and a service time as well as two homogenous vehicle fleets with limited capacities. Considering there are time windows in the model which create an interdependency of time and the flat network, the time-space model considers a Δ discretization which partitions time into Δ time periods and duplicates nodes at each time period and assigns departing and arriving times to an (i,j) arc (Escobar-Vargas and Crainic, 2024). Copies of different nodes are done in relevant times and the service time at customers is embedded at the inbound nodes for simplification of the formulation.

Besides all previously researched problems in relations to DDD, some authors also tackled solving the shortest path problem using the DDD algorithm. In this subsection, we'll go over the time dependent shortest path problem also covering the variant with minimum duration component. When it comes to transportation problems, one of the most important components of the algorithm handles finding the shortest path between two locations in a network especially when the travel time along the arc is set as a function of the travel start time (He et al., 2022). One of the key assumptions is FIFO, first-in first-out, which states that

it is impossible to arrive earlier to the end if we start travelling the arc later (He et al., 2018). The most basic version of the problem handles the minimum arrival time path problem, which gives a travel time at the origin and seeks the earliest time at destination possible, however the two variants He et al. (2022) present are minimum duration and minimum travel time time dependent path problems, which allow waiting at nodes. He et al. (2018) also presents work on the minimum duration variant, sometimes also called as minimum delay TDSPP, which finds the minimum difference between the arrival time at the sink and departure time at the source. In this context, DDD algorithm is modified to solve a sequence of shortest paths in a partially time expanded network, where each provides a better lower bound and allows the upper bound to be calculated. The algorithm ends when the length of the shortest path in the partially time expanded network matches the best found upper bound (He et al., 2022). For the purposes of defining SPP problem, the authors introduced a directed graph network $\mathcal{D}$=($\mathcal{N}$, $\mathcal{A}$) along with a time interval t ranging from zero to T and a piecewise linear travel times, that satisfies the previously mentioned FIFO property for every arc (i,j) (He et al., 2022, He et al., 2018). With node 1 representing the origin, node n represents the destination. For the timed path, each path consists of node and time pairs, where each node in the sequence has its associated time representing the beginning time of the arc traversal (He et al. 2022). For the path P to be feasible, the path starts with node one at the beginning of time period and finishes at the destination before the end of period T with a possibility of having some waiting nodes. However, when minimizing the arrival time at the destination, the arrival time of one arc is the departure time of the next arc and there is no benefit in having waiting arcs (He et al., 2022, He et al., 2018). Finding a timed path that departs the origin as late as possible in order to reach destination by a given time can be solved by precomputing reverse travel time functions which helps the algorithm solving time dependent SPP provide both the forward and backward shortest path tree (He et al. 2022). For an optimal path that contains an arc (i,j) there exists a departure time that occurs exactly at a breakpoint, where (i,t) as well as the timed nodes for the source (1,0) and sink (n,T) could be considered as a breakpoint. For both the MDP and MTTP, the DDD algorithm is developed to investigate a very small number of breakpoints in order to solve a problem to optimality with the precomputed inverse costs (He et al. 2022, He et al. 2018).

3. **Mathematical models**

Now that the problem was explained with regards to different approaches in the literature and the same physical network was extended to the time-expanded formulation to account not only for the physical network, but for time as well, in the next section we will touch upon some of the mathematical formulations for different problem variants presented in the literature. These formulations are important since they are the base of the relaxation that the DDD algorithm uses. We will go over couple of different mathematical models to be able to better understand the role of different formulations for the DDD algorithm we will see later in this paper.

3.1. **Service Network Design Problem**

In order to ensure that a feasible solution of SNDP on the time-expanded network can be converted to a feasible solution in real continuous time, travel time and time the commodities are available need to be round up and times that commodities are due need to be rounded down (Boland et al. 2017). The SND($\mathcal{D}_T$) is then defined as a service network design problem defined over a time-expanded network $\mathcal{D}_T$. Boland et al. (2017) provides a SND($\mathcal{D}_T$) model in a form which we can see in Figure 1 (left), where the objective function seeks to minimize the sum of fixed and variable costs, where it is assumed that holding freight in a location has a cost of zero. As we can observe, constraint (1-4) ensure that each commodity departs from its origin when it becomes available and arrives at its destination when it'd due and that trailer capacity is available for the commodities that are sent from location i to j at time t'. Last two constraints define the domain of the variable (Boland et al. 2017). When using a regular and fully time-expanded network, no approximations are introduced when τ_{ij}/Δ, e_K/Δ and l_K/Δ are integer in which case a feasible solution to SND($\mathcal{D}_T$) is also feasible in continuous time (Boland et al. 2017). This arguably makes the fully time-expanded network large for any practical instances. However, it is proven to be sufficient to include only time point in T that are determined by either direct travel time paths starting at the origin at the time commodity becomes available or of the form $e_{K+}\sum_{a}\in_P \tau_a$ for some commodity k and some path originating at o_K (Boland et al. 2017). This set of time points can still be large and still not enough to enable a solution to CTSNDP but by iteratively refining a set of time points T it can be proven that the solution of SND($\mathcal{D}_T$) for a carefully chosen arc set $\mathcal{A}_T$ can be converted to an optimal solution to CTSNDP.

Since in their model, Boland et al. (2017) consider the cost for holding freight at a certain location i to be zero, Hewitt(2019) on the other hand includes these cost in his objective function which is the slight adaptation of the SND($\mathcal{D}_T$) model presented by Boland et al. (2017), but later on in their paper they still assign zero holding cost since the correctness of DDD relies on the zero holding cost assumption. Besides the holding cost h_i, Hewitt (2019) adds additional constraint (6-10) which we can see in Figure 1 (right) in order to adapt DDD algorithm to the CTLPDP he is trying to solve. One of the requirements for a successful adaptation of the SNDP to CTLPDP in the In-Tree Network which seeks to model all shipments at the terminal with same destination terminal to travel next to the same terminal (Hewitt 2019). This is done with additional constraints (5-7) in the Figure 1(right) which define the tree structure, ensure that the path followed by each commodity follows the structure and the domain of the new variables. Another requirement to the CTLPDP is that the carrier has

predefined sets of paths that shipments can follow, where for each commodity k there is a set of paths $\mathcal{P}_k$ in the flat network from commodity's origin to its destination (Hewitt 2019). Additional constraints (8-10) in Figure 1 (right) ensure that commodity flow variables are synchronized with chosen paths, a path is chosen for each commodity and the domain of the commodity path variables is defined. With both of these operational rules in place, Hewitt (2019) defines CTSNDP with additional constraints as CTLPDP.

$$z(\mathcal{D}_{\mathcal{T}}) = \min\left\{ \sum_{((i,t),(j,\bar{t}))\in\mathcal{A}_{\mathcal{T}}} f_{ij} y_{ij}^{t\bar{t}} + \sum_{k\in\mathcal{K}} \sum_{((i,t),(j,\bar{t}))\in\mathcal{A}_{\mathcal{T}}} c_{ij} q_k x_{ij}^{kt\bar{t}} \right\}$$

subject to

$$\sum_{((i,t),(j,\bar{t}))\in\mathcal{A}_{\mathcal{T}}\cup\mathcal{H}_{\mathcal{T}}} x_{ij}^{kt\bar{t}} - \sum_{((j,\bar{t}),(i,t))\in\mathcal{A}_{\mathcal{T}}\cup\mathcal{H}_{\mathcal{T}}} x_{ji}^{k\bar{t}t} = \begin{cases} 1 & (i,t) = (o_k, e_k), \\ -1 & (i,t) = (d_k, l_k), \\ 0 & \text{otherwise;} \end{cases} \quad \forall k \in \mathcal{K}, (i,t) \in \mathcal{N}_{\mathcal{T}}, \quad (1)$$

$$\sum_{k\in\mathcal{K}} q_k x_{ij}^{kt\bar{t}} \leqslant u_{ij} y_{ij}^{t\bar{t}} \quad \forall ((i,t),(j,\bar{t})) \in \mathcal{A}_{\mathcal{T}}; \quad (2)$$

$$x_{ij}^{kt\bar{t}} \in \{0,1\} \quad \forall ((i,t),(j,\bar{t})) \in \mathcal{A}_{\mathcal{T}} \cup \mathcal{H}_{\mathcal{T}}, k \in \mathcal{K}; \quad (3)$$

$$y_{ij}^{t\bar{t}} \in \mathbb{N}_{\geqslant 0} \quad \forall ((i,t),(j,\bar{t})) \in \mathcal{A}_{\mathcal{T}}. \quad (4)$$

$$\sum_{j:(i,j)\in\mathcal{A}} z_{ij}^{d} \leq 1 \quad \forall i \in \mathcal{N}, d \in \mathcal{N}, \quad (5)$$

$$\sum_{t\in\mathcal{T}_i,\bar{t}\in\mathcal{T}_j:((i,t),(j,\bar{t}))\in\mathcal{A}_{\mathcal{T}}} x_{ij}^{kt\bar{t}} \leq z_{ij}^{d_k} \quad \forall (i,j) \in \mathcal{A}, k \in \mathcal{K}, \quad (6)$$

$$z_{ij}^{d} \in \{0,1\} \quad \forall (i,j) \in \mathcal{A}, d \in \mathcal{N}. \quad (7)$$

$$\sum_{t\in\mathcal{T}_i,\bar{t}\in\mathcal{T}_j:((i,t),(j,\bar{t}))\in\mathcal{A}_{\mathcal{T}}} x_{ij}^{kt\bar{t}} = \sum_{p\in\mathcal{P}_k(i,j)} v_p^k \quad \forall (i,j) \in \mathcal{A}, k \in \mathcal{K}, \quad (8)$$

$$\sum_{p\in\mathcal{P}_k} v_p^k = 1, \quad k \in \mathcal{K}, \quad (9)$$

$$v_p^k \in \{0,1\} \quad k \in \mathcal{K}, p \in \mathcal{P}_k. \quad (10)$$

Figure 1: **(left)** Photo of SND($\mathcal{D}_T$) model from Boland et al. (2017); **(right)** Photo of additional SND($\mathcal{D}_T$) constraints from Hewitt (2019).

$$\sum_{((j,\bar{t}),(i,t))\in A_{LT}\cup H_{LT}} y_{ji}^{\bar{t}t} = \sum_{((i,t),(j,\bar{t}))\in A_{LT}\cup H_{LT}} y_{ij}^{t\bar{t}} \quad \forall i \in N_c, (i,t) \in N_{\mathcal{T}}$$

$$\sum_{t\in T_i} \sum_{((j,\bar{t}),(i,t))\in A_{LT}} y_{ji}^{\bar{t}t} = \sum_{t\in T_i} \sum_{((i,t),(j,\bar{t}))\in A_{LT}} y_{ij}^{t\bar{t}} \quad \forall i \in N_{bb}.$$

subject to

$$\min \sum_{k\in K} \sum_{a\in\mathcal{A}_{\mathcal{T}}} v_a q^k x_a^k + \sum_{a\in\mathcal{D}_{\mathcal{T}}} f_a z_a, \quad (1a)$$

$$\sum_{a\in\delta_i^+} x_a^k - \sum_{a\in\delta_i^-} x_a^k = r_i^k, \quad \forall k \in K, i \in \mathcal{N}_{\mathcal{T}}, \quad (1b)$$

$$\sum_{k\in K: a\in\mathcal{A}_{\mathcal{T}}} q^k x_a^k \leq z_a u_a, \quad \forall a \in \mathcal{D}_{\mathcal{T}}, \quad (1c)$$

$$x_a^k \in \{0,1\}, \quad \forall k \in K, a \in \mathcal{A}_{\mathcal{T}}, \quad (1d)$$

$$z_a \in \mathbb{Z}_+, \quad \forall a \in \mathcal{D}_{\mathcal{T}}. \quad (1e)$$

Figure 2: **(left)** Photo of additional SND($\mathcal{D}_T$) constraints from Medina et al. (2019); **(right)** Photo of Interval-based SND($\mathcal{D}_T$) model from Marshall et al. (2021).

Another extension of an SNDP from Boland et al. (2017) can be observed in the following paragraph. Medina et al. (2019) seek to model their problem SNDRP which seeks to transport a set of shipments from a set of supplier locations to customer locations through a consolidation network, where each customer is assigned to a single terminal from which shipments are delivered and each supplier is assigned to a single terminal to which it sends goods. At a destination terminal, each vehicle is assigned with a set of shipments to deliver with a derived route that begins at the depot, visits the destination of each shipment at a time before it's due and then returns to the depot. This layer of the problem could be modeled as a Vehicle Routing Problem with Time Windows, but as the time at which the route may begin

is unknown, since it depends on the time at which the shipments are delivered to the terminal, the problem tries to synchronize the timing of the deliveries to the terminal with the departure of the local delivery routes that deliver the shipments (Medina et al. 2019). Medina et al. (2019) use the same mathematical formulation as Boland et al (2017) for SNDP but to ensure that the SNDRP is formulated correctly, they add some additional constraints to ensure accurate routing and corresponding costs of shipments both at the line-haul and local delivery routes. Medina et al. (2019) create a set of local arcs $\mathcal{A}_L$ which contains arcs that model movement from breakbulks to customer locations as well as arcs between two customer locations. To ensure that variables form such path Medina et al. (2019) add additional two constraints (6-7) as shown in Figure 2 (left) which ensure that every arrival at customer location is followed by a departure to another node and enforce that a number of routes starting from a breakbulk is the same as the number of routes arrival at the same breakbulk. This way an arc leaving a customer either goes to another customer or returns to the breakbulk itself, hence why the route starts and ends at the same breakbulk.

Another extension to the CTSNDP solved by Boland et al. (2017) is constructing an interval-based time-expanded network studied by Marshall et al. (2021). Key element of the DDDI algorithm is the calculation of the lower bound to an instance of CTSNDP by solving an integer program defined over a time-expanded network that is based on time intervals (Marschall et al. 2021). Since every node is termed as node-internal, meaning that for every node n there is a (t,t') interval of time connected to the node which assumes that any commodity dispatched from node n_1 at some time in the interval (t_1,t'_1) can be next dispatched from node n_2 at any time in the interval (t_2,t'_2). To model this interval-based CTSNDP, Marshall et al. (2021) present an integer program which can be seen in Figure 2 (right), where δ^+_i and δ^-_i denote the set of timed-arcs leaving and entering node-interval I, respectively. The objective is to minimize the total cost, both variable and fixed cost, with respect to certain constraints (1b-1e). The flow constraints ensure that each commodity leaves its origin node during some interval which contains e^k and reaches its destination node during some interval that contains l^k (Marshall et al. 2021). Consolidation constraints require enough vehicles to be assigned to each dispatch-arc at some time during the time interval of the dispatch-arc's first node-interval and the last two constraints define the variable domains.

Now that we've covered and explained the basic SNDP mathematical formulation and its extensions from different papers, without looking too much into detail, other previously mentioned SNDP variants' integer programming models will be summarized as follows by pointing some special characteristics and formulation the other authors used.

The SNDRP problem consist of planning the pick-up and delivery of each commodity by selecting routes from a predefined set and assigning trucks so that the total transportation and transfer costs are minimized while taking capacity and time window constraints into account (He et al. 2023). Based on their route-time expanded network, He et al. (2023) solve the SNDRP problem with a mathematical formulation where similarly to the previously mentioned papers they try to minimize fixed and variable transfer costs for all commodities and hubs. Besides the usual constraints defining pick-up and delivery of each commodity, flow conservation constraints, capacity constraints on the transportation arcs and the domain defining constraints, He et al. (2023) also define constraints that ensure that the commodity enters the hub node with at least one physical route entering that same hub and that the commodity leaves the hub node.

In other formulations there was never a question about the vehicles transporting the freight throughout the network, since the assumption was that the fleet was homogeneous.

However, Scherr et al. (2020) study a variant of the SNDP whit mixed autonomous fleet, where they adapt the SNDP problem to account for the zone suitable for autonomous vehicle (AV) driving on its own and the zone where manually operated vehicles (MV) need to pull AVs in a platoon. In their adapted, more complex mathematical formulation they seek to minimize the total cost while respecting constraints for capacity of MVs and AVs, platooning, design-balance constraints that ensure the MV and AV connections to the routes are feasible and variable defining constraints. Since there is a restriction of travel for AVs in certain zones, additional constraints seek to model the disaggregate coupling and further tighten the transportation capacity (Scherr et al. 2020). In their additional pre-processing program, they seek to minimize the transshipment of commodities between vehicles by measuring the number of vehicles a commodity is loaded on. They introduce constraints that load commodities to MVs and AVs, ensure commodity flow, enforce capacity restriction for MVs and AVs, ensure platooning of AVs with MVs on MV arcs, time related constraints and variable defining constraints (Scherr et al. 2020). With these programs they adapt the general SNDP setting to account for the mixed vehicle fleet, time and different available routing zones.

As mentioned before, one of key assumptions for the successful execution of DDD algorithm was the fact that holding costs were zero allowing for waiting times at certain nodes and easier consolidation (Boland et al. 2017). However, in case the holding costs aren't disregarded then the cost of CTSNDP-HC solution can be computed as a function of the holding and consolidation plans (Shu et al. 2024). In order to properly account for holding cost and consolidation opportunities, Shu et al. (2024) incorporated holding cost into their objective function and where holding arcs allow the commodity to arrive early at its destination and depart later from its origin. One additional constraint in the model defines the variable w^k_i computed as the difference between departure and arrival time which in combination with h^k_i calculates the total holding cost in the objective function, which the problem tries to minimize (Shu et al. 2024). In case where holding costs were equal zero, the dispatch times of a path P could be shifted to be as early as possible without any changes in total cost or consolidation plans, but for CTSNDP-HC because of the presence of non-zero holding costs in the objective function this statement is not valid anymore (Shu et al. 2024). Shu et al. (2024) hence define, given a flat solution S, an LP to determine optimal departure times for each commodity k in order to show that a complete time-expanded model exists for the CTSNDP-HC problem, which wasn't immediately straight forward if they used just the formulation for CTSNDP from Boland et al.(2017).

So far, we've covered different studies which focused on models where times were encoded on a node-basis. In order to improve solving problems which have a lot of different departure times at nodes, Van Dyk and Koenemann (2024) enhanced and modeled an approach where the departure times were determined on the arc level rather than the node level, where arcs are grouped into subsets that have the same set of departure times in each iteration. They model the problem SND-RR where each commodity has a designated feasible network in space through which it must be routed. In order to formulate the arc-based model, Van Dyk and Koenemann (2024) create an auxiliary graph G, where each node in $\mathcal{D}$ is replicated, for each part in the partition $\delta^+(v)$, which allows to store the set of departure times for each arc in that part. The integer program for the proposed SND-RR is defined analogous to the formulation from Boland et al. (2017) with the objective of minimizing the sum of fixed and variable cost. Constraints ensure that each commodity is assigned a feasible trajectory and for each timed arc there is sufficient capacity to accommodate scheduled trajectories at that timed arc (Van Dyk and Koenemann 2024).

3.2. **Traveling Salesman Problem**

Based on the flat network and the time-expanded network formulations, authors propose formulations to solve TSP problems that are the base of DDD algorithm. For the most basic case of a TSPTW, the objective is to minimize the total sum of costs of using certain arcs (Boland et al. 2017b). With $\delta^{+}_{()}$ and $\delta^{-}_{()}$ denoting a set of timed arcs that are coming into and going out of a timed node, the constraints try to force at least one unit of flow to leave the city i for all $i \in \mathcal{N}$ and regulate the balance flow to ensure that the flow into a timed note is followed by the flow out of the timed node at all cities except the depot. The model imbeds the assumption that any waiting in the tour occurs prior to the start of the time window, where the vehicle will depart the customer as soon as he arrives or latest at the start of the time window of that customer (Boland et al. 2017b). The formulation from Boland et al. (2017b) was found through experiments that it doesn't need subtour elimination constraints since the LP relaxation yields a very strong lower bound.

To formulate an integer program for TD-TSPTW from Vu et al. (2019) the program seeks to satisfy the objective function similar to the one shown in Figure 3 (left) with the constraints similar to the ones from Boland et al. (2017b) that ensure the vehicle arrives once at each location during the time window and that the vehicle departs every location at which it arrives with the exception of depot. Since the paper from Vu et al. (2019) seeks to optimize on two objectives, one being the makespan objective and the other the duration objective, the paper sets the objective functions for the makespan objective to count the cost only when returning to the depot and for the duration objective the makespan cost function is then modified to calculate cost as a difference between the end time point of the tour and the beginning of the tour. To make sure the impact of time windows and time dependent travel times are accounted for, these are embedded into the time-expanded network by removing the arcs that arrive at the node outside of its time window. The time dependency is then also embedded into the arcs through the choice of t' (Vu et al. 2019).

$$c_{a=((i,t),(j,t'))} = \begin{Bmatrix} -t, & \text{for } i = 0 \\ t + \tau_{ij}(t), & \text{for } j = 0 \\ 0, & \text{otherwise} \end{Bmatrix} \quad \forall a \in \mathcal{A}. \tag{1}$$

With these decision variables and this notation a formulation of the TD-MTDP is as follows:

$$\text{minimize} \sum_{a \in \mathcal{A}} c_a x_a \tag{2}$$

subject to

$$\sum_{t=e_i}^{l_i} \sum_{a \in \delta^{+}_{(i,t)}} x_a = 1, \quad \forall i \in N, \tag{3}$$

$$\sum_{a \in \delta^{+}_{(i,t)}} x_a - \sum_{a \in \delta^{-}_{(i,t)}} x_a = 0, \quad \forall (i,t) \in \mathcal{N},\ i \neq 0, \tag{4}$$

$$x_a \in \{0,1\}, \quad \forall a \in \mathcal{A}. \tag{5}$$

$$\text{minimize} \sum_{k \in N \setminus \{0\}} (t_k - t_0) \tag{6}$$

Formulated on a time-expanded network, $\mathcal{D} = (\mathcal{N}, \mathcal{A})$, the objective (6) can be expressed as follows:

$$\text{minimize} \sum_{((i,t),(j,t')) \in \mathcal{A} \| j \neq 0} \max(e_j, t + \tau_{ij}(t)) x_{((i,t),(j,t'))} - (|N| - 1) \sum_{((0,t),(i,t'))} t x_{((0,t),(i,t'))} \tag{7}$$

Figure 3: **(left)** Photo of the TD-MTDP formulation; **(right)** formulation of the TD-DMP from Vu et al. (2022)

Vu et al. (2022) also focus of the formulation with the minimum duration objective function that we can see in Figure 3 (left) that minimizes the total time of the tour while also respecting the same constraints of departing the customer once during their TW and the flow conservation constraints (constraints 3 and 4, Figure 3 (left)). What the assumption of the

problem states is that the time is discretized into a finite number of time points. However authors of this paper also study a TD-DMP problem for which the previous formulation is then extended as we can see in the Figure 3 (right) to compute the arrival at customer j at first term in constraint (7) and the departure time from the depot , then aggregated for all customers. This adaptation of TSP seeks to minimize the aggregated time of how long it takes to visit each individual customer when the vehicle starts from the depot while also respecting the constraints (3-5) (Vu et al. 2022).

3.3. **Scheduling problem**

Mathematical models for the previously introduced scheduling problems are also based on the time expanded formulations of the problem, where the initial flat network was adapted to account for the time component. For the train rescheduling optimization problem, the objective function seeks to minimize the total cost of selected paths across all trains and resources (Croella et al. 2024). The first constraint makes sure that each train gets only one valid path on each of its route segments, while the second constraint prevents two conflicting paths, that might be used by two trains overlapping resources, being chose at the same time. If path p for train i and path p' for train j cannot happen safely together then this constraint ensures that at most one of them is selected as shown in Figure 4 (left). As discussed before, the decision variable is binary, choosing or not choosing a path without any fractional values. Figure 4 (right) also depicts another mathematical formulation for a scheduling problem, where the objective function is to minimize the total delay with respect to the target due date at the loading doors (Ojha and Erera, 2025). The constraint (2b) for the cross dock scheduling ensure that the trailer starts processing after its arrival time and at an unloading door at exactly one time point using a certain number of workers, whereas constraint (2c) determines delay with respect to the target due date at the loading door (see Figure 4 (right)). While constraint (2d) ensures that at most one trailer can be processed at the unloading door at given time, constraint (2e) that the maximum worker availability is respected at all times and the other constraints again define the domain of variables.

$$
\begin{aligned}
&\min && \textstyle\sum_{i\in I}\sum_{r\in R_i}\sum_{p\in \Pi^{ir}} c^{ir}(p)\cdot x_p^{ir} && \\
&s.t. && && \\
&(1) && \textstyle\sum_{p\in \Pi^{ir}} x_p^{ir} = 1, && i\in I,\ r\in R_i \\
&(2) && x_p^{ir} + x_{p'}^{jq} \le 1, && p\in \Pi^{ir} \text{ incompatible with } p'\in \Pi^{jq} \\
& && && \Pi^{ir},\Pi^{jq}\in \Pi \text{ and } (i,r,p)\neq(j,q,p') \\
& && x_p^{ir}\in\{0,1\} && i\in I,\ r\in R_i,\ p\in \Pi^{ir}.
\end{aligned}
$$

$$
\begin{aligned}
\text{Minimize} \quad & \sum_{\ell\in L} z_\ell && && (2a)\\
\text{s.t.} \quad & \sum_{m\in M}\sum_{\substack{(u,t)\in \widehat{N}\\ t\ge r_i}} x_{ut}^{im} = 1, && \forall i\in I && (2b)\\
& \sum_{m\in M}\sum_{\substack{(u,t)\in \widehat{N}\\ t\ge r_i}} (t+\rho_u^{im})x_{ut}^{im} - z_\ell \le d_\ell, && && \\
& && \forall i\in I, \ell\in D_i && (2c)\\
& \sum_{a_{ut'}^{im}\in \widehat{A}_{ut}} x_{ut'}^{im} \le 1, && \forall (u,t)\in \widehat{N} && (2d)\\
& \sum_{u\in U}\sum_{a_{ut'}^{im}\in \widehat{A}_{ut}} m x_{ut'}^{im} \le Q, && \forall t\in \widehat{T} && (2e)\\
& x_{ut}^{im}\in\{0,1\}, && \forall i\in I, m\in M, (u,t)\in \widehat{N}, t\ge r_i && \\
& && && (2f)\\
& z_\ell \ge 0, && \forall \ell\in L && (2g)
\end{aligned}
$$

Figure 4: (left) Train rescheduling formulation from Croella et al. (2024); **(right)** XDTS-W problem formulation from Ojha and Erera (2025)

For the time dependent scheduling problem, the objective function is to minimize the completion time (Pottel and Goel, 2022). Constraints ensure the start of one activity is after the end of the previous one, the cumulative resource consumption doesn't exceed the limited available amount and the last constraint defines the domain of the variables. The model also assumes the completion time functions are non-decreasing, meaning that completion times satisfy the FIFO property, which we have seen in other previous cases of time dependent problems the emphasis and the importance of the FIFO assumption (Pottel and Goel, 2022). Similar as before, since the resource consumption depends on congestion levels which vary throughout the day, it may be beneficial to start the activities later.

3.4. **Routing problem**

After problem introduction for routing problems, their space network also known as 'flat' network is then extended to contain time component in the index, making the new time-space network which serves as a base for the introduction of a mathematical formulation. Regarding multi attribute two echelon location routing problem, the objective is very clearly trying to minimize total cost of the system, including fix cost of selecting facilities and variable travel cost of the vehicles (Escobar-Vargas and Crainic, 2024). Over thirty constraints in the mathematical formulation of the problem make sure the two echelons are connected, customer is visited at least once, vehicle starts and ends at the depot, flow conservation constraints, ensure multiple visits to satellites in different time periods and no split demands, enforce spatial and temporal flow synchronization and maximum capacity of the vehicle and the facility, handle the arrival and departure times and regulate the waiting time (Escobar-Vargas and Crainic, 2024). Something similar to this complex mathematical problem formulation was already seen in the work of Scherr et al. (2020), where they also tackle different driving zones for vehicles, but also platooning, non-homogenous fleet and different capacities which we could categorize under multi attribute side of the problem. They adapt their original SNDP problem to account for the additional attributes, making the formulation and following relaxation large, where same can be said in the case of Escobar-Vargas and Crainic (2024). However, slightly simpler mathematical formulations were presented by Lagos et al. (2020) and Lagos et al. (2022) in their works regarding continuous time inventory routing problem and CT-IRP with out-and-back routes, respectively. As we can see from the Figure 5, both formulations have cost minimization as their objective with a slight difference in the formulation. Since in the IRP with out-and-back routes, each route is represented only once, travelling from depot to one customer and back, in the objective formulation only one half of the path is represented, hence why there is a factor of two that makes sure that the cost in both directions is accounted for (Figure 5 (left)). In the regular IRP from Lagos et al. (2020), one vehicle can visit multiple customers before returning to the depot and the total cost is the sum of all small parts of the route. In Figure 5 (right), constraints ensure vehicle flow balance and all vehicles return to the depot at the end of planning horizon, only one vehicle visits a customer at the same time, product flow balance, model product usage at the customer and ensure the sufficiency and the maximum capacity of the inventory. Since in this case, when all travel times are positive and the network is acyclic, there is no need for formal subtour elimination constraints (Lagos et al. 2020). For the case of IRP-OB, the constraints in Figure 5 (left) do the same job as in Figure 5 (right). However, the difference is in the indexing of the binary variables, where we see a zero representing the depot and i representing a

customer, which makes sure that the vehicle that departs the depot visits one customer only and then returns back, without the possibility of inter-customer connections.

$$\min \sum_{i\in N}\sum_{k\in\mathcal{T}_0} 2c_i x_{0i}^k,$$

$$\text{s.t.}\quad x_{i0}^k + x_{ii}^k = x_{ii}^{k-1} + x_{0i}^{\ell}, \qquad i\in N, k\in\mathcal{T}_i, (0,\ell)\in\delta^-(i,t_i^k), \tag{1a}$$

$$x_{00}^k + \sum_{i\in N} x_{0i}^k = x_{00}^{k-1} + \sum_{(i,\ell)\in\delta^-(0,t_0^k)} x_{i0}^{\ell}, \quad k\in\mathcal{T}_0\setminus\{0,K_0\}, \tag{1b}$$

$$x_{00}^0 + \sum_{i\in N} x_{0i}^0 = m, \tag{1c}$$

$$w_{0i}^{\ell} + w_{ii}^{k-1} - w_{ii}^k = y_i^k, \qquad i\in N, k\in\mathcal{T}_i, (0,\ell)\in\delta^-(i,t_i^k), \tag{1d}$$

$$x_{ii}^k + x_{i0}^k \le 1, \qquad i\in N, k\in\mathcal{T}_i, \tag{1e}$$

$$0 \le w_{ij}^k \le Q x_{ij}^k, \qquad a_{ij}^{k\ell}\in\mathcal{A}^{\mathcal{T}}, \tag{1f}$$

$$z_i^k = z_i^{k-1} + y_i^k - \bar{u}_i^k, \qquad i\in N, k\in\mathcal{T}_i\setminus\{0\}, \tag{1g}$$

$$z_i^0 = I_i^0 + y_i^0, \qquad i\in N, \tag{1h}$$

$$\bar{u}_i^{k+1} \le z_i^k \le C_i, \qquad i\in N, k\in\mathcal{T}_i, \tag{1i}$$

$$0 \le y_i^k, \qquad i\in N, k\in\mathcal{T}_i,$$

$$x_{ij}^k \in \{0,1\}, \qquad a_{ij}^{k\ell}\in\mathcal{A}^{\mathcal{T}}.$$

$$\min \sum_{(i,t)\in\mathcal{N}^{\mathcal{T}}}\sum_{j\in\delta^+(i,t)} c_{ij}x_{ij}^t$$

$$\text{s.t.}\quad \sum_{j\in\delta^+(i,t)} x_{ij}^t = \sum_{j\in\delta^-(i,t)} x_{ji}^{t-\tau_{ji}} \qquad (i,t)\in\mathcal{N}^{\mathcal{T}}\setminus\{(0,0),(0,T)\} \tag{2a}$$

$$\sum_{i\in\delta^-(0,T)} x_{i,0}^{T-\tau_{i,0}} = m \tag{2b}$$

$$\sum_{j\in\delta^+(i,t)} x_{ij}^t \le 1 \qquad (i,t)\in\mathcal{N}^{\mathcal{T}} \tag{2c}$$

$$\sum_{j\in\delta^-(i,t)} w_{ji}^{t-\tau_{ji}} - \sum_{j\in\delta^+(i,t)} w_{ij}^t = y_i^t \qquad (i,t)\in\mathcal{N}^{\mathcal{T}}, i\neq 0 \tag{2d}$$

$$0 \le w_{ij}^t \le Q x_{ij}^t \qquad (i,t)\in\mathcal{N}^{\mathcal{T}}, j\in\delta^+(i,t) \tag{2e}$$

$$z_i^t = z_i^{t-1} + y_i^t - u_i \qquad i\in N, t\in\mathcal{T}\setminus\{0\} \tag{2f}$$

$$z_i^0 = I_i^0 + y_i^0 \qquad i\in N \tag{2g}$$

$$u_i \le z_i^t \le C_i \qquad i\in N, t\in\mathcal{T} \tag{2h}$$

$$0 \le y_i^t \qquad i\in N, t\in\mathcal{T}$$

$$x_{ij}^t \in \{0,1\} \qquad \big((i,t)(j,t+\tau_{ij})\big)\in\mathcal{A}^{\mathcal{T}}.$$

Figure 5: (left) CT-IRP-OB formulation from Lagos et al. (2022); **(right)** CT-IRP formulation from Lagos et al. (2020)

4. **Problem relaxation properties**

4.1. **Service Network Design Problem**

The authors (Boland et al. 2017) define the algorithm for solving the CTSNDP as a dual ascent procedure, because it repeatedly solves the relaxation of the CTSNDP and checks to see if the solution of the relaxation can be converted to a solution od CTSNDP of the same cost. If the solution to the relaxation can be converted then the algorithm terminates with an optimal solution, if not then the algorithm refines the relaxation in another iteration. One of the main computational challenges is solving the fully time expanded network which may become very large and yield unreasonable solve time which is mitigated by solving the so called partially time-expanded network, where not every location is represented in every time point (Vu et al. 2022). The relaxation itself is an instance of the SNDP defined over a partially time-expanded network(Medina et al. 2019). Boland et al. (2017) refer to $\mathcal{D}_T$ as a partially time-expanded network since T_i could only contain a small subset of time points for $i \in \mathcal{N}$ and this set of time may differ for different locations. DDD scheme outlined in Figure 6 uses a concept of a previously mentioned partially time-expanded network. The DDD initially constructs sets $\mathcal{N}_T$,$\mathcal{H}_T$ and $\mathcal{A}_T$, and then modifies them in each iteration of the algorithm (Shu et al. 2024). The algorithm starts by initializing the partially time-expanded network $\mathcal{D}_T$=($\mathcal{N}_T$,$\mathcal{H}_T$ U $\mathcal{A}_T$) and at each iteration a relaxation model is solved to compute the dual bounds (valid lower and upper bound) (Shu et al. 2024). The upper bound is updated based on the lower bound solution obtained by solving the relaxation model, where in each iteration the partially time-expanded network is refined until the gap between the two bound is under the optimality tolerance. In case the lower bound solution isn't optimal for the CTSNDP the network is furthermore refined by adding new time points and updating the corresponding arcs (Shu et al. 2024). Once after a finite number of iterations a solution can be successfully converted to a feasible solution of CTSNDP of the same cost this solution is then at the same time also an optimal solution (Boland et al. 2017).

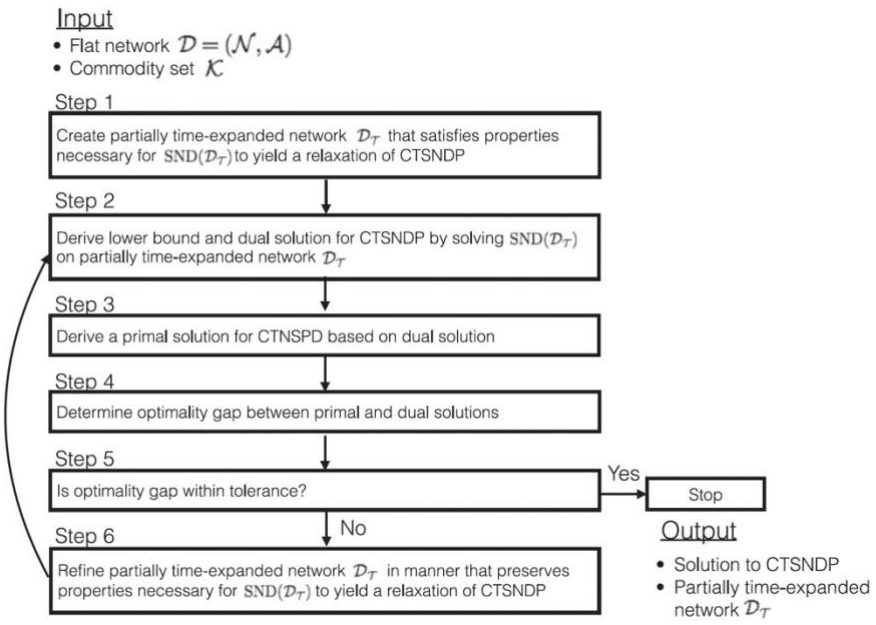


Figure 6: Dynamic discretization discovery scheme based on Hewitt (2019)

The partially time-expanded networks introduced by Boland et al. (2017) have four important properties which make sure that the SND($\mathcal{D}_T$) is a relaxation of the CTSNDP. In the following part, these four initial properties as well as some additional properties introduced for other variants of the problem will be introduced first before going over other steps of the algorithm. Following properties from Boland et al. (2017) need to be satisfied in order to have a relaxation of CTSNDP:

Property 1 states that for all commodities k$\in \mathcal{K}$ the nodes (o_k, e_k) and (d_k, l_k) are in $\mathcal{N}_T$.

Property 2 states that every arc ((i,t),(j,t')) $\in \mathcal{A}_T$ has t'$\leq$ t+ $\tau_{ij.}$

Property 3 states that for every arc a=(i,j) $\in \mathcal{A}$ in the flat network $\mathcal{D}$ and for every node (i,t) in the partially time-expanded network $\mathcal{D}_T$=($\mathcal{N}_T$,$\mathcal{H}_T$ U $\mathcal{A}_T$) there is a time copy of a in $\mathcal{A}_T$ starting at (i,t).

Property 4 states that if arc ((i,t),(j,t')) $\in \mathcal{A}_{T,}$ then there doesn't exist a node (j,t'') $\in \mathcal{N}_T$ with t'<t''$\leq$ t+ $\tau_{ij.}$

The first property ensures that there exist a source and sink node for every commodity k in $\mathcal{D}_T$ (Hewitt 2019, Scherr et al. 2020). Properties two and three are mainly responsible and ensure that SND($\mathcal{D}_T$) is indeed a relaxation of CTSNDP by implying that $\mathcal{D}_T$ has the early arrival property and that we work with time copies of an arc that are either of the correct length or shorter than their physical travel time, meaning we underestimate travel times (Boland et al. 2017; Scherr et al. 2020; Hewitt 2019, He et al. 2018). For a better visualization of this property we refer to the Figure 7 (Medina et al. 2019). Figure 7(a) shows a simple flat network which consists of four nodes, three arcs connecting these nodes and physical travel times listed below every arc. In Figure 7 (b) however we depict a short arc since the travel time in the time-expanded network for arc j->k is 'too short' and underestimates time. Both i-> j-> k and j-> k->l are early arrival paths since the arc j->k is 'too short' (Medina et al. 2019). Authors (Boland et al. 2017) then proceed to extensively prove that in fact by satisfying property one and the early arrival property, SND($\mathcal{D}_T$) is in fact a relaxation of CTSNDP. If $\mathcal{D}_T$ also satisfies property four then we say that $\mathcal{D}_T$ has the longest-feasible-arc property, meaning there doesn't exist such feasible arc in $\mathcal{D}_T$ that is longer than the current arc connecting two nodes meaning that the arc that originates from some node is the longest of all others feasible arcs but at most as long as the correct travel time (Boland et al. 2017; Scherr et al. 2020, Shu et al. 2024).

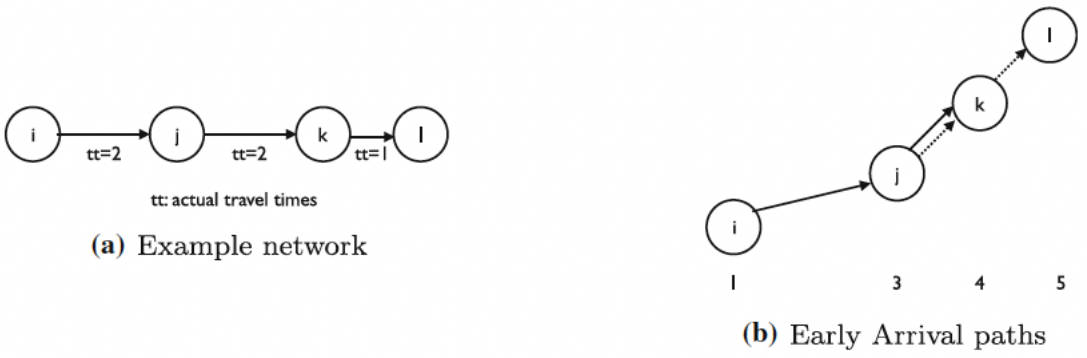


Figure 7: Example network and early arrival property from Medina et al. (2019)

Largest optimal objective function value is induced by the partially time-expanded network $\mathcal{D}_T$, that among other networks that satisfy the early arrival property also satisfy the longest-feasible-arc property (Boland et al. 2017).

These four properties are mainly applicable for a wide variety of SND problems, for more variants though there needs to be introduced some additional properties or valid inequalities to ensure and further strengthen the relaxation of the problem variant. One such case is when we have holding costs that are included in the objective function. Besides the original four properties, an additional set of valid inequalities was introduced to further strengthen SND-HC($\mathcal{D}_T$) (Shu et al. 2024). For each commodity k that's passing through the terminal, the difference between the lower bound on the true departure time and the upper bound on the true arrival time provides a lower bound on the holding time, which is particularly important in the initial partially time-expanded network (Shu et al. 2024).

In order to account for the problem characteristics that the vehicle that departs from an external zone must return to any external zone at the end of planning, Scherr et al. (2020) add two additional properties. Properties five and six state that there must be at least one path from one of the external zones at the beginning on the planning horizon towards any node in $\mathcal{D}_T$ and that $\mathcal{D}_T$ must include arcs that return the vehicles to an external zone not later than t_{MAX}, respectively. This way the partially time-expanded network for SNDMAF includes auxiliary arcs that "wrap around" connecting the last with the first node which is the external zone (Scherr et al. 2020).

For an interval-based variant of CTSNDP there is no explicit use of the proposed properties, but rather a set of five conditions that describe the time-expanded commodity network (Marshall et al. 2021). The five conditions define the dispatch and holding arcs, the path from o^k to d^k and early-late time window constraints.

In their arc-dependent variant, Van Dyk and Koenemann(2024) conclude the existence of a relaxation based on the first two properties analogous to the ones presented in Boland et al. (2017), that are modified for an arc based formulation of the problem. Similar to the previous article, He et al. (2023) also deal with arc-based formulations, hence why they introduced seven properties that prove that the solution of the partial time-expanded network is a relaxation of the CTSNDRP problem. The properties of the route-time expanded network state that the earliest time of every route can be computed according to available times of every commodity and if a route has no timed copy in the route-time-expanded network then this route cannot carry any commodity and should be eliminated from the set of routes since it is useless (He et al. 2023). Other properties state that every incoming timed route copy visiting a hub is linked to a central hub node at the same time when the hub is visited by this route, early outgoing of a hub by a timed route copy before the actual visiting time is possible, the outgoing arc of a central hub copy can be linked to the latest timed route copy only that is still earlier than the actual transfer time and that for every hub visited in multiple time points we can build holding arcs. We can see that in the route-time expanded network, the properties that were presented for node-based problems earlier were adapted slightly to account that the time is actually encoded on the arcs and not nodes. This algorithm is different form the one in Boland et al. (2017) since only some of the transfer arcs have a travel duration that may be shorter than the real duration or even be negative, whereas in Boland et al. (2017) and other similar problems there is a possibility that every arc can be a 'short arc' (He et al. 2023). The assumptions of the algorithm are that travel times of transportation arcs are fixed to their exact duration, transfer arcs associated to the central hub have zero duration, transfer arcs from a central hub to a timed route copy of a hub node can have

shorter duration, linking arcs from a timed origin node to a timed route copy of the same origin can have a shorter duration and linking arcs from a timed route copy of a delivery node to the timed destination node can only have positive or zero duration, to ensure that the delivery cannot be later than the latest delivery time (He et al. 2023).
We see that for all papers above, if $\mathcal{D}_T$ is a partially time expended network and satisfies properties related to their respective problems then the SND($\mathcal{D}_T$) is a relaxation of the original problem (Boland et al. 2017; Hewitt 2019; Scherr et al. 2020; He et al. 2023; Van Dyk and Koenemann 2024; Medina et al. 2019; Marshall et al. 2021, Shu et al. 2024).

4.2. **Traveling Salesman Problem**

For the case of TSP problems, a relaxation with a partially time expanded network needs to be formulated as well. The TSPTW formulated on the partially time expanded network $\mathcal{D}_T$ and optimized with respect to the cost is then TSPTW($\mathcal{D}_T$) (Vu et al. 2022). Similar to SND cases, TSP problems also define properties necessary for $\mathcal{D}_T$ that ensure TSPTW($\mathcal{D}_T$) is a relaxation of the original problem. Generally, for all TSP problems, the properties optimize on every i ∈ $\mathcal{N}$ compared to the SND problems where properties had to be satisfied for every commodity k∈ $\mathcal{K}$, which is given by the structure of the problems itself. While SNDP tried to find the minimum cost for delivering commodities to customer, TSP try to find minimum cost of visiting every customer, which is then reflected in the properties and algorithm in general. Hence why the properties above are then adjusted to account for the change. One of the properties for a TSP states that for every i ∈ $\mathcal{N}$ there must be a timed node for the beginning and the end of time window, similar to the SNDP where for every commodity there exist a source and sink node for every commodity (Boland et al. 2017b; Vu et al. 2019; Vu et al. 2022). Similarly, there is a property that ensure travel times on arcs are either accurate or underestimated for every i ∈ $\mathcal{N}$ with t'≤ max {ej, t+ τ_{ij}}, where we say the arc is short if t'≤ t+ τ_{ij} (Boland et al. 2017b; Vu et al. 2019; Vu et al. 2022). In case of Boland et al. (2017b), if $\mathcal{D}_T$ satisfies the two properties above, then the value of TSPTW($\mathcal{D}_T$) is a lower bound of the value of TWPTW. Another important property of TD-TSPTW variants states that for every node i ∈ $\mathcal{N}$ $e_i \leq t \leq l_i$ holds, which strictly models that every for every customer only times within time windows of a location are represented in $\mathcal{N}_T$(Vu et al. 2019; Vu et al. 2022). Also, for the variants with time dependent travel time, one additional property states that the cost is underestimated by considering the smallest cost of dispatching the vehicle on travel arc (i,j) between time t and the latest time at which such dispatch could occur. For TSP problems, the previously introduced longest-arc-property remains the same as in SND problems and with it, the partially time expanded network induces an instance of TSPTW($\mathcal{D}_T$) with the largest objective function value (Boland et al. 2017b; Vu et al. 2019; Vu et al. 2022). Regarding these properties, the algorithm can map a path P into a path P' which with respect to these properties doesn't need to begin and end at the depot which is critical to show that the relaxation of the problem exists (Vu et al. 2022). Even though the fully time expanded network may contain a large number of nodes, it can still be represented in a partially time expanded network in which departures occur at earlier times and the cost of this solution cannot exceed the cost of the solution on a fully time expanded network. Same as in SND, in case a solution on a partially time expanded network contains only arcs of the correct length, then this solution is the optimal solution of TSPTW ($\mathcal{D}$) problem (Vu et al. 2019; Vu et al. 2022).

4.3. **Scheduling Problem**

In case of scheduling problems, the authors introduce a relaxation to the time expanded network before the start of the DDD algorithm. The XDTS-W problem considers a partial network $\mathcal{G}=(\mathcal{N},\mathcal{A})$ that is constructed in such a way that it satisfies a set of properties, that ensure any solution of the XDTS-W can be mapped to a feasible solution in $\mathcal{G}$ with the cost not greater that the cost of the feasible solution of XDTS-W (Ojha and Erera, 2025). The properties for this problem are adapted to the nature of the problem in such a way that the first property resembles the one presented before for SND problems regarding nodes, which states that the node set $\mathcal{N}$ minimally contains all nodes of type (u,r_i) where r_i is the planned arrival time of the trailer. Where the property one of SND adds both the nodes at the end and start of the time window, this adapted property adds minimally timed nodes at the planned arrival time of the trailer. Second property is concerned with arc creation based on the service duration. For every trailer, door and worker it creates arcs from each time point t to the future time point representing feasible unloading durations and only adds arcs if those future time point exist in the node set $\mathcal{N}$ (Ojha and Erera, 2025). This property is important in showing how long a trailer can occupy a door, when the unloading process starts and finishes and how time restraints interact with the schedule. The last property states that all door use the same synchronized time grid which makes conflict modeling easier and the overall time expanded network simpler (Ojha and Erera, 2025). In the XDTS-W-LB from a partial network $\mathcal{G}$ properties one and two ensure that a possible assignment exists and begins unloading trailer i at its arrival with some workers and the assignment to an arc in $\mathcal{G}$ creates an optimistic mapping of the unloading completion time of the trailer, not later than the actual completion time, which resembles the properties in SND and TSP problems where times on arcs could be underestimated as well (Ojha and Erera, 2025). If $\mathcal{G}$ doesn't satisfy the third property, the XDTS-W-LB is shown not to yield a lower bound for the XDTS-W problem.

Similarly, Pottel and Goel (2022) also introduce a set of properties that generate and modify partially time expanded networks, where the set of vertices is a subset of the fully time expanded network. The first property is the same as the one in SND and TSP problems, where for each activity i, at least one node for the start and end of time window is included in the node set of the partially time expanded network (Pottel and Goel, 2022). The second property includes, for every vertex its immediate successor, while the third property assigns to every vertex a lower bound on the actual resource consumption, which in a way underestimates the resource consumption between time points t and t'. The optimal solution of a TDASP generated on a partially time expanded network satisfying the properties 1-3 represents a lower bound to the optimal solution of the fully time expanded TDASP with discretized times (Pottel and Goel, 2022). This assumption is the same as in the other papers covering SND and TSP problems, where the existence of a lower bound of the problem is tied to the properties presented and the construction of the partially time expanded network.

For the train scheduling problem, the existence of an approximated model is given by introducing the Interval Assignment Problem (IAP), whose optimal solution defines a lower bound for the optimal solution of the TRP problem (Croella et al. 2024). Instead of having the previously seen properties that ensure the relaxation of the original problem, for the TRP, the optimal solution of IAP provides a lower bound. For each train i and each track segment r, a feasibility interval is defined representing the time within which the train can enter that segment , where each feasibility interval is divided into sub-intervals and the cost of the interval is set as the cost of train i entering track segment r at the beginning of that interval

(Croella et al. 2024). Regarding incompatibility, trains can be either rail path incompatible, where the train-rail precedence constraint is violated for any time point within the intervals, or conflict incompatible, where disjunctive precedence constraint is violated. Compared to the full TI formulation where incompatibility is based on start times, in IAP, intervals are incompatible if all pairs of time points within their Cartesian product are incompatible and in case two intervals are incompatible, all sub-intervals within are also incompatible. The goal of IAP is to find feasible assignment with minimum cost, where if TRP has an optimal solution then the IAP also has an optimal solution and the optimal cost of the IAP is a lower bound for the optimal cost of the TRP (Croella et al. 2024). IAP is then presented as an LP, with the objective function that minimizes the sum of costs for all selected intervals under constraints that ensure only one interval is chose for each train and track segment and that enforce feasibility by preventing the incompatible intervals. Since the IAP formulation uses only binary variables, the problem can be converted to MaxSat problem, where some hard clauses must be satisfied, while the soft clauses are weighted, and the goal is to minimize the sum of weights of satisfied soft clauses (Croella et al. 2024). Different methods exist for converting constraints of IAP into clauses which offer different trade-offs. In order to improve solvers performance for incremental problems, alternative MaxSat encoding for the IAP is designed. In this reformulation, a binary variable is defined to be not only true for the selected interval but also for all preceding intervals in the partition, which is beneficial because iteratively subdividing intervals, which is in the core of DDD, does not invalidate existing constraints, but only possibility makes them redundant (Croella et al. 2024). However, this allows the solver to incrementally add new constraints and leverage derived constraints and lower bounds from previous iterations, which potentially leads to significant performance improvement when solving multiple IAP problems. This reformulation is finally the necessary relaxation of the original problem, that the algorithm uses to build the DDD solution.

4.4. **Routing Problem**

When we talk about routing problems, when it comes to finding a partially time expanded network, there are different approaches to setting the previously mentioned properties for other types of problems. In the work regarding a two echelon multi attribute location routing problem, the existence of a reduced time-space network is defined over four major properties, which resembles most of the previously seen solutions that also involve defining properties that guarantee the existence of a partial network (Escobar-Vargas and Crainic, 2024). While the first property defines the existence of every timed node from the fully time expanded network in the partial network, the second property defines the possibility of having short travel arcs, similarly to what we've seen before. Third property makes sure that a waiting time arc exists and the last property states that there exists at least one time-space node that handles early inbound connections exclusively (Escobar-Vargas and Crainic, 2024). The four properties, which are fundamental aspects of the time-space representation of the problem, ensure that the 2E-MALRPS formulated on a reduced time-space networks is in fact a relaxation of the original problem and the solution obtained for the hybrid formulation is a lower bound of the problem, regardless of the discretization granularity (Escobar-Vargas and Crainic, 2024). When it comes to other IRP problems solved by the means of DDD, the works from Lagos et al. (2020) solving CT-IRP and Lagos et al. (2024) solving CT-IRP-OB, there is no straightforward introduction of the properties that guarantee the relaxation of the problem. The work from Lagos et al. (2020) argues that the time expanded network could become very

large in size and that the correct discretization parameter that guarantees an optimal continuous time solution could exist only theoretically. With this in mind, a selection of a value for a discretization parameter Δ and adjustment of all of the time-related parameters can however lead to a smaller MIP formulation that provides either a lower or an upper bound of the original formulation. While adjusting the travel time, depending on the choice of rounding, the resulting formulation is either a lower or upper bound (Lagos et al. 2020). In case of a round down the implication is that the travel time is decreased, and we get a lower bound, otherwise the resulting formulation yields an upper bound. It was observed in other papers before as well that introducing a property of underestimating travel times yields a relaxation to the original formulation and a lower bound, meaning that in the paper from Lagos et al. (2020) a similar idea was just presented more implicitly. Lagos et al. (2024) for their special case of CIRP-OB assume that there exists such time discretization fine enough in which a time indexed formulation finds an optimal solution as long as a feasible solution exists. In this paper, a linear relaxation optimal value is given by a proposition which is the analytical lower bound on the objective function that says that no feasible solution can be cheaper than that value (Lagos et al. 2024). This linear relaxation is weak when the integrality condition on variables is relaxed hence why the optimal value can contain 'fractional' vehicles that deliver only what's consumed in real time, suggesting that the lower bound will have to be further strengthened.

Since the algorithm itself tries to find a solution based on the relaxation of the problem as stated before, in the next section we will go over how this relaxation is used to create an initial network that is then refined to get an optimal solution.

5. **Initial network**

The initial partially time-expanded network consist of nodes (o_k,e_k) and (d_k,l_k) for all k∈ $\mathcal{K}$, where for each node (i,t) ∈ $\mathcal{N}_T$ and arc (i,j) ∈ $\mathcal{A}$, we find the node (j,t') with largest t' such that t'≤ t+ τ_{ij} and add arc ((i,t),(j,t')) to $\mathcal{A}_T$ (Boland et al. 2017). The partially time-expanded network again satisfies all of the previously introduced properties. To get a better overview on the creation of the initial network we refer to the Figure 8 where two different initialization algorithms are presented. In Figure 8 (left) is the initialization method defined above where the network is made by adding a time a timed node for every node, connecting the nodes with arcs while respecting the property that not every arcs travel time has to be accurate, meaning the travel times on the times arcs are allowed to be underestimated. In case that the freight has to wait at a certain node, then the holding arcs are also included in the initial network (Boland et al. 2017). Without going into too much detail about the DDD algorithm in general, Medina et al. (2019) refer their readers to the same algorithms used by Boland et al. (2017) and Van Dyk and Koenemann(2024) for their problem initialize the network with the nodes satisfying property one and arcs only of the form satisfying property two. Shu et al. (2024) in their CTSND problem with holding cost creates the same previously described initial network with the addition of functions that determine the lower and upper bound on the true departure and arrival times at the terminals, which then help them determine the amount of holding cost that need to be added to the objective function.

Some authors however introduced some preprocessing steps into their algorithm before the actual initialization (Hewitt 2019; He et al. 2023). For the CT-LPDP problem the DDD algorithm is designed as a Two-Phase DDD algorithm that seeks to begin the solution of the CTSNDP by DDD that yields a stronger relaxation than the one initially introduced (Hewitt 2019). It first applies DDD to a relaxation of CTSNDP to get a partially time-expanded network and an optimal value which is the lower bound of the solution, with which the solution of CTSNDP by DDD can begin and then in the second phase the DDD is executed again starting with the $\mathcal{D}_T$ network from the end of phase one.

Another preprocessing procedure was introduced by He et al. (2023) for their CT-SNDRP problem where the algorithm computes the departure time window for each route in the relation to the earliest available and latest delivery time of each commodity. They implement a Dijsktra algorithm on the route-time-expanded network repeating the process until all the routes reaching the destination node have been tested and the latest time of each route found is in fact the latest time that the route can depart in order to serve any commodity in the network (He et al. 2023). From the Figure 8 (right) we see that routes that start at the earliest departure time are added to the network and the initial route-time-expanded network contains more feasible timed copies in the beginning resulting in fewer timed copied that need to be added in the future iterations (He et al. 2023). To the initial network the CREATE-INITIAL algorithm (Figure 8 (right)) also adds holding and transfer arcs, as well as linking arcs and central hub nodes if needed.

Algorithm 2 (CREATE-INITIAL)

Require: Directed network $\mathcal{D} = (\mathcal{N}, \mathcal{A})$, commodity set $\mathcal{K}$

for all $k \in \mathcal{K}$ **do**
  Add node (o_k, e_k) to $\mathcal{N}_{\mathcal{T}}$
  Add node (d_k, l_k) to $\mathcal{N}_{\mathcal{T}}$
end for
for all $u \in \mathcal{N}$ **do**
  Add node $(u, 0)$ to $\mathcal{N}_{\mathcal{T}}$
end for
for all $(i,t) \in \mathcal{N}_{\mathcal{T}}$ **do**
  for all $(i,j) \in \mathcal{A}$ **do**
    Find largest t' such that $(j,t') \in \mathcal{N}_{\mathcal{T}}$ and $t' \leqslant t + \tau_{ij}$ and add arc $((i,t),(j,t'))$ to $\mathcal{A}_{\mathcal{T}}$
  end for
  Find smallest t' such that $(i,t') \in \mathcal{N}_{\mathcal{T}}$ and $t' > t$ and add arc $((i,t),(i,t'))$ to $\mathcal{H}_{\mathcal{T}}$
end for.

Algorithm 3 CREATE-INITIAL

Require: Route set $\mathcal{R}$
  add origin/destination nodes into $\mathcal{N}_{\mathcal{R},\mathcal{T}'}$
  for all $r \in \mathcal{R}$ **do**
    add one timed copy that starts at the earliest starting time
    add linking arcs if needed
    add central hub node if needed
  end for
  add holding and transfer arcs

Figure 8: Example initialization algorithms: **(left)** from Boland et al. (2017) and **(right)** from He et al. (2023)

For the SND problem with mixed autonomous fleet, the original formulation is very extensive encompassing a large number of constraints that model the use of different vehicles, restricted routes and vehicle platooning (Scherr et al. 2020). Despite the relaxation, the initial network algorithm for this problem variant contains a large number of steps. The initial network starts with nodes at the time point each commodity is available at the origin and the time point it's due at the destination, as well as nodes for each external zone at the beginning and end of the planning horizon (Scherr et al. 2020). Nodes for satellites are added at the earliest time point they can be reached from the external zone and nodes at the latest time point at which the external zone can be reached before t_{MAX}. To the initial network movement, holding and waiting arcs are added as well as auxiliary arcs between last and first time point of external zone (Scherr et al. 2020). Movement arcs are added to the network if the destination time of the arc would not exceed the last time point of its location. The partially time-expanded network is obtained after performing all of the steps in the initial algorithm with respect to the properties defined for the problem (Scherr et al. 2020). In the earliest stage, most of the arcs are underestimated with incorrect travel times, some of them also going 'backwards' in time.

When it comes to building an initial network for the interval-based problems, the algorithm BUILD_TIME-EXPANDED_MODEL creates the initial interval-based time expanded network and builds an IP model defined by the programming model that can be seen in Figure 2(right) (Marshall et al. 2021). Solutions to CTSNDP(T) are known as timed-solutions and if feasible have a timed-path-solution following the conditions discussed in previous section that ensure that arcs and paths for each commodity k exists within its required early-late time windows.

Now that we've discussed how different models build their initial network, let's go over the learning path of each algorithm that at the end hopefully converges into an optimal solution. Since, as we've discussed before, DDD algorithm is a dual bound algorithm, meaning it calculates both the lower and upper bound in each iteration and checks if the difference is under the optimality gap (Boland et al. 2017), we will start the next section by discussing how different model calculate these bounds.

For TSP problem, the initial algorithm in some papers is not explicitly and formally explained, but rather a part of the Solve-TSPTW algorithm which states that at the beginning, an initial time-expanded network is to be produced by satisfying the previously introduced properties (Boland et al. 2017b). However, in case of Vu et al. (2019), similar to the ideas of Hewitt (2019) and He et al. (2023) the authors introduce first a preprocessing step and then proceed to the initialization. Before the main iterative loop of the algorithm, similar to the techniques from the literature for routing problems with TW, the preprocessing technique applied aims to prune arcs from $\mathcal{A}$ and tighten time windows at locations (Vu et al. 2019). The technique is based on computing the difference Δij(t) between the earliest time the vehicle can depart location j, given that it departed location i at time t. If the travel times satisfy the triangle inequality, then is Δij(t) calculated as follows Δij(t)= max {ej, t+ τ_{ij}(t)}, otherwise it solves a shortest path problem, relying on the FIFO property for validity. In order to update the values of time windows, four rules are introduced, where the first two cover all variants of TSPTW and TD-TSPTW and the other two are only valid for variants of TD-TSPTW in which waiting at a location is not beneficial, like e.g. with makespan objective (Vu et al. 2019). Rule one states that the vehicle cannot depart from k earlier than it could arrive from any other location and rule two says the vehicle can depart k at the time that allows it to reach a subsequent location within its time window. Rule three states it's not beneficial to depart k early only to wait for the TW to open at the next location and rule four states that the vehicle shouldn't depart later than the latest time it can arrive to the other location (Vu et al. 2019). Regarding the arc pruning rules based on the time window analysis, they mostly cover precedence rules as it can be seen in Figure 9. They conclude best on precedence rules weather an arc is infeasible and then removes it from the solution, which in return tightens the time windows.

Algorithm 1 PREPROCESSING

Require: Graph $G=(N,A)$ and time windows $[e_i,l_i],\forall i\in N$
1: Set $P=\emptyset$ {P is the set of known precedence relationships.}
2: **while** changes found **do**
3: Update time windows per rules discussed above
4: **for all** $(i,j)\in A$ **do**
5: Check if $i\to j$ is infeasible and add $j\prec i$ to P if so.
6: **end for**
7: **for all** $(i,j,k)\in N$ **do**
8: Check if $i\to j\to k$, $k\to i\to j$, and $i\to k\to j$, are all infeasible and add $j\prec i$ to P if so.
9: Check if $i\to j\to k$, and $k\to i\to j$ are both infeasible and if so, remove arc (i,j) from A
10: **end for**
11: Add transitive closure of precedences to P so that it contains all precedence pairs (if $i\prec j$ and $j\prec k$ then $i\prec k$)
12: Remove all arc (j,i) from A if arc $(i,j)\in P$
13: Remove all arcs (i,k) from A if there is j such that $(i,j),(j,k)\in P$
14: Remove all arcs $(i,0)$ and $(0,j)$ from A if $(i,j)\in P$
15: **end while**

Figure 9: Preprocessing algorithm from Vu et al. (2019)

After successful preprocessing, the algorithm then proceeds to make an initial partially time expanded network by creating a set of nodes $\mathcal{N}_T$ ensuring it satisfies all of the properties, which in return allows for arcs to be created because of the set of nodes included in $\mathcal{N}_T$ (Vu et al. 2019). Following the same preprocessing and initialization method, Vu et al. (2022) also create their initial partially time expanded network for their TD-TSPTW problem with the same steps.

For the TDASP scheduling problem, an algorithm is used to create an initial partially time expanded network, where the algorithm according to property one first adds all vertices that correspond to the start and end of the time window in order from (n, l_n), (n, e_n),…,(1,l_1),(1,e_1) and then their immediate successors recursively according to the second property (Pottel and Goel, 2022). After adding the vertex (i,t), the consumption values of the new and the preceding vertex (i, t') are set according to the third property, by assigning a lower bound on the actual resource consumption (Pottel and Goel, 2022). For this initialization process, the main characteristic that sets it apart from the previously introduced initialization algorithms is the fashion in which the nodes are added, starting from n to one in a decreasing fashion as if can be seen in Figure 10. Similarly, the algorithm underestimates the vertices, what in other papers was referred to as arcs, according to the third property which in return provides us a lower bound whose solution might be unfeasible due to these short vertices and in course of the algorithm will have to be refined. For the trailer scheduling problem, there was no explicit algorithm for the initialization, but rather one algorithm for the whole DDD (Ojha and Erera, 2025). The algorithm constructs an initial partially time expanded network based on the relaxation properties providing a XDTS-W-LB solution which can be used to directly identify a feasible solution to the XDTS-W. The algorithm initially and repeatedly solves instances of XDTS-W-LB over a partially time expanded network satisfying the 1-3 properties (Ojha and Erera, 2025). For the TRP problem, the authors implement the steps of DDD paradigm while exploiting the IAP, which provides a relaxation of the original problem (Croella et al. 2024) The initialization starts with one big time interval allowing the train i to enter the track segment r at any time point. Instead of diving the time into many small steps, initially the algorithm creates a small model, starting with one large time interval for each train-track combination and then later on divides it into smaller intervals if conflicts or timing constraints require it (Croella et al. 2024).

In the case of routing problems, for the two echelon multi attribute problem specifically, the authors suggest the previously observed standard initialization that tries to prune arcs and tighten possible relevance-period sets, after which it creates an initial reduces network $\mathcal{G}\Delta$ that satisfies the previously defined properties (Escobar-Vargas and Crainic, 2024). Due to time dependency of demand that suggests some couples of location and time might be unreachable when travelling from the origin to the destination of the demand, a preprocessing analysis through a breadth-first search strategy is then introduced and performed on the physical network in hopes of tightening the availability time at platforms and the customer time window of the particular demand (Escobar-Vargas and Crainic, 2024). This particular preprocessing procedure iteratively explores a set of commodities, enumerating the platform-satellite combinations of interest that could potentially link the origin and destination at all time periods when the commodity could be available at the platform. The resulting set of feasible partial paths for each commodity defines the unreachable time periods that can be excluded from the network which helps generate an initial reduced time-space network at each relevant time period, that also satisfies the four properties (Escobar-Vargas and Crainic, 2024). In case of the CT-IRP problem by Lagos et al. (2020) however, we can observe a slightly different approach to solving the DDD algorithm in the sense that the algorithm doesn't start with the standard initialization but rather with a lower bound integer programming model that has the mathematical model as the base of the problems, but modifies some constraints. Constraint that ensures inventory at the customer is sufficient to meet the customer demand and the constraint that allows only one vehicle at customer location are relaxed, resulting in a model called LBM (Lagos et al. 2020). The LBM

model produces a feasible solution to the CIRP of the same cost and in the complete construction of the LBM feasible solution, for any vehicle route making a delivery to customer i at any time in the CIRP, there should be a corresponding route feasible to the LBM that delivers at the corresponding time index (Lagos et al. 2020). The LBM solution can contain two vehicles visiting a customer at the same time which can reduce cost, ultimately making the LBM solution not CIRP feasible, which then produces a need for strengthening the model. Similar process of introducing an LBM can be observed in the CT-IRP-OB variant, where the lower bound is obtained from the original problem formulation by removing one or more time points from the discretization, adjusting input parameters and relaxing some constraints (Lagos et al. 2022). However, one distinction from the Lagos et al. (2020) is in the introduction of the Optimality preserving conditions that are used to limit the search for an optimal solution. These OPC conditions cannot be directly imposed on the LBM model, but it can be checked if the time expanded network satisfies some conditions that allows to include the OPC in the model (Lagos et al. 2022). Finding a lower bound in a time expanded network is the first step of the DDD and in this case a sort of initialization algorithm to start the search of an CIRP-OB optimal solution.

Algorithm 2: Initializing a partial time-expanded network.

Data: $(e_i, l_i, \tau_i, \theta_i, \rho_i)_{i\in\{1,\dots,n\}}, Q, \varepsilon$

Function `DDD::initialize()`
  $V \leftarrow \emptyset$
  for $i \leftarrow n$ **to** 1 **do**
    $(V, (q_v)_{v\in V}) \leftarrow$ `DDD::addRecursive`$(V, (q_v)_{v\in V}, (i, l_i))$
    $(V, (q_v)_{v\in V}) \leftarrow$ `DDD::addRecursive`$(V, (q_v)_{v\in V}, (i, e_i))$
  end
  return $(V, (q_v)_{v\in V})$
end
Function `DDD::addRecursive`$(V, (q_v)_{v\in V}, (i, t))$
  if $(i, t) \in V$ **then return** $(V, (q_v)_{v\in V})$
  $V \leftarrow V \cup \{(i, t)\}$
  if $t = l_i$ **then**
    $q_{(i,t)} \leftarrow \rho_i(t)$
  else
    $t' \leftarrow \min\{t' \mid (i, t') \in V, t' > t\}$
    $q_{(i,t)} \leftarrow \min_{\bar{t}\in\{t, t+\varepsilon, \dots, t'-\varepsilon\}} \rho_i(\bar{t})$
    if $t > e_i$ **then**
      $t' \leftarrow \max\{t' \mid (i, t') \in V, t' < t\}$
      $q_{(i,t')} \leftarrow \min_{\bar{t}\in\{t', t'+\varepsilon, \dots, t-\varepsilon\}} \rho_i(\bar{t})$
    end
  end
  if $i = n$ **then**
    return $(V, (q_v)_{v\in V})$
  else
    return `DDD::addRecursive`$\Big(V, (q_v)_{v\in V}, \big(i+1, \max\{e_{i+1}, \varepsilon\lceil\frac{\theta_i(t)}{\varepsilon}\rceil\}\big)\Big)$
  end
end

Figure 10: Initialization algorithm for TDASP from Pottel and Goel (2022)

For the very specific problem of solving the shortest path, we've seen that the relaxation also follows the underestimation of the actual travel times on the arc, but since the problem itself doesn't have a classical mathematical model like the other problems, the initial network is done by computing an arc-completed backwards shortest path trees where for each node a backward shortest path tree is computed and provides a node-time pair (i,t_i) where the ABSPT

is then used to compute the upper and lower bound on the duration of an optimal path (He et al. 2018). For the initialization process for the MDP where the optimal solution is assumed to be waiting free at intermediate nodes, the algorithm is fed with the digraph $\mathcal{D}=(\mathcal{N},\mathcal{A})$, the latest time T and the arc travel time functions $c_{i,j}(t)$ for all times $t\in[0,T]$ and each arc (i,j) (He et al. 2018, He et al. 2022). Given a specific arrival time at the node, the corresponding backward shortest path tree (BSPT) represents a time expanded network (TEN) rooted at the destination node n at specific arrival time t and defined by the timed-nodes (i,t_i) and timed arcs $((i,t_i), (i,t_j))$ (He et al. 2022). In this case, the time t_i is the latest departure time form node i that allows arrival at n by the time t and there is a unique path from (i,t_i) to (n,t) in the TEN, induced by the solution from origin i to destination n. Due to the case that from the node i at any positive time, the destination might not be reached, those timed-nodes are then eliminated from the network in a preprocessing step, but for the simplicity of the problem, there is a timed-node for every node with a $t<0$ (He et al. 2022). The algorithm initializes with a list of ABSPTs, which are formed from the BSPTs by adding timed arcs $((i,t_i),(j,t_j))$ to every underlying arc (i,j), ordered by their arrival time at the end node n. The list is initialized with the ABSPTs corresponding to the the earliest time that the end node can be reached and ABSPT corresponding to the end of the time horizon T (He et al. 2022). After the ordered list of ABSPTs is initialized, the first lower and upper bound are calculated and then further used in the refinement procedure (He et al. 2022). All additional ABSPTs are then generated dynamically by splitting the interval and adding new time points. In case of the minimum travel time dependent path problem (MTTP) that differs from the MDP in such way that the optimal solution may require waiting at a node meaning that the algorithm has to calculate the lower bound using a single TEN with incorporated waiting arcs (He et al. 2022).

DDD algorithm for the minimum travel time time-dependent path problem (MTTP) differs from the one for MDP since its optimal solution may require waiting at intermediate nodes which then diminishes the breakpoints property utilized by the MDP problem (He et al. 2022). Instead, MTTP calculates lower bounds using a single TEN with incorporated waiting arcs. Timed path in MTTP can be viewed as a sequence of travel subpaths that are connected by waiting periods, where a travel subpath can be described as a maximum sequence of consecutive travel paths connected without waiting periods (He et al. 2022). For MTTP, we can prove that an optimal timed path exists such that each of the subpaths includes at least one timed node at an arc travel time function breakpoint, successfully extending the idea from the MDP (He et al. 2022). For the algorithm, time-expanded network with associated arc lengths (TENL) is formed by taking the union between forward and backwards shortest path trees for every breakpoint along with waiting arcs added between chronologically adjacent timed copies of nodes (He et al. 2022). The algorithm then tries to solve the MTTP by exploring the small fraction of the breakpoints, relying on the concept of mangrove, which is the previously mentioned union between forward and backwards SP trees for every breakpoint (i,t) (He et al. 2022).

6. **Learn the network**

6.1. **Lower and Upper Bound**

As was previously introduced, the DDD algorithm solves a relaxation of the original problem in each iteration which produces a lower bound that is then checked if it can be converted to a feasible solution of the problem of the same cost by calculating the current solution with the actual travel times (Medina et al. 2019). In order to obtain a lower bound to their problem, Scherr et al. (2020) solve an IP with valid inequalities that strengthen the relaxation. Since the arcs are allowed to travel backwards in time, services on auxiliary arcs are installed if transportation capacity for routing is required which can result in cycles that normally wouldn't happen in the conventional time-expanded networks. Scherr et al. (2020) propose incorporating some valid inequalities to strengthen the lower bound in each iteration. In order to obtain a feasible solution which is the upper bound of the objective function value, they consider the actual travel times of all arcs. The aim is to include all arcs that already have the correct length in $\mathcal{D}_T$ and lengthen the short arcs with the refinement procedure, which are used in the current solution of the SNDMAF($\mathcal{D}_T$) (Scherr et al. 2020).
Van Dyk and Koenemann (2024) argue that when a partial solution satisfies properties that define the relaxation for nodes and arcs of the original problem, the solution that is obtained is then the lower bound of the original problem. They solve a lower bound formulation defined on the valid partial auxiliary networks Gs. The optimal value of SND-RR(Gs) provides a lower bound on the optimal value of the time-expanded SND-RR problem. Given this optimal solution to the partially time-expanded network with corresponding trajectories in Gs, in each iteration the algorithm provides a feasible solution to the fully time-expanded network which represents the upper bound to the problem (Van Dyk and Koenemann 2024). In case the feasible solution is not optimal, in each iteration the set of nodes to be added to the partially time-expanded network is also added with the refinement process.
When we also include holding cost into the objective function, it can be proven that the relaxation of the problem is in fact the lower bound solved in each iteration of the algorithm and in order to derive a feasible CTSNDP-HC solution, which is the upper bound of the problem, the algorithm solves an LP, also called the implementable model IM(S) (Shu et al. 2024).
For TSP problems, similar to SND problems, the algorithm that satisfies the relaxation properties provides a lower bound to the objective function value of an optimal solution to the TSPTW($\mathcal{D}$) (Boland et al. 2017b; Vu et al. 2019). By carefully designing a partially time expanded network, the algorithm is guaranteed to provide either a lower bound for TSPTW's optimal value or an upper bound, if feasible. The algorithm iteratively refines $\mathcal{D}_T$ to produce a sequence of improving lower bound and consequently upper bounds as well until optimality is proven (Vu et al. 2019). While TD-TSPTW($\mathcal{D}_T$) provides a lower bound, its optimal solution might not directly correspond to a feasible solution of TD-TSPTW($\mathcal{D}$). This occurs for two reasons where for the first reason, $\mathcal{D}_T$ can contain arcs which induce subtours which would be detected by the flow balance constraints and for the second reason, the arc set may contain infeasible location sequence due to time window violation at a subsequent location, when actual travel times are applied (Vu et al. 2019). However, if an optimal solution to TD-TDPTW($\mathcal{D}_T$) doesn't include subtours and accurately model time, it provides and upper bound and proves optimality.

In the case of TDASP, the optimal solution of the problem restricted to a partially time expanded network which also satisfies the properties 1-3 is a lower bound on the optimal solution of the fully time expanded network with discretized times (Pottel and Goel, 2022). Due to the third property, the resource consumption is feasible for the partial network but still has a completion time smaller or equal to the solution of the fully expanded TDASP, hence making it a lower bound. In case an optimal solution to the TDASP on a partial network has all the right calculation of all resource consumptions, that means that solution is then the optimal solution of the fully expanded TDASP (Pottel and Goel, 2022). Similarly, for the trailer scheduling problem, by repeatedly solving the XDTS-W-LB over a partial network satisfying the properties 1-3, the solution produces a sequence of nondecreasing objectives function lower bounds (Ojha and Erera, 2025). The solver produces in each iteration a best lower bound and a best upper bound solution with an objective value UB' such that LB'<UB'. If all trailers are assigned a correct processing time and there are no more short arcs to refine, then the algorithm terminates with a provable optimality gap between dual bounds (Ojha and Erera, 2025). However, in case the LB to the partial network is not the LB to the optimal objective value of the original problem, then there exists a gap between a lower and upper bound and the optimality gap is greater than zero. In order to avoid this, the algorithm further refines the network and updates the current solution (Ojha and Erera, 2025).

Similar to other problems, if a reduced time-space network satisfies all four properties, then the optimal solution of the 2E-MALRPS formulated on the reduced network is a lower bound for the optimal solution of the problem on the complete time-space network (Escobar-Vargas and Crainic, 2024). Good quality lower bound is key for the DDD since it guides the search through the solution space, but since the LB is obtained by solving the integer program defined by the hybrid formulation on the reduced time-space network, the obtained solution may not always be feasible for the original problem (Escobar-Vargas and Crainic, 2024). Even if a solution is feasible from the point of capacity constraints and location decisions, vehicles schedules might be infeasible with the original travel times. Routing involves a high number of interconnections between nodes, hence why each arc may be represented by multiple time-space arcs depending on the discretization granularity, which tends to increase the size of the reduced time-space network (Escobar-Vargas and Crainic, 2024). Multiple candidate solutions could have the same cost structure which brings the need of introducing a degeneracy procedure that checks whether the structure of the current solution is degenerate and mitigates it by fixing the location and allocation decisions. Degeneracy mitigation procedure helps the refinement step of the algorithm since in case there is degeneracy, the same routes with refined granularity may no longer be observed (Escobar-Vargas and Crainic, 2024). When estimating the feasibility of a solution on a partially time expanded network with actual travel times, then the obtained solution is an upper bound of the original problem and if the solution value is better than the current best upper bound, the upper bound is updated and the procedure terminates (Escobar-Vargas and Crainic, 2024). When the solution is infeasible, the algorithm stops and forwards the solution to the refinement step of the DDD, which refines the solution until the stopping criteria is reached or the solution is proven to be infeasible (Escobar-Vargas and Crainic, 2024). For the CIRP case, the optimal solution of the lower bound model is a lower bound on the optimal value of the CIRP (Lagos et al. 2020). The lower bound itself is produced from a relaxation of some capacity constraints as well as omitting the one vehicle at the same time per customer constraint. This way, the lower bound of the original CIRP problem is mapped in such way that it can be transformed into a feasible solution for the original CIRP with the same cost

(Lagos et al. 2020). The lower bound can have more than one vehicle visiting a customer at the same time and possibly allow split deliveries, hence the need for a refinement process. Similar to the LBM, the upper bound model uses same variables and parameters, but the travel times are rounded up and overestimated, some constraints are modified, and the resulting feasible solution is also CIRP feasible (Lagos et al. 2020). In order to see if the LBM solution is optimal, discrete-time solution has to be successfully converted into continuous-time solution of the same cost. Lagos et al. (2022) also suggest that by removing some time points from the discretization, adjusting the input parameters and relaxing some constraints, the given formulation produces a lower bound to the original CT-IRP-OB problem. Since the travel times are rounded down, it is possible that the lower bound is not optimal for the original problem when estimated with actual travel times (Lagos et al. 2022). On the other hand, also deriving from the original problem formulation, by rounding up the travel times in the time expanded network the upper bound model is produced. This model finds feasible solutions for the CIRP-OB and is a valid upper bound for the problem (Lagos et al. 2022).
For the shortest path problem, the smallest lower bound over all ABSPTs in the list is the lower bound of the original problem and any feasible timed path (He et al. 2022). The shortest path from the departure node to the sink with the underestimated travel times gives a lower bound on a MDSPP and such shortest path estimated with actual travel times provides an upper bound (He et al. 2018). Computing the LB is essentially constructing underestimated travel times (UTTs) for each arc in the ABSPTs and then finds the least UTT path from beginning to the sink and returns its value (He et al. 2022). Such lower bound can be improved by refining the discretization by inserting a new ABSPT with the new end time in the interval followed by the end time of the ABSPT that cause the lower bound increase (He et al. 2022). The upper bound can easily be calculated from the ABSPT by calculating the time difference between t_n and t_1, where t_1 is the departure time at the origin and t_n is the arrival time at the destination, and the algorithm maintains the best upper bound (He et al. 2022). These two bounds are used to determine whether an MDSPP has been found and in case it wasn't then the algorithm determines for which parts of the time expanded network the time discretization need to be refined further (He et al. 2018). For the MTTP the algorithm constructs a lower bound based on all arc completed mangroves corresponding to the breakpoints, where the TEN itself includes travel arcs with assigned underestimated travel times and zero-UTT waiting arcs connecting different subpaths (He et al. 2022). The proposition guarantees the existence of a lower bound for MTTP if the least UTT path exists. The upper bound is also constructed, where any path from its start to end node is a feasible solution for the MTTP and its total travel time gives and UB on the value of MTTP (He et al. 2022). Minimizing the total travel time of a path and calculating it with actual travel times provides the best UB.

6.2. **Refinement**

6.2.1. **Service Network Design Problem**

When the solution of SND($\mathcal{D}_T$) cannot be converted to a feasible solution of the original CTSND problem with the same cost, then it's necessary to refine the network because some arcs in the solution are 'too short' (Boland et al. 2017). The refinement algorithm discovers an arc that is enabling so called 'phantom consolidation' which is in reality impossible because of the existence of at least one short arc and then adds a new node to the partially

time-expanded network that enables the true travel time of the arc to be modeled (Medina et al. 2019). The refinement process still has to ensure that all of the properties for a relaxation are respected and at the end of refinement at least one short arc is corrected. Since the algorithm allowed for short arcs ((i,t),(j,t')) where the length of t' is defined like $t' \leq t + \tau_{ij}$ (τ_{ij} being the time to traverse the arc between two nodes), during the refinement process the time is corrected so that the arc has the actual travel time of $t + \tau_{ij}$, which would be noted as ((i,t),(j, $t + \tau_{ij}$)) (Boland et al. 2017). The refinement proves and lengthening the arcs is a two-step process that first adds the new timed node (j, $t + \tau_{ij}$) to $\mathcal{N}_T$ and the new arc ((i,t),(j, $t + \tau_{ij}$)) to the $\mathcal{A}_T$, as well as a new arc ((i,t^i_{NEW}),(j, t') so that the resulting network still has the early arrival property. After adding the new times node of the form (j, t'') where $t'' = t + \tau_{ij}$ to the network, the previous node (j,t') with $t' \leq t + \tau_{ij}$ and the corresponding arc don't satisfy the longest arc property which then in the second step of the refinement process has to be restored (Boland et al. 2017).

Algorithm 3 (LENGTHEN-ARC$((i,t),(j,t'))$)
Require: Arc $((i,t),(j,t')) \in \mathcal{A}_{\mathcal{T}}$
1: REFINE$(j, t+\tau_{ij})$
2: RESTORE$(j, t+\tau_{ij})$.

Algorithm 4 (REFINE(i, t^i_{new}))
Require: Node $i \in \mathcal{N}$; time point $t^i_{\text{new}} \in \mathcal{T}_i$ with $t^i_k < t^i_{\text{new}} < t^i_{k+1}$
1: Add node (i, t^i_{new}) to $\mathcal{N}_{\mathcal{T}}$;
2: Delete arc $((i,t^i_k),(i,t^i_{k+1}))$ from $\mathcal{A}_{\mathcal{T}}$; add arcs $((i,t^i_k),(i,t^i_{\text{new}}))$ and $((i,t^i_{\text{new}}),(i,t^i_{k+1}))$ to $\mathcal{A}_{\mathcal{T}}$
3: **for** $((i,t^i_k),(j,t)) \in \mathcal{A}_{\mathcal{T}}$ **do**
4: Add arc $((i,t^i_{\text{new}}),(j,t))$ to $\mathcal{A}_{\mathcal{T}}$
5: **end for**.

Algorithm 5 (RESTORE(i, t^i_{new}))
Require: Node $i \in \mathcal{N}$; time point $t^i_{\text{new}} \in \mathcal{T}_i$ with $t^i_k < t^i_{\text{new}} < t^i_{k+1}$
1: **for all** $((i,t^i_k),(j,t)) \in \mathcal{A}_{\mathcal{T}}$ **do**
2: Set $t' = \arg\max\{s \in \mathcal{T}_j \mid s \leqslant t^i_{\text{new}} + \tau_{ij}\}$.
3: **if** $t' \neq t$ **then**
4: Delete arc $((i,t^i_{\text{new}}),(j,t))$ from $\mathcal{A}_{\mathcal{T}}$; add arc $((i,t^i_{\text{new}}),(j,t'))$ to $\mathcal{A}_{\mathcal{T}}$
5: **end if**
6: **end for**
7: **for all** $((j,t),(i,t^i_k)) \in \mathcal{A}_{\mathcal{T}}$ such that $t + \tau_{ji} \geqslant t^i_{\text{new}}$ **do**
8: Delete arc $((j,t),(i,t^i_k))$ from $\mathcal{A}_{\mathcal{T}}$; add arc $((j,t),(i,t^i_{\text{new}}))$ to $\mathcal{A}_{\mathcal{T}}$
9: **end for**.

Algorithm 3 (Refinement Strategies)
Input: Network $\mathcal{D}_{\mathcal{T}}$, LB solution (x, y, w), set $\underline{\mathcal{A}}$ consisting of all short-arcs used in the LB solution, and set $\underline{\mathcal{A}}_1$ consisting of short-arcs associated to IISs obtained by Algorithm 2
Output: Updated network $\mathcal{D}_{\mathcal{T}}$

```
//Strategy 1: Lengthen short-arcs
forall short-arcs ((i,t),(j,t')) ∈ A_1 do
    Apply lengthen-arc procedure presented in Boland et al. (2017) for short-arc ((i,t),(j,t'));
end
Sort the arcs in A \ A_1 = (a_1, a_2, ..., a_{|A \ A_1|}) for increasing order of the departure times;
forall n ← 1 to min{|A \ A_1|, |K|/5} do
    ((i,t),(j,t')) ← a_n;
    Apply lengthen-arc procedure presented in Boland et al. (2017) for short-arc ((i,t),(j,t'));
end
//Strategy 2: Tighten upper bounds of the departure times
Set t^min_{i,t} = +∞ for all (i,t) ∈ N_T and N = ∅;
forall k ∈ K do
    forall variables x^{ktt'}_{ij} equal to 1 in the LB solution do
        Compute θ^k_i according to (27);
        if θ^k_i ≥ 0 and θ^k_i < w^k_i then
            N = N ∪ {(i,t)};
            t^min_{i,t} = min{t^min_{i,t}, t + max{⌊(w^k_i − θ^k_i)/2⌋, 1}};
        end
    end
end
forall (i,t) ∈ N do
    if t^min_{i,t} < t̄_i(t) then
        Apply refine and restore procedures presented in Boland et al. (2017) for node i and new time point t^min_{i,t};
    end
end
end
```

Figure 11: Example refinement algorithms: (left) from Boland et al. (2017) and (right) from Shu et al. (2024)

In Figure 11 (left) we see the full refinement algorithm, which consists of three smaller algorithms LENGHTEN-ARC, REFINE and RESTORE, there the algorithms add the new time point, consequently a new node and arcs that connect the other nodes to the new node, they delete the previous short arcs form the arc set $\mathcal{A}_T$ such that both the longest arc property and the early arrival property are restored in the newly obtained solution. Since the LENGHTEN-ARC takes a short arc and restores it to its correct length and the actual travel time, the length of the arc is done at most once meaning that even if all arcs were refined and restored the algorithm would still finish in a finite number of iterations (Boland et al. 2017). For the

LENGHTEN-ARC problem a MIP is formulated to determine the fewest number of arcs that are too short for the consolidation that occur in the SND($\mathcal{D}_T$) with the objective function of minimizing the number of arcs that are assigned a short rather than the actual travel time with constraints that count the number of short arcs, check the dispatch times in accordance to the assigned travel times and the availability of the commodity ensuring the consolidation of SND($\mathcal{D}_T$) is maintained (Boland et al. 2017). The optimal value of the MIP would then be zero, meaning that there are no more short arcs that need to be lengthened and the current SND($\mathcal{D}_T$) can be converted to the solution of CTSNDP with equal cost.

In the category of other SNDP problems, He et al. (2020), Van Dyk and Koenemann (2024) and Scherr et al. (2023) use similar procedures as Boland et al. (2017) to refine the network.

Scherr et al. (2023) uses the same method of identifying nodes that are most likely to contribute to improving the solution quality in the next iteration and then with their refine and restore algorithms they add nodes respecting the six properties for their problem, where the new node receives outgoing arcs that mirror the ones of the earlier node. Since not all arcs can satisfy the longest arc property due to a limitation of not exceeding the last time point of the destination location $t^j_{MAX(k)}$, they restore the arcs in such way that they arrive to the furthest possible node which is at most as far as the correct travel time.

He et al. (2020) use a similar procedure as Boland et al. (2017) for their refinement but instead of manipulating arcs they manipulate routes. They use a MIP to discover and add new timed route copies by linking them to the existing network. The refinement procedure includes new timed route copies, linking arcs for origin and destination nodes, replaces holding arcs by the new ones and links new central hub to the existing times route copies. After the refinement, for each new timed route copy added to the partially time-expanded network, unnecessary linking and transfer arcs are removed (He et al. 2020).

Van Dyk and Koenemann (2024), similar to the node-based refinement presented by Boland et al. (2017), during their refinement a timed node is added to the node set Vs to lengthen the earliest-departing short timed arc in each trajectory Qk. Since their formulation is arc-based, the increase in the lower bound formulation is less than the increase in node-based formulation.

Differing on the methods used to determine the new time points, Shu et al. (2024) apply the refinement strategy one to refine and restore short-arcs used in LB solution and then they use the refinement strategy two to further tighten the relaxation of the holding times since their variant included holding cost in the objective function. The refinement strategy one is the same as the one in Boland et al. (2017) where they invoke the refine and restore strategy to add a new time point and lengthen the arcs to satisfy the properties. When no more commodities use any short arc of the form ((i,t),(j, t')), where t'<t+ τ_{ij}, they identify the halfway time points for each node i and add the following new time points to T that aim to tighten the relaxation of holding times for each timed node (i,t) $\in \mathcal{N}_T$ and then reapply the refine and restore procedure for new time points (Shu et al. 2024). For reference, Figure 11 (right) depicts all of the exact steps of this refinement process.

Speaking of changes to the original refinement procedure, Marshall et al. (2021) and Hewitt (2019) also produce enhanced refinement procedure for their particular variants.

Hewitt (2019) among some of the enhancements to the algorithm he proposed for the original DDD, also included an enhancement for a smoother refinement procedure. He proposes to break symmetries in commodity paths solved at an iteration, which impact the execution of DDD in a way that with the existing symmetries the refinement process in charge of removing infeasible consolidation in an iteration doesn't remove them. Hence why, the

modified refinement scheme for DDD used to lengthen arcs, lengthens both the short arc and the earlier times copy of that same underlying arc in the flat network that yield a symmetric path (Hewitt 2019). The refinement scheme is based on identifying the earliest symmetric timed copy for the arc that needs to be lengthened and a commodity k and lengthens it as well. The refinement is concluded with more than one symmetric arc being lengthened.
On the other hand, for the interval-based variant of DDD, also known as DDDI, there were several refinement strategies proposed for the implementation, each of which addresses different issues in the variant (Marshall et al. 2021). Default refinement strategy first checks the solution graph for any cycles and in if it's acyclic it checks for any violated paths and then it adds time points in either case by fixing only one violation per iteration on order of the commodity index. As an extension to the default refinement strategy, Reduced Iterations strategy fixes multiple issues in each iteration. In case of disconnected subgraphs where the commodities in different components don't interact, instead of adding just one fix, adding multiple fixes for each component should reduce the number of iterations. For independent failures, like consolidation and path-infeasible failures of a commodity k, the algorithm adds the independent fixes and time points without having concern about potential redundancy. Another method of refinement in this paper called Fewer Time Points was introduced with the purpose of keeping the algorithms' size as small as possible since the majority of DDDI computational effort was spent on solving the IP for the lower bound (Marshall et al. 2021). The total and solve times are able to be reduced by fixing more issues in each iteration using fewer time points, which directly stems from the application of different theorems in the paper. The algorithm, however, prioritizes adding fewer time point, more specifically unary/single timed-nodes which are computationally preferred, and strategically chooses commodity pairs in the failed consolidation which have the smallest time-window gap, while also breaking cycles more efficiently.

6.2.2. **Traveling Salesman Problem**

For the most basic case of a TSP problem, the refinement strategy is very similar to that presented for solving SND problems. Given a partially time expanded network $\mathcal{D}_T$ containing 'too short' arcs, the resulting time underestimation can be corrected with a LENGHTEN-ARC algorithm that still satisfies all of the properties and behaves very similar as the general one for SNDP (Boland et al. 2017b). In this very simplistic approach, a new node is added to the node set that reflects the correct travel time and the arcs are then updated to maintain all of the properties, most importantly the longest arc and the time window properties.
A little more complex refinement process was undertaken for TD-TSPTW variants, where the process tries to more aggressively find high quality primal solutions and improve upper bound by generating and solving two primal partially time expanded networks $\mathcal{D}_T^1$ and $\mathcal{D}_T^2$ in each iteration that use the same node set as the lower bound network $\mathcal{D}_T$ (Vu et al. 2019). These two differ in their arc set, where for $\mathcal{D}_T^1$ we model arcs whose travel times are at least as long as the correct time where any feasible solution to $\mathcal{D}_T^1$ is guaranteed to yield a feasible solution to TD-TSPTW($\mathcal{D}$). In $\mathcal{D}_T^2$ the arcs included are 'too short' but it doesn't include modeling non-positive travel times ($t' \leq t$), hence why this candidate solution cannot include subtours (Vu et al. 2019). In both cases a time point t'' must exist for all arcs $((i, t), (j, t')) \in \mathcal{A}_T$ since $t + \tau_{ij}(t) \leq l_j$ and $(j, l_j) \in NT$ are maintained throughout the algorithm (Vu et al. 2019). The DDD-TD-TSPTW algorithm attempts to convert each candidate solution into a feasible solution of TD-TSPTW($\mathcal{D}$) which is not always possible due to subtours and infeasible location sequence

when introducing original travel times. The primary method of strengthening the relaxation and enabling the convergence is by lengthening the short arcs while still ensuring the properties of the relaxation are satisfied to guarantee a lower bound. The arc lengthening procedure is based on arcs present in a candidate solution s (Vu et al. 2019). If s doesn't contain a subtour, the path itself is processed but if it does, a path derived from each subtour is processed. This process can in detail be seen in Figure 12, where we see two algorithms: one that lengthens arcs and the other that adds nodes. Algorithm two iterates through arcs of the path p', where it checks for each arc if it's 'too short' and if it can be lengthened with respect to time windows. If it can, then the Add-Node algorithm adds new node, the arc is lengthened to the new node with respect to the longest-arc property, and the old arc is deleted (Vu et al. 2019). This procedure reminds very much of the original refinement procedure for the SNDP problems, but here the refinement works with paths. The lengthen arc algorithm, besides adding the new node also adds nodes and arcs that enable the vehicle to follow the remainder of the original path. Algorithm 2 in Figure 12 also considers sub-paths starting at intermediate nodes and lengthens their arcs, making these "opportunistic" additions of arcs and nodes computationally beneficial, since they make $\mathcal{N}_T$ and $\mathcal{A}_T$ as closer approximations of $\mathcal{N}$ and $\mathcal{A}$ of the full network $\mathcal{D}$, which ultimately strengthens the lower bound more quickly (Vu et al. 2019).

Algorithm 2 LENGTHEN-ARCS-PATH(p')

Require: Path $p' = ((u_0, t_0), \ldots, (u_k, t_k), (u_{k+1}, t_{k+1}), \ldots, (u_m, t_m)), (u_i, t_i) \in \mathcal{N}_T$
1: **for** $i \leftarrow 0$ to $m-1$ **do**
2: $t \leftarrow t_i$ {Consider departure from u_i at t_i}
3: **for** $j \leftarrow i+1$ to m **do**
4: $t' \leftarrow \max\{e_{u_j}, t + \tau_{u_{j-1}u_j}(t)\}$
5: **if** $t' > l_j$ **then**
6: Break
7: **end if**
8: Call Add-Node$((u_j, t'))$
9: $t \leftarrow t'$
10: **end for**
11: **end for**

Algorithm 3 ADD-NODE(j, t')

Require: Node (j, t')
1: **if** $(j, t') \in \mathcal{N}_T$ **then**
2: Return
3: **end if**
4: $\mathcal{N}_T \leftarrow \mathcal{N}_T \cup (j, t')$
5: **for all** $k \in N$ such that $t' + \tau_{jk}(t') \leq l_k$ **do**
6: $t = \text{argmax}\{\bar{t} | (k, \bar{t}) \in \mathcal{N}_T, \bar{t} \leq t' + \tau_{jk}(t')\}$.
7: Add $((j, t')(k, t))$ to $\mathcal{A}_T$ with cost $\underline{c}_{jk}(t')$
8: **end for**
9: **for all** $a \in \mathcal{A}_T$ such that $a = ((i, t), (j, t")), t" < t'$ and $t + \tau_{ij}(t) \geq t'$ **do**
10: Delete arc $((i, t), (j, t''))$ from $\mathcal{A}_T$
11: Add arc $((i, t), (j, t'))$ to $\mathcal{A}_T$ with cost $\underline{c}_{ij}(t)$.
12: **end for**

Figure 12: Example refinement algorithm from Vu et al. (2019)

In addition, when a new feasible solution with a lower objective function value than the best-found solution so far is found, $\mathcal{D}_T$ can be then pruned by removing arcs, updating depot latest return time and repeating the preprocessing algorithm (Vu et al. 2019).
While Vu et al. (2022) create the same two candidate solutions as Vu et al. (2019), they have one more candidate solution by examining solution s to see whether it can be used to derive a solution to TD-TSPTW($\mathcal{D}$). They check if s consists solely of arcs with correct travel times which makes it feasible and optimal, but even if it contains 'short arcs' the sequence of visiting locations can be the basis of a feasible solution. DDD then tries to determine departure times associated with each location following properties that vehicle departs depot at the beginning of its TW and arrive before the end of it and departure times agree with travel times and occur during the TW of the location (Vu et al. 2022). These departure times and the sequence are used then to generate a solution to the TD-TSPTW($\mathcal{D}$). Following the refinement process of Vu et al. (2019), Vu et al. (2022) use the same steps of the algorithm to correct the 'short arc', while also noting that the algorithm doesn't necessarily try to correct all short arcs at an iteration.

6.2.3. **Scheduling Problem**

For the scheduling problems, in the case of TDASP, the algorithm adds at least one vertex to the partial network in each iteration if the solution is feasible but not optimal (Pottel and Goel, 2022). The algorithm takes off from the initial network and in each iteration begins finding a solution restricted to the partial network and in case there is an activity where the total consumption associated with the solution may exceed the capacity, a new vertex is then added to the partially expanded network V between (i,t_i) and the next vertex of activity i after time t_i. Since there is also the case of having replenishments, like having a new electric battery or the workers are allowed to resume after a sufficiently long break, the algorithm dynamically adds new vertices corresponding to a fully replenished resource (Pottel and Goel, 2022). Compared to the SND and TSP cases, where we lengthened arcs and deleted nodes that were too short in order to satisfy the longest arc property, for this specific case where we need to make sure we don't run out of a resource, the refinement adds vertices that are actually shorter than the current ones in order to assure the solution remains feasible.
For the trailer scheduling problem, the algorithm adds new timed nodes in each iteration in a way that at least one assignment of an unloading trailer with an infeasible short processing timed arc is eliminated from the solution (Ojha and Erera, 2025). If the current solution is still not optimal, the algorithm adds additional time points and regenerates the arc set for the next iteration. The authors also introduce two different procedure that enable extraction of a feasible solution to the original problem from a feasible solution to the XDTS-W-LB, since sometimes it is not straight forward (Ojha and Erera, 2025). In both approaches all of the following elements are fixed in order to create a feasible solution that respects the total workforce constraint: trailer-to-door assignments, number of workers per trailer and trailer sequence for each unloading door (Ojha and Erera, 2025). The first greedy approach generates a feasible solution by fixing the decision regarding unloading doors and the number of workers for each trailer, after which a delaying scheme of selecting a feasible unloading start is applied. The second approach introduces a MIP called worker dispatch model (WDM) that determines the actual optimal unloading time whose optimal objective function will equal the objective function value for XDTS-W (Ojha and Erera, 2025).

For the train rescheduling problem, if the schedule from the optimal IAP solution is infeasible for the original TRP problem, a repair mechanism involving an iterative process of solving an LP is applied (Croella et al. 2024). The LP includes variables for train entry times, train-rail path precedence constraints and LB constraints with the objective of minimizing delays. In case that the current schedule is infeasible, additional disjunctive precedence constraint that is violated is added to the LP always choosing the 'first-one' violated in chronological order, that minimizes the increase in the objective function value (Croella et al. 2024). This process continues until the feasible schedule is found or the LP becomes infeasible, either way with a finite number of iterations. If the solution is not feasible for the TRP due to intervals, which were compatible in the IAP but lead to an infeasible TRP schedule, the refinement procedure is introduced (Croella et al. 2024). For the violated train-rail precedence, in case a train violated its own sequence of track segments by e.g. entering a later segment too early, the problematic interval in question is then refined and split into two smaller intervals, the new vertex and edge are added to the vertex and edge sets, and then one of the newly created intervals becomes incompatible with the earlier segment , leading to a new constraint. In case of violated disjunctive precedence where two trains conflict on a shared resource, both of their tracks are split into smaller intervals, the new vertices and edges are added to the vertex and edge sets, creating new incompatible interval pairs (Croella et al. 2024). After the new intervals are defined, for the subsequent track segments along the trains' path, time consistency is propagated which recursively adds further intervals and incompatibilities if needed. At the end, in some k-th iteration of DDD-TRP, solving the IAP, checking for optimality and then refining the IAP is involved if necessary, by checking for incompatibilities and propagation changes (Croella et al. 2024).

6.2.4. **Routing Problem**

When we talk about routing problems and their refinement procedures, starting with a 2E-MALRP problem, as the last step of the DDD solution method, the refinement is performed then when the solution obtained from a reduced time-space network has short arcs which violate temporal constraints when evaluated with original travel times (Escobar-Vargas and Crainic, 2024). Refining the reduced time-space network when it comes to the short arcs extracted from the lower bound that is obtained in each iteration ultimately strengthens the lower bound in the future. This procedure extends the short arcs, while ensuring the four properties are still valid, by adding two new time-space customer nodes and arcs from both of its extreme points to the reduced network (Escobar-Vargas and Crainic, 2024). In this problem it is not enough to only refine the short arc in one of its extreme points since it doesn't exclude the appearance of the similar solution in the next iterations due to solution degeneracy. The algorithm also ensure that any newly added time-space node is within customer's time window and connects to other time-space customers only before or during their time window openings (Escobar-Vargas and Crainic, 2024). The algorithm, apart from adding two new time-space nodes at the extreme points of an arc, has the same three smaller algorithms that ensure adding nodes, updating the waiting arcs and restoring the connections between nodes, similar to what we've seen in other papers presented above. For the CIRP problem where the LBM model was used to create an initial partially time expanded network and get a solution, the refinement process entails eliminating depot subtours and introducing some valid inequalities (Lagos et al. 2020). For large discretization parameter Δ, the LBM has a weakness that with underestimation of the actual travel times between two nodes i and j,

it can happen that the model uses these zero-travel-time cycles even if the tour is disconnected from the depot (Lagos et al. 2020). Since this is a direct violation of the property for any CIRP feasible solution, such evasion of the depot and the use of zero travel time paths represents a depot subtour in the LBM. To combat this problem, auxiliary variables and constraints are introduced that represent the integer flow of a commodity that must be transported from node (0,0) to node (0,T) with at least one unit of flow carried by each vehicle (Lagos et al. 2020). Following the same idea as Hewitt (2019), Marshall et al. (2021) and Vu et al. (2019), another way to strengthen the LBM was by introducing several valid inequalities from the period inventory-routing problem. First inequality introduced regulates the lower bound on the total visits to a customer over the planning horizon, second regulates the minimum inventory level at the start of the planning period and the last one is the triangle inequality for travel times, that makes sure the vehicle departs from depot early enough to be able to arrive to the customer at a specific time (Lagos et al. 2020). For the CIRP-OB variant of a routing problem, incorporating valid inequalities for the number of visits as an LBM strengthening procedure was adapted for the specific variant with out-and-back routes (Lagos et al. 2022). Other two techniques like vehicle waiting and visit time conditions were also enforced by additional constraints into the LBM model. In order to get an LBM solution, travel times, customer storage capacities and number of visits have to be modified at the same time with constraints that recover customer storage capacity (Lagos et al. 2022). Since travel times are rounded down, inventory capacity is enlarged and multiple visits to the customer at the same time are allowed in the LBM, if the solution isn't feasible for the CIRP-OB problem, correcting travel times, customer storage capacities or number of visits are three techniques used to refine the solution in each iteration (Lagos et al. 2022). For travel times, the algorithm checks whether the travel times starting at time t^k_i are correct and if shorter time to depot is detected, after which a new time point is added in order to get a solution whose travel times aren't rounded down. For storage capacity, the algorithm checks for any storage violations by vehicle deliveries in the LBM solution and if the inventory exceeds the maximum capacity, after which a new time point in the middle of the current interval is added (Lagos et al. 2022). When correcting the number of visits, the constraint regarding the maximum number of visits is added to impose a maximum number of one visit per time interval, after which new time point is added to the solution. With these refinement procedures, the refinement of the CIRP-OB is finalized in a finite number of steps (Lagos et al. 2022).

6.2.5. **Shortest Path Problem**

For the shortest path problem, the algorithm iteratively refines the discretization in order to close the gap between the LB and UB, after computing the full list of ABSPTs by exploiting the property that the optimal solution uses arc travel time function breakpoints (He et al. 2022). The algorithm refines the time discretization by exploring the gap in the time expanded network between two timed nodes in a fashion that the new time node (i,t) fulfills following $t^k_i<t<t^{k+1}_i$ criterion and the algorithm updates the underestimated travel time which improves the lower bound given that the interval previously used to calculate the underestimated travel times shortened (He et al. 2018). If the current best LB<UB, the algorithm tries to identify these unexplored breakpoints and if such breakpoint exists between departure times defined by two consecutive ABSPTs, this breakpoint can be used to construct a new ABSPT (He et al. 2022). The corresponding ABSPT is created and inserted into the ordered list L, the calculation starts from the new breakpoint and determines the resulting arrival time at the

destination and recalculates the bounds (He et al. 2022). Under the assumption that there is an optimal solution that uses some arc travel time function breakpoint, the subintervals can be eliminated without making new ABSPTs and the breakpoint can be inserted to the list between two ABSPTs (He et al. 2022). If between two consecutive ABSPTs there are no more breakpoints, then is the earlier ABSPT marked as resolved, its LB updated to the UB (He et al. 2022). The algorithm also doesn't specify the exact breakpoint that needs to be chosen in case there are multiple breakpoints between two points in the TEN of the backwards shortest path tree (He et al. 2022). For the case of DDD solving MTTP problem, where we have travel subpaths, defined as waiting free timed paths, the algorithm considers mangroves for each breakpoint instead of the ABSPTs and uses waiting arcs to connect subpaths to one another (He et al. 2022). After the algorithm has finished calculating a SPP and both the UB and LB, if the two bounds aren't equal, that means at least one of the arc-completed mangroves has a breakpoint that's not resolved, because otherwise there would be no difference between the two bounds (He et al. 2022). For each node an ordered list of breakpoints is maintained and when there are no more unexplored breakpoints between two entries in the list, the whole mangrove is marked as resolved (He et al. 2022). When there is a difference between the bounds, the algorithm chooses one or more breakpoints to add to the list and refines the discretization until the mangroves are resolved and the UB and LB meet. Since the whole algorithm is a class of smaller algorithms, the performance depends on which set of unresolved breakpoints the algorithm chooses to refine in each iteration (He et al. 2022).

This concludes the part of the algorithm refinement. From the initial idea presented, we've summarized some changes to the original algorithm and enhancements made by different authors that help speed up the refinement process. As previously stated, the whole point of the refinement is closing the gap between the lower and upper bound on the solution of the problem, hence why in the next section we will go over the process of deriving a solution for the original problem.

6.3. **Convergence and optimality**

In each iteration of the algorithm, we get both the lower and upper bound and if LB<UB it means the current solution still has one short arc that needs to be lengthened or at least one missing time point that needs to be added (Shu et al. 2024). Since the fully time-expanded network $\mathcal{D}$ has a finite number of arcs and time points, the algorithm converges after a finite number of iterations or the partially time-expanded network $\mathcal{D}_T$ becomes equivalent to the fully time-expanded network $\mathcal{D}$, which means that in both cases there is an optimal solution to the original problem (Shu et al. 2024). Termination of DDD is guaranteed since $\mathcal{A}$ is of finite cardinality and during refinement at least one arc is lengthened in $\mathcal{A}_T$, meaning that after a finite number of iterations, $\mathcal{A}_T$ would contain sufficient arcs of the $\mathcal{A}$ which means that solving a partially time expanded network is equivalent of solving the fully time expanded network (Vu et al. 2022). In case the optimality tolerance is set to zero, the algorithm for the original problem converges to optimality after a finite number of iterations.
SOLVE-CTSNDP terminates with the optimal solution, when the optimal solution to SND($\mathcal{D}_T$) can be converted to the CTSNDP solution with the same cost, which after a finite number of iterations, all arcs in $\mathcal{A}_T$ must have travel times corresponding to their actual travel times in the flat network (Boland et al. 2017). If a solution on the partially time-expanded network can

be converted to one on the complete network with the same route starting times and consolidation plans, then is the solution optimal for the original continuous time SNDRP problem (He et al. 2020). On the other hand, if the solution cannot be converted, then it means that some of the consolidations need to be separated, inducing higher cost and the upper bound instead of optimal solution is found.

The solution to the relaxation is checked in each iteration if it can be converted to a feasible and provably optimal solution to the original problem, if it can then the algorithm terminates, otherwise the partially time expanded network is further refined and the algorithm continues until it converges to the optimality (Vu et al. 2022). In case of a TSP problem, some algorithms work with candidate solutions which are checked for infeasible location sequence, subtours and short arcs. Whenever a candidate solution is successfully converted to a feasible solution, its objective function value is compared against the best found upper bound and optimality tolerance (Vu et al. 2019). In case the optimality tolerance is satisfied, the found feasible solution is then also optimal. However, the algorithm may also collect all integer solutions found by the solver, meaning not just the optimal ones, for conversion attempts.

For the train rescheduling problem, the authors show that the DDD-TRP converges to the optimal solution after a finite number of iterations, where they solve a dual procedure of both a row and column generation (Croella et al. 2024). At each iteration, a relaxation of the original problem is solved and checked if the solution of the relaxation can be converted to the optimal solution of the original problem. In the case it cannot be converted, at least one row is added to the constraints group and at least one variable is generated and added to the problem during the refinement process we've explained before (Croella et al. 2024). Either way, DDD-TRP terminated either with an optimal solution or a prove that the problem is infeasible. DDD-TRP at every iteration gets a solution value which is a lower bound of the optimal solution for the TRP, but if the algorithm fails to produce a feasible solution with the repair procedure, then the problem is concluded to be infeasible (Croella et al. 2024). Similar to the TRP, for the XDTS-W problem the optimal solution of the problem is found either by obtaining the solution of the XDTS-W-LB that doesn't contain short processing arcs of trailers or when the current best feasible solution identified has the same objective function value as the lower bound solution (Ojha and Erera, 2025). In case the optimal solution is not found and there are no more short arcs to be refined, the solver finds the lower and upper bound and terminates the algorithm by computing a provable optimality gap. The algorithm again converges in a finite number of iterations (Ojha and Erera, 2025). If a feasible solution for the TDASP exists, the algorithm either terminates with an optimal solution or further refines the partially expanded network by adding at least one vertex in each iteration until algorithm converges to optimality in finite number of iterations, which in the worst case means expanding the network to the fully time expanded one (Pottel and Goel, 2022).

For the CIRP problem, the first step of successfully converting an LBM solution into a continuous-time optimal solution is extracting the delivery routes and independent vehicle itineraries with associated product flows (Lagos et al. 2020). Second step is to revise the time of each visit and the quantity delivered during the visit while preserving the vehicle and customer sequences and feasible route's timing. The first step is accomplished by solving a MIP and the second step with and LP or IP. The results imply there exists such discretization that an optimal solution of the time-expanded network produces a continuous time optimal solution (Lagos et al. 2020). If the travel times, storage and vehicle capacities, initial inventories and the usage rates are all rational and if the CIRP instance is feasible, then there exists an optimal rational solution (Lagos et al. 2020).

Following a similar idea, Lagos et al. (2022) argue that in order to determine if a continuous time feasible solution with the specified number of visits from the LBM exists, they need to introduce a MIP program that revises the visiting times and delivered quantities, while preserving the number of visits to each customer. This model is then called a Feasibility Check Model which may deduce that no feasible CIRP-OB solutions using the number of visits exists, in which case a different optimal LBM solution is needed and attained by the refinement procedure (Lagos et al. 2022). If FCM is feasible and has an optimal value, then it provides a feasible solution to the CIRP-OB with the same cost as the LBM solution.

Since the optimal solution obtained on the reduced network must remain optimal when evaluated with actual travel times for all arcs, Escobar-Vargas and Crainic (2024) decided to employ a feasibility check analysis, similar to Lagos et al. (2022), to evaluate the routes of the second echelon and verify the feasibility of customer time windows. By taking the vehicle schedules defined by the optimal solution obtained from the reduced network, the procedure then traverses the sequence of nodes iteratively evaluating whether all routes are feasible or if any of the customer time windows are infeasible. Once the feasibility is verified the obtained solution is an upper bound of the original problem and if it's better that the current best upper bound then the procedures updates the bound and terminates (Escobar-Vargas and Crainic, 2024). In case the solution is infeasible it is then further refined by the DDD.

When we speak of convergence in the DDD algorithm with regards to solving the shortest path problem, we can say that there exists a termination in finite number of steps because we can find an optimal path containing at least one node-time pair corresponding to a breakpoint and since there is only a finite number of breakpoints, the algorithm terminates in a finite number of steps (He et al. 2018). When there are no more breakpoints to explore in the gap between two ABSPTs, the algorithm can still terminate even if the smallest LB among the ABSPTs is larger than the best UB obtained as far. With the efficient pre-computation of the look-up table with the local minimum of each breakpoint, the efficiency of the algorithm in computing the minimum arc travel time is guaranteed (He et al. 2018). Choosing a new ABSPT that splits the subinterval and then recomputing the bounds can produce an algorithm that converges to an optimal solution even without exploiting the breakpoints (He et al. 2022). However, by using the breakpoints, the subintervals can be created more carefully and unnecessary bisection of the subintervals can be eliminated resulting in a termination with a finite number of steps (He et al. 2022). Even though the algorithm uses a list of discrete ABSPTs, the MDP problem can be interpreted as a single partially time expanded network with zero-UTT arcs for waiting at the origin and destination (He et al. 2018, He et al. 2022). For the MTTP case, the lists are used to compute the value of the time-expanded network and the bounds and by adding new breakpoints and refining the algorithm, the exploration continues until the two bounds are equal, which means they algorithm converged to optimality (He et al. 2022). Instead of comparing the two bound, alternatively the algorithm can terminate when all arcs used in the LB solution originate from resolved mangroves and provide a feasible, and consequently, an optimal MTTP solution.

7. **Enhancements**

We've already mentioned that some of the authors built upon the initial DDD algorithm idea to improve it in some shape or form. In this section we will discuss some of these enhancements made to the algorithm as well as some heuristics based on the original DDD algorithm.

Since the longest time necessary for the execution of the algorithm is spent on solving the program to derive a lower bound, some of the authors made enhancements to the algorithm in order to provide some sort of preprocessing step or a so called "warm start". During this preprocessing, the algorithm could compute the departure time window of each route in relation to the earliest available and latest delivery of each commodity (He et al. 2020). In the initialization, routes that start at their earliest departure time are added to the partial network in a way that the initial route-time-expanded network contains more feasible timed copies in the beginning which results in fewer timed copies added in the future iteration and better initial lower and upper bounds (He et al. 2020). Following the same idea of a "warm start" of sorts could be by applying a two-phase DDD, where this enhancement seeks to begin the solution of continuous time SNDP by DDD with a partially time-expanded network that yields a stronger relaxation than the usual relaxation of the problem (Hewitt 2019). This enhancement in summary executes the DDD algorithm twice, first time to obtain the partially time-expanded network and its initial solution, leveraging the computational advantage of solving a linear instead of an integer program, and then again but skipping the prior first step of creating the initial network. Based on the idea from Hewitt (2019), Scherr et al. (2023) apply a similar two-phase procedure but they further relax the problem by ignoring the capacity constraints for transportation in the first phase, thus the LP finds the largest possible amount of consolidation. Ignoring the capacity constraint weakens the lower bound even more but it builds a smaller initial partially time-expanded network, where some of the arcs found by the LP can still potentially be used in IP once the capacity is restored (Scherr et al. 2023). Another idea of relaxation strengthening stems from solving the TD-MTDP which is a problem similar to TSP, where the authors modify the objective function by increasing the cost coefficients in the second term of the objective function, which measure when the vehicle departs from the depot (Vu et al. 2022). After focusing on the time the vehicle departs from the depot, they introduced a mathematical formulation as a stronger relaxation to the original problem compared to their previous work. To prove optimality of this stronger relaxation, they prove that all arcs have to be of the correct length and the next potential departure time from the depot is represented in the network (Vu et al. 2022).

In their efforts to improve the performance of the DDD algorithm solving the TSPTW problem, Boland et al. (2017b) also introduce a preprocessing step to tighten the time windows and remove unnecessary arcs inspired from previous works. Somewhere along the lines of a 'warm start' can also be solution harvesting, where all integer solutions found by the IP solver are checked for feasibility (Boland et al. 2017b). Any new best solution is recorded, and all 'too short arcs' used in infeasible solutions are immediately lengthened in the current iteration. In one of the previous sections, the preprocessing technique for a TD-TSPTW problem from Vu et al. (2019) was explained in more detail, but to sum it up shortly here, it uses arc pruning to eliminate arcs from a solution with the use of precedence rules. Authors Vu et al. (2022) also use the same preprocessing technique before the algorithm initialization.

In order to reduce both the number of integer programs solved and the time spent solving them, the idea of introducing some valid inequalities at each iteration derived from the

CTSNDP to the SSNDP comes to play (Hewitt 2019). At the heart of problem, it lays that even when the valid inequality is added to the partially time-expanded network there exists a feasible solution that can be optimal for a fully time-expanded network. The problem that DDD solves at each iteration is a relaxation because of the short arcs that allow infeasible consolidations, hence why the introduction of arc-time window inequalities calculate how many timed-arcs need to be actually installed for commodities whose time windows do not overlap, in order to allow enough capacity to serve these commodities separately (Hewitt 2019). Similar to the ATW inequalities, there can also be path-arc-time window (Path-ATW) inequalities introduced, that in the defined-path variant enable more precise analysis which returns narrower time windows and stronger valid inequalities, following the assumption that each commodity follows the shortest path from its origin to destination (Hewitt 2019). As a way of eliminating subtours and bad paths that violate time windows if we add actual travel times, valid inequalities can be added to the TSP problem formulation (Boland et al. 2017b). This is specifically done by detecting node sets that give rise to subtours and bad paths during the course of the algorithm and adding elimination constraints to the partially time expanded network models. Similar to this, in hopes of eliminating undesirable attributes form a feasible region, Vu et al. (2019) also add valid inequalities to eliminate subtours and path-based inequalities. If a candidate solution contains a subtour, the algorithm adds the inequality defined over locations $\mathcal{N}$ in order to rule out all timed copies of a subtour and prevent it from appearing in further iterations. If a solution contains an infeasible sequence of locations, the added valid inequality is defined over locations $\mathcal{N}$ and forbid all timed copies of the infeasible sequence as well (Vu et al. 2019). Following similar process, Lagos et al. (2020) introduce depot subtour elimination constraints to the LBM model that eliminate the possibility that the vehicle traverses zero-travel-time cycles even when connected to the depot node. To do this, authors introduce auxiliary variables and constraints that eliminate these depot subtours. Inspired by the period inventory routing problems, Lagos et al. (2020) also introduce several valid inequalities like the lower bound on the total number of visits to a customer, lower bound on the number of visits that must occur in a given time interval as well as triangle inequality for travel times. Lagos et al. (2022) then built on the idea of the inequalities and adapted the inequalities from Lagos et al. (2020) for the number of visits to their CIRP-OB variant of the problem.

An additional valid inequality and a preprocessing technique can be introduced to remove redundant time-arcs, where an earlier alternative with the same or broader commodity usage exists, which ultimately reduces the model size. Regarding the IP model, another enhancement idea states that instead of regenerating the model every time, updates can be performed on the current model, variables and constraints can be labeled for reuse, facilitating a 'warm start' for the IP solver (Marshall et al 2021).

Another idea for the enhancement is to just partially relax the DDD algorithm in a way that the algorithm starts by solving an IP to obtain a lower bound in iteration zero and then we set service and commodity flow variable to continuous if they haven't been used in previous iterations (Scherr et al. 2023). Over the iteration, these variables are then set to binary and integer once they have been part of a lower bound solution and the produced partially relaxed model produces a valid lower bound just as the fully relaxed and integer model would do.

Another enhancement introduced directly impacted the refinement process. Hewitt (2019) introduced a Symmetry-Refine algorithm and Shu et al. (2024) introduced a two-phase refinement process, that were explained in detail earlier in this paper. The idea of both of the algorithm enhancements was to speed up the process of refinement and explore less time

points that need to be added, in a way to tighten the relaxation. For the interval-based DDD there were also different extensions for the refinement process that check for cycles and violated paths, also the reduced iterations extension that tackles multiple problem in one iteration and adding fewer time points to save up some computational effort which was covered in greater detail earlier in the refinement section of this paper (Marshall et al. 2021). Some other efforts presented tackle strengthening the IP model by safely removing some of the time-arcs, since there exists an alternative feasible solution with the same cost (Marshall et al. 2021). Another direct refinement procedure enhancement is dynamically adding waiting opportunities at the depot for the TSP problem with duration objective (Vu et al. 2022). Unlike modeling all waiting nodes from the start, this procedure dynamically adds nodes of the form (0,t) representing waiting at the depot. The procedure helps minimize the underestimation of the objective function and leads to a better solution by adding another waiting node at the depot between time points t and t', in a way bisecting the interval with a time point t'' (Vu et al. 2022). With this procedure, smaller partially time expanded networks are made which help speed up the convergence to optimality.

In practice, for the first couple of iterations of DDDI, there is a large gap between the upper and lower bound, hence why there is very little benefit in using a small optimality tolerance in the IP solver used to find the next lower bound. Since the DDD operates with lower and upper bounds which are constantly calculated and it finishes with an optimal solution in case the difference between the two bounds is under a certain fixed optimality tolerance, the enhancement idea introduces an adaptive way of calculating the IP optimality tolerance (Marshall et al. 2021). Based on the current DDDI gap, the solver can adjust the target optimality dynamically. So, instead of having a fixed tolerance across all iterations, we can adapt it as the DDDI algorithm progresses (Marshall et al. 2021).

In order to speed up the search of feasible solutions, some authors added heuristics that use the current node set to generate feasible solutions (Boland et al. 2017b; Vu et al. 2019; Vu et al. 2022). First heuristic is more conservative and contains only timed arcs with positive travel times and guarantee a feasible solution for the TSPTW. It is used to find a feasible TSPTW tour sooner, but it disabled after the first such is found (Boland et al. 2017b). The second heuristic is less conservative and contains short arcs but is designed to provide better solutions since it's solved at every iteration with addition of bad path elimination inequalities. Vu et al. (2022) also introduce two heuristics for discovering candidate solutions. The first mechanism optimizes departure times for a given tour by taking a feasible TSP tour, focusing only on arcs between subsequent locations in the sequence, and constructing a restricted integer program to find departure times for that sequence that yield a smaller duration (Vu et al. 2022). The second mechanism is a repair heuristic for infeasible solutions of the relaxation that are not directly feasible for the original problem. It first identifies subtours and partitions the remaining locations into disjoint sets, then constructs a partially time expanded network with specific arc sets that enable travel within base subtour, between subtour and the partitioned set and within partitioned sets (Vu et al. 2022). It also distinguishes between cases whether the subtour contains time window violations or only subtours, where the algorithms then detect the infeasible sequences and create an artificial location set.

8. **DDD based metaheuristic**

At the time of writing this literature review, the only heuristic developed based on DDD was from Belieres et al. (2021), where the authors tried to solve a logistics network design problem, crucial for logistics operations planning for third party logistics service providers specialized in warehousing and transportations. Since the original problem is objectively too large to be handled by traditional solvers in reasonable time, the authors developed a Time-Expanded Network Reduction Matheuristic (TENMR) inspired by DDD. Compared to a traditional SNDP problem, LSNDP seeks to design a 'forward flow' network and instead of determining the origin and destination of a shipment a priori, for each customer request LSNDP sources deliveries from different suppliers (Belieres et al. 2021). The study focuses on operations of a 3PL supporting restaurant supply chain in France within four-echelon network over a near future horizon where all shipment pass through a primary and secondary warehouse (Belieres et al. 2021). In LSNDP a service is defined as a possibility to transport a product between facilities, defined by locations and the departure and arrival times, with vehicles of a certain capacity where 3PL determines the transportation plan but relies on a third-party carrier for execution without managing the fleet (Belieres et al. 2021). The current problems' strategy uses a centralized delivery plans where all products have to go through primary and secondary warehouse before reaching the customer which is visible in Figure 13 (left), enabling consolidation which lowers cost and increases vehicles fill rate but also leading to longer travel distances and involving multiple handling steps, which causes additional costs and impacts delivery times (Belieres et al. 2021). In order to reduce costs, the extended strategy could involve direct delivery paths from suppliers to customers and indirect delivery paths where products transit through only secondary warehouse as portrayed in Figure 13 (right).

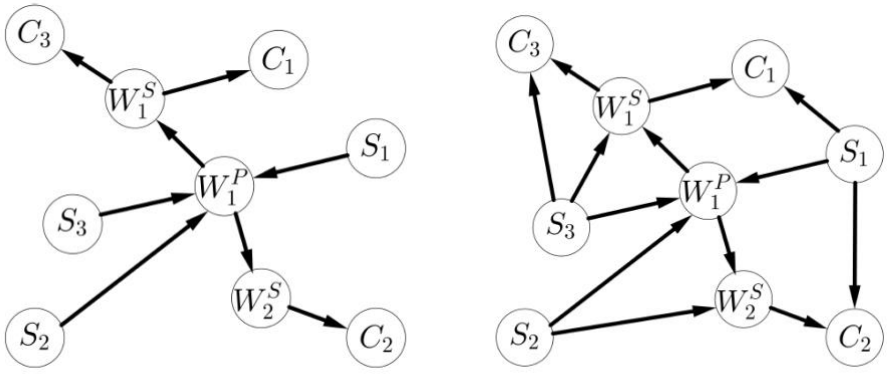


Figure 6: Current distribution strategy

Figure 8: Extended distribution strategy

Figure 13: Example of distribution strategies from Belieres et al. (2021)

When it comes to a mathematical formulation, the problem is modeled on a directed graph $\mathcal{G}$=($\mathcal{N}$, $\mathcal{A}$) where $\mathcal{N}$ is a set of stakeholders and $\mathcal{A}$ is a set of arcs that model freight transportation with specific origins and destinations (Belieres et al. 2021). This physical network is then extended to capture the temporal component, like it's usually done in DDD as well, to a time expanded network $\mathcal{G}_T$=($\mathcal{N}_T$,$\mathcal{H}_T$ U $\mathcal{A}_T$) where each node and arc become timed nodes and arcs, defined by the granularity of discretization. $\mathcal{H}_T$ are holding arcs

modeling storage at warehouses, $\mathcal{A}_T$ are transportation arcs where travel times from a static network are expressed in time intervals and the model accounts for overestimation of real transit times due to time discretization by introducing an approximation, which calculates the difference between estimated and original transit time and is taken into account when calculating linear costs of transportation arcs (Belieres et al. 2021). This is best illustrated with an example of time granularity of Δ=4, with 24 hours being divided into 6 hour intervals, and travel time on an arc of 5 hours The approximation then is one hour which implies that a product shipped on this arc is kept one hour in facility j before any operation can be proceeded (Belieres et al. 2021). With this in mind, the mathematical model for LSNDP is an integer program with an objective function that minimizes total cost, summing fixed and variable costs on transportation arcs and storage costs as it can be seen in Figure 14. Regarding the constraints in Figure 14, this model has flow conservation (2), demand fulfillment (3), vehicle allocation (4), warehouse capacity (5) and variable domain constraints (6,7) (Belieres et al. 2021). These constraints make sure enough vehicle are serving customers, meeting their demand while balancing the inflow and outflow at warehouses and the overall storage capacities. The main reason for the TENMR heuristic adaptation based on DDD is the fact that regular DDD algorithm is not suitable for the LSNDP problem due to different reasons (Belieres et al. 2021). In LSNDP unlike DDD, storage costs are not assumed to be automatically zero and the initial and final destination of shipments is unknown, which, in a direct application of DDD, would lead to a dense initial network and slow convergence (Belieres et al. 2021). With the generic network repair procedure from DDD, the direct application is not suitable for a multi echelon structure which needs different repair procedures for arcs directed towards customer compared to arc directed to warehouses (Belieres et al. 2021).

$$\text{minimize } z(\mathcal{G}_T) = \sum_{((i,t),(j,t'))\in\mathcal{A}_T} f_{ij} y_{ij}^{tt'} + \sum_{((i,t),(j,t'))\in\mathcal{A}_T} \sum_{p\in\mathcal{P}} \dot{c}_{ij} x_{ij}^{ptt'} + \sum_{((i,t),(i,t+1))\in\mathcal{H}_T} \sum_{p\in\mathcal{P}} \dot{c}_i x_{ii}^{ptt+1} \quad (1)$$

Under the following constraints :

$$\sum_{((i,t),(j,t'))\in\mathcal{A}_T\cup\mathcal{H}_T} x_{ij}^{ptt'} - \sum_{((j,t'),(l,t''))\in\mathcal{A}_T\cup\mathcal{H}_T} x_{jl}^{pt't''} = 0, \quad \forall (j,t') \in \mathcal{W}_T^{\mathcal{P}} \cup \mathcal{W}_T^{\mathcal{S}}, \forall p \in \mathcal{P} \quad (2)$$

$$\sum_{((i,t),(j,t'))\in\mathcal{A}_T} x_{ij}^{ptt'} \geq d_{jt'}^{p}, \quad \forall (j,t') \in \mathcal{C}_T, \forall p \in \mathcal{P} \quad (3)$$

$$\sum_{p\in\mathcal{P}} x_{ij}^{ptt'} \leq \hat{u} y_{ij}^{tt'}, \quad \forall ((i,t),(j,t')) \in \mathcal{A}_T \quad (4)$$

$$\sum_{p\in\mathcal{P}} x_{ii}^{ptt+1} \leq wlim_i, \quad \forall ((i,t),(i,t+1)) \in \mathcal{H}_T \quad (5)$$

$$x_{ij}^{ptt'} \in \mathbb{R}^+, \quad \forall ((i,t),(j,t')) \in \mathcal{A}_T \cup \mathcal{H}_T, \ \forall p \in \mathcal{P}^i \quad (6)$$

$$y_{ij}^{tt'} \in \mathbb{N}^+, \quad \forall ((i,t),(j,t')) \in \mathcal{A}_T \quad (7)$$

Figure 14: Mathematical model from Belieres et al. (2021)

The TENMR retains DDD's concepts but adapts it for LSNDP problem by iteratively constructing a sparse time expanded network, likely to contain a high-quality solution, after which is solves an IP on the reduced network (Belieres et al. 2021). During the initial network (χ_T) construction process the heuristic only solves LPs to speed up the process then starts two repair mechanisms, one for short arcs and one for storage costs, before finally solving the IP (Belieres et al. 2021). In order to ensure feasibility, the initial network is initialized with all nodes and arcs from the optimal solution of the linear relaxation of the complete network G_T. Short arcs underestimating the real transit times are axed to χ_T, but unlike in DDD these short arcs are given linear and fixed costs inferior to their original copies which are proportional to their transit time approximation, making them more attractive when solving LP relaxations (Belieres et al. 2021). The costs of storage arcs are initially set to zero, like in DDD. LP(χ_T) is solved after the initial construction and in case it leverages either short arcs of free storage, the two repair mechanisms are then initiated. The first phase of refinement repairs short arcs in a modified approach where it extends short arcs 'forwardly', like DDD, but also 'backwardly' meaning it not only created new destination time nodes and connects them with other existing nodes with arcs but also new origin nodes, which then prevents the addition of dead-end nodes (Belieres et al. 2021). In the second phase the algorithm repairs free storage arcs until none appear in the optimal LP(χ_T) solution. After there are no more short arcs or free storage arcs, the refinement ends in finite number of iterations. All remaining short arcs are removed from the χ_T to ensure feasibility for the original IP problem and the cost of all remaining free storage arcs are updated to their original values and finally the IP(χ_T) is solved to obtain a solution for the IP(G_T) (Belieres et al. 2021). Regarding the computational study for the heuristic, it can be found in the next section along other computational studies.

9. **Computational study**

Now that we've gone over different problem variants, partially time-expanded network relaxations and network learning paths, in this section we will go over some of the computational studies and their success when it comes to DDD.
For the computational study, we can quickly mention that different papers used different types of solvers, using anything from the commercial software Gurobi (Scherr et al. 2023; Van Dyk and Koenemann 2024; Boland et al. 2017b; Vu et al. 2022) to CPLEX (Hewitt 2019; He et al. 2020; Marshall et al. 2021; Vu et al. 2019; Escobar-Vargas and Crainic, 2024; Lagos et al. 2022). Also worth mentioning is that the computational time limit varied amongst different papers with anywhere from one hour (Marshall et al. 2021), two hours (Medina et al. 2019, He et al. 2020, Boland et al. 2017; Vu et al. 2022; Pottel and Goel 2022; Lagos et al. 2020; Lagos et al. 2022), three hours (Van Dyk and Koenemann 2024) all the way up to five hours (Scherr et al. 2023; Boland et al. 2017b) of total run time, with different types of memory, cores and speeds of processors. For most papers the fixed optimality gap was set to be less than or equal to 1% (Boland et al. 2017; Hewitt 2019; Van Dyk and Koenemann 2024; Scherr et al. 2023; Shu et al. 2024; Medina et al. 2019; Vu et al. 2022). Now that we've covered the general setup of the computational analysis, in the next section we will introduce instances used in different studies and then proceed with the results and comparison.

9.1. **Instances**

In most of the papers covering the SNDP variants of problems, researches used the C and C+ instances described in Crainic et al. (2001) that have been used as a benchmark of many different algorithms for the network design problems (Boland et al. 2017; Hewitt 2019; Van Dyk and Koenemann 2024; Shu et al. 2024; Marsall et al. 2021). Some papers however did expand these initial 432 instances with some additional ones, where Van Dyk and Koenemann (2024) used a geometric approach for instance generation just for their hub-and-spoke networks and the 432 instances from Boland et al. (2017) for other network designs. Marshall et al. (2021) and Shu et al. (2024) use the 558 instances from Boland et al. (2019), containing the initial 432 instances from Boland et al. (2017) extended with the additional 126 instances. Papers like Scherr et al. (2023), He et al. (2020) and Medina et al. (2019) generate their own sets of instances to tackle the rest of the SNDP variants, some inspired by the real-world scenarios.
Starting with the 432 instances from Boland et al. (2017), these instances vary with respect to the number of nodes (20,30), arcs (230,300,520,700), commodities (40,100,200,400), whether the variable costs outweigh the fixed costs and whether the arcs are loosely or tightly capacitated. The results were done on instances with 100, 200 and 400 commodities since the smaller ones are instantly solved. Travel times were set in minutes, the speed at 60 miles per hour and the commodity available time was drawn from three different normal distributions. The discretization parameters were split into a finer discretization of 1, 5 and 15 minutes and coarse with 30 and 60 minutes.
Regarding the instances for a CTSNDRP problem from He et al. (2020), the 37 instances they used are based on the historical data over one year of operations that contains more than 33,000 orders, where each order was modeled as distinct commodities. To generate the actual instances, they partitioned the data by week, then randomly sampled orders in that

week to generate commodities and added 4 hubs, 6 suppliers and 291 delivery points scattered around France (He et al. 2020). All available 33 trucks are the Europe standard semi-trailers and the cost information regarding specific sets of inbound routes from supplier to hubs, outbound routes from hubs to delivery points, inter-hub routes and direct delivery routes were provided by the industrial partner.

Based on a logistics network from a French 4S Network company, Medina et al. (2019) generated 320 instances for their computational study differing in number of commodities (25,50,100 or 200) , time windows width (tight- 12-24h or wide- 24-36h) and a number of breakbulks (4,6,8,10).

For the special SNDP problem with mixed autonomous fleet, Scherr et al. (2023) create their own set of 50 instances in total for the computational study which defer in the number of external zones, satellites, travel times, cost for using manually guided vehicles compared to autonomous vehicles, cost per unit with a discretization of 5 min.

For a TSPTW variant presented by Boland et al. (2017b), a total of 337 instances were used, mostly taken from previous papers covering TSP problems, varying between 'easy' and 'hard' instances in general.

Vu et al. (2019) test the DDD algorithm on two sets of instances from the literature, containing 952 and 960 instances, respectively, varying in number of locations the vehicle must visit (15, 20, 30,40), degradation of travel speed due to congestion, division of time periods (morning, middle of the day, evening) and number of time intervals (a.k.a. breakpoints). Same set of instances was then later on used by Vu et al. (2022) to make comparisons. Both papers do however add a third set of instances, generated with the similar methodology like the second set, but with a larger number of locations that need to be visited (60,80, 100) totaling with 720 instances for this set.

In hopes of trying to assess their approach in solving TDASP, Pottel and Goel (2022) use instances inspired by an industry example on last-mile deliveries with a homogenous fleet of electric vehicles that are highly dependent on the traffic conditions in the metropolitan areas. Based on the instances generated according to the same characteristics like Solomon (1987), Pottel and Goel (2022) divide the instances between two different clustered customer locations, randomly distributed customer locations and mixed locations each with a time dependent congestion factor between [0,1] for each point in time.

For the IRP problems, Escobar-Vargas and Crainic (2024) presented their 5 new sets of 60 instances each with anywhere between 5, 10, 15, 30 and 50 origin to destinations demands with different variations of time and capacity, varying time window width and demand values. For two different echelons, the vehicle capacity also differs between 200 units for the first and 50 units for the second echelon level.

Lagos et al. (2020) for their CT-IRP problem define two base instances, altered to create additional instances with anywhere between 5 and 15 customers for their base instances with different locations chosen randomly in the first set of instances and in clusters for the second set. Travel times are taken as Euclidian instances rounded to two decimal places with costs that equal travel times and the planning horizon equaling a maximum time of 18 to make sure some customers can be visited multiple times before their return to the depot. For the two types of customer locations, five numbers of customers, three different usage rates and vehicle capacities the total instance set is 90 identified and referenced as a four-tuple (Lagos et al. 2020).

Lagos et al. (2022) for their CR-IRP-OB problem generated 3 instances for each of the 81 types totaling 243 instances, ranging between 10,20 and 30 customers, with 25%, 50% or 75% of

the customers having their storage capacity less than the total vehicle capacity and with three different usage rates. Tests showed that very easy instances often don't provide enough insights into how DDD operates so they replace those with re-sampled instances that need more DDD iterations. For the number of vehicles, UBM provides a number that seeks to minimize the total number of vehicles but doesn't make the problem infeasible or too easy (Lagos et al. 2022).

In order to assess the effectiveness of DDD on the shortest path problem, He et al. (2018) generated random instances with n=20 and T=200 and n=30 and T=200, where the travel time on the arc (i,j) is a piecewise linear interpolant of a function at every integer time point that also satisfies the FIFO property. Authors also decided to multiply time T by the factor of 2,5 and 5, in order to assess the impact of the number of breakpoints on the overall DDD performance.

He et al. (2022) for their SPP problem generate instance by first considering how three different types of network topology affect the effectiveness of the DDD algorithm and introduce corridor, grid and triangle networks that resemble urban city layouts with bidirectional arcs and base travel time that satisfies FIFO properties. For the travel function, authors consider continuous cubic splines which are converted into piecewise linear functions, designed to mimic 10-hour urban period with morning and evening rush hours. Parameter T indicates the total time horizon and serves as an indicator of the fineness of the discretization through the total number of linear pieces (He et al. 2022). The total number of instances is over 10, for n=50, T={20,40,60,80,100}, three types of networks and two types of travel time functions.

9.2. **Comparing results**

9.2.1. **Service Network Design Problem**

We will start the comparative analysis of the computational studies with the paper from Boland et al. (2017), since they introduced the DDD algorithm first and then we will compare the results from other authors over the same instance set to see how some potential improvements in the algorithms affect the overall results. They start their computational analysis by comparing the results between using the full discretization (FD) and the SOLVE-CTSNDP (for the remainder of this paper referred to as DDD), comparing the two over different discretization parameters, optimality gap at termination and percentage of instances solved. It is noted that DDD never exceeds the 16GB memory limit, whereas FD does so, however only with Δ=1 for nearly 39% out of 432 instances, hence why we will report the results for other discretizations Δ separately (Boland et al. 2017). On coarser discretization with Δ=30 and Δ=60 min both the FD and DDD perform similarly in terms of time needed to termination and instances solved to optimality, but DDD clearly outperforms the FD on finer discretization (with Δ=1, with Δ=5 and Δ=15 min) since it requires less time to solve more instances and even for the ones that it couldn't solve to optimality it provides a smaller optimality gap (refer to the Appendix- Figure 1 & 2) (Boland et al. 2017). While DDD remains effective when Δ becomes finer, FDs performance deteriorates, which could be explained with the fact that DDD works with a significantly smaller time-expanded networks while searching for a provably optimal solution to CTSNDP compared to FD, especially for instance with Δ=1 (Boland et al. 2017). This is further corroborated by the analysis that for all Δ, both fine and course, the last integer programs solved by DDD are significantly smaller than those

by FD, never exceeding 45% in FD over all Δ and 25% in FD maximum for finer discretizations (refer to the Appendix Figure 3). For finer discretizations, compared to the FD, DDD provides significantly smaller IPs showing that the finer the discretization, the smaller the relative size of the final SND($\mathcal{D}_T$) (Boland et al. 2017). When it comes to the size of partially time-expanded networks created and refined by DDD, we see their size remains stable over different discretization during the execution of the algorithm, with relative size increasing from 10% to 18% (Boland et al. 2017). In conclusion, DDD outperforms FD because it starts with a significantly smaller time-expanded network and refines it in such a way that it grows only modestly, resulting in IPs significantly smaller to solve by DDD compared to FD (Boland et al. 2017). When it comes to the primal and dual gap conversion, it can be observed that both gaps converge quickly with the quality of the primal solution and the strength of the dual bounds produced performing consistently across discretizations, making DDD robust to the change in the discretization parameter Δ (Boland et al. 2017).

One of the biggest assumptions of the algorithm that affects the objective function was that the holding cost was set to zero (Boland et al. 2017). To see how the holding costs can affect the effectiveness of the DDD algorithm, Shu et al. (2024) run experiments with Δ=5 min comparing the DDD algorithm solving CTSNDP from Boland et al. (2017), which is taken as a baseline algorithm to compute heuristic solutions for CTSNDP-HC and noted as EXM-0, and their enhanced DDD algorithm that accounts for the holding costs, noted as EXM. As a measure of tractability, the instances were also grouped according to their flexibility and cost ratio, either being high or low (HC, LC, HF, LF). The results suggest that EXM algorithm performs better than the EXM-0 when used to solve instances with zero holding cost, solving more instances to optimality and providing better optimality gaps for the unsolved instances (Shu et al. 2024). The algorithm EXM also requires smaller number of iterations and a shorter computing time to converge to optimality, which can be accounted to the effectiveness of the valid consolidation inequalities (refer to the Appendix Figure 4). Instances of groups LC/LF and LC/HF in the CTSNDP-HC can easily be solved with EXM, whereas instances with HC/LF and HC/HF are more difficult to solved to optimality, because a trade-off between fixed, flow and holding cost is difficult to achieve (Shu et al. 2024). Compared to EXM-0, EXM produces better lower and upper bounds for the CTSNDP-HC and for the most difficult category of instances HC/HF it improves the solutions derived from EXM-0 significantly, meaning that significant cost savings can be gained by taking holding cost into account when solving CTSNDP (Shu et al. 2024). Comparing the size of the final size of the algorithm, for some categories of instances it's obvious that EXM requires sometimes twice the number of variables and constraints (refer to the Appendix Figure 5), indicating an increased complexity of the problem when incorporating holding cost. Since the refinement strategy is done in two phases, study shows that most of the time points are added in the first refinement stage for all groups of instances (Shu et al. 2024). Solutions computed with EXM achieve only negligible reduction in holding times and cost compared to the solutions derived from EXM-0. This is achieved through a decrease in the percentage of consolidations, which reveals the significance of adding HC on the decision between consolidation and higher holding costs (Shu et al. 2024). Constraints that are added to tighten the lower bounds and restrict infeasible consolidations significantly affect the effectiveness of the algorithm, since by introducing them the algorithm is able to solve more HC/HF instance to optimality by 4% and tighten the lower bound in first iteration and overall (Shu et al. 2024).

In attempts to make the DDD algorithm even more effective, Hewitt (2019) introduced a set of enhancements in hopes of achieving even better results comparing the two-phase DDD

against the original DDD algorithm. His study shows that the two-phase DDD was able to solve 2% more instances in less time, although it required more iterations, from which the majority was spent completing the first phase (Hewitt 2019). Since linear programs are solved in the first phase, the time required to complete the first phase was low, which contributes to the fact that two-phase DDD in general needs less time to solve more instances than DDD, as it solves less integer programs on average. Regarding partially time-expanded networks, we observe that they are smaller when solving two-phase DDD which could suggest that the IPs it solves in the second phase may be easier to solve as well (Hewitt 2019). In the first phase of two-phase DDD, the algorithm was able to solve 25,83% of the instance to optimality and get almost near optimal solutions by the end of the first phase alone, typically in under two minutes (refer to the Appendix Figure 6). For instances that weren't solved, the reason was a weak lower bound found at the end of first phase but nevertheless, the algorithm still produced solutions with 3-5% optimality gap at the end of phase one (Hewitt 2019). Two-phase DDD was able to produce many of the necessary time points in the first phase alone and for 40% of instances a primal solution within 1% of optimal and for 86% of instance primal solution within 2% of optimal. Although with the two-phase enhancement the algorithm showed superior results to DDD already, but with the addition of ATW inequalities, the algorithm was able to solve additional 2% more instance and, in less time than just the two-phase DDD (Hewitt 2019). The ATW inequalities produced higher quality primal solution and stronger dual bounds in both phases. The enhanced refinement process, the Symmetry-Refine, was also able to help solve more instances to optimality in less time, on average under 15 minutes (Hewitt 2019). Combining the two enhancements, the Two-phase DDD with ATW and Symmetry-Refine excels on every statistic, form number of instances solved, optimality gaps to the quality of primal solution by the end of first phase (refer to the Appendix Figure 7).

Based on the same set of instances, we can also compare the effectiveness of the interval based DDD algorithm to the original DDD from Boland et al. (2017) and the full discretization (Marshall et al. 2021). Comparing DDDI with full discretization shows that all DDDI variants presented by the authors significantly outperform the full discretization model (refer to the Appendix Figure 8). The four DDDI variants studied were the Default, Reduced Iterations, Fewer Time Points and Adaptive (uses Fewer Time Points refinement strategy, incorporates the redundant timed-arcs removal and employs an adaptive optimality tolerance for the IP solver), where all four variants used the additional strengthening constraints and warm-starting or the solver (Marshall et al. 2021). The most challenging instance to solve were the HC/HF, but despite the challenge, DDDI variants demonstrated the ability in finding high quality solutions for these difficult instances within the one-hour limit (Marshall et al. 2021). The Adaptive DDDI variant managed to achieve the best average optimality gap for HC/HF instance of 0,84%, compare to Default, Fewer Time Points, Reduced Iterations and FD with optimality gaps of 13,12%, 8,53%, 8,94% and 49,69%, respectively (refer to the Appendix Figure 8). With its quick initial convergence even for the HC/HF instances, Adaptive spent most of its time dedicated to proving the optimality (Marshall et al. 2021). In the HC/HF category of instances, Fewer Time Points strategy resulted in a smaller average gap but the Reduced Iterations solved more instances to optimality, which highlights the trade-off in refinement strategies between reducing model size and rapidly proving optimality that might benefit from adding extra time points (Marshall et al. 2021). Compared to FD, DDDI has a crucial advantage in the ability to generate significantly smaller IP models in terms of the number of variables and constraints, which makes DDDI a much more attractive approach for

solving large-scale instances (Marshall et al. 2021). Across all instance groups, DDDI variants maintain stable model sizes compared to FD, where most of the model growth occurs near the beginning of algorithm's execution. Regarding model size, it's expected that Reduced Iterations produces largest models due to fixing multiple issues per iterations, while Fewer Time Points variant induces smallest models due to its emphasis on unary time points (Marshall et al. 2021). One of the significant enhancements to DDDI is the valid inequality for strengthening the IP model which helps the Adaptive variant nearly halve the average optimality gap and solve nearly 10% more HC instances to optimality, although these constraint roughly doubled the total number of constraints in the IP which is still relatively small compared to the FD model (Marshall et al. 2021).

DDDI compared to DDD introduces several fundamental differences that lead to better performance. For the most challenging HC/HF group of problems, the Adaptive variant of DDDI compared to DDD solves more instances to optimality with a much smaller model size, with 12 times less the number of variables and over 5 times less the number of constraints in compared to DDD variant (Marshall et al. 2021). However, DDD has on average a lower final gap, meaning for the instances it didn't solve it produced a smaller gap. For instances not solved to optimality, DDD's gap is up to 3 percentage points lower than DDDI's, but DDDI tends to solve more instances in much less time. DDDI is able to converge rapidly for all groups except for HC/Hf where it struggles, while DDD on average continues to find better solutions (Marshall et al. 2021). While the refinement process for DDDI excels in terms of model size it still lacks in terms of finding better upper bounds to close the final gap, making the refinement of DDD more effective in this respect. For all categories of instances on average, we can say that DDD produces a better final gap in a smaller number of iterations, but DDDI needs a lot less time, variables and constrains as well as solves more instances to optimality (refer to the Appendix Figure 9).

The original and other problems we covered in this section were all node-based, but we also want to see how the algorithm behaves if times are encoded over the arcs producing an arc-based DDD (Van Dyk and Koenemann 2024). Both arc-based DDD and traditional node-based DDD solve the majority of instances (182 out of 192) within the three-hour limit in SND variant with designated paths, but the arc-based DDD completed in 42,6% of the time of node-based approach, reaching an improvement of 57,4% in runtime on average (Van Dyk and Koenemann 2024). For each decile for the runtime of each algorithm, arc-based DDD clearly offers a significant improvement over the node-based approach (refer to the Appendix Figure 11). Even though arc-based DDD terminates with an average increase in additional iterations of 18% compared to node-based DDD, there is still 55% less variables and 60% less constraints in the final iteration of arc-based DDD, which in return helps reduce the overall runtime. The average size of the lower bound formulation solved in the arc-based DDD is also significantly smaller which is a more significant difference than the average gap between dual bounds in each iteration for the two algorithms (Van Dyk and Koenemann 2024). Due to a significantly smaller size in the formulation, arc-based approach takes less time to reach a fixed percentage gap between these dual bounds. In an SND with hub-and-spoke variant, for the instances solve we can also observe a significant improvement in runtime with the arc-based DDD. The majority of instances (139 out of 192) were again solve to optimality by both approaches within the time limit, with arc-based DDD completing in 58,2% of the time of the node-based approach, representing an improvement of 41,8% in runtime on average (Van Dyk and Koenemann 2024). In this variant as well, the arc-based approach terminates with 2% more iterations on average, but there are only 66% of variable and 69% of constraints in

the final iteration compared to node-based approach (Van Dyk and Koenemann 2024). The fraction of instances solved over time is also consistently better for the arc-based approach compared to the node-based (refer to the Appendix Figure 12). The comparative performance for hub-and-spoke instances is consistent with the one for designated path instances (Van Dyk and Koenemann 2024). For the instances with critical times, a moderate improvement was achieved when using arc-based approach (Van Dyk and Koenemann 2024). Majority of the instances (174 out of 192) were again solved within 1% of the optimality within the set time limit by both approaches, with arc-based finishing in 87,6% of the time of the node-based approach, representing an improvement of 12,4% in runtime. When comparing each decile of runtime and optimality gap for both approaches, in the critical time instances, arc-based approach shows a larger optimality gap compared to node-based approach (refer to the Appendix Figure 13). But since for all previous deciles, arc-based approach consistently showed improvement over node-based, this might not be a compelling argument against an arc-based approach, but rather a hint that for some instances one approach may perform better than the other based on different factors (Van Dyk and Koenemann 2024).

We can now go over some results on the enhancements to the original DDD algorithm proposed by Scherr et al. (2023) to see if they improve the original algorithm and are worth exploring. Compared to the Gurobi solver solutions, all three DDD variants solving SNDMAF variant (the DDD adapted for this variant of SND, two-phase DDD and partially relaxed DDD) are able to outperform the results in terms of computational runtime and solutions quality (Scherr et al. 2023). The results show that DDD requires the smallest partially time-expanded network since the algorithm solves IPs to produce a lower bound with a solution quality better than the one from Gurobi. However, two-phase DDD is the fastest exact algorithm that requires the smallest number of iterations and provides the best solutions with the smallest gap and solves 90% of instances to optimality, but with the cost of having the largest partially time-expanded network to explore compared to other DDD variants (refer to the Appendix Figure 17). After one-hour runtime Gurobi is able to solve merely 32% of instances to optimality while in the same amount of time the best variant of DDD, the two phase DDD, solves 80% of instances (refer to the Appendix Figure 17). The partially relaxed DDD variant requires more iterations to solve due to exploring fewer nodes and arcs, which in return provides slightly better solution quality and runtime compared to DDD (Scherr et al. 2023). Regarding runtime, all variants of two-phase DDD require only half of the runtime of the solver and outperform the DDD and partially relaxed DDD, which are still 38% faster than the solver (Scherr et al. 2023). When we look closer into the two-phase DDD, we can say that it requires far less time solving the phase one involving on average only 3% of the runtime and that adding the valid inequalities and a cap doesn't contribute to much to the solution quality or runtime (refer to the Appendix Figure 17). In conclusion, all DDD variants show better results compared to the use of commercial solver and the two-phase DDD regarding runtime, solution quality and solving instances to optimality exceled the other DDD variants which shows the benefit of using DDD to create initial network.

9.2.2. **Traveling Salesman Problem**

Now that the results for different SNDP variants were introduced and compared, most even on the same set of instances, in this subsection we will introduce results of the computational studies for TSP problems based on the previously introduce characteristics and instance sets.

DDD solving the TSPTW from Boland et al. (2017b) managed to solve 313 out of 337 instances to optimality and for 329 instance it managed to produce equal or better value of the objective function then the best-known in the literature. For most of the instances that timed out, the solutions were still very close to optimality, close to best-known solutions with small gaps as well. Compared to the fully time expanded networks, study shows that DDD on a partially time expanded network needs a very small number of arcs and nodes to produce high quality solutions (refer to the Appendix Figure 18). The adaptation of the DDD algorithm to cover TSPTW problem shows great results, especially if combined with enhancements which were covered in previous section.

When looking at the adapted DDD algorithm that solve a TD-TSPTW problem, Vu et al. (2019) first compare it to benchmark results of TTBF-BC on both sets if instances that were introduced earlier. On the first set of instances with fewer breakpoints, DDD-TD-TSPTW managed to solve all 952 instances in a shorter amount of time on average compared to TTBF-BC that struggled with solving 12 instances in total. While for a traffic pattern A and B, DDD-TD-TSPTW needed 12,91 s and 8,79s on average, TTBF-BC needed 31,99s and 20,44s, respectively, which means that DDD manages to solve all instances on average with more than twice the speed (refer to the Appendix Figure 19). On the set two, that has more breakpoints, we see that DDD remains very robust and solves all instances for traffic patterns A and B in 1,31s and 1,14s, compared to TTBF-BC which needs 100,13s and 93,10s on average, respectively, meaning DDD significantly outperforms the other algorithm (Vu et al. 2019). The robustness to breakpoints and time-dependent granularity is visible in DDD algorithm in a sense that while the number of breakpoints increased from set one going to the set two, DDD actually needed less time to solve the instances, which can mainly be contributed to DDD operating on a time expanded network, where changes in speed due to dispatch times only require updating arcs (refer to the Appendix Figure 19). Notably, even with adjustments for integral values, DDD still produced the same objective function value as TTBF-BC when evaluated with the original problem data (Vu et al. 2019). DDD also outperforms the other algorithm in terms of scalability with locations, where, as the number of locations increased, DDD showed a much smaller growth rate in solve times compared to TTBF-BC, which is largely affected from the number of locations and breakpoint, probably due to weaker LP relaxations (Vu et al. 2019). For DDD it was noted that the calculated dual and primal gaps were not greatly affected by the increasing number of locations, with dual gap even decreasing (Vu et al. 2019). Contrasting TTBF-BC, the dual gap was smaller in the second set with more breakpoint than the first set. DDD also showed no conclusive evidence of a relationship between congestion level or time window width and DDD solve times, further reiterating the algorithms robustness to different parameters and superiority against TTBF-BC for makespan objective (Vu et al. 2019). Next the authors start by comparing the performance of base DDD algorithm and the proposed enhancements in form of adding valid inequalities and primal IPs (heuristic candidate solutions). The results show that the DDD was quite effective in solving the second set but struggled with the first set and that by adding the valid inequalities the performance didn't improve significantly (Vu et al. 2019). However, the primal IPs drastically reduced the solve time and the number of iterations, especially for the first set, and had fewer nodes at termination, which could be attributed to the ability to quickly produce strong dual bounds allowing for accurate termination once a near optimal solution is found (Vu et al. 2019). Using both primal solutions yielded the best overall results, compared to solving them separately where the first primal IP was more effective alone than the second. The best achieved performance was when primal IPs and valid inequalities were combined,

accomplishing the fastest run time for both instance sets (Vu et al. 2019). For the third large set of instances, DDD still solve almost all instances with 60 and 80 location, with gaps under 3% for the ones it didn't solve. With 100 locations, DDD began to struggle, solving fewer instances with larger optimality gaps for unsolved ones, which could be improved with better IP solvers (Vu et al. 2019). Speaking of the duration objective and the necessary adaptations that were also investigated in the paper, the computational study made on the same instance sets shows that DDD can solve almost all instances of both sets, but compared to the makespan objective function takes longer time, with averages for set one and set two of 143,45s and 106,20s for the duration objective and 10,94s and 1,22s for makespan objective, respectively for each set (Vu et al. 2019). Notably, the larger integer programs solved at each iteration due to adding additional waiting nodes at the depot make both the initial and final sizes of the node set roughly double in size of the duration objective compared to makespan objective, which could explain the longer solve times of duration objective (Vu et al. 2019).
For the duration objective, the TD-MTDP problem, in order to combat the double growth of the problem when adding additional waiting nodes at the beginning of the problem, Vu et al. (2022) proposed an enhancement to add the waiting nodes at the depot dynamically. This enhancement creates only 12,85% of the nodes of DDD from Vu et al. (2019) for set one and 17,05 % of node from set two, ultimately creating fewer nodes with dynamic node addition (Vu et al. 2022). This enhancement also significantly reduced the overall network size enabling convergence in less time and greatly improving the DDD performance for TD-MTDP. When assessing the repair mechanism, authors conclude that repair enabled DDD-TD-MTDP to produce feasible solutions and solved more instances, but with a slightly larger solve time (Vu et al. 2022). Also, for the 317 out of 592 instances for DDD-DT-MTDP that produced a primal solution, repair mechanism managed to find better primal solution, meaning that all of the proposed enhancements together significantly improved the effectiveness of solving instances than Vu et al. (2019). When assessing TD-DMP, authors benchmarked the DDD's performance against Gurobi, and the results showed that for both set one and two, DDD solved all instances of DDD-TD-DMP while Gurobi solved all but three (Vu et al. 2022). Also, on the set of instances MM-TDTSPTW, DDD didn't solve all instances but still managed to solve more than Gurobi and in much less time, with an average optimality gap of 1,73% for the ones it couldn't solve (Vu et al. 2022). DDD-TD-DMP, in 5,35 iteration on average, is able to produce feasible solutions and stronger dual bounds for TD-DMP, than DDD-TD-MTDP, with 10,12 iterations on average, can for TD-MTDP, with the only difference between the two being the objective function (Vu et al. 2022).

9.2.3. **Scheduling Problem**

So far, we've seen the impact of time dependency on TSP problems, but in this sub-section we will go over some computational results for the time dependent activity scheduling problem with and without replenishments. The first set of experiments compares the effectiveness of the DDD algorithm compared to the fully time expanded network with the influence of the changing choice of discretization parameter. Compared to the fully time expanded network, DDD generates a small percentage of vertices for more or less all discretization parameters, also showing that by reducing the discretization parameter the number of vertices only grows slowly (Pottel and Goel, 2022). Although in general, increasing the discretization parameter doesn't lead to better results, it was shown that going from 0.1 to 1.0 has a decrease in completion time, which is explained by different savings values

calculated for different discretization parameters. When the batteries have the recharge option, DDD results with discretization parameter of 0.1 are compared to results by CPLEX solver with default settings based on some additional instances where vehicles are allowed to recharge batteries at the depot and public charging places (Pottel and Goel, 2022). For the two-hour limit, CPLEX fails to produce a solution for many instances compared to DDD approaches which are much faster in general and terminate for close to all instances on average in case of TDASPR (refer to the Appendix Figure 20). Since instances in C1, R1 and RC1 have shorter planning horizons and smaller freight capacities, more vehicles are used for them, the vehicles visit fewer customers in routes and require less computational effort compared to C2, R2 and RC2 instances, hence why the likelihood of running out of energy is smaller. For sets of R1 and RC1 instances, vehicles make no stops to replenish the energy since the instances have such short planning horizons leaving little to no time for replenishments (refer to the Appendix Figure 20). With increasing number of charging stations, the routes change as well as the computational effort of evaluating the alternatives, which can be countered with preloading of vertices, which helps stabilize the run time (Pottel and Goel 2022). The results conclude that the recharge option reduces the number of routes in some cases by one third and the number of vehicles as well as the total completion time (refer to the Appendix Figure 20). When it comes to different recharging places, allowing to recharge at the depot already accounts more than enough to the solution quality and completion time even without public recharge stations, which provide only small contribution to the completion time.

9.2.4. **Routing Problem**

When it comes to routing problems, in this subsection we will focus on the analysis of three different computational studies covering 2EMALRP, CT-IRP and CT-IRP-OB problems. Starting with the 2EMALRP problem from Escobar-Vargas and Crainic (2024) the first study focuses on the performance of the hybrid formulation which isolates constraints 2-22 (from a total of 32) from the problem alongside the objective function where the hybrid formulation and the standalone time-space formulations are solved for the complete time-space network with Δ time periods and different granularity values (Escobar-Vargas and Crainic, 2024). For small and medium instances (5,10 and 15 ODs) and the reduced time-space networks with granularity values of Δ= 50 and Δ=25 the results reported show the similar performance with respect to the upper bounds but a significant difference in the lower bound and run time values (refer to the Appendix Figure 21). Linear relaxation of the hybrid formulation provides better lower bounds on average by 14% compared to the standalone formulation, but with the downside of slower convergence to optimality. Vehicle index in the commodity flow variables and redundant continuous time constraints are what is improving the overall lower bound in the hybrid formulation. For the medium instances with 15 origin to destination deliveries, the problem gets more difficult to solve for both formulations, providing far less feasible and optimal solutions, which can be explained by the longer schedule lengths and larger number of OD (refer to the Appendix Figure 21). The standalone time-space model shows scalability issues with larger instances due to the size of the time-space representation of the network leading to large, less solvable integer programs (Escobar-Vargas and Crainic, 2024). Same issue is present in hybrid formulations, where size of the problem, especially when valuated on the complete time-space network, results in long exploration times and fewer optimal solutions. From this can be concluded that both formulations get more

comparable when the number of ODs gets larger and they both struggle with finding optimal solutions within given time (refer to the Appendix Figure 21). Speaking of discretization granularity, the results present a trade off between result accuracy and the performance of the time-space models, where finer discretization shows a more accurate representation of the problem with the downside of having a larger integer problem (Escobar-Vargas and Crainic, 2024). While calculating with solution quality losses of 1,3% and 1,9% for hybrid and standalone formulation, respectively, the reduction in computational time od 77% and 50% can be observed by coarsening the discretization. Overall results, however, clearly show the hybrid solution superiority even with only using a commercial solver, since it consistently provides less-degraded solutions compared to the classical time-space formulation.

Continuing with the IRP problems, Lagos et al. (2020) prove their concept by experimenting with different discretization parameters and report the results on solving the LBM and using an RPO model that converts the solution into a CIRP feasible solution. The experiments show that for most instances high quality solutions were found, especially with instances where the locations were clustered compared to random instances where twelve out of 90 instances had an optimality gap of more than 10%. Since the objective was to minimize the number of vehicles used, this objective doesn't take the overall cost efficiency into account resulting in a large gap in both the UBM and RPO models (Lagos et al. 2020). It was shown that for almost all large instances with random location, in case one extra vehicle was available, low cost solutions could be obtained and all of the solutions were either optimal or close to optimal, showing that the impact of the number of vehicles was significant for lowering the overall costs. When experimenting with different finer and coarser discretizations, experiments were less than favorable and showed that the best lower bounds were obtained on coarser discretizations, which means that more time points doesn't lead to a better result but also on the other hand that fewer time points can deliver a great result (Lagos et al. 2020). Another valid observation was that the incorporation of valid inequalities showed an improvement in both the upper and lower bounds for different discretization values and enabled RPO to extract two more feasible delivery schedules when the LBM had valid inequalities incorporated (Lagos et al. 2020).

For the extension of the CIRP problem with out and back routes, Lagos et al. (2022) state that their algorithm either finished with and optimal or feasible solution or without a solution at all, where in all final solutions there were no multiple vehicles visiting a customer at the same time. The results show that DDD is able to solve 206 out of 243 instances to optimality, finds a feasible solution for 25 of them and the remaining 12 instances remain without solution, mostly the difficult instances (refer to the Appendix Figure 22). DDD had no issue solving almost all instances with 10 customers but struggled with 30 customers solving 57 out of 81 instances optimally, which shows that the increase in the number of customers makes solving the problem more challenging (Lagos et al. 2022). Similar to Lagos et al. (2020), experiments show that instances with clustered customers are easier to solve and require fewer time points over non clustered. With an increasing number of customers and the usage rate, the number of necessary vehicles also increases (refer to the Appendix Figure 22). With the increasing number of customers, the number of necessary time points in the final solutions also increase, which means that more time points are needed possibly due to more flexibility in the vehicle itineraries and combination of customer visits (Lagos et al.2022). With an increasing number of customers, time to solve also increases which is shown in the results that 80% of instances with 10 customers are optimally solved in 20 minutes compared to 70% of instances with 30 customers that require the full two hours to reach optimality. For a

significant number of instances, the DDD algorithm needed more than 50 iterations, which further confirms previous results (Lagos et al. 2022). Lagos et al. (2022) introduced different variants of DDD which incorporate branching, valid inequalities, visits and waiting conditions and they experiment all of these mechanisms on instances with 20 customers over two hours. The results show that all strengthening mechanisms lead to an improvement in the performance, where the full DDD has more instances with optimal and feasible solutions over other variants without these algorithm strengthening measures, where not using the valid inequalities has the worst impact on the solution quality over all five measures (Lagos et al. 2022). These results correspond with other papers like Lagos et al. (2020) and further confirm the impact of the valid inequalities.

9.2.5. **Shortest Path Problem**

For the shortest path problem, He et al. (2018) compare the performance of DDD and the enumeration algorithm and report different metrics. The results show that DDD investigates only a small fraction of breakpoints compared to the enumeration algorithm and a small number of breakpoints from the overall number of them. This fraction of explored breakpoints decreases with the increase in both n and T (He et al. 2018). For the coarse discretization, enumeration outperforms the DDD, but when T is multiplied by 2,5 and 5, in those cases of finer discretization, DDD significantly outperforms the enumeration with regards to the average solve time.
He et al. (2022) try to investigate the impact of the choice of the next breakpoint on the algorithm and consider the scenarios of picking the breakpoint at random (RAN), picking median breakpoint (MED) or the minimizer of $c_{i,j}(t)$ over t (MIN). For the MTTP, they compare adding a single (SIN) breakpoint over multiple (M) for each travel subpaths in each iteration. Since the experiments show that in MTTP, more than 90% of the time is spent solving SPP on the time expanded network, the authors chose to add multiple breakpoints aggressively (AGG) in order to reduce the number of SPP solved (He et al. 2022). The scheme (MIN, S) results in the minimal number of breakpoints explored, whereas (AGG, M) results in the minimum number of iterations, which shows a tradeoff between the number of iterations and number of explored breakpoints, ultimately affecting the solve time. Same as He et al. (2018), He et al. (2022) compare the effectiveness of DDD with enumeration algorithms, in which they use MIN scheme. For the MDP, the algorithm explores a significantly smaller number of breakpoints and compared to MTTP, with the fraction decreasing with finer discretization (refer to the Appendix Figure 23 and 24). MDP benefits in terms of breakpoints with the smaller computing time, whereas MTTP's computing time relies on the class of instances, outperforming enumeration for corridor and triangle topology and finer discretization (He et al. 2022). Even with mixes results in the case of MTTP, it still shows a clear advantage as T increases and the discretization gets finer. When comparing MDP and MTTP with regards to breakpoint number, MTTP requires 2n mangroves in the first iteration compared to only 2 ABSPTs for MDP, which affects the number of explored breakpoints with a significant fraction for smaller T values and coarser discretization. DDD shows its biggest strength solving instances in finer discretization but struggles with coarser ones, which shows that in the MTTP case it's more challenging to develop a DDD algorithm compared to MDP (He et al. 2022). In order to see how well DDD algorithm for MDP scales, additional large instances were generated with n=1000 and n=10000 and the final experiment showed that DDD indeed scales well. Only a small fraction of the number of breakpoints is investigated,

requiring far less computational time compared to the enumeration algorithm (He et al. 2022). With the increasing number of nodes in the networks, the algorithm still explores only a fraction of total number of breakpoints thanks to using the ABSPT, which are able to represent all node breakpoints simultaneously (He et al. 2022). The algorithm is able to perform very well with finer discretization, where the use of finer discretization doesn't make instances harder to solve for the DDD, thanks to the breakpoint selection scheme that identifies the optimal solution faster with finer discretization (He et al. 2022).

9.3. **Performance on real-life data**

In order to test how DDD performs in practice on real life data, Boland et al. (2017) created new five instance of an LTL carrier from the US, operating in nearly all states over a five-day planning horizon with freight originating at five different time points. According to the carrier, finest implementable discretization was Δ=30 min. From the results it's visible that DDD is able to solve only instances with four states, unable to solve larger instances but still able to produce a primal and dual bound with a provably small gap (Boland et al. 2017). For large instances, with 3-4% optimality gap, DDD could be used as a heuristic. Compared to DDD, FD is also able to solve only smaller instance but with a larger optimality gap with lower quality primal gap than DDD. The difficulty at solving the larger instance could be attributed to a large number of commodities and length of planning horizon, impacting the size of the IP solved at each iteration (Boland et al. 2017).

On the same instances from an LTL carrier from the US, over a two-hour computational time, the results clearly show that DDDI outperforms DDD and FD for these instances, since it finds high quality solutions quickly and also proves these solutions are of high quality (Marshall et al. 2021). DDDI compared to DDD on average finds a better gap in much less time, never hitting the two-hour limit like DDD for four out of five instances (refer to the Appendix Figure 10).

Another case study-based assessment of DDD algorithms' strength was done on a US LTL carrier based on data over five days, covering 38 states with the integrated of local and long-haul transportation, where the freight is available at the origin terminal at 7 p.m. and due at its destination terminal by 8 a.m. on the delivery day (Hewitt 2019). Due to a large number of states and instances with over 50,000 commodities, the optimization was focused only on the states in the Pacific Northwest, based on the freight that both originates and terminates in the Pacific Northwest, but also freight that originates outside and is destined within and originates within and terminates outside of the region (Hewitt 2019). The carrier operates a hub-and-spoke structure with the finest possible discretization of Δ=30 min. The 10 instances derived from carrier's data differ in number of states, commodities, nodes and arcs. The analysis shows that the Two-phase DDD with Symmetry-Refine and Path-ATW is able to solve all instances that optimize for six states or less within 1% of optimality tolerance, but struggles with more states and is able to produce a primal solution and a dual bound that yield an optimality gap of 2,48% on average (Hewitt 2019). If the first phase was used as a heuristic, for instances that weren't solved, the algorithm is able to produce in 382 seconds a primal solution with an objective function value within 2,96% of optimal on average, for instances it is able to solve, the first phase requires only 63 second to produce an objective function value within 1,96& of optimal. The solutions obtained from DDD are able to affect managerial decisions in a sense that the loading and unloading process determine the dispatch times of

trucks, which could be optimized if enough of the work force is scheduled to work during dispatch timings determined by the algorithm (Hewitt 2019).
In order to understand computational advantages of different strategies for their CTSNDRP problem based on the previously explained real world instances, He et al. (2020) introduced different scenarios like the base scenario that contains routes of the form P-H, H-H and H-D or H-D-D, scenario D which is scenario B with direct routes of the form P-D, scenario MD that allows delivery points to receive deliveries from any hub in the network and scenario PDR that is scenario B where routes are allowed to visit different types of facilities of the form P-H-H and P-P-H and P-H-D. Combining the strategies is also possible like e.g. MS+PDR+D, which allow sourcing from different hubs, direct deliveries from hub to delivery point and routes to visit multiple facilities. Allowing more flexibility in the routing strategy can significantly reduce the total cost, which we can see in the scenario D whit cost reduction with an average of 22%, in scenario PDR with an average reduction of 18,3% and scenario MS with an average reduction of 8,4% (He et al. 2020). In general, direct shipping is advantageous when it's allowed which in return reduces total costs and number of routes in each category (refer to the Appendix Figure 14). The combined scenarios MS+PDR enable greater savings than the two independent scenarios, but in return are more difficult to solve due to a huge number of initial routes added to the route-expanded network (He et al. 2020). When direct shipping is allowed, the impact of MS and PDR routes is smaller, which could be due to the fact that every two points in France could be reached during one day and if the consolidation opportunities are reduced it is more advantageous to use direct delivery (He et al. 2020). For every scenario, DDD-based algorithm outperforms the full discretization network solved using CPLEX since it finds feasible solutions for every instance CPLEX doesn't, produces far more provably optimal solutions and solutions of higher quality in less time (refer to the Appendix Figure 15). For difficult cases (MS+PDR and MS+PDR+D), CPLEX couldn't find feasible solutions for 17 out of 37 instances because if reached the memory limit, proving that the DDD outperforms the benchmark CPLEX since it produces feasible solutions with average gaps performing much better than the one provided by CPLEX (He et al. 2020). If the difficult scenarios were warm started with the initial solution from the easier scenarios, DDD is able to prove optimality for more instances and produce solutions of provably high quality for instances it couldn't solve without the warm start (He et al. 2020).
Solving an SNDRP problem with DDD on the instances generated based on the geographical and temporal aspects of a real-world company was the focus of computational study from Medina et al. (2019). Since the SNDRP contains both the service network and vehicle routing component, the computational study researched the effects of both arc-based and route-base formulations. For route-based formulations we see that while it allows us to model complex rules of local delivery routes, it become easily difficult to solve primarily due to the size of these instances that results from enumerating routes a priori (Medina et al. 2019). The number of commodities has a significant impact on DDDs ability to solve the instances as well as the number of breakbulks, as in the case of 25 commodities the percentage of instances it can solve to optimality decreases with the number of breakbulks (refer to the Appendix Figure 16). For instances with 25 and 50 commodities, for all numbers of breakbulks the optimality gap is under 1% and 2% on average and the time to optimal on average under 12s and 670,5s, respectively, which suggests that for a smaller number of commodities DDD is able to produce high quality solutions in a short amount of time (refer to the Appendix Figure 16). Similarly, can be said for instances with 4 breakbulks where in reasonable amount of time, for all commodities, DDD is able to produce high quality solutions with a maximum optimality gap

of 1,55% (200 commodities) and 100% optimal solution for smaller numbers of commodities. For instances solved with an arc-based formulation, DDD produces feasible solutions for every instance and for many instances needs a short amount of time and for overall all but the largest instances it's able to produce a solution with a small optimality gap (Medina et al. 2019). While the percentage of instances solved as the number of breakbulks grows remains roughly the same, the time to solve increases (Medina et al. 2019). The analysis showed that the performance of DDD is highly sensitive to the width of commodity time windows, since, on average, it's able to produce a feasible solution for 97,06% of instance with a tight and only 48,57% of wide instances (Medina et al. 2019). When comparing arc-based and node-based DDD formulations, we can say that arc-based formulation is able to solve more instances and is much more robust with respect to the increasing number of breakbulks with a much slower degradation of solving instances compared to the node-based formulation (Medina et al. 2019). When trying to validate combining SNDP and VRP into a combined SNDRP problem, results have shown that the integrated problem yields nearly 3% cost savings on average, with higher savings in instances with narrower time windows, but at the cost of computational time increasing compared to individual problems (Medina et al. 2019). Also notable is that the savings increased with the number of commodities suggesting that even higher savings may exist for larger instances (Medina et al. 2019).

In order to compare the MILP and MaxSat solver effectiveness on a DDD-TRP problem, Croella et al. (2024) use 72 real-life instances derived from a single-tracks Norwegian railroad network, later named as Line A and Line B, which vary in the length of the rail, number of stations, trains and track segments. Real world train rescheduling and dispatch optimizations are presented with bigger challenges and have more value to dispatchers when many trains have large delays. With mandatory dwelling time at the station similar to the real world scenario, trains are then unable to catch up with the delays by introducing shorter dwelling time and to simulate slow-downs caused by signaling equipment error, longer travel times are introduced to 24 instances without correcting the timetables (Croella et al. 2024). With a timeout of 2 minutes, the study compares the effectiveness of Gurobi Big-M model, Gurobi IAP MILP and MaxSat incremental RC2DDD model. With the number of edges and nodes defined the complexity grows in each iteration, since a new IAP has to be solved (Croella et al. 2024). DDD-TRP solved with MaxSat requires more iterations to finish which could be explained due to merging the number of refinements of solving the IAP and refinements of the objective function. However, both the SAT and MILP versions of DDD spend most of their computational time in the solving phase compared to the small fraction of time of less than 3% on building, refining and conflict checking (Croella et al. 2024). In the case of linear continuous cost function, MaxSat is found to be the fastest approach in 60% of the cases but for the remaining instance Big-M shows a superior performance. Cost representation when differentiating between couple of seconds compared to couple of minutes is expensive for SAT, where MILP shows superiority on handling such instances, solving 60 out of 72 instances to optimality, at the cost of computational speed which is 20 times slower than MaxSat (Croella et al. 2024). For the linear rounded cost function, all 68 out of 72 instances were both solved by Big-M and MaxSat with MaxSat showing a superior speed ranging from 2x to 10x faster solve times. MILP solver on the other hand exceeds the time limit for 10 instances and exibits high computational time, making the MaxSat approach the most favorable (Croella et al. 2024). For the Stepwise objective function, computational times overall are lower with MaxSat being between 2x-20x faster compared to Big-M while solving all the instances. MILP solver again, fails to solve seven instances and exceeds the timeout, making it the worst solver

of all (Croella et al. 2024). When applied to the dynamic real time scheduling, it is observed that the stepwise objective function is the most favorable since it can be solved faster and provide feasible solutions when optimal ones take too long. Compared to Big-M and MILP solvers regarding the stepwise objective function, MaxSat solves hardest instances in far less time, providing anywhere between 4x to 281x speed ups (Croella et al. 2024). When using DDD adaptation for train rescheduling, the overall results show that MaxSat can perform better than the Big-M formulations and outperform it in real-world data and models.

Another study based on 64 real life instances was done by Ojha and Erera (2025) based on a US LTL carriers' L-shaped cross-dock with 44 unloading an57 loading doors where the average speed of forklifts was used to calculate travel times. The planning horizon was split between a 6-hout sunrise-day and 10-hour twilight-night sort, with an average of three- four trailer unloading at the same time. The instances were divided into four categories ranging between extra small all the way to large (Ojha and Erera, 2025). With a randomly drawn deadlines for outbound trailer, the maximum number of workers unloading one trailer was set to three. With relatively coarse discretizations in the early stages of solving DDD, the authors decide to omit solving WDM since the optimality gaps are unreasonably high anyways, hence why WDM is deployed in case the optimality gap is under 2%, using the computational time solving WDM only when the solutions are near-optimal (Ojha and Erera, 2025). With a 60-minute limitation and four different size categories of instances with different parameters, authors of this study compare DDD with FullTI and two heuristics. Result show that when all trailers arrive to the unloading dock at the start of the planning horizon, both FullTI and DDD show weak performance, with FullTI not being able to solve any of the instances to optimality and DDD having worse optimality gaps than FullTI (Ojha and Erera, 2025). Interesting observation is that the upper bounds for M and L sized instances (larger instances) are better than the ones from the FullTI, since solving the linear relaxation in FullTI is computationally expensive, the solver makes only small improvements in large instances. Contrary to the FullTI, DDD has a better time solving first couple of iterations due to a small number of constraints and decision variables, where WDM also often creates good primal solutions of the XDTS-W from the XDTS-W-LB (Ojha and Erera, 2025). When the value of *alpha*, representing a larger spread of trailer arrivals over the early portion of the planning period, increases, all FullTI, DDD and heuristic approaches need less time and have a smaller optimality gap. DDD solves these instances faster to optimality and with maximum of 40% of the nodes in the partial network compared to the FullTI and also outperforms heuristics even for easier instances, where heuristics generate average optimality gaps between 2-13% (Ojha and Erera, 2025). For most of the instances, DDD provides better primal solutions but weaker dual bounds due to smaller number of time points in the partial network compared to the complete network. DDD is also able to solve 10 instances in the first iteration with 0% optimality gap and for some instances with wide enough arrival TW, DDD is performing much faster than FullTI due to smaller optimization model size and number of time points (Ojha and Erera, 2025). While for large *alpha* values DDD is much faster than FullTI, for *alpha*=0 solver needs to prioritize the unloading of some trailers making the instance difficult to solve. Compared to DDD when *alpha*=0, metaheuristics find better solutions due to DDD using a significant time trying to identify time points to be added dynamically instead of choosing these time points at the initialization (Ojha and Erera, 2025). For other instances where the trailer arrival times are more spread out, DDD finds better and optimal solutions faster than metaheuristics in only two to four iterations on average. When solving trailer scheduling, we can conclude that DDD is very effective in most instances compared to both FullTI and metaheuristics and only

struggles when the trailer arrival times are all the beginning of the planning horizon. With fast solving times and generally good optimality gaps and instances solved to optimality, DDD performs great compared to the other methods (Ojha and Erera, 2025).

Apart from their previous study on generated instances, He et al. (2022) also conducted a study on real life data to assess the effectiveness of the DDD over solving the SPP. The study is run on real-world data from a part of Atlanta road network, comprising of 306 nodes and 618 arcs, obtained from Open Street Map. With arc travel times obtained from different streams like navigation system and cell phones, the collected data was aggregated in order to obtain the necessary piecewise linear travel functions with 15-minute breakpoints over a 24-hour period (He et al. 2022). In order to assess the differences, the origin was selected in North Atlanta and the destination in Atlanta's Midtown with the earliest departure at 9 a.m. and latest arrival at 1 p.m. The minimum duration path leaves at 10:37, starting at the time between the morning and afternoon rush hours, with a duration of 45,8 minutes whereas the earliest arrival and latest arrival take 47,5 and 47,8 minutes, respectively, using the street in residential areas instead of interstates (He et al. 2022). Even though the experiment was able to achieve different results for minimum duration and earliest arrival and latest departure, the small difference was attributed to the small section of Atlanta chosen for the experiment, which is also known to have lots of schools and speed limits. Since obtaining the results took a mere matter of second, authors conclude that the algorithm has a great potential in further real-life use (He et al. 2022).

Regarding the TENMR heuristic based on the DDD solving an LSNDP problem, the computational study is based on a 3PL's network with 460 stakeholders across four areas in France, where the transportation fixed costs, linear costs, warehouse capacities and costs, product lines and demand patterns are provided by the 3PL or extracted from some other data (like National Road Committee) (Belieres et al. 2021). The instances are divided between 90 easy instances, generated randomly from 20 stakeholders with two demand seasonalities over a 14-day planning period and smaller time discretizations and radiuses, and 24 difficult instances, based on the whole network with larger discretizations and covering radiuses (Belieres et al. 2021). For the comparison, the study compared solving LSNDP on the complete time expanded network using CPLEX only and its branch and cut method over performing TENMR heuristic before using CPLEX on the reduced program with 5-hour maximum runtime and within 1% optimality gap. Regarding easy instances, all were solved to optimality by CPLEX with a primal gap below 0,5% for over 75% of the instances (Belieres et al. 2021). For 26 of the easy instances, TENMR and CPLEX combined found the optimal solutions, suggesting that solving the reduced formulation has little impact on solution quality. However, as the Δ increased from 1 to 3, CPLEX solving time increased 120 times more , while at the same time using TENMR and CPLEX together increased the solve time only by the factor of 15, meaning that while the TENMR and CPLEX combined didn't solve all instances to optimality the computational time was significantly smaller compared to using only CPLEX (Belieres et al. 2021). While the service radius in has less impact than Δ increase in the easy instances, it can be concluded that TENMR and CPLEX combined produced solutions of comparable quality to the exact method but in a significantly less time. Regarding difficult instances, neither CPLEX alone or the TENMR and CPLEX combined managed to converge to optimality within the 5-hour maximum time limit (Belieres et al. 2021). While CPLEX alone provided very weak optimality gaps, TENMR and CPLEX together provided better overall solution quality with a 26,5 % better on average solution than the one from CPLEX within the given time limit. Even if the time limit for TENMR and CPLEX together was reduced to 30 minutes, it still managed

to provide better results by 9,1% compared to CPLEX alone after 5 hours. This means that for larger industrial instances, it's more favorable to solve a simplified model (Belieres et al. 2021). Regarding the heuristic behaviors, it's reported that it spent most time solving the reduced program, where with the finer discretizations the number of arcs created in phase one of the refinement and time spent increased, but on the other hand, the number of variables decreased compared to the full model , indicating a sparser time expanded network with finer discretizations. During phase two, the heuristic spent around 6 iterations and in the worst case scenarions 4 minutes to construct the sparse network (Belieres et al. 2021). Even though TENMR performed better than CPLEX alone it still didn't solve difficult instances to optimality, hence why the authors created an additional hybrid metaheuristic combining Meta Partial Benders Decomposition with TENMR, which achieved a 4,17% optimality gap compared to 4,77% gap with TENMR alone and computed better upper bounds for 83,3% of instances (Belieres et al. 2021). Regarding delivery paths and operations, with the connectivity radius increasing the percentage of centralized delivery path decreased but even still almost 90% of all shipment still used them, suggesting that the extended strategies do not represent a significant departure from current operations, allowing the company to make significant changes without many network management changes (Belieres et al. 2021). Introduction of alternative delivery path increased the number of vehicles required but it decreased vehicle fill rate and the average distance traveled per truck. These extended distribution strategies resulted in significantly cheaper vehicle fleet utilization and reduction in handling costs but an overall increase in storage costs likely due to combination of different delivery paths and longer storage time that allows consolidation to maintain higher fill rate (Belieres et al. 2021). Finally, from the computational study presented, it can be seen that the reduction metaheuristic does have a positive impact on certain types of instances, their solve time and optimality gaps compared to exact methods.

10. **Conclusion**

This conclusion is a reflection of the comprehensive study of the topic of Dynamic Discretization Discovery algorithm (DDD), summarizing its core methodology, application diversity, key findings and performance and the algorithms limitations and future potential outlook. This thesis serves as a detailed literature review and analysis of the DDD algorithm, which resurfaced as a novel approach to solving continuous-time optimization problems, first introduced in 2017 with its main idea of addressing the computational intractability of fully time-expanded networks. DDD, instead, utilizes a partially time-expanded network that acts as a relaxation of the original problem in order to get the optimal results. DDD is an iterative process which starts by constructing an initial static network, solving a mathematical model to get a lower bound and then uses a refinement procedure to correct short arcs where travel time is underestimated and hence the proposed travel plan impossible. The relaxation strength is achieved through several properties, which ensure that the solution either proves optimal solution or provides a near-optimal or feasible solution through dual bounding. Through the research, DDD has been shown to be very versatile with a wide application domain extending from the original SND problems. DDD has been adapted to other problems and currently it allows TSP to have an efficient solution of time-dependent travel times and makespan objectives, for train scheduling it outperforms traditional solvers in speed, for IRP it solves complex problems by exploiting inventory capacities and visit frequencies and for SPP it uses breakpoints in travel time functions to reach optimality without excessive enumerations. What the literature also shows is that DDD is adaptable to cater to the needs of the specific problems and its individual parts like e.g. refinement procedure can be executed with slight alterations in order to achieve best results. What DDD shows is that if constantly outperforms a full discretization, especially with fine discretization and small intervals since FD often exceeds memory limits and becomes unsolvable in a certain time limit due to its size. On the other hand, DDD keeps the number of variables and constraints manageable by strategically adding only necessary time points and details to the plan when it's really needed for a realistic schedule. In the enhancements section of this thesis, we've seen that some authors like to add additional valid inequalities or preprocessing steps that were proven to have a positive effect on the overall execution of the algorithm.

When it comes to shortcoming and restrictions, despite its success, DDD can have some computational bottlenecks. Majority of the execution time is spent solving an IP that derives the lower bound and in the case of a problem with a large number of instances, DDD can still go over the time limit and result in large optimality gaps. Another restriction of the model is the assumption of zero holding costs which makes the complexity higher when holding costs are introduced, with the formulation sometimes having double the amount of variables in order to reach convergence. DDD algorithm is also very instance and problem sensitive. For example, in trailer scheduling, DDD struggles with the refinement when all trailers arrive at the beginning of the planning horizon and in SNDRP wide time windows significantly reduce the success rate compared to tighter windows. Another restriction is the assumption of FIFO property since without it, the vehicles would just wait in order to arrive earlier later, which makes the relaxation more complicated.

The future of DDD is not so dark and there are some promising ideas that can be explored. Simply moving from node-based to arc-based or interval-based formulations has shown great improvements in runtime and model size and is a valid approach for further research. The

early development of a metaheuristic based on DDD ideas allowed easier handling of large-scale problems that are currently not solvable with exact methods in feasible time. With this in mind, heuristics based on DDD have a promising exploration path. Another DDD potential lies in the real-world application cases. DDD could have a potential of influencing decision making regarding e.g. workforce scheduling and consolidation strategies in next-day shipping and is a potential future research area.

All in all, DDD is a robust and scalable algorithm that can cover a gap between practical solvability and continuous-time models. While certain instances and complex problems might be challenging, DDDs ability to dynamically discover the detailed plan is a good stepping stone for further operational research.

References

Belieres, S.,Hewitt, M.,Jozefowiez, N. & Semet, F. (2021): *A time-expanded network reduction matheuristic for the logistics service network design problem*, Transportation Research Part E: Logistics and Transportation Review, vol. 147(2), 102203

Boland, N., Hewitt, M., Marshall, L. & Savelsbergh, M. (2017): *The Continuous-Time Service Network Design Problem*, Operations Research, vol. 28(5), pp. 1115-1428

Boland, N., Hewitt, M., Vu, D.M. & Savelsbergh, M. (2017b): Solving the traveling salesman problem with time windows through dynamically generated time-expanded networks, In International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, pp. 254- 262. Springer

Boland, N. & Savelsbergh, M. (2019): *Perspectives on integer programming for time-dependent models*, Transactions in Operations Research, vol .27(2), pp. 147-173

Boland, N., Hewitt, M., Marshall, L. & Savelsbergh, M. (2019b): *The price of discretizing time: a study in service network design,* EURO Journal on Transportation and Logistics, vol. 8(2), pp. 195-216

Croella, A. L., Luteberget, B., Mannino, C. & Ventura, P. (2024): *A MaxSAT approach for solving a new Dynamic Discretization Discovery model for train rescheduling problems*, Computers and Operations Research, vol. 167, 106679

Escobar-Vargas, D. and Crainic, T. G. (2024): *Multi-attribute two-echelon location routing: Formulation and dynamic discretization discovery approach*, European Journal of Operational Research, vol. 314(1), pp. 66-78

He, E., Boland, N., Nemhauser, G. & Savelsbergh, M. (2018): *A Dynamic Discretization Discovery Algorithm for the Minimum Duration Time- Dependent Shortest Path Problem*, in book: Integration of Constraint Programming, Artificial Intelligence, and Operations Research (pp. 289-297), Lecture Notes in Computer Science, vol. 10848, Springer

He, E.Y., Boland, N., Nemhauser, G. & Savelsbergh, M. (2022): *Dynamic Discretization discovery algorithms for time-dependent shortest path problems*, Informs Journal of computing, vol. 34(2), pp. 1086-1114

He, Y., Hewitt, M., Lehuede, F. & Medina, J. (2023): *A continuous-time service network design and vehicle routing problem*, Networks, vol. 83(2), pp. 300-323

Hewitt, M. (2019): *Enhanced Dynamic Discretization Discovery for the Continuous Time Load Plan Design Problem*, Transportation Science, vol. 53(6), pp. 1501-1799

Lagos, F., Boland, N. & Savelsbergh, M. (2020): *The Continuous-Time Inventory-Routing Problem*, Transportation Science, vol. 54(2), pp. 299-654

Lagos, F., Boland, N. & Savelsbergh, M. (2022): *Dynamic Discretization discovery for solving the Continuous Time Inventory Routing Problem with Out-and-Back Routes*, Computers an Operations Research, vol. 141, 105686

Marshall, L., Boland, N. & Savelsbergh, M. (2021): *Interval-Based Dynamic Discretization Discovery for Solving the Continuous-Time Service Network Design Problem*, Transportation Science, vol. 55(1), pp. 29-51

Medina, J., Hewitt, M., Lehuédé, F. & Péton, O., (2019): *Integrating long-haul and local transportation planning: the Service Network Design and Routing Problem*, EURO Journal on Transportation and Logistics, vol. 8 (2), pp. 119–145

Ojha, R. & Erera, A. (2025): Cross-Dock Trailer Scheduling with Workforce Constraints: A Dynamic Discretization Discovery Approach

Pottel, S. and Goel, A. (2022): *Scheduling activities with time-dependent durations and resource consumptions*, European Journal of Operational Research, vol. 301(2), pp. 445-457

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

Shu, S., Xu, Z. & Baldacci, R. (2024): *Incorporating Holding Costs in Continuous-Time Service Network Design: New Model, Relaxation, and Exact Algorithm*, Transportation Science, vol. 58(2), pp. 279-556

Van Dyk, M. and Koenemann, J. (2024): *Sparse dynamic discretization discovery via arc-dependent time discretizations*, Computers and Operations Research, vol. 169, 106715

Vu, D. M., Hewitt, M., Boland, N. & Savelsbargh, M. (2019): *Dynamic Discretization Discovery for Solving the Time-Dependant Traveling Salesman Problem with Time Windows*, Transportation Science, vol. 54(3), pp. 703-720

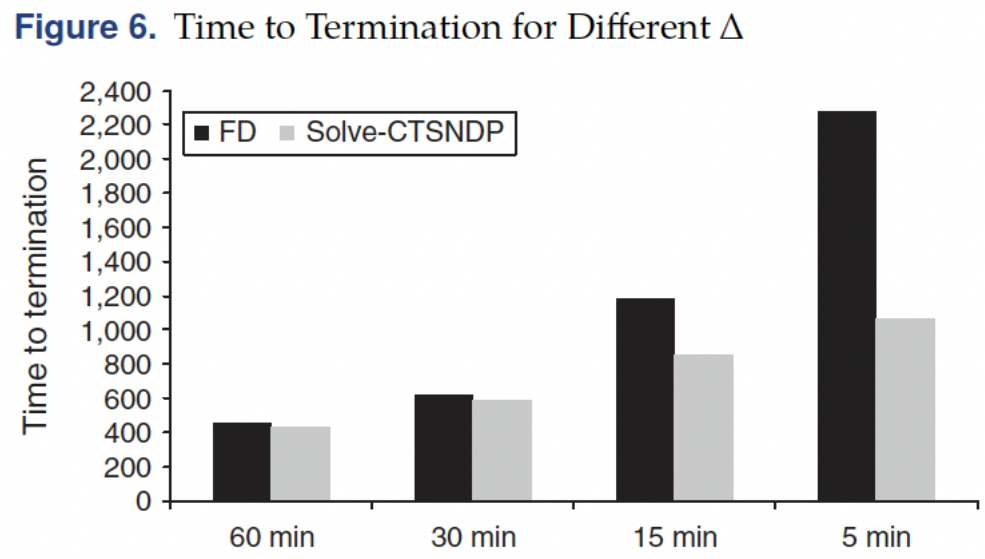
Vu, D. M., Hewitt, M. & Vu, D.D. (2022): *Solving the time dependent minimum tour duration and delivery man problems with dynamic discretization discovery*, European Journal of Operational Research, vol. 302(3), pp. 831-846

Appendix

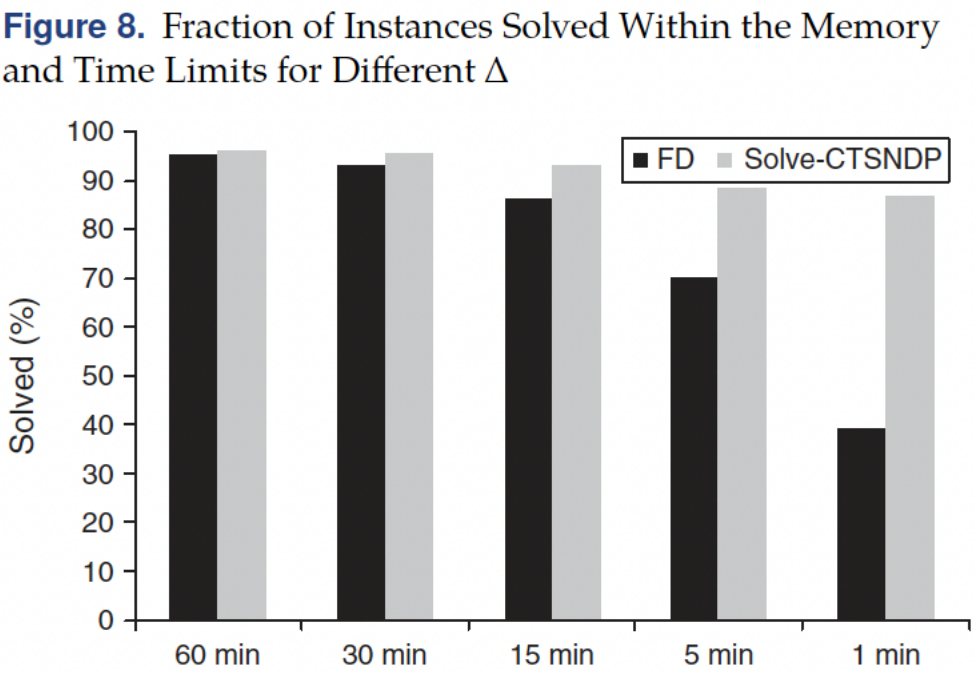
A. Verpflichtende deutsche Zusammenfassung der Arbeit

Das Hauptziel dieser Masterarbeit besteht darin, den Dynamic Discretization Discovery (DDD)-Algorithmus zu untersuchen, der zur Lösung komplexer Optimierungsprobleme in kontinuierlichen Zeitnetzwerken entwickelt wurde. Das zentrale Forschungsanliegen ist eine umfassende Literaturanalyse aller verfügbaren wissenschaftlichen Arbeiten zum Thema DDD sowie die Untersuchung der Vielseitigkeit des Algorithmus in verschiedenen Anwendungsfeldern, darunter Gütertransport, Fahrzeugroutenplanung und industrielle Ablaufplanung. Durch die Analyse unterschiedlicher mathematischer Modellierungen und rechnergestützter Studien zeigt die Arbeit, dass DDD eine hohe Skalierbarkeit und eine schnellere Konvergenz im Vergleich zu kommerziellen Lösungsverfahren bietet sowie insgesamt zu einer effektiven Kostenminimierung und geringeren operativen Lücken in großskaligen logistischen Problemen führt.

B. Appendix- Figures

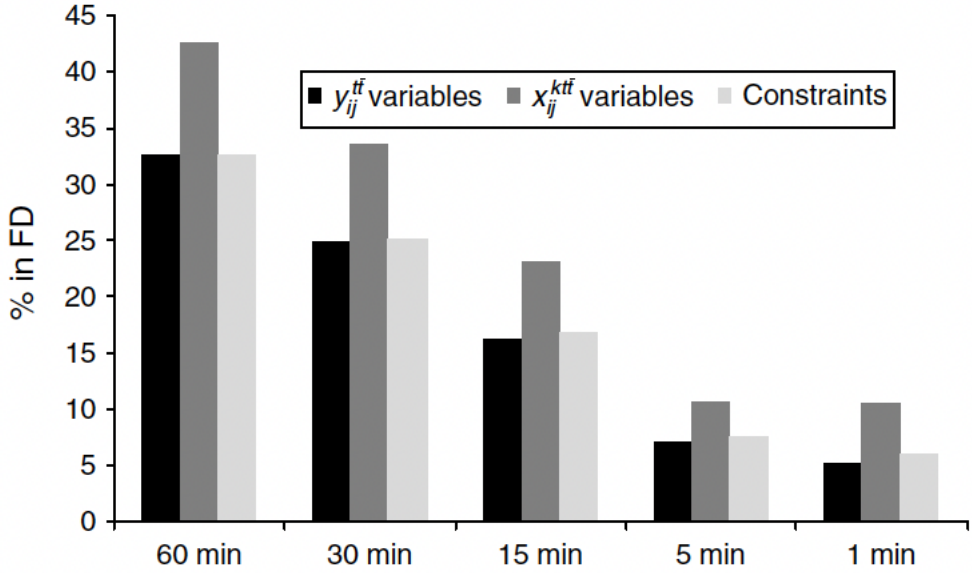


Appendix- Figure 1: Photo from Boland et al. (2017)



Appendix- Figure 2: Photo from Boland et al. (2017)

Figure 12. Relative Integer Programming Size Associated with the Final SND($\mathscr{D}_{\mathcal{T}}$)



Appendix- Figure 3: Photo from Boland et al. (2017)

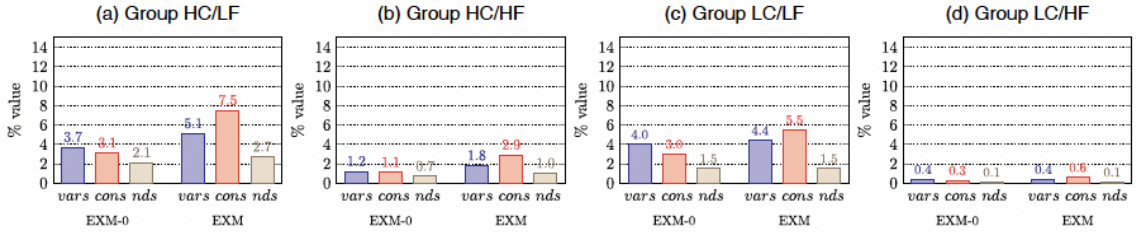
Table 2. Summary Results on the CTSNDP-HC Instances

	Zero holding costs													Nonzero holding costs									
	EXM-0							EXM						EXM									
		%UB0							%UB						%UB						%LB0	%UB1	
Group	%opt	min	max	avg	time	%tLB	iter	%opt	Min	max	avg	time	%tLB	iter	%opt	min	max	avg	time	%tLB	iter	avg	avg	max
HC/LF	96.7	1.1	4.5	3.3	345.3	90.4	10.1	99.5	1.1	1.1	1.1	181.8	86.0	4.2	98.4	1.1	1.9	1.6	279.3	86.6	4.4	4.1	0.8	5.3
HC/HF	71.8	1.0	23.3	7.5	2,726.1	96.3	11.6	81.9	1.1	5.0	2.4	1,936.0	93.6	6.5	65.5	1.0	6.1	2.9	2,902.7	94.4	6.2	10.6	3.9	17.9
LC/LF	100.0	—	—	—	0.5	66.6	3.5	100.0	—	—	—	0.6	68.6	1.8	100.0	—	—	—	0.7	62.3	1.8	0.7	0.0	1.0
LC/HF	100.0	—	—	—	0.1	52.8	1.6	100.0	—	—	—	0.1	68.6	1.2	100.0	—	—	—	0.2	57.0	2.3	0.8	1.1	8.0

Note. "—" represents that all instances in the corresponding instance group are solved to optimal by the corresponding method.

Appendix- Figure 4: Photo from Shu et al. (2024)

Figure 4. (Color online) Comparison of Partially and Fully Time-Expanded Networks



Appendix- Figure 5: Photo from Shu et al. (2024)

Table 3. Performance of Two-Phase DDD on Instances It Could and Could Not Solve

		Phase 1					Phase 2					
Instances	% all instances	Time	Iterations	Opt. gap (%)	Primal opt. gap (%)	$\lvert\mathcal{N}_{\underline{\mathcal{T}}}\rvert$	Time	Iterations	Primal imp. (%)	Dual imp. (%)	Opt. gap (%)	$\lvert\mathcal{N}_{\underline{\mathcal{T}}}\rvert$
Solved	90.50	94.72	7.24	1.77	1.21	629.24	369.41	2.99	0.58	0.57	0.64	656.52
Unsolved	9.50	527.46	11.93	4.47	2.58	963.49	6,517.27	10.93	0.89	1.94	1.71	1,055.80
All	100	135.79	7.69	2.03	1.34	660.97	952.89	3.74	0.61	0.70	0.74	694.42

Note. Opt., optimality; imp., improvement.

Appendix- Figure 6: Photo from Hewitt (2019)

Table 5. Performance by Phase

	Phase 1						Phase 2		
Method	% solved (%)	Time	Iterations	Opt. gap (%)	Primal opt. gap (%)	$\lvert\mathcal{N}_{\underline{\mathcal{T}}}\rvert$	Time	Iterations	$\lvert\mathcal{N}_{\underline{\mathcal{T}}}\rvert$
Two-Phase DDD	23.38	135.79	7.69	2.03	1.34	660.97	952.89	3.74	694.42
Two-Phase DDD + ATW	23.84	131.72	7.30	1.98	1.30	645.57	841.14	4.23	683.27
Two-Phase DDD + ATW + Symmetry-Refine	27.55	107.63	5.60	1.91	1.25	678.76	782.73	3.25	723.90

Note. Opt., optimality.

Appendix- Figure 7: Photo from Hewitt (2019)

Algorithm	Gap %	Time (s)	# Iterations	% Optimal
HC/LF				
Full Discretization	18.28	2,595.03		32.8
Default	3.53	1,751.68	92.6	60.1
Reduced Iterations	1.70	1,386.03	11.2	70.5
Fewer Time Points	1.34	1,258.35	11.2	74.3
Adaptive	0.12	677.76	14.8	85.8
HC/HF				
Full Discretization	49.69	3,290.23		12.4
Default	13.12	2,224.00	53.6	45.2
Reduced Iterations	8.94	2,021.67	10.5	51.4
Fewer Time Points	8.53	1,991.27	11.9	48.6
Adaptive	0.84	1,693.76	17.5	56.5
LC/LF				
Full Discretization	0.00	270.24		95.7
Default	0.00	3.85	28.0	100.0
Reduced Iterations	0.00	1.01	6.0	100.0
Fewer Time Points	0.00	0.67	6.1	100.0
Adaptive	0.00	0.59	6.5	100.0
LC/HF				
Full Discretization	0.00	715.94		93.3
Default	0.00	0.39	7.8	100.0
Reduced Iterations	0.00	0.18	3.4	100.0
Fewer Time Points	0.00	0.12	3.0	100.0
Adaptive	0.00	0.13	3.2	100.0

Table 4 **Computational Results for DDDI Variants vs Full Discretization (1hr solve time limit)**

Appendix- Figure 8: Photo from Marshall et al. (2021)

Algorithm	Gap %	Time (s)	Variables	Constraints	# Iterations	% Optimal
HC/LF						
DDD	0.08	1,391.1	205,540.2	177,149.5	5.3	77.1
DDDI	0.12	677.8	14,013.6	25,757.0	14.8	85.8
HC/HF						
DDD	0.56	1,966.7	161,097.2	128,888.2	6.0	53.7
DDDI	0.84	1,693.8	13,044.6	23,496.2	17.5	56.5
LC/LF						
DDD	0.00	28.6	20,633.5	27,177.9	3.7	100.0
DDDI	0.00	0.6	1,382.4	2,535.4	6.5	100.0
LC/HF						
DDD	0.00	1.5	4,500.3	6,917.1	2.5	100.0
DDDI	0.00	0.1	376.7	639.0	3.2	100.0

Table 6 Computational Results DDD vs DDDI (1hr solve time limit)

Appendix- Figure 9: Photo from Marshall et al. (2021)

				FD			DDD			DDDI		
States	$\|N\|$	$\|A\|$	$\|K\|$	Time (s)	Value	Gap (%)	Time (s)	Value	Gap (%)	Time (s)	Value	Gap (%)
ID,MT,OR,WA	10	54	224	213	1,503,240	0.94	30	1,507,222	0.92	5	1,510,629	0.93
CO,ID,MT,OR,WA	14	81	341	7,200	1,780,041	3.63	7,200	1,757,287	1.68	78	1,757,395	0.69
CO,ID,MT,OR,WA,NV	16	109	469	7,200	2,939,430	25.28	7,200	2,291,691	2.74	803	2,273,707	0.65
CO,ID,MT,OR,WA,UT	15	104	458	7,200	2,805,518	19.71	7,200	2,325,602	1.45	223	2,320,710	0.78
ID,MT,OR,WA,NV,UT	13	97	429	7,200	3,130,964	23.78	7,200	2,501,827	3.00	137	2,493,752	0.89

Table 7 Performance on instances derived from U.S. LTL carrier

Appendix- Figure 10: Photo from Marshall et al. (2021)

Table 4
Runtime and optimality gap comparison for designated path instances.

Decile	Ave. total runtime (s)			Ave. optimality gap		
	Arc-based	Node-based	% improvement	Arc-based	Node-based	% improvement
0.1	42	138	69.2%	<1%	<1%	N/A
0.2	75	239	68.5%	<1%	<1%	N/A
0.3	128	437	70.6%	<1%	<1%	N/A
0.4	173	629	72.4%	<1%	<1%	N/A
0.5	250	833	70.0%	<1%	<1%	N/A
0.6	451	1,188	62.0%	<1%	<1%	N/A
0.7	727	1,832	60.3%	<1%	<1%	N/A
0.8	1,252	3,328	62.4%	<1%	<1%	N/A
0.9	3,142	7,247	56.6%	<1%	<1%	N/A
1	10,800	10,800	N/A	5.87%	8.49%	30.8%

Appendix- Figure 11: Photo from Van Dyk and Koenemann (2024)

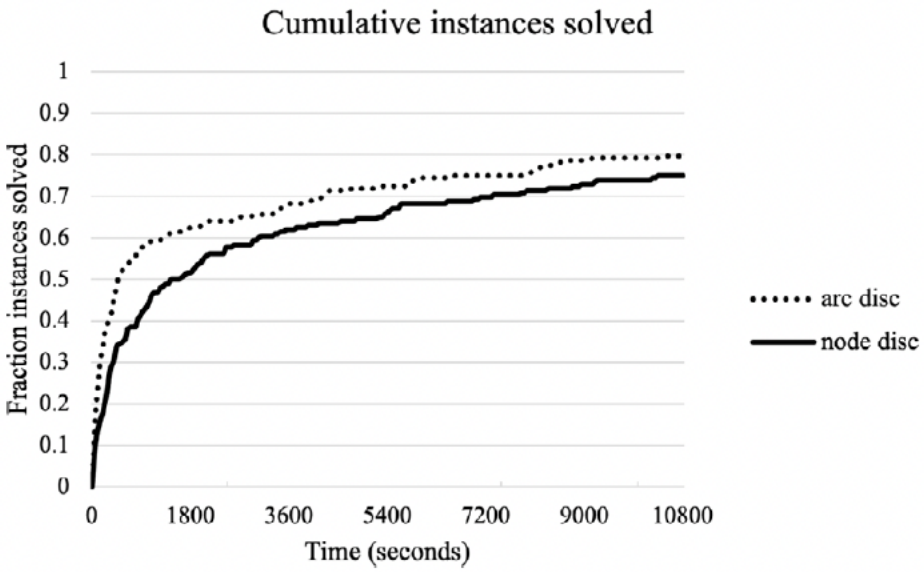


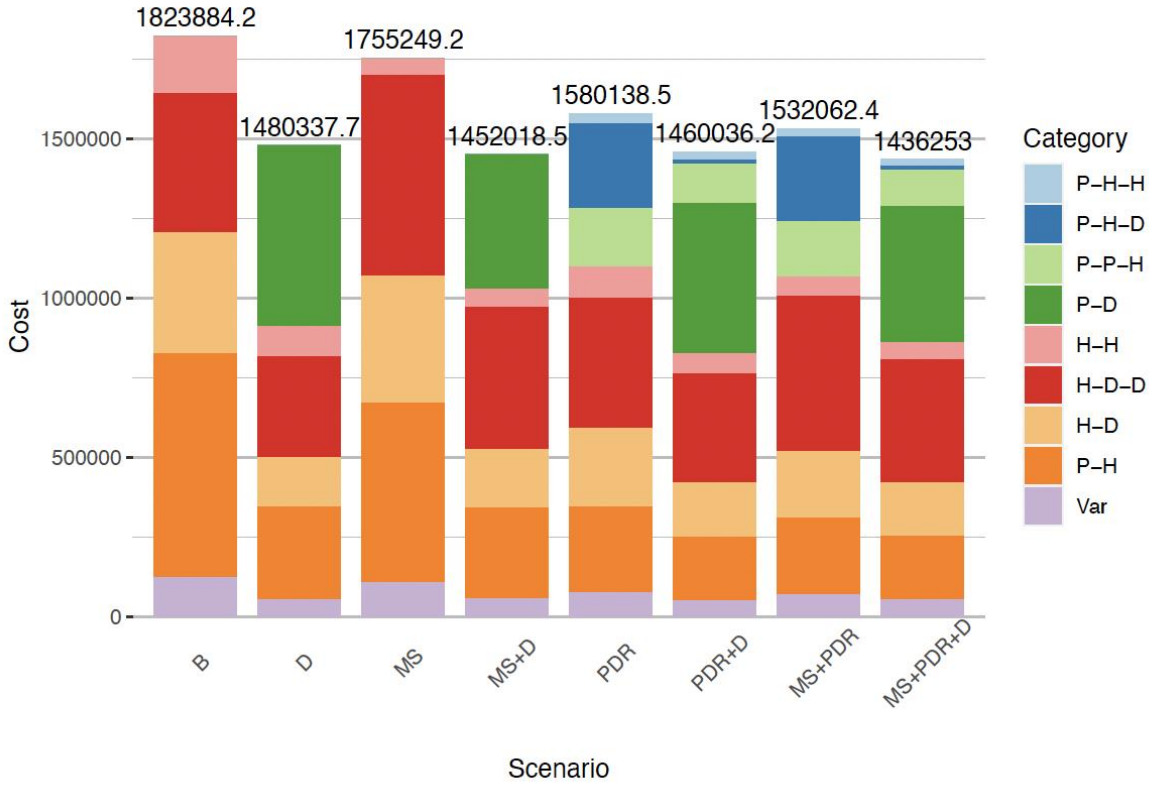
Fig. 14. Fraction of instances solved.

Appendix- Figure 12: Photo from Van Dyk and Koenemann (2024)

Table 12
Runtime and optimality gap comparison for instances with critical times.

Decile	Ave. total runtime (s)			Ave. optimality gap		
	Arc-based	Node-based	% improvement	Arc-based	Node-based	% improvement
0.1	72	91	20.6%	<1%	<1%	N/A
0.2	117	153	23.6%	<1%	<1%	N/A
0.3	192	247	20.0%	<1%	<1%	N/A
0.4	312	375	16.7%	<1%	<1%	N/A
0.5	420	485	13.4%	<1%	<1%	N/A
0.6	701	867	19.2%	<1%	<1%	N/A
0.7	1,131	1,256	10.0%	<1%	<1%	N/A
0.8	2,330	2,775	16.0%	<1%	<1%	N/A
0.9	7,328	7,469	1.9%	<1%	<1%	N/A
1	10,800	10,800	N/A	44.6%	36.3%	−22.9%

Appendix- Figure 13: Photo from Van Dyk and Koenemann (2024)



Appendix- Figure 14: Photo about cost distribution for each scenario from He et al. (2020)

Scenario	CPLEX				DDD				avg
	nbFeas	nbOpt	avgGap	avgT	nbFeas	nbOpt	avgGap	avgT	UBGap
B	34	2	13.9%	6776	37	27	1.2%	2819	3.3%
D	34	4	12.4%	6389	37	28	1.0%	2394	2.9%
MS	28	3	24.9%	6515	37	19	5.5%	4043	19.9%
MS+D	29	3	24.3%	6547	37	19	4.4%	4117	15.8%
PDR	30	3	15.9%	6519	37	17	3.3%	4697	3.2%
PDR+D	29	3	16.2%	6496	37	22	2.9%	3773	4.7%
MS+PDR	20	3	33.2%	6326	37	7	10.4%	6056	48.3%
MS+PDR+D	20	3	24.8%	6261	37	10	10.0%	5548	15.6%
Average	29.25	3.00	20.70%	6,478.63	37.00	18.63	4.84%	4,180.88	14.21%

Table 1: Performance analysis - comparison with CPLEX

Appendix- Figure 15: Photo from He et al. (2020)

Table 2 Computational results with the arc-based formulation ($\mathcal{F}_A$)

# Commodities	# CRCs	% Feasible	% Optimal	Time to optimal (s)	Optimality gap (%)
25	4	100	100	7	0.44
	6	100	100	8	0.42
	8	100	100	13	0.49
	10	100	100	20	0.37
	Average	100	100	12	0.43
50	4	100	100	75	0.51
	6	100	90	680	2.55
	8	100	80	1119	1.07
	10	100	60	808	1.75
	Average	100	82.50	670.5	1.47
100	4	100	80	1398	0.59
	6	100	40	56	3.15
	8	100	45	89	4.64
	10	100	45	145	7.54
	Average	100	52.5	422	3.98
200	4	100	50	901	1.55
	6	100	40	402	3.60
	8	100	45	670	8.49
	10	100	45	1223	13.59
	Average	100	45	799	6.81

Appendix- Figure 16: Photo from Medina et al. (2019)

Table 5
Results for the exact algorithms.

Algorithm	Runtime	#Iter.	|A|	Gap	Sol. qual.	%Opt. 5 h	%Opt. 1 h
Gurobi	8,412 s	–	3204.60	1.47%	–	82.00%	32.00%
1-DDD	5,163 s	14.82	725.86	1.06%	−0.24%	86.00%	62.00%
2-DDD	3,109 s	10.16	964.98	0.88%	−0.36%	90.00%	80.00%
R-DDD	4,768 s	13.66	840.30	1.06%	−0.29%	86.00%	70.00%

Table 6
Results for variants of two-phase DDD with different components.

Algorithm	Phase	Runtime	#Iter.	|A|	Gap	Sol. qual.	%Opt. 5 h
2-DDD	1+2	3,109 s	10.16	964.98	0.88%	−0.36%	90.00%
	1	90 s	6.13	773.58	–	−6.65%	–
2-DDD w/o VI	1+2	3,705 s	10.08	956.14	0.95%	−0.30%	88.00%
	1	119 s	6.48	831.28	–	−6.37%	–
2-DDD w/ cap	1+2	3,694 s	9.62	976.38	0.92%	−0.35%	90.00%
	1	122 s	5.60	849.46	–	−6.08%	–

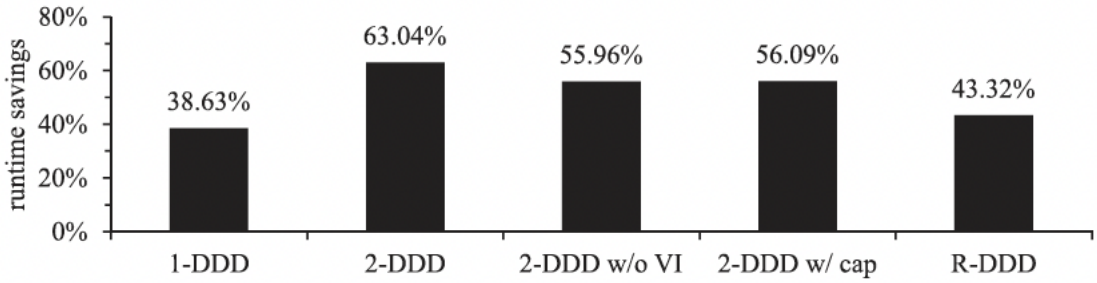


Fig. 8. Runtime savings of DDD algorithms compared to Gurobi.

Appendix- Figure 17: Photo from Scherr et al. (2023)

Table 2. Time-expanded network size.

Instances	Full time-expanded network		Final partially time-expanded network	
	#Nodes	#Arcs	#Nodes	#Arcs
AFG-Easy	69 k	1,336 k	232	3,549
AFG-Hard	133 k	10,441 k	2,190	50,100
DDGS	3 k	185 k	400	4,773
SU	45 k	7,195 k	1,292	8,512
SPGPR	107,672 k	2,631,490 k	564	14,308

Appendix- Figure 18: Photo from Boland et al. (2017b)

Instance set	Traffic pattern	TTBF-BC			DDD-TD-TSPTW		
		Inst	Slv	Time	Inst	Slv	Time
Set 1	A	478	470	31.99	478	478	12.91
	B	474	470	20.44	474	474	8.79
Set *w100*	A	108	107	100.13	480	480	1.31
	B	108	107	93.10	480	480	1.14

Table 1 Performance on Arigliano et al. (2015) instances

Appendix- Figure 19: Photo from Vu et al. (2019)

Table 3
Results averaged over 29 instances of type C1, R1, and RC1.

Solver	Charging	Avg. CPU	CPU per Evaluation	Avg. Veh.	Avg. Compl.	Replenishments	Routes w. Repl.	Terminated
DDD	none	82.4	0.000182	18.0	5949.4	0	0	29
DDD	Depot	93.9	0.000161	18.0	5948.2	6	6	29
DDD	Depot + 1 per city	134.8	0.000203	18.1	5958.3	9	9	29
DDD	Depot + 3 per city	135.3	0.000186	18.0	5954.8	10	10	29
DDD	Depot + 5 per city	146.9	0.000198	18.0	5960.5	10	10	29
DDD-PL	none	72.6	0.000162	18.0	5949.4	0	0	29
DDD-PL	Depot	75.8	0.000128	18.0	5948.2	6	6	29
DDD-PL	Depot + 1 per city	84.0	0.000127	18.1	5958.3	9	9	29
DDD-PL	Depot + 3 per city	81.1	0.000113	18.0	5954.8	10	10	29
DDD-PL	Depot + 5 per city	88.4	0.000121	18.0	5960.5	10	10	29
MIP	none	2508.0	0.005232	18.1	5950.6	0	0	29
MIP	Depot	3299.9	0.005505	18.4	5752.5	4	4	28
MIP	Depot + 1 per city	3621.5	0.005620	18.4	5753.5	7	7	28
MIP	Depot + 3 per city	3345.9	0.005407	19.3	5622.5	5	5	23
MIP	Depot + 5 per city	3282.6	0.005342	19.5	5753.8	6	6	22

Table 4
Results averaged over 27 instances of type C2, R2, and RC2.

Solver	Charging	Avg. CPU	CPU per Evaluation	Avg. Veh.	Avg. Compl.	Replenishments	Routes w. Repl.	Terminated
DDD	none	718.2	0.001021	9.6	8386.9	0	0	27
DDD	Depot	934.5	0.000802	6.7	6936.7	188	124	27
DDD	Depot + 1 per city	1254.9	0.000853	6.8	6884.8	190	122	27
DDD	Depot + 3 per city	2021.6	0.000739	6.2	6430.8	189	113	26
DDD	Depot + 5 per city	2306.4	0.000761	6.3	6858.1	172	105	23
DDD-PL	none	721.2	0.001020	9.6	8386.9	0	0	27
DDD-PL	Depot	640.4	0.000552	6.7	6936.7	188	124	27
DDD-PL	Depot + 1 per city	739.2	0.000513	6.8	6884.8	190	122	27
DDD-PL	Depot + 3 per city	1030.3	0.000351	6.2	6654.4	197	117	27
DDD-PL	Depot + 5 per city	1045.7	0.000305	6.2	6854.9	194	116	26
MIP	none	3729.6	0.005853	10.0	9056.6	0	0	21
MIP	Depot	4877.2	0.005443	7.1	8100.1	55	40	8
MIP	Depot + 1 per city	5542.7	0.005701	6.2	7588.0	29	25	5
MIP	Depot + 3 per city	6725.1	0.005562	4.0	10778.2	1	1	1
MIP	Depot + 5 per city	6230.4	0.005494	4.0	10746.7	3	3	1

Appendix- Figure 20: Photo from Pottel and Goel (2022)

Table 1
Performance of the hybrid time–space formulation.

Instances				Hybrid								
				$\bar{\Delta}$					$\Delta = 50$		$\Delta = 25$	
$\lvert\mathcal{P}^{ph}\rvert$	$\lvert\mathcal{Z}^{ph}\rvert$	OD	NI	FUB	OUB	CPUsec	RG (%)	OG (%)	Dif UB	Dif CPUsec	Dif UB	Dif CPUsec
2	3	5	20	20	20	4561.33	16.38	0.00	0.71	49.70	1.54	83.94
3	5	5	20	20	20	3759.97	18.34	0.00	0.31	39.91	0.84	72.72
6	4	5	20	20	20	3118.99	21.99	0.00	1.38	40.05	3.19	75.45
2	3	10	20	20	15	4356.59	16.88	4.14	2.24	67.19	2.71	92.60
3	5	10	20	20	15	8586.08	17.86	9.13	2.79	64.20	3.28	87.36
6	4	10	20	20	15	8960.59	18.84	9.61	1.44	66.34	1.72	86.11
2	3	15	20	11	0	9000.00	36.43	29.30	10.41	4.31	11.78	36.83
3	5	15	20	0	0	9000.00	31.01	24.13	N.A	0.00	N.A	31.89
6	4	15	20	0	0	9000.00	37.85	31.19	N.A	0.00	N.A	33.19

Table 2
Performance of the standalone time–space formulation.

Instances				Standalone								
				$\bar{\Delta}$					$\Delta = 50$		$\Delta = 25$	
$\lvert\mathcal{P}^{ph}\rvert$	$\lvert\mathcal{Z}^{ph}\rvert$	OD	NI	FUB	OUB	CPUsec	RG(%)	OG(%)	Dif UB	Dif CPUsec	Dif UB	Dif CPUsec
2	3	5	20	20	20	1521.48	19.79	0.00	1.17	25.81	1.95	46.96
3	5	5	20	20	20	1290.38	21.55	0.00	0.86	26.36	1.22	45.24
6	4	5	20	20	20	2306.77	30.51	0.00	2.73	25.90	3.48	54.74
2	3	10	20	20	14	3208.01	21.57	3.19	2.54	71.70	3.07	81.01
3	5	10	20	20	14	4897.51	21.25	4.32	3.10	75.26	3.78	83.08
6	4	10	20	20	15	5255.78	21.94	5.03	1.64	74.93	2.14	84.85
2	3	15	20	17	1	9000.00	37.50	27.76	3.80	41.00	5.21	76.97
3	5	15	20	17	0	8961.78	31.59	23.06	3.71	39.48	4.66	79.89
6	4	15	20	16	0	9000.00	38.94	31.15	3.92	37.83	5.05	73.88

Appendix- Figure 21: Photo from Escobar-Vargas and Crainic (2024)

Table 1
Results summary.

	N			U			G			P		
	10	20	30	1	2	3	1	2	3	25	50	75
Optimal	80	69	57	80	72	54	79	68	59	68	72	66
Feasible	1	10	14	0	8	17	2	9	14	5	8	12
No Solution	0	2	10	1	1	10	0	4	8	8	1	3
Vehicles	7.83	15.47	23.64	13.36	15.67	17.91	15.99	15.65	15.30	15.15	15.67	16.12
Visits	13.52	25.86	37.92	21.85	25.61	28.63	25.85	24.90	24.92	22.79	25.41	27.35
Time Points	23.22	36.07	42.09	34.43	33.16	33.79	29.20	33.61	38.56	37.42	32.61	31.35
Initial Points (%)	27.45	17.54	13.60	17.16	19.59	21.84	21.67	19.47	17.45	18.99	20.16	19.43

Appendix- Figure 22: Photo from Lagos es al. (2022)

Table 2. MDP Results

T	ntype	ttype	BP	Total number of BP	Percentage of BP	Time DDD, ms	Time enum, ms	Percentage time	Number of arcs
20	C	1	37.4	933	4.0	13.3	197.0	6.8	7.5
40	C	1	49.0	1,913	2.6	17.6	389.4	4.5	7.6
60	C	1	57.3	2,893	2.0	20.6	587.4	3.5	7.6
80	C	1	68.4	3,873	1.8	25.0	789.0	3.2	7.6
100	C	1	73.7	4,853	1.5	28.3	1,010.6	2.8	7.6
20	C	2	51.6	933	5.5	18.4	214.7	8.6	7.5
40	C	2	54.9	1,913	2.9	19.1	423.3	4.5	7.5
60	C	2	64.9	2,893	2.2	23.9	636.7	3.8	7.5
80	C	2	71.1	3,873	1.8	25.8	860.9	3.0	7.5
100	C	2	83.0	4,853	1.7	31.7	1,079.0	2.9	7.5
20	G	1	48.6	933	5.2	8.2	110.2	7.4	13.0
40	G	1	53.9	1,913	2.8	9.2	217.9	4.2	13.0
60	G	1	71.4	2,893	2.5	12.4	331.6	3.7	13.0
80	G	1	80.8	3,873	2.1	14.0	444.9	3.1	13.0
100	G	1	91.0	4,853	1.9	16.0	558.3	2.9	13.0
20	G	2	47.6	933	5.1	8.3	119.1	7.0	13.0
40	G	2	58.6	1,913	3.1	9.8	238.0	4.1	13.0
60	G	2	71.7	2,893	2.5	12.4	362.3	3.4	13.0
80	G	2	82.2	3,873	2.1	14.1	487.2	2.9	13.0
100	G	2	94.9	4,853	2.0	16.6	609.9	2.7	13.0
20	T	1	47.9	933	5.1	15.2	162.7	9.3	9.0
40	T	1	59.1	1,913	3.1	18.1	321.1	5.6	9.0
60	T	1	69.7	2,893	2.4	21.0	488.3	4.3	9.0
80	T	1	81.5	3,873	2.1	25.0	653.8	3.8	9.0
100	T	1	94.8	4,853	2.0	28.2	822.7	3.4	9.0
20	T	2	42.8	933	4.6	12.6	173.8	7.2	8.9
40	T	2	53.7	1,913	2.8	16.4	349.0	4.7	8.9
60	T	2	66.3	2,893	2.3	20.2	531.6	3.8	8.9
80	T	2	81.9	3,873	2.1	26.2	722.7	3.6	8.9
100	T	2	87.8	4,853	1.8	26.6	902.2	2.9	8.9

Appendix- Figure 23: Photo from He es al. (2022)

Table 3. MTTP Results

									Optimal path	
T	ntype	ttype	BP	Total number of BP	Percentage of BP	Time DDD, ms	Time enum, ms	Percentage of time	Number of arcs	Number of subpaths
20	C	1	258.2	933	27.7	850.7	562.0	151.4	7.6	3.9
40	C	1	415.7	1,913	21.7	1,240.9	1,240.8	100.0	7.6	4.1
60	C	1	581.8	2,893	20.1	1,782.6	2,154.2	82.8	7.6	3.8
80	C	1	750.0	3,873	19.4	2,433.1	3,218.9	75.6	7.6	3.6
100	C	1	916.8	4,853	18.9	3,092.9	4,398.8	70.3	7.6	4.1
20	C	2	257.2	933	27.6	728.7	562.4	129.6	7.5	4.9
40	C	2	421.7	1,913	22.0	1,146.3	1,325.9	86.5	7.5	5.5
60	C	2	592.5	2,893	20.5	1,633.6	2,325.8	70.2	7.5	5.2
80	C	2	733.9	3,873	18.9	1,979.0	3,491.6	56.7	7.5	5.2
100	C	2	897.7	4,853	18.5	2,512.4	4,724.3	53.2	7.5	5.3
20	G	1	364.4	933	39.1	1,793.1	494.1	362.9	13.0	4.6
40	G	1	581.6	1,913	30.4	2,506.2	1,228.1	204.1	13.0	4.6
60	G	1	905.9	2,893	31.3	4,657.2	2,198.9	211.8	13.0	4.1
80	G	1	1,185.9	3,873	30.6	6,625.8	3,348.4	197.9	13.0	3.7
100	G	1	1,364.5	4,853	28.1	7,253.0	4,623.4	156.9	13.0	4.0
20	G	2	406.8	933	43.6	2,141.4	534.3	400.8	13.0	5.9
40	G	2	582.3	1,913	30.4	2,149.0	1,318.4	163.0	13.0	6.7
60	G	2	809.5	2,893	28.0	3,014.4	2,377.8	126.8	13.0	6.4
80	G	2	1,055.0	3,873	27.2	4,078.2	3,617.7	112.7	13.0	6.2
100	G	2	1,301.0	4,853	26.8	5,239.4	4,988.0	105.0	13.0	6.5
20	T	1	277.2	933	29.7	1,060.3	476.9	222.3	8.7	4.6
40	T	1	430.7	1,913	22.5	1,351.2	1,164.4	116.0	8.7	4.4
60	T	1	619.8	2,893	21.4	2,124.4	2,068.7	102.7	8.7	3.8
80	T	1	800.3	3,873	20.7	2,878.3	3,123.8	92.1	8.7	3.8
100	T	1	982.0	4,853	20.2	3,686.9	4,267.5	86.4	8.7	4.2
20	T	2	304.0	933	32.6	1,226.8	509.6	240.7	8.6	4.9
40	T	2	456.6	1,913	23.9	1,362.5	1,255.5	108.5	8.7	5.1
60	T	2	621.8	2,893	21.5	1,841.9	2,216.4	83.1	8.7	5.1
80	T	2	801.9	3,873	20.7	2,474.0	3,344.6	74.0	8.7	5.0
100	T	2	991.2	4,853	20.4	3,216.1	4,550.3	70.7	8.7	5.4

Appendix- Figure 24: Photo from He es al. (2022)